Voxel Data Management System (VDMS) Report

User requirements and conceptual benchmark

The user requirements have been listed in three categories: selections, calculations and analyses.

The user requirements have been listed in a number of categories: functionality within the system (selections, calculations and analyses), presentation functionality (access, view, navigate) and download functionality. Some functionality was examined in more detail (attributes).

Selections

- Selection: rectangle
- Selection: line + buffer
- Selection: polygon
- Selection: attributes

Calculations

Many researchers remarked that they prefer to make calculations using their own software. If possible they would like to be able to preview the results of calculations within the system before downloading the points to their local system and doing their own re-calculations. The most important calculations that the researchers would like to be able to make are: (1) calculation of statistical values, like the average height of an area, lowest point, highest point and (2) voxels density.

- Calculations: statistic values
- Calculations: point clouds density

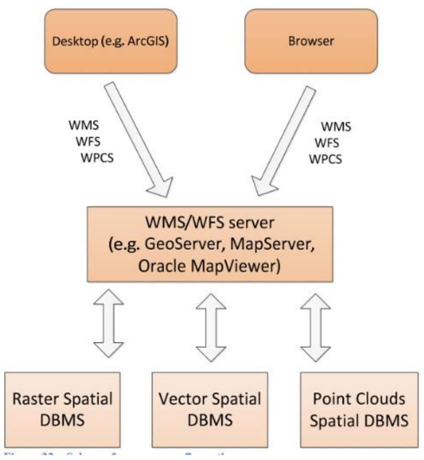
Analyses

Researchers would like to compare point clouds with data coming from other sources of the same area.

- Analysis: spatial analyses with other tables
- Analysis: pattern (nearest neighbor analysis)

Access

Most researchers (84 %) want to access the new point clouds spatial DBMS through a web service, either directly through their browser or by letting their desktop software (e.g. ArcGIS) connect to the DBMS.



View

Like with access, most researchers want to view the selected data in a web application using one of the standard browsers, or by means of a desktop application.

Navigate

Most researchers want to navigate the point cloud data using traditional techniques like zooming and panning.

Download form and format

The raster data is still the most popular format but it is reassuring to find that a lot of researchers also would like to download the point cloud data, if available. A number of download formats have been mentioned. For raster data the researchers would like to use image or publication formats (jpg, pdf) or GeoTIFF and NetCDF format [Website: NetCDF, Website: OpenDAP]. For point clouds downloads in LAS an LAZ formats are the current standard.

Co-ordinate systems

Different CPS transform.

Functionality in the various categories

http://www.pointclouds.nl/index.html

Benchmark stages

Dataset name	Benchmark	#Voxels	#Files	Format	Disk size (GB)	Description
Village	Small	22,333,289	1	xyz	0.32	'UNSW Village' building
building	Medium	241,613,693	52	xyz	3.52	All buildings in UNSW lower campus
campus	Large	942,878,048	54	xyz	13.62	All buildings, terrain and tree in UNSW lower campus

在不同的 benchmark 上测试不同的查询

small benchmark 还用来探索 block size 和 compression level 的 varying

General VDMS description

Name	Software	Software description	Type
pyf	Python/PySpark	Python and its libraries	File-based
pf	PostgreSQL	Open-source RDBMS	Flat table
pm	PostgreSQL/PostGIS	Open-source RDBMS	Multipoint geometry
pp	PostgreSQL/PostGIS	Open-source RDBMS	PcPatch
mf	MongoDB	Document-oriented NoSQL	Flat table

mm	MongoDB	Document-oriented NoSQL	Multipoint geometry

评价:

1. compression technique: PostgreSQL 使用’dimensional’压缩的结果更好

2. data types (double precision, integer, numeric): 选择 double 更好

3. file formats:

4. block size: block 越小越好，大概 3000 个 voxels 每个 block

workstation 是 4 核的 CPU

laptop 是 2 核的 CPU

Management of voxels using files

Binary file

Feather file

Management of voxels using RDBMS

PostgreSQL

Create database

```
postgres=> CREATE DATABASE VDMS WITH ENCODING='UTF-8' OWNER=wei CONNECTION
LIMIT=100;
CREATE DATABASE
```

```
postgres=> \c VDMS
psql (11.2, server 12.3)
WARNING: psql major version 11, server major version 12.
         Some psql features might not work.
WARNING: Console code page (850) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
You are now connected to database "VDMS" as user "wei".
```

Create tables

Management of voxels using NoSQL

MongoDB

先在 HP laptop 做单机测试, 然后部署到 workstation 做单机数据库

Reference

- http://c.biancheng.net/view/6567.html
- https://www.runoob.com/mongodb/mongodb-tutorial.html
- https://www.w3cschool.cn/mongodb/
- https://piaosanlang.gitbooks.io/mongodb/content/01day/README1.html
- https://www.w3cschool.cn/mongodb/mongodb-window-install.html
- MongoDB 工具篇】MongoDB Compass 介绍 https://blog.csdn.net/Alen_Liu_SZ/article/details/100716346
- MongoDB 系列五（地理空间索引与查询） https://yq.aliyun.com/articles/616432
- MongoDB 地理空间数据存储及检索 https://www.cnblogs.com/oloroso/p/9777141.html
-

MongoDB 下载和安装

- 下载: https://www.mongodb.com/try/download/community
- Widnows 安装: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/
  - 默认将 MongoDB 配置成 Windows service,一旦完成安装, MongoDB service 伴随 OS 自动开启
  - 使用默认的 data 目录和 log 目录
  - 单独安装 MongoDB Compass

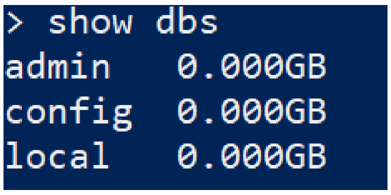
连接 MongoDB 和启动 MongoDB 后台管理 Shell

```
PS C:\WINDOWS\system32> cd 'C:\Program Files\MongoDB\Server\4.4\bin'
PS C:\Program Files\MongoDB\Server\4.4\bin> .\mongo.exe
MongoDB shell version v4.4.1
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
```

