**Агрегации и обработка данных в MongoDB**

**1. Введение в агрегационный фреймворк**

Агрегационный фреймворк в MongoDB предоставляет мощный инструмент для обработки и анализа данных.

Ключевая концепция — конвейер (pipeline). Данные проходят через последовательность стадий, каждая из которых выполняет определённое преобразование.

**2. Стадии конвейера**

Каждая стадия конвейера представляет собой оператор, который обрабатывает данные и передаёт результат на следующую стадию. Основные стадии:

* **$match**: фильтрация документов.
* **$group**: группировка данных.
* **$sort**: сортировка.
* **$project**: выбор полей для вывода.
* **$limit**: ограничение количества документов.
* **$skip**: пропуск первых N документов.

**3. Пример работы с конвейером**

Рассмотрим пример: анализ продаж товаров.  
Задача: найти среднюю цену товаров бренда "XYZ".

db.products.aggregate([

  { $match: { "brand": "XYZ" } }, // Фильтрация

  { $group: {

      \_id: null,

      avgPrice: { $avg: "$price" }

  } } // Группировка и расчёт среднего значения

])

Результат:

[

  { "\_id": null, "avgPrice": 25000 }

]

**4. Основные стадии агрегации**

**$match (фильтрация)**

Фильтрует документы на основе условий. Аналогично find.  
Пример: отфильтровать товары дороже 10,000.

db.products.aggregate([

  { $match: { "price": { $gt: 10000 } } }

])

**$group (группировка)**

Позволяет группировать данные и выполнять агрегатные вычисления.  
Пример: подсчитать общее количество товаров по бренду.

db.products.aggregate([

  { $group: {

      \_id: "$brand",

      totalProducts: { $sum: 1 }

  } }

])

Результат:

[

  { "\_id": "XYZ", "totalProducts": 2 },

  { "\_id": "ABC", "totalProducts": 1 }

]

**$sort (сортировка)**

Сортирует документы по заданным полям.  
Пример: отсортировать товары по цене в порядке убывания.

db.products.aggregate([

  { $sort: { "price": -1 } }

])

**$project (проекция)**

Позволяет выбрать поля для отображения.  
Пример: показать только название и цену товаров.

db.products.aggregate([

  { $project: { "name": 1, "price": 1, "\_id": 0 } }

])

**$limit и $skip**

Используются для ограничения количества документов.  
Пример: показать 3 самых дешёвых товара.

db.products.aggregate([

  { $sort: { "price": 1 } },

  { $limit: 3 }

])

**5. Примеры использования агрегатных функций**

**$sum**: вычисляет сумму.

Пример: общая стоимость всех товаров.

* db.products.aggregate([
* { $group: {
* \_id: null,
* totalCost: { $sum: { $multiply: ["$price", "$stock"] } }
* } }
* ])

**$avg**: вычисляет среднее значение.

Пример: средняя цена товаров.

* db.products.aggregate([
* { $group: {
* \_id: null,
* avgPrice: { $avg: "$price" }
* } }
* ])

**$max и $min**: находят максимальное и минимальное значения.

Пример: самая дорогая и самая дешёвая цена товара.

db.products.aggregate([

  { $group: {

      \_id: null,

      maxPrice: { $max: "$price" },

      minPrice: { $min: "$price" }

  } }

])

**6. Пример комплексной агрегации**

Задача: найти среднюю стоимость товаров каждого бренда, отсортировать по убыванию средней цены.

db.products.aggregate([

  { $group: {

      \_id: "$brand",

      avgPrice: { $avg: "$price" }

  } },

  { $sort: { "avgPrice": -1 } }

])

Результат**:**

[

  { "\_id": "ABC", "avgPrice": 60000 },

  { "\_id": "XYZ", "avgPrice": 25000 }

]

**Преимущества агрегационного фреймворка**

* Высокая производительность.
* Удобная обработка больших объёмов данных.
* Гибкость за счёт комбинирования стадий.

**Недостатки**

* Повышенные требования к памяти при сложных конвейерах.
* Сложность отладки многоступенчатых запросов.