HackZone computer

HackZone
computer

Índice:

1. ¿Quién soy?

Soy un trabajador de HackZone, una tienda que se encarga de venta de Hardware. Nuestra tienda cuenta con un gran almacén donde tenemos todos nuestros artículos.

A diario tenemos que recurrir a una base de datos donde tenemos toda la información de nuestros artículos.

Veremos algunas soluciones para afrontar algunas búsquedas y encontrar aquello que necesitamos.

2. Query Operators

1. $eq:

Compara documentos donde el valor de un campo es igual al valor especificado.

```
{ <field>: { $eq: <value> } }
```

2. $gt:

Selecciona aquellos documentos donde el valor de `field`es mayor que (es decir `>`) el especificado `value`.

Sintaxis :`{field: {$gt: value} }`

3. $gte:

Selecciona los documentos donde el valor de `field`es mayor o igual a (es decir `>=`) un valor especificado (p `value`.

Sintaxis :`{field: {$gte: value} }`

4. $in:

Selecciona los documentos donde el valor de un campo es igual a cualquier valor en la matriz especificada.

```
{ field: { $in: [<value1>, <value2>, ... <valueN> ] } }
```

5. $lt:

Selecciona los documentos donde el valor de `field`es menor que (es decir `<`) el especificado `value`.

Syntax: `{field: {$lt: value} }`

6. $lte:

Selecciona los documentos donde el valor de `field`es menor o igual a (es decir `<=`) el especificado `value`

Sintaxis :`{ field: { $lte: value} }`

7. $ne:

Selecciona los documentos donde el valor de `field`no es igual al especificado `value`. Esto incluye documentos que no contienen el `field`.

Syntax: `{field: {$ne: value} }`

8. $nin:

Selecciona los documentos donde:

- el `field`valor no está en el especificado `array` **o**
- el `field`no existe.

Syntax: `{ field: { $nin: [ <value1>, <value2> ... <valueN> ]} }`

3. Logical

1. $and:

Realiza una lógica `AND` de operación en una matriz de *una o más* expresiones (`<expression1>`, `<expression2>`y así sucesivamente) y selecciona los documentos que satisfacen *todas* las expresiones.

Syntax: `{ $and: [ { <expression1> }, { <expression2> } , ... , { <expressionN> } ] }`

2. $not:

Realiza una `NOT` operación lógica en el especificado `<operator-expression>`y selecciona los documentos que *no* coinciden con el `<operator-expression>`. Esto incluye documentos que no contienen el `field`.

Syntax: `{ field: { $not: { <operator-expression> } } }`

3. $nor:

Realiza una `NOR` operación lógica en una matriz de una o más expresiones de consulta y selecciona los documentos que **fallan en** todas las expresiones de consulta de la matriz.

```
{ $nor: [ { <expression1> }, { <expression2> }, ...  { <expressionN> } ] }
```

4. $or:

Operador realiza una `OR` operación lógica en una matriz de *dos o más* `<expressions>` y selecciona los documentos que satisfacen *al menos* uno de los `<expressions>`.

```
{ $or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }
```

4. Element

1. $exists:

Cuando `<boolean>` es verdadero, `$exists` coincide con los documentos que contienen el campo, incluidos los documentos en los que se encuentra el valor del campo `null`. Si `<boolean>` es falso, la consulta devuelve solo los documentos que no contienen el campo.

Syntax: `{ field: { $exists: <boolean> } }`

5. Evaluation

1. $regex:

Proporciona capacidades de expresión regular para *cadenas de* coincidencia de patrones en consultas.

6. Array

1. $all:

Operador selecciona los documentos donde el valor de un campo es una matriz que contiene todos los elementos especificados.

```
{ <field>: { $all: [ <value1> , <value2> ... ] } }
```

2. $elemMatch:

Operador compara documentos que contienen un campo de matriz con al menos un elemento que coincide con todos los criterios de consulta especificados.

```
{ <field>: { $elemMatch: { <query1>, <query2>, ... } } }
```

3. $size:

Operador hace coincidir cualquier matriz con el número de elementos especificados por el argumento.

```
db.collection.find( { field: { $size: 2 } } );
```

7. Aportación personal

1. Aggregate:

Las operaciones de agregación procesan varios documentos y devuelven resultados calculados. Puede utilizar operaciones de agregación para:

- Agrupe los valores de varios documentos juntos.
- Realice operaciones en los datos agrupados para devolver un único resultado.
- Analice los cambios de datos a lo largo del tiempo.

2. $group:

Agrupa los documentos de entrada por la `_id` expresión especificada y para cada agrupación distinta, genera un documento. El `_id` campo de cada documento de salida contiene el grupo único por valor.

```
{
  $group:
    {
      _id: <expression>, // Group By Expression
      <field1>: { <accumulator1> : <expression1> },
      ...
    }
}
```

3. $sum:

Calcula y devuelve la suma colectiva de valores numéricos. `$sum`ignora los valores no numéricos.

```
{ $sum: <expression> }
```

4. $project:

Pasa los documentos con los campos solicitados a la siguiente etapa del proceso. Los campos especificados pueden ser campos existentes de los documentos de entrada o campos recién calculados.

```
{ $project: { <specification(s)> } }
```

5. $subtract:

Resta dos números para devolver la diferencia, o dos fechas para devolver la diferencia en milisegundos, o una fecha y un número en milisegundos para devolver la fecha resultante.

```
{ $subtract: [ <expression1>, <expression2> ] }
```

6. $multiply:

Multiplica números y devuelve el resultado. Pase los argumentos a `$multiply`en una matriz.

```
{ $multiply: [ <expression1>, <expression2>, ... ] }
```

7. $avg:

Devuelve el valor medio de los valores numéricos. `$avg` ignora los valores no numéricos.

```
{ $avg: <expression> }
```

8. $size:

El `$size`operador hace coincidir cualquier matriz con el número de elementos especificados por el argumento.

```
db.collection.find( { field: { $size: 2 } } );
```

9. $max:

```
{ $max: <expression> }
```

Devuelve el valor máximo.

10. $min:

Devuelve el valor mínimo.

```
{ $min: <expression> }
```

11. $match:

Filtra los documentos para pasar solo los documentos que coinciden con las condiciones especificadas a la siguiente etapa de canalización.

```
{ $match: { <query> } }
```