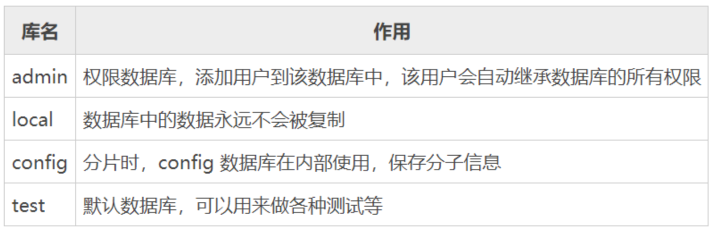
MongoDB 创建和查看数据库

SQL术语/概念	MongoDB术语/概念	解释/说明
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
column	field	数据字段/域
index	index	索引
table joins		表连接,MongoDB不支持
primary key	primary key	主键,MongoDB自动将_id字段设置为主键

- 保留数据库

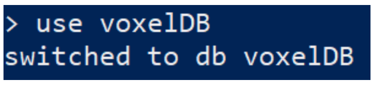


```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```



库名	作用
admin	权限数据库，添加用户到该数据库中，该用户会自动继承数据库的所有权限
local	数据库中的数据永远不会被复制
config	分片时，config 数据库在内部使用，保存分子信息
test	默认数据库，可以用来做各种测试等

- 创建数据库



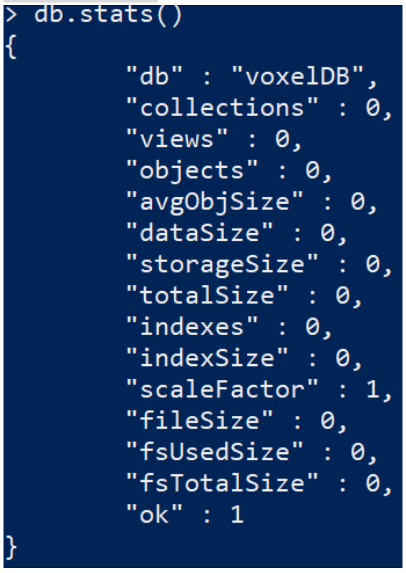
```
> use voxelDB
switched to db voxelDB
```

- 查看数据库

  > show dbs

- 统计数据库信息

  > db. stats()　　//统计数据信息



```
> db.stats()
{
        "db" : "voxelDB",
        "collections" : 0,
        "views" : 0,
        "objects" : 0,
        "avgObjSize" : 0,
        "dataSize" : 0,
        "storageSize" : 0,
        "totalSize" : 0,
        "indexes" : 0,
        "indexSize" : 0,
        "scaleFactor" : 1,
        "fileSize" : 0,
        "fsUsedSize" : 0,
        "fsTotalSize" : 0,
        "ok" : 1
}
```

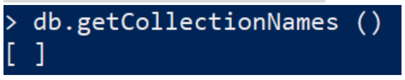
- 删除数据库

  > db.dropDatabase ()　//删除当前数据库

  ?

- 查看集合

  > db.getCollectionNames ()　//查询当前数据下所有的集合名称

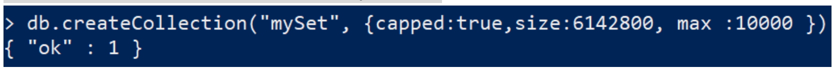


```
> db.getCollectionNames ()
[ ]
```

MongoDB 创建集合

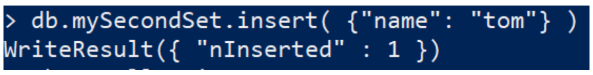
- 创建集合
  - 显示创建集合

    > db.createCollection(name, options)



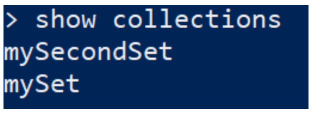
```
> db.createCollection("mySet", {capped:true,size:6142800, max :10000 })
{ "ok" : 1 }
```

  - 隐式自动创建集合



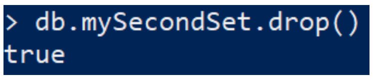
```
> db.mySecondSet.insert( {"name": "tom"} )
WriteResult({ "nInserted" : 1 })
```

- 查看集合的详细信息



