Devuelve los Pokémon con altura entre 0.5 y 1.7 metros, mostrando nombre, poder y elemento.

```
db.pokemons.find(
 { altura_m: { $gte: 0.5, $lte: 1.7 } },
 { nombre: true, poder: true, elemento: true, _id: false }
)
```

Muestra solo el elemento de los Pokémon en ese rango de altura.

```
db.pokemons.find(
 { altura_m: { $gte: 0.5, $lte: 1.7 } },
 { _id: false, elemento: true }
)
```

Muestra nombre, altura y peso de Pokémon con altura menor a 1 y peso mayor a 5.

$and: devuelve documentos que cumplen todas las condiciones.

```
db.pokemons.find({$and: [
 { altura_m: {$lt :1}},        { peso_kg: {$gt : 5}} ]}, {nombre:true, altura_m:true, peso_kg:true, _id:false})
```

Devuelve los Pokémon que tengan un poder menor a 50 **o** cuyo elemento sea "fuego".

$or: cumple **al menos una** de las condiciones

```
db.pokemons.find({ $or: [{ nivel_poder: { $lt: 50 } }, { elemento: "fuego" }]},{ nombre: true, elemento: true, nivel_poder: true })
```

```
db.pokemons.find({ nombre: /^S/ })       // Comienza con 'S'
db.pokemons.find({ nombre: /ff$/ })      // Termina en 'ff'
db.pokemons.find({ nombre: /ba/ })       // Contiene 'ba'
```

Devuelve Pokémon cuyo elemento **es uno de los indicados**. `$in`: es como un "OR múltiple".

db.pokemons.find({ elemento: { $in: ['fuego', 'planta', 'agua'] } })

Devuelve Pokémon **que no tienen esos elementos**. $nin: es el “no está en”.

db.pokemons.find({ elemento: { $nin: ['fuego', 'planta', 'agua'] } })

Muestra los Pokémon que tienen el campo vidas $exists: verifica la existencia de un campo.

```
db.pokemons.find({ vidas: { $exists: true } })
```

Muestra Pokémon cuyo campo fechanac es de tipo fecha.

```
db.pokemons.find({ fechanac: { $type: 'date' } })
```

Actualiza el peso a 46 donde el elemento es "normal". $set: cambia o agrega un valor a un campo.

```
db.pokemons.updateOne({ elemento: 'normal' },{ $set: { peso_kg: 46 } })
```

Suma 0.1 a la altura si es menor a 0.5. $inc: incrementa numéricamente.

```
db.pokemons.updateOne({ altura_m: { $lt: 0.5 } },{ $inc : { altura_m: 0.1 } })
```

Elimina el campo fechanac si existe. $unset: elimina campos.db.pokemons.updateOne({ fechanac: { $exists: true } },{ $unset: { fechanac: true } })

Si no existe el Pokémon charmander, lo crea con ese campo.

upsert: mezcla update + insert.

```
db.pokemons.updateOne({ nombre: 'charmander' },{ $set: { elemento: 'fuego' } },{ upsert: true })
```

Inserta 50 documentos de prueba en la colección demos

```
for(i=1; i<=50; i++){db.demos.insertOne({test: 'test'+i})}
```

Devuelve los Pokémon que aparecen en el juego "Red".

```
db.pokemons.find({ games: "Red" }, { nombre: 1, games: 1, _id: 0 })
```

Muestra una lista de elementos diferentes (eléctrico, fuego, etc.).

```
db.pokemons.distinct("elemento")
```

Calcula el promedio del array scores para cada Pokémon.$avg: operador de agregación para promediar arrays.

```
db.pokemons.aggregate([{$project: {nombre: 1,promedioScore: { $avg: "$scores" }}}])
```

Filtra Pokémon que tengan scores.

Toma solo 1 documento ($limit: 1).

Crea un nuevo arreglo newScores, donde cada score original se **multiplica por 5**.

$match: filtra documentos (como WHERE).

$project: selecciona campos o crea nuevos.

$map: recorre un array y transforma sus elementos.

$multiply: operador matemático.

```
db.pokemons.aggregate([{ $match: { scores: { $exists: true } } },{ $project: { _id: false, nombre: true, scores: true } },{ $limit: 1 },{$project: {newScores: {$map: {input: "$scores",as: "score",in: { $multiply: ["$$score", 5] }}}}}])
```

Filtra Pokémon tipo **fuego** que tengan scores.

