Лабораторна робота 5(II). Загальні принципи проектування та використання бази даних MongoDB

Зміст

Мета: Познайомитися з документо-орієнтованим моделюванням даних і з тим, як організовані дані в MongoDB - на рівні бази даних, колекції та документа. Опанувати виконання деяких операцій в MongoDB.

Теоретичні положення

Лабораторна робота демонструє принципи проектування схеми даних для MongoDB. Це корисно, тому що багато користувачів MongoDB раніше проектували схеми тільки традиційних реляційних СУБД. Розглянемо в якості приклада перепроектування схеми технічної служби, яка була перевантажена в MongoDB в лабораторній роботі 14, а також з'ясуємо, чим ця схема відрізняється від еквівалентної реляційної схеми, і дізнаємося про те, як в MongoDB представляються типові зв'язки між сутностями, наприклад, типу один-до-багатьох і багато-до-багатьох. Сконструйована схема буде потім використовуватися при обговоренні запитів та агрегування. Проектування схеми бази даних - це процедура вибору найкращого подання набору даних з урахуванням можливостей СУБД, природи даних і вимог додатка.

Принципи проектування схеми для реляційних баз даних вже давно визначені. В цьому випадку пропонується прагнути до нормалізованої моделі, яка дозволяє забезпечити можливість запитів загального вигляду і уникнути такого оновлення даних, яке могло б привести до аномалій та розбіжностей даних. До того ж, наявні перевірені практикою прийоми як моделювати зв'язок один-до-багатьох і багато-до-багатьох. Втім, проектування схеми ніколи не було точною наукою, навіть у разі реляційних баз даних. Для додатків, критичних до продуктивності або працюють з неструктурованими даними, може знадобитися більш загальна модель, що досить часто може не бути нормалізованою.

Оптимальний проект схеми завжди є результатом глибокого розуміння вибраної СУБД, правильного уявлення про функціональність програми, що розробляється, а також досвіду розробника.

Практичні вправи

Вправа 5(II).1. Перепроектування схеми БД та доповнення даними

Завдання: Перепроектувати схему даних предметної області для MongoDB, розмістивши дані декількох таблиць в одній колекції, використовуючи масиви. Кожна колекція має містити не менше ніж 4 документи. Описати **переваги** та **недоліки** нової схеми.

Розглянемо виконання цього завдання на прикладі колекції cComp, що містить інформацію про всі деталі, які можуть бути на будь-яких складах технічної служби автотранспортного підприємства або взагалі відсутні.

Для представлення інформації про деталі на складах в нормалізованій моделі РСУБД потрібно декілька таблиць. cComp – це таблиця для основної інформації про деталі на складах. Але крім неї є таблиці VRest, cStore та інші, котрі пов'язують деталі з інформацією на якому складі та в якій кількості є кожна деталь в наявності.
Моделювати в MongoDB простіше. Оскільки в будь-якому документі про деталі є місце для довільних динамічних атрибутів. А за рахунок застосування масивів для зберігання внутрішніх структур документа зазвичай можна розмістити багато табличне реляційне уявлення в одну колекцію MongoDB. Таким чином, можна об’єднати в колекції cComp ввідомості про залишки деталей на складах (ця інформація зберігається в VRest) та назви складів (ця інформація зберігається в cStore). Для цього в консолі MongoDB в колекції cComp для деяких документів, що містять інформацію про певні деталі, додамо інформацію про те, на яких складах ці деталі наявні:

```
db.cComp.update(
    { Id : -2024963176},
    { $set: { Stores: [
                { "store": "Центральний склад", quantity: 1},
                { "store": "Перехідний склад", quantity: 1}
             ]
            }
    }
);

db.cComp.update(
    { Id : -2039630098},
    { $set: { Stores: [
                { "store": "Центральний склад", quantity: 5},
                { "store": "Перехідний склад", quantity: 2}
             ]
            }
    }
);

db.cComp.update(
    { Id : -1430801777},
    { $set: { Stores: [
                { "store": "Центральний склад", quantity: 2},
                { "store": "Склад цивільної оборони", quantity: 3}
             ]
            }
    }
).
```

Якщо документ колекції cComp не містить масива Stores, то це означає, що ця деталь взагалі відсутня на складах.