```
> show collections
mySecondSet
mySet
```

- 删除集合



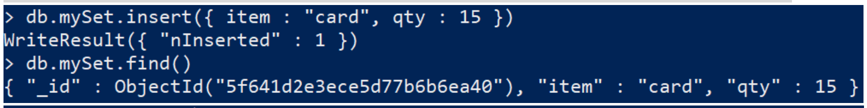
```
> db.mySecondSet.drop()
true
```

MongoDB 插入更新删除文档(数据)

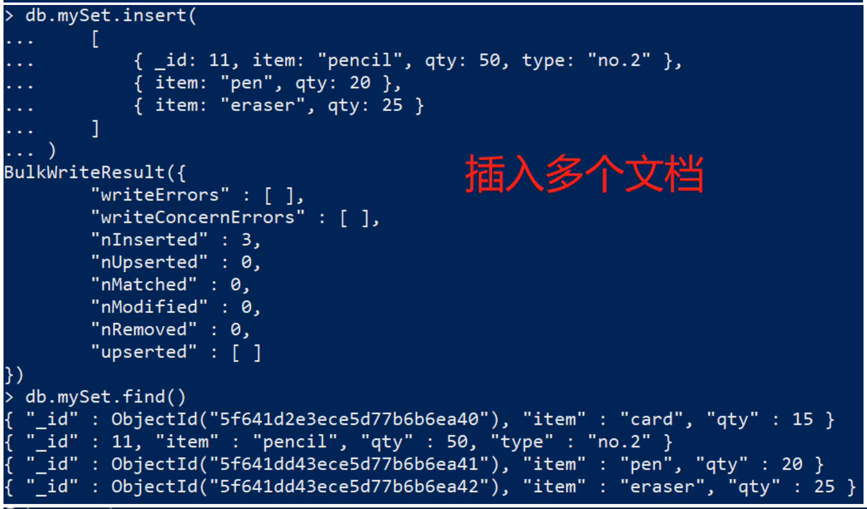
所有存储在集合中的数据都是 BSON 格式。BSON 是一种类 json 的一种二进制形式的存储格式，简称 Binary JSON。

- 插入文档(数据)

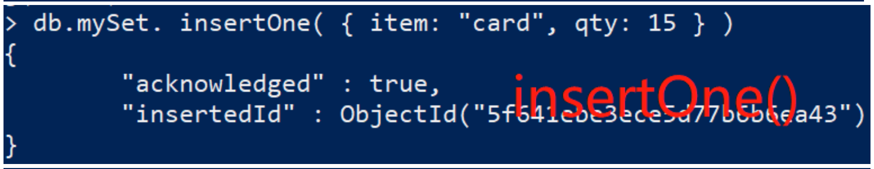
  db.COLLECTION_NAME.insert(<document or array of documents>)



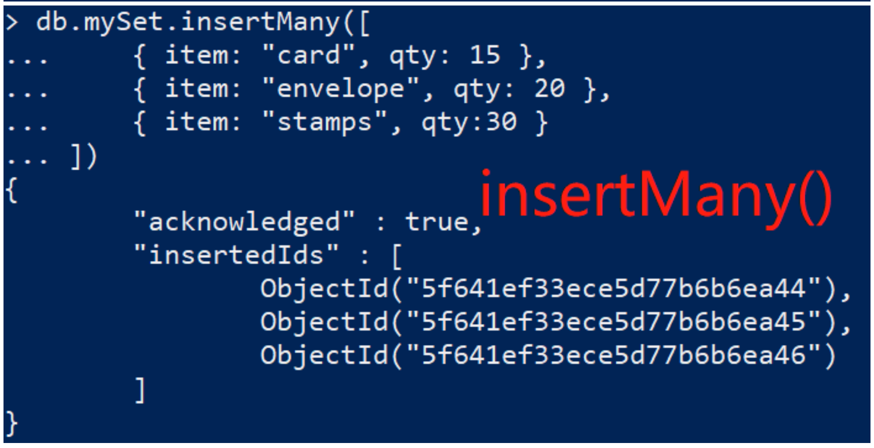
```
> db.mySet.insert({ item : "card", qty : 15 })
WriteResult({ "nInserted" : 1 })
> db.mySet.find()
{ "_id" : ObjectId("5f641d2e3ece5d77b6b6ea40"), "item" : "card", "qty" : 15 }
```



```
> db.mySet.insert(
...     [
...         { _id: 11, item: "pencil", qty: 50, type: "no.2" },
...         { item: "pen", qty: 20 },
...         { item: "eraser", qty: 25 }
...     ]
... )
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
> db.mySet.find()
{ "_id" : ObjectId("5f641d2e3ece5d77b6b6ea40"), "item" : "card", "qty" : 15 }
{ "_id" : 11, "item" : "pencil", "qty" : 50, "type" : "no.2" }
{ "_id" : ObjectId("5f641dd43ece5d77b6b6ea41"), "item" : "pen", "qty" : 20 }
{ "_id" : ObjectId("5f641dd43ece5d77b6b6ea42"), "item" : "eraser", "qty" : 25 }
```



```
> db.mySet. insertOne( { item: "card", qty: 15 } )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5f641ebe3ece5d77b6b6ea43")
}
```



```
> db.mySet.insertMany([
...     { item: "card", qty: 15 },
...     { item: "envelope", qty: 20 },
...     { item: "stamps", qty:30 }
... ])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5f641ef33ece5d77b6b6ea44"),
                ObjectId("5f641ef33ece5d77b6b6ea45"),
                ObjectId("5f641ef33ece5d77b6b6ea46")
        ]
}
```

- 更新或修改文档(数据)

  > db.collection.update(criteria, objNew, upsert, multi)

  > db.collection.save(obj)

```
> db.collection.updateOne(<filter>, <update>, <options>)
> db.collection.updateMany(<filter>, <update>, <options>)
> db.collection.replaceOne(<filter>, <update>, <options>)
```

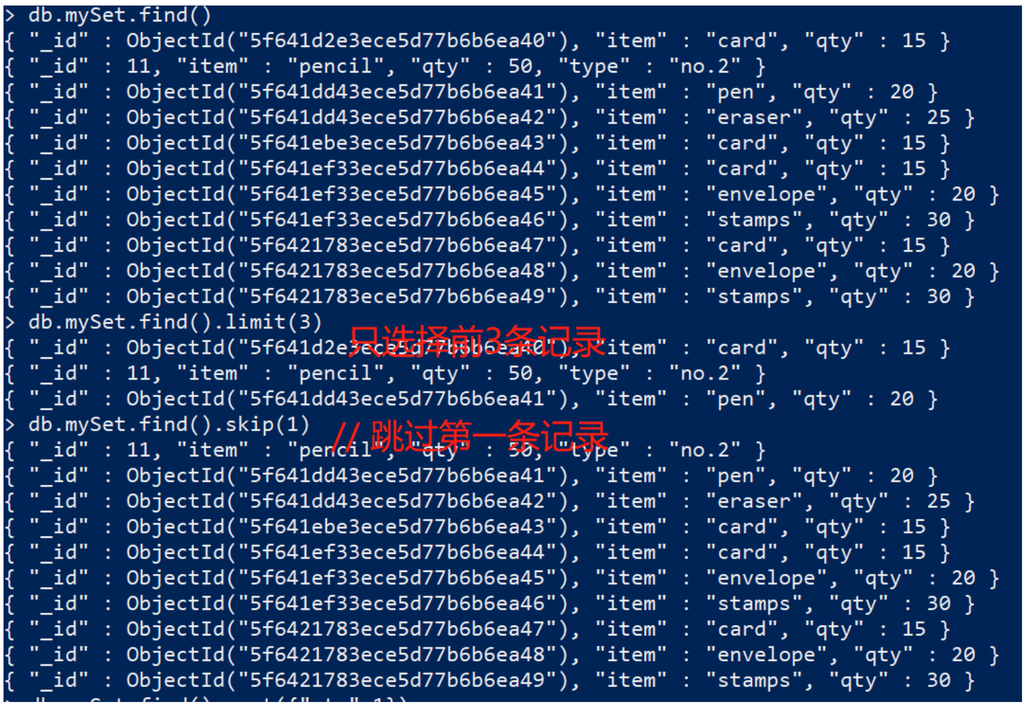
- 删除文档(数据)

