Aggregation pipeline

Como vimos en el apartado anterior el *aggregation pipeline* es el array de documentos que define las etapas o *stages* de agregación. Cada etapa de agregación es un documento que define una operación de agregación. Hay un total de 42 operaciones de agregación (a fecha del 16 de enero de 2024) y por este motivo sólo vamos a ver una pequeña selección de las mismas.

Operadores de etapa / fase / *stage*

La forma de definir una operación de agregación es mediante un documento que contiene un único campo cuyo nombre es el de la operación de agregación. Así, por ejemplo, nombre del campo que hemos de usar para definir una fase de filtrado será `$match`. El valor de dicho campo será otro documento que contiene los parámetros de la operación de agregación. En el caso de `$match` el valor de dicho campo será un *query document* como los que usamos en la operación `find()`.

```
[
    { $match: {
        $and: [ {price: { $lte: 100 } }, { minimum_nights: { $lte: 3 } } ]
    } }
]
```

Las operaciones que veremos en estos apuntes serán las siguientes:

- `$match`: Filtra los documentos de entrada.
- `$addFields`: Añade campos a los documentos de entrada.
- `$project`: Proyecta (añade y elimina) los campos de los documentos de entrada.
- `$limit`: Permite realizar varias operaciones de agregación en una sola etapa.
- `$group`: Agrupa los documentos de entrada por un campo especificado.
- `$lookup`: Permite realizar una operación de *join* entre dos colecciones.
- `$out`: Permite guardar el resultado de una agregación en una colección.

Además, por su interés veremos también el operador `$function` que permite definir funciones JavaScript que se ejecutarán sobre los documentos en la etapa de agregación.

`$match`

La operación `$match` filtra los documentos de entrada y devuelve solo los documentos que coinciden con la condición especificada.

```
db.listings.aggregate([
   { $match: {
         $and: [ {price: { $lte: 100 } }, { minimum_nights: { $lte: 3 } } ]
      }
   }
])
```

Como podemos ver el valor que admite es un *query document* como los que usamos en `find()`.

$addFields

La operación `$addFields` agrega nuevos campos a los documentos. Similar a `$project`, `$addFields` reforma cada documento de la secuencia; específicamente, agregando nuevos campos a los documentos de salida que contienen tanto los campos existentes de los documentos de entrada como los campos recién agregados. `$set` es un alias para `$addFields`.

En el momento de crear nuevos campos podremos hacer referencia a los campos existentes en el documento de entrada. Para ello utilizaremos la sintaxis `$<nombre_campo>`. También podremos usar operadores de expresiones, como `$toDate` o `$function`, para crear nuevos campos.

En el siguiente ejemplo añadimos un campo `last_review_date` que contiene la fecha de la última reseña que se ha hecho sobre el listing. Para ello usamos el operador `$toDate` que convierte una cadena de texto en una fecha.

```
db.listings.aggregate([
   { $addFields: { last_review_date: { $toDate: '$last_review' } } }
])
```

$project

La operación `$project` permite proyectar los campos de los documentos de entrada. Es decir, permite añadir y eliminar campos de los documentos de entrada. Para indicar los campos que queremos proyectar usaremos un documento que contiene los campos que queremos proyectar y un valor que indica si queremos que el campo se incluya o no en el documento de salida. Si el valor es 1 o `true` el campo se incluirá en el documento de salida, si es 0 o `false` no se incluirá. Admite además crear nuevos campos de la misma forma que `$addFields`.

```
db.listings.aggregate([
   { $project: { _id: 0, name: 1, neighbourhood: 1, price: 1, last_review_date: {
$toDate: '$last_review' } } }
])
```

$limit

Limit se usa para limitar el número de documentos que se procesan en una etapa de agregación. Si se coloca al principio de un pipeline de agregación, limita el número de documentos de entrada en el pipeline de agregación. Si se coloca al final de un pipeline de agregación, limita el número de documentos en el flujo de salida.

```
db.listings.aggregate([
   { $limit: 10 }
])
```

$group

La operación `$group` agrupa los documentos de entrada por uno o más campos y calcula las agregaciones sobre los documentos agrupados.

La sintaxis para indicar el campo por el que se agrupan los documentos es la siguiente:

```
{ $group: { _id: <campo> } }
```

en el caso de usar más de un campo:

```
{ $group: { _id: { <campo1>: <valor1>, <campo2>: <valor2>, ... } } }
```


```
{ $group: { _id: { <campo1>: <valor1>, <campo2>: <valor2>, ... } } }
```


Dentro de una etapa de agregación `$group` podremos utilizar expresiones de acumulación para calcular valores sobre los documentos agrupados.