Вправа 5(II).2. Групування та агрегування в MongoDB.

Завдання: Написати до своєї предметної області 2 запити з групуванням та 2 запити з використанням агрегатних функцій.

Групування в MongoDB переважно забезпечує функція Aggregate(). Достатньо повну документацію по ній можна знайти в http://docs.mongodb.org/manual/aggregation/ та http://docs.mongodb.org/manual/meta/aggregation-quick-reference/ . Наведемо частково перший документ.

Синтаксис команди:
db.collection.aggregate([{ <stage> }, ...])

Стадії

Стадії обробки потоку документів вказуються в масиві підкоманд команди aggregate. Документи проходять через стадії один за одним. Всі стадії крім $out і $geoNear можуть з'явитися багато разів у потоці.
Короткий опис параметру Pipeline (потоку підкоманд) команди aggregate():

Ім'я	*Опис*
$geoNear	Повертає замовлений потік документів, грунтованих на близькості до геопросторового пункту. Об'єднує функціональність $match, $sort, і $limit для даних про місце розташування. Вихідні документи включають додаткове поле відстані і можуть включати поле ідентифікатора розташування.
$group	Групує вхідні документи по вказаному виразу ідентифікатора і застосовує вираз накопичувача(ів) до кожної групи, якщо це обумовлено. Споживає усі вхідні документи і видає один документ для кожної групи.Вихідні документи містять тільки поле ідентифікатора і, якщо обумовлено, поля накопичувача.
$limit	Передає перші n документів немодифікованими у потік, де n - вказане обмеження. Для кожного вхідного документу, виводить або один документ (для перших n документів), або пусті документи (після перших n документів).
$match	Фільтрує потік документів, пропускаючи на наступну стадію лише вказані документи. $match використовує стандартні запити Mongodb.
$out	Пише вихідні документи потоку агрегації до колекції. Щоб користуватися стадією $ out, вона має бути останньою стадією в потоці підкоманд.
$project	Надає нового вигляду кожному документу в потоці, як-от додаючи нові поля або видаляючи існуючі поля. Для кожного вхідного документу виводить один документ.
$redact	Надає нового вигляду кожному документу в потоці, обмежуючи зміст вхідних документів, грунтований на інформації, збереженій у документах безпосередньо. Об'єднує функціональність $ project і $ match. Може бути використаний для редагування рівня поля. Для кожного вхідного документу виводить або один, або пустий документ.
$skip	Пропускає перші n документів, де n є вказаною кількістю пропусків, і передає документи, що залишилися, немодифікованими в потік. Для кожного вхідного документу виводить або пусті документи (для перших n документів), або один документ (після перших n документів).
$sort	Переупорядковує потік документів згідно вказаного ключа сортування.

	Змінюється тільки порядок; документи залишаються немодифікованими. Для кожного вхідного документу виводить один документ.
$unwind	Розкриває поле масиву з вхідного документу в вихідний для кожного елементу. Кожний вихідний документ замінює массив значенням елементу. Для кожного вхідного документу виводиться n документів, де n є числом елементів масиву і може бути нульовим для порожнього масиву.

Порівняння команд SQL і MongoDB

З документації MongoDB (http://docs.mongodb.org/manual/reference/sql-aggregation-comparison/) наведемо порівняльну таблицю команд SQL та MongoDB**.** Приклади в цій таблиці наводяться для документу типу:

```
{
  cust_id: "abc123",
  ord_date: ISODate("2012-11-02T17:04:11.102Z"),
  status: 'A',
  price: 50,
  items: [ { sku: "xxx", qty: 25, price: 1 },
           { sku: "yyy", qty: 25, price: 1 } ]
}
```

SQL Example	MongoDB Example	Description
`SELECT COUNT(*) AS count FROM orders`	`db.orders.aggregate( [ { $group: { _id: null, count: { $sum: 1 } } } ] )`	Count all records from `orders`
`SELECT SUM(price) AS total FROM orders`	`db.orders.aggregate( [ { $group: { _id: null, total: { $sum: "$price" } } } ] )`	Sum the `price` field from `orders`
`SELECT cust_id, SUM(price) AS total FROM orders GROUP BY cust_id`	`db.orders.aggregate( [ { $group: { _id: "$cust_id", total: { $sum: "$price" } } } ] )`	For each unique `cust_id`, sum the `price` field.