```
> db.collection.remove(<query>, {justOne: <boolean>, writeConcern: <document>})
> db.collection.deleteMany()
> db.collection.deleteOne()
```

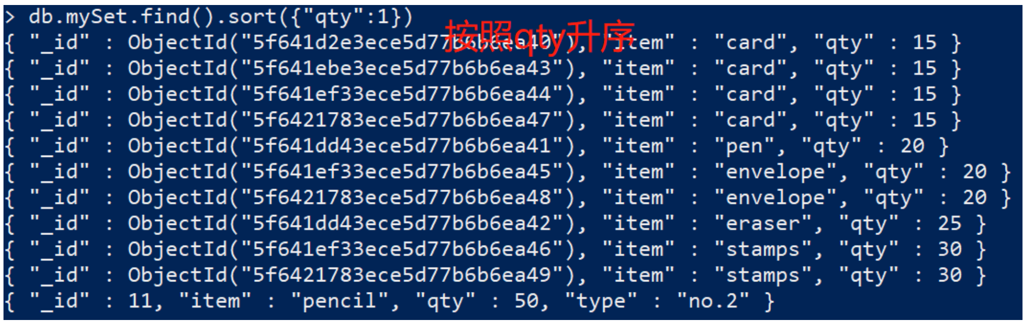
- 查询文档(数据)

```
> db.collection.find(query, projection)
> db.collection.find(query, projection).pretty()
```

操作符	格式	实例	与 RDBMS where 语句比较
等于（=）	{<key> : {<value>}}	db.test.find({price : 24})	where price = 24
大于（>）	{<key> : {$gt : <value>}}	db.test.find({price : {$gt : 24}})	where price > 24
小于（<）	{<key> : {$lt : <value>}}	db.test.find({price : {$lt : 24}})	where price < 24
大于等于（>=）	{<key> : {$gte : <value>}}	db.test.find({price : {$gte : 24}})	where price >= 24
小于等于（<=）	{<key> : {$lte : <value>}}	db.test.find({price : {$lte : 24}})	where price <= 24
不等于（!=）	{<key> : {$ne : <value>}}	db.test.find({price : {$ne : 24}})	where price != 24
与（and）	{key01 : value01, key02 : value02, ...}	db.test.find({name : "《MongoDB 入门教程》", price : 24})	where name = "《MongoDB 入门教程》" and price = 24
或（or）	{$or : [{key01 : value01}, {key02 : value02}, ...]}	db.test.find({$or:[{name : "《MongoDB 入门教程》"},{price : 24}]})	where name = "《MongoDB 入门教程》" or price = 24



```
> db.mySet.find()
{ "_id" : ObjectId("5f641d2e3ece5d77b6b6ea40"), "item" : "card", "qty" : 15 }
{ "_id" : 11, "item" : "pencil", "qty" : 50, "type" : "no.2" }
{ "_id" : ObjectId("5f641dd43ece5d77b6b6ea41"), "item" : "pen", "qty" : 20 }
{ "_id" : ObjectId("5f641dd43ece5d77b6b6ea42"), "item" : "eraser", "qty" : 25 }
{ "_id" : ObjectId("5f641ebe3ece5d77b6b6ea43"), "item" : "card", "qty" : 15 }
{ "_id" : ObjectId("5f641ef33ece5d77b6b6ea44"), "item" : "card", "qty" : 15 }
{ "_id" : ObjectId("5f641ef33ece5d77b6b6ea45"), "item" : "envelope", "qty" : 20 }
{ "_id" : ObjectId("5f641ef33ece5d77b6b6ea46"), "item" : "stamps", "qty" : 30 }
{ "_id" : ObjectId("5f6421783ece5d77b6b6ea47"), "item" : "card", "qty" : 15 }
{ "_id" : ObjectId("5f6421783ece5d77b6b6ea48"), "item" : "envelope", "qty" : 20 }
{ "_id" : ObjectId("5f6421783ece5d77b6b6ea49"), "item" : "stamps", "qty" : 30 }
> db.mySet.find().limit(3)
{ "_id" : ObjectId("5f641d2e3ece5d77b6b6ea40"), "item" : "card", "qty" : 15 }
{ "_id" : 11, "item" : "pencil", "qty" : 50, "type" : "no.2" }
{ "_id" : ObjectId("5f641dd43ece5d77b6b6ea41"), "item" : "pen", "qty" : 20 }
> db.mySet.find().skip(1)
{ "_id" : 11, "item" : "pencil", "qty" : 50, "type" : "no.2" }
{ "_id" : ObjectId("5f641dd43ece5d77b6b6ea41"), "item" : "pen", "qty" : 20 }
{ "_id" : ObjectId("5f641dd43ece5d77b6b6ea42"), "item" : "eraser", "qty" : 25 }
{ "_id" : ObjectId("5f641ebe3ece5d77b6b6ea43"), "item" : "card", "qty" : 15 }
{ "_id" : ObjectId("5f641ef33ece5d77b6b6ea44"), "item" : "card", "qty" : 15 }
{ "_id" : ObjectId("5f641ef33ece5d77b6b6ea45"), "item" : "envelope", "qty" : 20 }
{ "_id" : ObjectId("5f641ef33ece5d77b6b6ea46"), "item" : "stamps", "qty" : 30 }
{ "_id" : ObjectId("5f6421783ece5d77b6b6ea47"), "item" : "card", "qty" : 15 }
{ "_id" : ObjectId("5f6421783ece5d77b6b6ea48"), "item" : "envelope", "qty" : 20 }
{ "_id" : ObjectId("5f6421783ece5d77b6b6ea49"), "item" : "stamps", "qty" : 30 }
```