Muestra su nombre y un nuevo campo newScores con los valores multiplicados por 3.

```
db.pokemons.aggregate([

  { $match: { scores: { $exists: true }, elemento: 'fuego' } },

  { $project: { _id: false, nombre: true, scores: true } },
```

```
  {$project: {nombre: true,newScores: {$map: {input: "$scores",as:
"score",in: { $multiply: ["$$score", 3] }}}}])
```

Relación **uno a uno**, una persona con una dirección

```
// Uno a uno

const user = {name: 'Jhon Doe',email: 'jhon@example.com',address:
{city: 'PNA',postalCode: '543050',complement: 'Calle 5 No 123'}}
```

Relación **uno a muchos**: un usuario con varias direcciones.

```
// Uno a muchos

const user = {name: 'Jhon Doe',email: 'jhon@example.com',address: [

{ city: 'PNA', postalCode: '543050', complement: 'Calle 5 No 123'
},{ city: 'CUCUTA', postalCode: '3050', complement: 'AV 0 111'}]}
```

Simula una relación **muchos a muchos** con referencias (ObjectId) a otra colección.

```
// Muchos a muchos (referencias)

const user = {name: 'Jhon Doe',email: 'jhon@example.com',address:
[ObjectId('68479aae49af3fb892d796bc'),...]}
```

Crea una nueva especie.

```
const specie = {name: 'especie1',summary: 'descripcion de especie
1'}db.species.insertOne(specie)
```

Asigna la especie a Pikachu con el campo species_id.

```
db.pokemons.updateOne({ nombre: 'Pikachu' },{ $set: { species_id:
ObjectId('685354918a83145b44abe9e8') } })
```

Asigna la especie a todos los Pokémon que aún **no** tienen species_id.

```
db.pokemons.updateOne({ species_id: { $exists: false } },{ $set: {
species_id: ObjectId('685354918a83145b44abe9e8') } })
```

Crea un índice compuesto para optimizar búsquedas por elemento y species_id.

```
db.pokemons.createIndex({elemento: 1,species_id: 1})
```

Une la colección pokemons con species, como un **JOIN** en SQL.

Devuelve los Pokémon con su species_id y el detalle de la especie en un array especies.

$lookup: Hace una especie de JOIN entre colecciones en MongoDB.

```
db.pokemons.aggregate([{$lookup: {from: 'species',localField: 'species_id',foreignField: '_id',as: 'especies'}},{$project: {_id: false,nombre: true,species_id: true,especies: true}}])
```

Devuelve Pokémon que tienen al menos 5 juegos en el array games.$where: permite usar código JavaScript (menos eficiente, pero poderoso).$and: combina condiciones.

```
db.pokemons.find({$and: [{ games: { $exists: true }},{ $where: 'this.games.length >= 5' }]},{nombre: true,games: true})
```

Devuelve solo una porción del array games desde el índice 3, máximo 5 elementos (aunque probablemente solo tenga menos).$slice: útil para paginar arrays.

```
db.pokemons.findOne({ nombre: 'Pikachu' },{_id: false,nombre: true,games: { $slice: [3, 5] }})
```

Cambia el primer valor 5 que aparece en el array scores a 15.'scores.$': operador **posicional**, actúa sobre el primer valor que coincide con la condición.

```
db.pokemons.updateMany({ scores: { $exists: true }, scores: 5 },{ $set: { 'scores.$': 15 } })
```

Actualizar valor del array por indice

Reemplaza el **segundo juego** (índice 1) con 'Soulsilver'.

```
db.pokemons.updateOne({ nombre: "Charizard" },{ $set: { 'games.1': 'Soulsilver' } })
```

Eliminar elementos especificos de un array

Elimina 'black' y 'white' si están presentes en el array games.

$pull: elimina coincidencias.
$in: compara múltiples valores.

```
db.pokemons.updateOne({ nombre: "Charizard" },{ $pull: { games: { $in: ['black', 'white'] } } })
```

Reemplazar todo el array

```
db.pokemons.updateMany({ nombre: "Pikachu" },{ $set: { scores: [7] } })
```

Añade varios elementos al array y luego los ordena.