SQL Example	MongoDB Example	Description
	} }])	

SELECT cust_id,
SUM(price) AS total
FROM orders
GROUP BY cust_id
ORDER BY total
	db.orders.aggregate([
	{
	$group: {
	_id: "$cust_id",
	total: { $sum: "$price" }
	}
	},
	{ $sort: { total: 1 } }
	])
		For each unique `cust_id`, sum the `price` field, results sorted by sum.

SELECT cust_id,
ord_date,
SUM(price) AS total
FROM orders
GROUP BY cust_id,
ord_date
	db.orders.aggregate([
	{
	$group: {
	_id: {
	cust_id: "$cust_id",
	ord_date: {
	month: { $month: "$ord_date" },
	day: { $dayOfMonth: "$ord_date" },
	year: { $year: "$ord_date"}
	}
	},
	total: { $sum: "$price" }
	}
	}
	])
		For each unique `cust_id`,`ord_date` grouping, sum the `price` field. Excludes the time portion of the date.

SELECT cust_id,
count(*)
FROM orders
GROUP BY cust_id
HAVING count(*) > 1
	db.orders.aggregate([
	{
	$group: {
	_id: "$cust_id",
	count: { $sum: 1 }
	}
	},
	{ $match: { count: { $gt: 1 } } }
	])
		For `cust_id` with multiple records, return the `cust_id` and the corresponding record count.

SELECT cust_id,
ord_date,
SUM(price) AS total
FROM orders
GROUP BY cust_id,
ord_date
HAVING total > 250
	db.orders.aggregate([
	{
	$group: {
	_id: {
	cust_id: "$cust_id",
	ord_date: {
	month: { $month: "$ord_date" },
	day: { $dayOfMonth: "$ord_date" },
	year: { $year: "$ord_date"}
	}
	},
	total: { $sum: "$price" }
	}
	},
		For each unique `cust_id`,`ord_date` grouping, sum the `price` field and return only where the sum is greater than 250. Excludes the time portion of the date.

SQL Example	MongoDB Example	Description
	{ $match: { total: { $gt: 250 } } }])	

SELECT cust_id,
SUM(price) as total
FROM orders
WHERE status = 'A'
GROUP BY cust_id
	db.orders.aggregate([
	{ $match: { status: 'A' } },
	{
	$group: {
	_id: "$cust_id",
	total: { $sum: "$price" }
	}
	}
	])
		For each unique `cust_id` with status `A`, sum the `price` field.

SELECT cust_id,
SUM(price) as total
FROM orders
WHERE status = 'A'
GROUP BY cust_id
HAVING total > 250
	db.orders.aggregate([
	{ $match: { status: 'A' } },
	{
	$group: {
	_id: "$cust_id",
	total: { $sum: "$price" }
	}
	},
	{ $match: { total: { $gt: 250 } } }
	])
		For each unique `cust_id` with status `A`, sum the `price` field and return only where the sum is greater than 250.

SELECT cust_id,
SUM(li.qty) as qty
FROM orders o,
order_lineitem li
WHERE li.order_id = o.id
GROUP BY cust_id
	db.orders.aggregate([
	{ $unwind: "$items" },
	{
	$group: {
	_id: "$cust_id",
	qty: { $sum: "$items.qty" }
	}
	}
	])
		For each unique `cust_id`, sum the corresponding line item `qty` fields associated with the orders.