```
> db.mySet.find().sort({"qty":1})
{ "_id" : ObjectId("5f641d2e3ece5d77b6b6ea40"), "item" : "card", "qty" : 15 }
{ "_id" : ObjectId("5f641ebe3ece5d77b6b6ea43"), "item" : "card", "qty" : 15 }
{ "_id" : ObjectId("5f641ef33ece5d77b6b6ea44"), "item" : "card", "qty" : 15 }
{ "_id" : ObjectId("5f6421783ece5d77b6b6ea47"), "item" : "card", "qty" : 15 }
{ "_id" : ObjectId("5f641dd43ece5d77b6b6ea41"), "item" : "pen", "qty" : 20 }
{ "_id" : ObjectId("5f641ef33ece5d77b6b6ea45"), "item" : "envelope", "qty" : 20 }
{ "_id" : ObjectId("5f6421783ece5d77b6b6ea48"), "item" : "envelope", "qty" : 20 }
{ "_id" : ObjectId("5f641dd43ece5d77b6b6ea42"), "item" : "eraser", "qty" : 25 }
{ "_id" : ObjectId("5f641ef33ece5d77b6b6ea46"), "item" : "stamps", "qty" : 30 }
{ "_id" : ObjectId("5f6421783ece5d77b6b6ea49"), "item" : "stamps", "qty" : 30 }
{ "_id" : 11, "item" : "pencil", "qty" : 50, "type" : "no.2" }
```

- 游标(cursor)
  游标是指对数据一行一行地进行操作，在 MongoDB 数据库中对游标的控制非常简单，只需使用 firid() 函数就可以返回游标。

```
>var cursor = db.test.find()
>while (cursor.hasNext()){
    var doc = cursor.next();
    print(doc.name);  //把每一条数据都单独拿出来进行逐行的控制
    print(doc);  //将游标数据取出来后，其实每行数据返回的都是一个［object BSON］型的内容
    printjson(doc);  //将游标获取的集合以JSON的形式显示
}
```

MongoDB 索引

- 索引类型+创建索引
  - 索引可以提升文档的查询速度，但建立索引的过程需要使用计算与存储资源，在已经建立索引的前提下，插入新的文档会引起索引顺序的重排。
  - MongoDB 的索引是基于 B-tree 数据结构及对应算法形成的。树索引存储特定字段或字段集的值，按字段值排序。索引条目的排序支持有效的等式匹配和基于范围的查询操作。
  - MongoDB 在创建集合时，会默认在 _id 字段上创建唯一索引。该索引可防止客户端插入具有相同字段的两个文档，_id 字段上的索引不能被删除。
  - 在分片集群中，如果不将该 _id 字段用作分片键，则应用需要自定义逻辑来确保 _id 字段中值的唯一性，通常通过使用标准的自生成的 ObjectId 作为 _id。

  > db.collection.createIndex(<key and index type specification>, <options>)
- 地理索引
  地理索引包含两种地理类型，如果需要计算的地理数据表示为类似于地球的球形表面上的坐标，则可以使用 2dsphere 索引。
  通常可以按照坐标轴、经度、纬度的方式把位置数据存储为 GeoJSON 对象。GeoJSON 的坐标参考系使用的是 wgs84 数据。如果需要计算距离（在一个欧几里得平面上），通常可以按照正常坐标对的形式存储位置数据，可使用 2d index。
  > db.collection.createIndex({ <location field> : "2dsphere"})
- 散列索引
  散列（Hashed）索引是指按照某个字段的散列值来建立索引，目前主要用于 MongoDB Sharded Cluster 的散列分片，散列索引只能用于字段完全匹配的查询，不能用于范围查询等。
  > db.collection.createIndex({ _id : "hashed" })
- 查看现有索引
  > db.records.getIndexes()
- 删除索引

> db.collection.dropIndex() // 删除特定索引
> db.collection.dropIndexes() // 删除除 _id 索引之外的所有索引

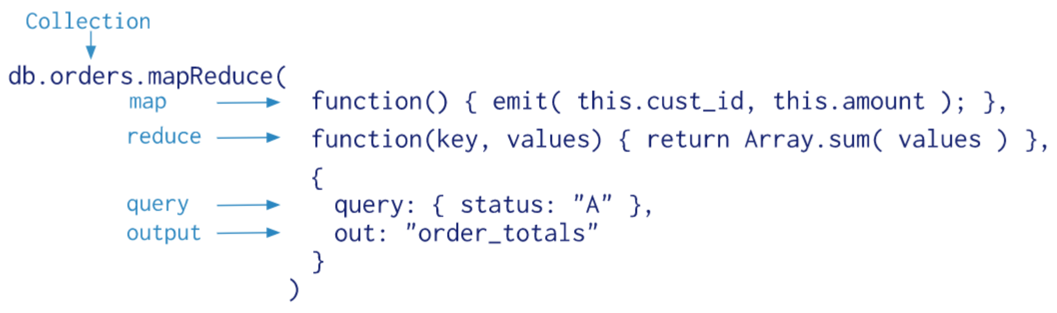
- 修改索引
  若要修改现有索引，则需要删除现有索引并重新创建索引。

MongoDB 聚合查询

聚合操作主要用于处理数据并返回计算结果。聚合操作将来自多个文档的值组合在一起，按条件分组后，再进行一系列操作（如求和、平均值、最大值、最小值）以返回单个结果。

MongoDB 提供了三种执行聚合的方法：聚合管道、map-reduce 和单一目标聚合方法。

- 聚合管道方法
  MongoDB 的聚合框架就是将文档输入处理管道，在管道内完成对文档的操作，最终将文档转换为聚合结果。
  最基本的管道阶段提供过滤器，其操作类似查询和文档转换，可以修改输出文档的形式。其他管道操作提供了按特定字段对文档进行分组和排序的工具，以及用于聚合数组内容（包括文档数组）的工具。
  管道阶段的 RAM 限制为 100MB。若要允许处理大型数据集，则可使用 allowDiskUse 选项启用聚合管道阶段，将数据写入临时文件。
  > db.collection.aggregate([{$match : {< query >},}{$group: {< fieldl >: < field2 >}}])
  - Query 设置统计查询条件，类似于 SQL 的 where，field1 为分类字段，要求使用 _id 名表示分类字段(group by)，field2 为包含各种统计操作符的数字型字段，如 $sum、$avg、$min 等。
- map-reduce 方法
  通常 map-reduce 方法有两个阶段：首先 map 阶段将大批量的工作数据分解执行，然后 reduce 阶段再将结果合并成最终结果。