Algunas de estas expresiones son:

- `$sum`: Calcula la suma de los valores de un campo.
- `$avg`: Calcula la media de los valores de un campo (ignora los valores no numéricos).
- `$median`: Calcula la mediana de los valores de una expresión sobre los documentos de la agregación.
- `$first`: Devuelve de una expresión sobre el primer documento de la agregación.
- `$min`: Calcula el valor mínimo de una expresión para los documentos de la agregación.
- `$max`: Calcula el valor máximo de una expresión para los documentos de la agregación.
- `$push`: Devuelve un array con los valores de una expresión sobre los documentos de la agregación.

pero hay muchas más.

En el siguiente ejemplo agrupamos los documentos por el campo `neighbourhood` y calculamos el número de documentos (listings) de cada grupo y el precio medio de los mismos.

```
db.listings.aggregate([
   { $group: { _id: '$neighbourhood', listings: { $count: {} }, average_price: {
$avg: '$price' } } }
])
```

Las agrupaciones también se pueden realizar sobre más de un campo. En el siguiente ejemplo agrupamos los documentos por el campo `host_id` y `neighbourhood` y calculamos el número de documentos (listings) de cada grupo y el precio medio de los mismos.

```
db.listings.aggregate([
   { $group: { _id: { host_id: `$host_id`, neighbourhood: '$neighbourhood' },
listings: { $count: {} }, average_price: { $avg: '$price' } } }
])
```

`$lookup`

La operación `$lookup` permite realizar una operación de *join* entre dos colecciones. La forma en la que lo hace es *seleccionando* los documentos cuyo campo de la colección externa (`foreignField`) coincidan con el del campo de la colección sobre la que se está aplicando la agregación (`localField`) y añadiendo dichos documentos a un array. La sintaxis de esta operación es la siguiente:

```
{
   $lookup:
     {
       from: <colección>,
       localField: <campo_colección_actual>,
       foreignField: <campo_colección_referenciada>,
       as: <nombre_campo_salida>
     }
}
```

En el siguiente ejemplo realizamos un *join* entre las colecciones `listings` y `reviews` para añadir a cada documento de `listings` un array con los documentos de `reviews` que tengan el campo `listing_id` igual al campo `id` del documento de `listings`.

```
db.listings.aggregate([
   { $lookup: { from: 'reviews', localField: 'id', foreignField: 'listing_id',
as: 'reviews' } }
])

{
 _id: ObjectId('6592eb004bf3534c97961c41'),
    id: 209673,
    name: 'Rental unit in Barcelona · ★4.78 · 2 bedrooms · 3 beds · 1 bath',
    host_id: 1033366,
    host_name: 'Elena',
    neighbourhood_group: 'Gràcia',
    neighbourhood: 'la Vila de Gràcia',
    latitude: 41.398,
    longitude: 2.1553,
    room_type: 'Entire home/apt',
    price: 81,
    minimum_nights: 2,
    number_of_reviews: 339,
    last_review: '2023-11-17',
    reviews_per_month: 2.54,
    calculated_host_listings_count: 1,
    availability_365: 235,
    number_of_reviews_ltm: 31,
    license: 'HUTB-010932',
    reviews: [
      {
        _id: ObjectId('6592eaab177df769e6d1c07a'),
        listing_id: 209673,
        date: '2013-03-04'
      },
      {
        _id: ObjectId('6592eaab177df769e6d1c07e'),
        listing_id: 209673,
        date: '2013-01-02'
      },
      {
        _id: ObjectId('6592eaab177df769e6d1c07f'),
        listing_id: 209673,
        date: '2013-05-05'
```

```
      },
      ...
},
...
```

`$out`

La operación `$out` permite guardar el resultado de una agregación en una colección. La sintaxis de esta operación es la siguiente:

```
{ $out: {
      db: <base_de_datos>,
      coll: <colección>
   }
}
```

`$function`

`$function` no define una fase si no que es un operador que se puede utilizan dentro de una. Permite definir funciones JavaScript que se ejecutarán en la etapa de agregación. La sintaxis de esta operación es la siguiente:

```
{
   $function:
     {
       body: <function>,
       args: [ <argument1>, <argument2>, ... ],
       lang: <language>
     }
}
```

En el siguiente ejemplo definimos una función que convierte cadenas de la forma "$." a valores numéricos decimales. Esta función la aplicamos sobre el campo `price` de los documentos de entrada.

```
db.listings.aggregate([
   { $addFields: { price: { $function: { body: function(price) { return
parseFloat(price.substring(1)); }, args: ['$price'], lang: 'js' } } } }
])
```

Nota: el uso de funciones JavaScript en las etapas de agregación puede tener un impacto negativo en el rendimiento de las mismas po lo que se recomienda usarlas únicamente **cuando no existen operadores que permitan realizar la operación requerida**.