SELECT COUNT(*)
FROM (SELECT cust_id,
ord_date
FROM orders
GROUP BY cust_id,
ord_date)
as DerivedTable
	db.orders.aggregate([
	{
	$group: {
	_id: {
	cust_id: "$cust_id",
	ord_date: {
	month: { $month: "$ord_date" },
	day: { $dayOfMonth: "$ord_date"
	},
	year: { $year: "$ord_date"}
	}
	}
	}
	},
	{
	$group: {
	_id: null,
	count: { $sum: 1 }
	}
	}
	])
		Count the number of distinct `cust_id`,`ord_date` groupings. Excludes the time portion of the date.

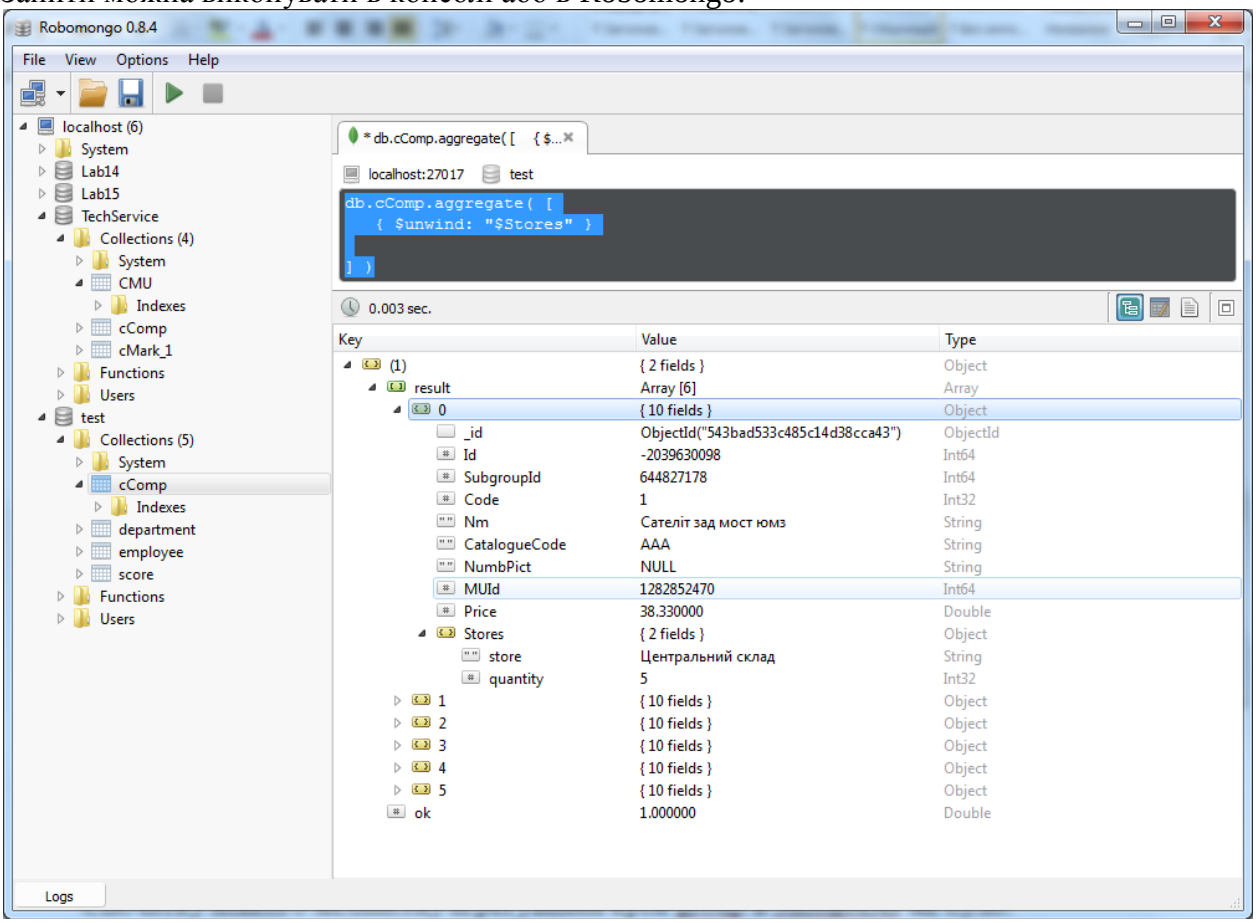
Виконання вправи

Розглянемо приклад таких запитів для колекції cComp.