```
db.orders.mapReduce(
                      function() { emit( this.cust_id, this.amount ); },
                      function(key, values) { return Array.sum( values ) },
                      {
                        query: { status: "A" },
                        out: "order_totals"
                      }
                    )
```

Create table and index

PS H:\> cd 'D:\Program Files\MongoDB\Server\4.4\bin'

PS D:\Program Files\MongoDB\Server\4.4\bin> .\mongo.exe

> show dbs

> use voxelDB

> db.createCollection("voxelpt")

> db.createCollection("voxelmpt")

> show collections

Import data

GeoJSON – Point

```
{
“category”: “building”,
“name”: “Red Centre-H13”,
“ifc”: “IfcDoor”,
"geom": {
        "type": "Point",
        "coordinates": [125.6, 10.1]
   }
}
```

GeoJSON – MultiPoint

```
{
“category”: “building”,
“name”: “Red Centre-H13”,
“ifc”: “IfcDoor”,
"geom": {
        type: "MultiPoint",
  coordinates: [
     [ -73.9580, 40.8003 ],
     [ -73.9498, 40.7968 ],
     [ -73.9737, 40.7648 ],
     [ -73.9814, 40.7681 ]
  ]   }
}
```

Step-1: EPSG: 28356 to EPSG:4326

EPSG:28356	EPSG:4326
• WGS84 Bounds: 150.0000, -37.8000, 156.0000, -21.7000 • Projected Bounds: 189586.6272, 5812134.5296, 810413.3728, 7597371.5494 • Area: Australia - 150°E to 156°E	• WGS84 Bounds: -180.0000, -90.0000, 180.0000, 90.0000 • Projected Bounds: -180.0000, -90.0000, 180.0000, 90.0000 • Area: World

Solution-1: Use ST_Transform() to convert EPSG: 28356 to EPSG:4326 in SQL script, and output it as a csv file.

-

Solution-2: Use GDAL API to code a program to do such transformation in C++.

Rasdaman

Background

Traditional databases do not support the information category of large multidimensional arrays, while rasdaman technology offers distinct array management features whose conceptual model supports arrays of any number of dimensions and over virtually any cell (“***pixel***”, “***voxel***”) type. The rasdaman query language, ***rasql***, is crafted along standard SQL and gives high-level, declarative access to any kind of raster data. Its architecture principle of tile stream processing, together with highly effective optimizations, has proven scalable into multi-Terabyte object sizes.

Array data model. Arrays are determined by their extent (“domain”) and their cell (“pixel”, “voxel”). Over such typed arrays, collections (similar to table in RDBMS) are built. Collections have two columns (attributes), a system-maintained object identifier (OID) and the array itself (i.e., array is a new attribute type). This allows to conveniently embed arrays into relational modeling: foreign keys in conventional tables allow to reference particular array objects, in connection with a domain specification even parts of arrays.

On server side, arrays are stored inside a standard database. To this end, arrays are partitioned into subarrays called **tiles**; each such tile goes into a BLOB (binary large object) in a relational table. This allows conventional relational database systems to maintain arrays of unlimited size. Besides, a spatial index allows to quickly locate the tiles required for determining the tile set addressed by a query.

Query language. The rasdaman query language, rasql, offers raster processing formulated through expressions over raster operations in the style of SQL. Rasql is a full query language, supporting select, insert, update, and delete. Additionally, the concept of a partial update is introduced which allows to selectively update parts of an array.

OGC geo standards support. Rasdaman implements the Open Geospatial Consortium standards for gridded coverages, i.e., multi-dimensional raster data. It offers spatio-temporal access and analytics through APIs based on the OGC data standard *Coverage Implementation Schema* (CIS) and the OGC service standards *Web Map Service* (WMS), *Web Coverage Service* (WCS), and *Web Coverage Processing Service* (WCPS).

Installation

Due to Rasdaman only tested on Linux OS, we choose Virtual Machine + Ubuntu 18.04 (https://releases.ubuntu.com/bionic/) as basic experimental environment. We get preconfectioned packages for installing DEB packages on Debian / Ubuntu; this is the recommended way - among others because the package manager will be able to manage the installation.

Note that, the rasdaman engine in the packages uses embedded **SQLite** for managing its array metadata, and geo service component, petascope, currently still relies on a **PostgreSQL** database.

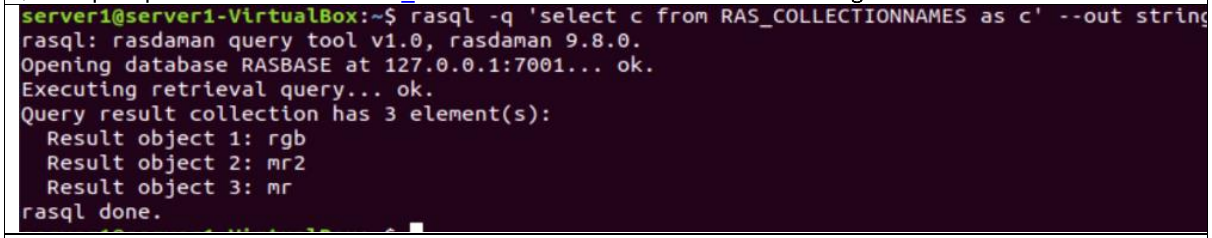
Once the rasdaman installation has been accomplished. We can customize a rasdaman script by updating environment variables, and then we can use rasdaman service script to start/stop rasdaman.

Installation