$push con $each y $sort: útil para mantener arrays ordenados.

```
db.pokemons.updateMany({ nombre: "Pikachu" },{$push: {scores: {$each: [5, 3, 11, 9],$sort: 1}}})
```

Elimina completamente el campo scores del documento.

```
db.pokemons.updateMany({ nombre: "Pikachu" },{ $unset: { scores: true } })
```

Agregar juego a bulbasar

Añade 'Yellow' al array games.

```
db.pokemons.updateOne({ nombre: 'Bulbasaur' },{ $push: { games: 'Yellow' } })
```

$pop se utiliza para eliminar elementos de un array en un documento. Con $pop, puedes elegir si eliminar el primer elemento del array o el último. Se utiliza -1 para eliminar el primer elemento y 1 para eliminar el último.

```
db.pokemons.updateOne({ nombre: 'Charizard' },{ $pop: { games: 1 } })
```

Busca Pokémon que tengan **ambos juegos**: 'Red' y 'Ruby'.

```
db.pokemons.find({games: { $all: ['Red', 'Ruby'] }})
```

Insertar scores donde no existan

Añade el array scores si no existe y el poder es mayor a 70.

```
db.pokemons.updateMany({$and: [{ nivelPoder: { $gt: 70 } },{ scores: { $exists: false } }]},{ $set: { scores: [7, 8, 9] } })
```

Renombrar campo

```
db.pokemons.updateMany({},{$rename: {nivel_poder: 'nivelPoder'}})
```

Findandmodify para insertar o actualizar

Actualiza (o crea) un Pokémon si coincide con el nombre.

findAndModify: comando **legacy**, preferible usar findOneAndUpdate.

```
db.pokemons.findAndModify({query: { nombre: "charmader" },update: {$set: {nivel_poder: 75,poder: "Impactrueno",...}},new: true})
```

ESTRUCTURA DE AGGREGATE

$aggregate es una **etapa de procesamiento por pasos** donde puedes transformar, filtrar y analizar datos en múltiples fases:

```
db.pokemons.aggregate([
  { $match: { ... } },      // Filtrado (como WHERE)
  { $project: { ... } },    // Selección o transformación de campos
  { $group: { _id: ..., total: { $sum: ... } } }, // Agrupación
  { $sort: { ... } }        // Ordenar resultados])
```

Contar cuantos pokemon hay por tipo(elemento)

```
db.pokemons.aggregate([{ $group: { _id: "$elemento", total: { $sum: 1 } } },{ $sort: { total: -1 } }])
```

Calcular la suma total de nivelPoder de todos los pokemon

```
db.pokemons.aggregate([
  {
    $group: {
      _id: null,
```

```
      totalPoder: { $sum: "$nivelPoder" }
    }
  }
])
```

Calcular el promedio de scores para cada pokemon

```
db.pokemons.aggregate([
  {
    $project: {
      nombre: 1,
      promedioScore: { $avg: "$scores" }
    }
  }
])
```

Listar los pokemon que tienen al menos 4 juegos en su array games

```
db.pokemons.aggregate([{$project: {nombre: 1,totalGames: { $size: "$games" }}},{$match: {totalGames: { $gte: 4 }}}])
```

Duplicar los socres de los pokemon tipo fuego

```
db.pokemons.aggregate([
  { $match: { elemento: "fuego", scores: { $exists: true } } },{$project: {nombre: 1,scores: 1,newScores: {$map: {input: "$scores",as: "score",in: { $multiply: ["$$score", 2] }}}}}])
```

Mostar el pokemon mas fuerte(niveldepoder mas alto)

```
db.pokemons.aggregate([{ $sort: { nivelPoder: -1 } },{ $limit: 1 },{ $project: { _id: 0, nombre: 1, nivelPoder: 1 } }])
```

Listar todos los nombres de pokenon y cuantos juegos tiene cada uno
db.pokemons.aggregate([{$project: {nombre: 1,totalGames: { $size: "$games" }}}])

Mostrar los pokemon que tienen al menos un socre mayor a 10

db.pokemons.aggregate([

{$match: {scores: { $elemMatch: { $gt: 10 } }}},{$project: {nombre: 1,scores: 1}}])

$elemMatch busca si algún elemento del array cumple una condición.

Obtener el pokemon mas pesado

db.pokemons.aggregate([{ $sort: { peso_kg: -1 } }, { $limit: 1 }, { $project: { nombre: 1, peso_kg: 1, _id: 0 } }])

Usa $sort descendente por peso.

Agrupar por elemento y obtener el promedio de nivelpoder por tipo

```
db.pokemons.aggregate([
  {
    $group: {
      _id: "$elemento",
      promedioPoder: { $avg: "$nivelPoder" }
    }
  },
  { $sort: { promedioPoder: -1 } }
])
```