Запит 1: Для кожної деталі показати склад, на якому вона знаходиться, та кількість на цьому складі.

```
db.cComp.aggregate( [
    { $unwind: "$Stores" }
] ).
```

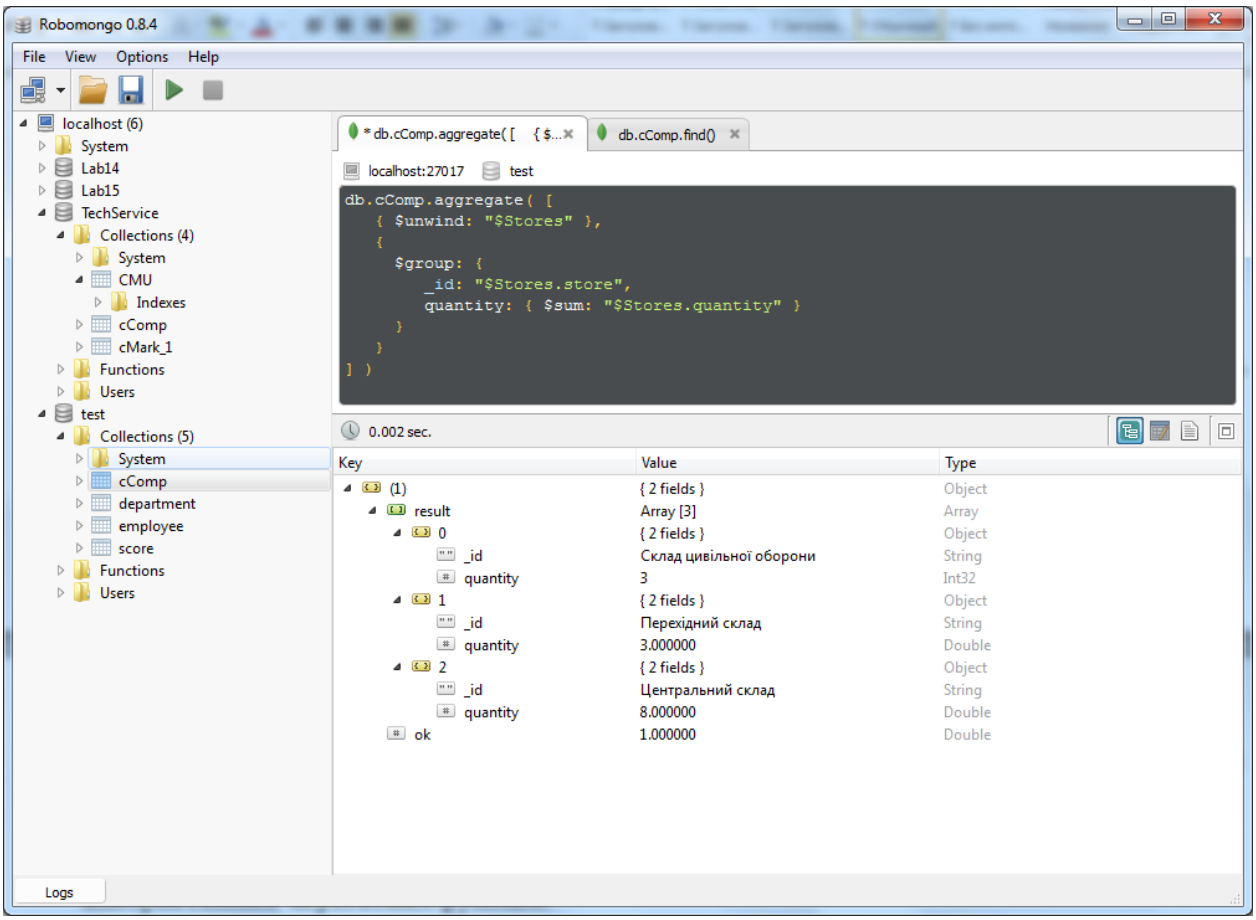
Запити можна виконувати в консолі або в Robomongo:



Результатом виконання, цього запиту є розвернення інформації про склади, на яких знаходиться деталь. Якщо деталь знаходиться на трьох складах, то в результаті виконання цього запиту, цій деталі буде відповідати 3 записи. Тобто **$unwind: "$Stores"** вказує що треба розвернути масив "$Stores".