```
$ wget -O - https://download.rasdaman.org/packages/rasdaman.gpg | sudo apt-
key add -
$ echo "deb [arch=amd64] https://download.rasdaman.org/packages/deb bionic stable
" \ | sudo tee /etc/apt/sources.list.d/rasdaman.list
$ sudo apt-get update
$ sudo apt-get install rasdaman
$ source /etc/profile.d/rasdaman.sh
```

Sample query

```
$ rasql -q 'select c from RAS_COLLECTIONNAMES as c' --out string
```



```
server1@server1-VirtualBox:~$ rasql -q 'select c from RAS_COLLECTIONNAMES as c' --out strin
rasql: rasdaman query tool v1.0, rasdaman 9.8.0.
Opening database RASBASE at 127.0.0.1:7001... ok.
Executing retrieval query... ok.
Query result collection has 3 element(s):
  Result object 1: rgb
  Result object 2: mr2
  Result object 3: mr
rasql done.
```

Rasdaman service

```
# rasdaman installation directory
RMANHOME=/opt/rasdaman
# local user running the rasdaman server
RMANUSER=rasdaman
# runuser, or sudo for older OS
RUNUSER=runuser
# login credentials for non-interactive rasdaman start/stop
RASLOGIN=rasadmin:d293a15562d3e70b6fdc5ee452eaed40
# port on which clients connect to rasdaman
RASMGR_PORT=7001
# options to be passed on to start_rasdaman.sh
START_RASDAMAN_OPTS="-p $RASMGR_PORT"
# options to be passed on to stop_rasdaman.sh
STOP_RASDAMAN_OPTS="-p $RASMGR_PORT"
```

```
$ service rasdaman start
$ service rasdaman stop
$ service rasdaman status
```

To clarify Rasdaman operation, we briefly check the directory structure (the default installation directory is *$RMANHOME= /opt/rasdaman*):

Directory	Description
bin	rasdaman executables, e.g. rasql, start_rasdaman.sh, …
data	Path where the server stores array tiles as files; this directory can get big, it is recommended to make it a link to a sufficiently large disk partition.
etc	Configuration files, e.g. rasmgr.conf

Small benchmark

This section contains the description of a small benchmark focussed on the storage and management of the voxel data. We use the term "small" to reflect the fact that we use a small test dataset and a small subset of test queries. The goal of the project is to develop a system that can deal with a much larger amount of points while providing a much larger set of functionalities.

We use a small dataset which consists of 22,333,289 voxels of the building – 'UNSW Village' in the UNSW Kensington campus in Australia. Figure 1 depicts the 3D spatial representation of the building. The voxels in this dataset belong to one of the more than 50 objects of UNSW Kensington campus. The input data is provided as a XYZ file and its size is 329 MB. Each voxel has 4 dimensions which are listed in the Table 1. Note that the spatial coordinates, which are given in meters, are stored in 32 bits. Actually, they are stored as scaled integers, which can also have an offset. The scale is 0.2 which means that the coordinates are stored in meter. In this case, the X, Y, Z coordinate also has an offset of 336000, 6245250 and 20 meters, respectively.

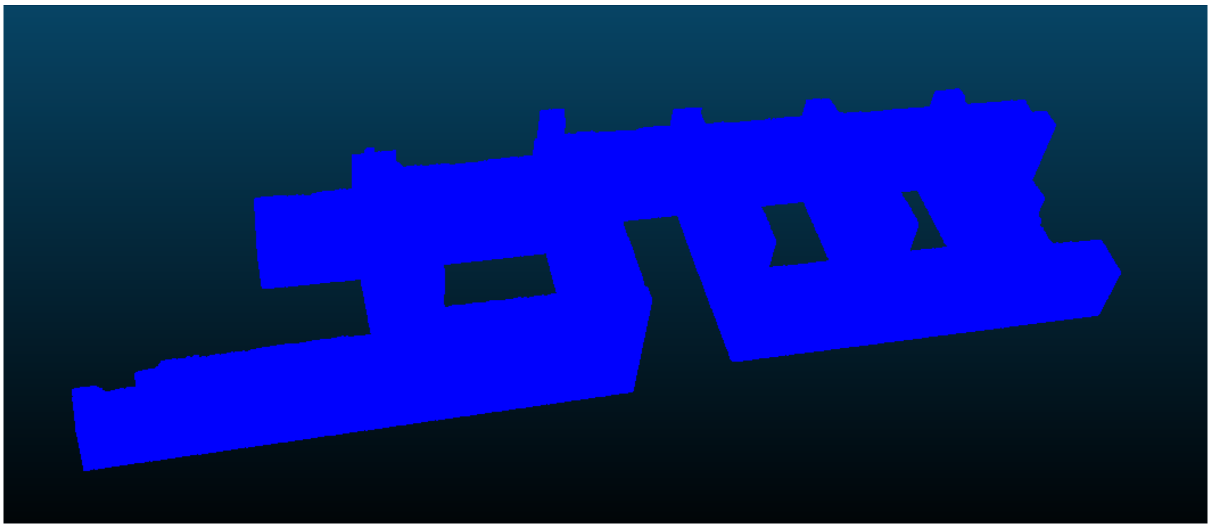


Figure 1: 3D representation of the area used in the small benchmark

Name	Size[bits]	Offset[bytes]
X	32	0
Y	32	4
Z	32	8
object ID	8	12

Table 1: Dimensions of the voxels in the input XYZ file. Only X, Y and Z contain valuable data

Medium benchmark

Large benchmark

Management of voxels under Distribution

PostgreSQL

Ref: https://cloud.tencent.com/developer/article/1563023

- **Citus** (https://www.citusdata.com/)
- pgxc && pgxl (https://www.postgres-xl.org/documentation/pgxc-ctl.html)
- **Greenplum** (https://greenplum.org/)

MongoDB

Reference

[1] A Distributed Storage and Access Approach for Massive Remote Sensing Data in MongoDB

基于 MongoDB 的 sharding technology, 设计并建立了分布式集群体系结构, 变化 sharding strategies,与 PostgreSQL 比较 storage performance and access performance.

[2] The Method of Cloudizing Storing Unstructured LiDAR Point Cloud Data by MongoDB

[3] Research on Storage and Processing of MongoDB for Laser Point Cloud under Distribution

BSON 与 GridFS 是相互补充的关系

Install MongoDB Community Edition on Ubuntu

