Agregaciones

Las agregaciones sirven para realizar un conjunto de operaciones que toman como entrada un subconjunto de documentos de una colección. La forma en la que se definen las operaciones que se realizarán sobre esta colección de documentos es mediante *pipelines* de agregaciones. Un pipeline de agregaciones consiste en un array de fases o etapas (*stages*) de agregación. Cada etapa de agregación es un documento que define una operación de agregación. Las etapas de agregación se ejecutan en orden, de forma que la salida de una etapa de agregación es la entrada de la siguiente etapa de agregación.

Un ejemplo de *pipeline* de agregaciones sería el siguiente:

```
[
   // Stage 1: Seleccionamos los listings que tengan un precio menor o igual a
100 y un mínimo de noches menor o igual a 3.
   {
      $match: { $and: [ { price: { $lte: 100 } }, { minimun_nights: { $lte: 3 } }
] }
   },
   // Stage 2: Agrupamos los listings por barrio y contamos el número de listings
válidos.
   {
      $group: { _id: 'neighbourhood', valid_listings: { $count: {} } }
   }
]
```

Una agregación que transforme un campo de texto a numérico podría ser la siguiente:

```
db.listings.aggregate( [
  {
    $addFields: {
      price: { $toInt: $substring: [ '$price', 1, { $strLenCP: '$price' } ] }
    }
  }
] )
```

Como podemos ver este *pipeline* tiene dos etapas de agregación. En la primera etapa de agregación filtramos los documentos de entrada y nos quedamos con los que tengan un precio menor o igual a 100 y un mínimo de noches menor o igual a 3. En la segunda etapa de agregación agrupamos los documentos de entrada por barrio y contamos el número de listings.

Para realizar agregaciones se utilizarán *pipelines* de agregaciones.

Pipeline de agregaciones

Los *pipeline* de agregaciones indican una o más etapas o fases (también llamados *stages*). En una de las fases podríamos, por ejemplo, seleccionar los documentos que nos interesen y en la siguiente agruparlos en relación al valor de un campo para calcular valores totales, medios, etc.

En el siguiente ejemplo estaremos realizando una agregación de *listings* de Airbnb de Barcelona. El *pipeline* de esta agregación consta de dos fases: una de filtrado (`$match`) y otra de agrupación (`$group`) en la que a su vez creamos un nuevo campo donde almacenamos el número de documentos (`$count`) que se han agrupado.

```
db.listings.aggregate( [
  {
      $match: { minimum_nights: { $lte: 3 } }
  },
  {
      $group: { _id: '$neighbourhood', listings: { $count: {} }, precio_medio: {
$avg: '$price' } }
  }
] )
```

Fases de agregación

Como acabamos de ver cada etapa de agregación es un documento que define **una operación** de fase.

Cada fase de agregación de un *pipeline* se define mediante un documento que contiene un único campo cuyo nombre es el de la operación de agregación a realizar en la fase. El valor de dicho champo será otro documento que contiene los parámetros de la operación de agregación.

Las fases de agregación que admite MongoDB son *unas cuantas* de entre las que podemos citar las siguientes:

- `$match`: Filtra los documentos de entrada y devuelve solo los documentos que coinciden con la condición especificada.
- `$group`: Agrupa los documentos de entrada por un campo especificado y calcula las agregaciones sobre los documentos agrupados.
- `$addFields`: Agrega nuevos campos a los documentos. Similar a `$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 `$addFields`.
- `$project`: Similar a `$addFields`, pero en lugar de agregar campos, puede tanto agregar como eliminar. Por cada documento de la entrada, devuelve un documento. `$unset` es un alias para `$project`.
- `$lookup`: Permite realizar operaciones de *join* entre dos colecciones.
- `$sort`: Ordena los documentos de entrada según los campos especificados.

Operaciones de acumulación

Las operaciones de acumulación son operaciones que se realizan sobre un conjunto de documentos y que devuelven un único valor. Estas operaciones se utilizan en las etapas de agregación para realizar cálculos sobre los documentos de entrada. Algunas de estas operaciones son:

- `$sum`: Suma los valores de los documentos de entrada.
- `$avg`: Calcula la media de los valores de los documentos de entrada.

- `$min`: Devuelve el valor mínimo de los documentos de entrada.
- `$max`: Devuelve el valor máximo de los documentos de entrada.

Cómo añadir un campo de una colección a otra

En este ejemplo de agregación vamos a añadir el campo (`first_review`) de una colección (`det_listings`) a otra (`listings`). Para ello utilizaremos la operación `$lookup` que nos permite realizar operaciones de *join* entre dos colecciones.

Las fases del *pipeline* de agregación serán las siguientes:

1. `$lookup`: Realizamos un *join* entre las colecciones `listings` y `det_listings` utilizando el campo `id` de ambas colecciones como campo de unión. El resultado de esta operación será un nuevo campo (`det_info`) que contendrá un array con los documentos de la colección `det_listings` que coincidan con el campo `id` de la colección `listings`. Como el campo `id` es único en la colección `det_listings` el array tendrá un único elemento.
2. `$limit`: Limitamos el número de documentos que pasarán a la siguiente fase a 1. *Esto es para que la agregación no tarde demasiado en ejecutarse, no tendría sentido en un caso real.*
3. `$addFields`: Añadimos un nuevo campo (`first_review`) que contendrá el valor del campo `first_review` (la fecha de la primera *review* que tomamos de la colección `det_reviews`). `$det_info.first_review` nos devolverá un array con todos los valores del campo `first_review` de los documentos del del array `$det_info` (en nuestro caso sólo habrá uno).
4. `$unwind`: Deshacemos el array que contiene el campo `first_review`, que será de un elemento, para que el campo `first_review` sea un campo normal y no un array de un elemento.
5. `$project`: Eliminamos el campo `det_info` que ya no nos interesa.

```
db.listings.aggregate( [
  {
    $lookup: {
        from: 'det_listings',
        localField: 'id',
        foreignField: 'id',
        as: 'det_info'
    }
  },
  {
    $limit: 1
  },
  {
    $addFields: {
      first_review: '$det_info.first_review',

    }
  },
  {
    $unwind: '$first_review'
  },
  {
    $project: {
```

```
      det_info: 0
    }
  }
] )
```

Si quisiésemos que los campos `first_review` y `last_review` fuesen de tipo `Date` en lugar de `String` podríamos añadir una fase `$addFields` utilizando la operación `$toDate` de la siguiente forma:

```
{
  $set: {
    first_review: { $toDate: '$first_review' },
    last_review: { $toDate: '$last_review' }
  }
}
```

Nótese que `$set` es un alias para `$addFields`.

```
db.det_listings.aggregate( [ {
    $match: { price: { $ne: "" } }
  },
  {
    $set: {
      first_review: { $toDate: '$first_review' },
      last_review: { $toDate: '$last_review' }
    }
  },
  {
    $group: {
      _id: '$neighbourhood',
      avg_price: { $avg: '$price' },
      min_nights: { $min: '$minimum_nights' },
      max_nights: { $max: '$maximum_nights' },
      avg_first_review: { $avg: '$first_review' },
      avg_last_review: { $avg: '$last_review' }
    }
  }
] )
```