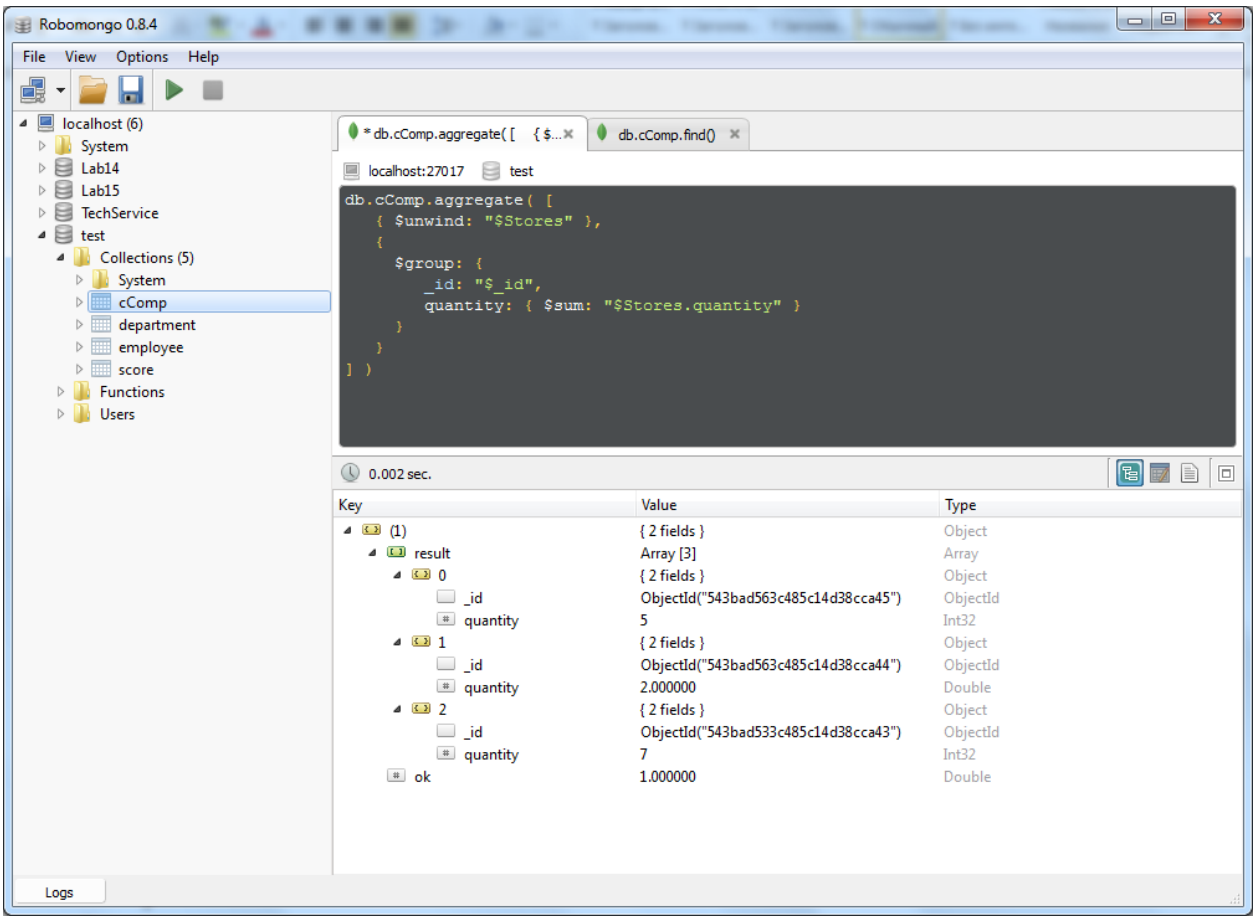
Запит 2: Показати кількість деталей на кожному складі:

```
db.cComp.aggregate( [
    { $unwind: "$Stores" },
    {
      $group: {
          _id: "$Stores.store",
          quantity: { $sum: "$Stores.quantity" }
      }
    }
] )
```