```
$ wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add –

$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.4
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list

$ sudo apt-get update

$ sudo apt-get install -y mongodb-org

$ ps --no-headers -o comm 1
```

Run MongoDB Community Edition

```
$ sudo systemctl start mongod

$ sudo systemctl status mongod

$ sudo systemctl stop mongod

$ sudo systemctl restart mongod

$ mongo
```

Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations. MongoDB shards data at the collection level, distributing the collection data across the shards in the cluster.

MongoDB 分布式集群架构

MongoDB 有三种集群部署模式，分别为主从复制（Master-Slaver）、副本集（Replica Set）和分片（Sharding）模式。Sharding 模式追求的是高性能，而且是三种集群中最复杂的。在实际生产环境中，通常将 Replica Set 和 Sharding 两种技术结合使用。

- 主从复制
  Master-Slaver 是一种主从副本的模式，目前已经不推荐使用。
- 副本集
  Replica Set 模式取代了 Master-Slaver 模式，是一种互为主从的关系。Replica Set 将数据复制多份保存，不同服务器保存同一份数据，在出现故障时自动切换，实现故障转移，在实际生产中非常实用。
  当集群中主节点发生故障时，副本集可以自动投票，选举出新的主节点，并引导其余的从节点连接新的主节点，而且这个过程对应用是透明的。
- 分片
  Sharding 模式适合处理大量数据，它将数据分开存储，不同服务器保存不同的数据，所有服务器数据的总和即为整个数据集。
  MongoDB 的分片机制允许创建一个包含许多台机器的集群，将数据子集分散在集群中，每个分片维护着一个数据集合的子集。与副本集相比，使用集群架构可以使应用程序具有更强大的数据处理能力。
- 构建一个 MongoDB 的分片集群，需要三个重要的组件，分别是分片服务器（Shard Server）、配置服务器（Config Server）和路由服务器（Route Server）。
  - 每个 Shard Server 都是一个 mongod 数据库实例，用于存储实际的数据块。整个数据库集合分成多个块存储在不同的 Shard Server 中。在实际生产中，一个 Shard Server 可由几台机器组成一个副本集来承担，防止因主节点单点故障导致整个系统崩溃。
  - 这是独立的一个 mongod 进程，保存集群和分片的元数据，在集群启动最开始时建立，保存各个分片包含数据的信息。
  - 这是独立的一个 mongos 进程，Route Server 在集群中可作为路由使用，客户端由此接入，让整个集群看起来像是一个单一的数据库，提供客户端应用程序和分片集群之间的接口。

主机名	IP	端口信息
Node1	149.171.161.38	momgod shard1: 27018（rs1） momgod shard2: 27018（rs2） momgod shard3: 27018（rs3） mongod config1: 27030 mongos router1: 27017
Node2	149.171.161.36	momgod shard1: 27018（rs1） momgod shard2: 27018（rs2） momgod shard3: 27018（rs3） mongod config1: 27030 mongos router1: 27017

Node3	149.171.161.37	momgod shard1: 27018（rs1） momgod shard2: 27018（rs2） momgod shard3: 27018（rs3） mongod config1: 27030 mongos router1: 27017

副本集和分片联合部署的基本思路是先建立副本集，然后将每个副本集作为整体建立分片，如在上表中，集群有两个副本集 rs1 和 rs2，每个副本集由三个成员组成，分别部署在三台机器 Node1、Node2 和 Node3 上。

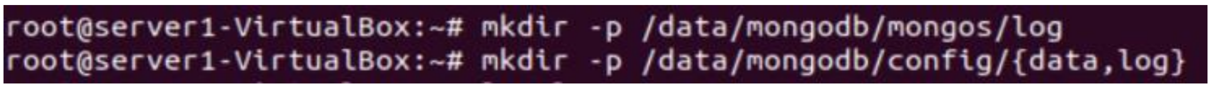
另外，在每台机器上启动一个 mongod 和 mongos 实例分别用于实现 Config Server 和 Route Server 的功能，使用三台机器备份的方式保证集群的可靠性。mongod is the daemon, mongo is the client, mongos is the 'MongoDB Shard Utility'.

先 stop mongod: sudo systemctl stop mongod

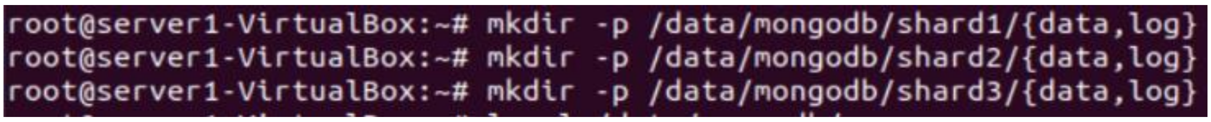
https://www.cnblogs.com/littleatp/p/8563273.html

https://www.thomas-krenn.com/en/wiki/Network_Configuration_in_VirtualBox

https://blog.csdn.net/Powerful_Fy/article/details/103615902



```
root@server1-VirtualBox:~# mkdir -p /data/mongodb/mongos/log
root@server1-VirtualBox:~# mkdir -p /data/mongodb/config/{data,log}
```



```
root@server1-VirtualBox:~# mkdir -p /data/mongodb/shard1/{data,log}
root@server1-VirtualBox:~# mkdir -p /data/mongodb/shard2/{data,log}
root@server1-VirtualBox:~# mkdir -p /data/mongodb/shard3/{data,log}
```

Ref:

- https://docs.mongodb.com/manual/sharding/
- https://www.cnblogs.com/duanxz/p/10730121.html
- https://blog.csdn.net/duanbeibei/article/details/89334866

HBase

Spark