Запит 3: Показати по кожній деталі сумарну кількість на всіх складах:

```
db.cComp.aggregate( [
    { $unwind: "$Stores" },
    {
      $group: {
          _id: "$_id",
          quantity: { $sum: "$Stores.quantity" }
      }
    }
] )
```



Вправа 5(II).3. Виконання аналогу JOIN за допомогою Map-reduce.

Завдання: Виконати запит в MongoDB, який за кодом, що зберігається в одній колекції, виводить назву, що зберігається в іншій колекції, тобто аналог JOIN в SQL.

Виконання цього завдання розглянемо на прикладі колекцій cComp (містить інформацію про деталь) та CMU (довідник, що містить назви одиниць вимірювання). А саме для кожної деталі виведемо її одиницю вимірювання.

Для виконання цього запиту застосуємо механізм Map-reduce. Це процес двоступінчатий. Спочатку робиться map (відображення), потім - reduce (згортка). На етапі відображення вхідні документи трансформуються (map) і породжують (emit) пари ключ=>значення (як ключ, так і значення можуть бути складеними). Породжені дані збираються в масиви по однаковому ключу, це також виконує emit. Під час згортки (reduce) на вході ключ і масив значень, породжених для цього ключа, а на виході - фінальний результат.

Для нашого прикладу процес буде складатися з виконання наступних функцій JavaScript:

```
/* Ця функція виводить колекцію, згруповану по ключу «Id одиниці виміру»,
 і всіма потрібними полями, але заповнену лише даними деталей cComp */
var mapComp = function () {
            var output= {MUid : this.MUId, name:this.Nm, MU:null}
                    emit(this.MUId, output);
                };

/* Ця функція виводить колекцію з ключем «Id одиниці виміру» і всіма
потрібними полями, але заповнену лише даними одиниць виміру cMU */
var mapCMU = function () {
                var output= {MUid : this.Id, name:null, MU:this.Nm}
```

```
                    emit(this.Id, output);
                };

/* Ця функція виконується двічі і перший раз заповнює вихідну колекцію
Outs назвами деталей, а другий раз – назвами одиниць виміру */
var reduceF = function(key, values) {
    var outs = { name:null, MU:null};
    values.forEach(function(v){
                    if(outs.name ==null){
                        outs.name = v.name
                     }
                     if(outs.MU ==null){
                        outs.MU = v.MU
                     }
     });
    return outs;
};

//Викликаємо mapReduce для колекції cComp
result = db.cComp.mapReduce(mapComp, reduceF, {out: {reduce: 'Comp_CMU'}});
//Викликаємо mapReduce для колекції cMU
result = db.CMU.mapReduce(mapCMU,reduceF, {out: {reduce: 'Comp_CMU'}});
db.Comp_CMU.find()
```

Контрольні питання

1. За допомогою яких команд можна доповнювати та змінювати дані та структуру документа в MongoDB?
2. Поясніть призначення команди **$unwind**?
3. Чим видрізняються Запит 2 та Запит 3 з Вправи 5(II).2?
4. Поясніть принцип роботи Map-reduce, назвіть переваги та недоліки в порівнянні з застосуванням команди group?

Використані джерела

1. Karl Seguin. The Little MongoDB Book (Маленькая книга о MongoDB). http://openmymind.net/mongodb.pdf
2. The MongoDB 2.6 Manual. http://docs.mongodb.org/manual/
3. http://blog.knoldus.com/2014/03/12/easiest-way-to-implement-joins-in-mongodb-2-4/
4. http://docs.mongodb.org/manual/reference/sql-comparison/
5. http://docs.mongodb.org/manual/reference/method/db.collection.update/#db.collection.update
6. http://habrahabr.ru/post/184130/
7. http://stackoverflow.com/questions/12831939/couldnt-connect-to-server-127-0-0-127017