Università degli studi di Messina

Dipartimento di Matematica e Informatica

Progetto Database 2

Realizzare un database basato su MongoDB di un Hospistal Information System (HIS)

Studenti: Pafumi Francesco, Gangemi Salvatore

Matricola: 446322, 443193

A.A 2015/2016

Professore : Antonio Celesti

Indice

1. *Richiesta*

Si richiede la realizzazione di un applicativo che simuli il funzionamento e le prestazioni di un database NoSql su uno schema preesistente. Il DBMS che è stato richiesto per l'utilizzo è MongoDB.

L'applicativo richiede da uno schema preesistente andare a generare in modo casuale i dati che andranno in seguito a popolare il database iniziale con dei dataset di 10,100,1000,10000,100000 elementi.

Si richiede di analizzare le differenze in termini di costi ed efficienza con altri gruppi che stanno effettuando lo stesso progetto su database di tipo NoSql differenti. Si richieste infine di andare a graficare i risultati ottenuti.

2. *Ambiente di sviluppo*

L'applicazione è stata implementata utilizzando il linguaggio di programmazione Python 2.7
L'editor utilizzato per la stesura del codice è PyCharm.

3. *Architettura applicazione*

L'architettura dell'applicazione è stata strutturata cercando di massimizzare la modularità e la riusabilità.

Di seguito sono riportati gli elementi che compongono la progettazione e il modo in cui sono stati collegati fra di loro:

- Connessione al DBMS

```
client = MongoClient("localhost", 27017)
db = client.test
```

Per l'utilizzo di MongoDB tramite python è stata utilizzata la libreria pymongo in grado di sincronizzare e permettere un utilizzo più efficiente e veloce di MongoDB e della sua architettura.
Tramite questo metodo è possibile connettersi al server specificando indirizzo e porta.

- caricaDatiDaJson

```
def caricaDatiDaJson(jsonFile):
    with open(jsonFile) as data_file:
        data = json.load(data_file)
        return data
```

Per andare ad effettuare i test sui tipi diversi di dataset composti da diversi elementi è stato utilizzato il seguente metodo che, dato il path, permette un facile import del file json

- queryInserisciDati

```
#PAZIENTE
def queryInserisciDati(db,id,name,width,height,l_shank,l_thigh,lokomat_shank,lokomat_thigh,
                       lokomat_recorded,version,legtype,lwalk_training_duration,lwalk_distance,step_datas):
    collection = db.Patients
    collection.insert({
        "id" : id,
        "name" : name,
        "width" : width,
        "height": height,
        "l_shank" : l_shank,
        "l_thigh" : l_thigh,
        "lokomat_shank" : lokomat_shank,
        "lokomat_thigh" : lokomat_thigh,
        "lokomat_recorded" : lokomat_recorded,
        "version" : version,
        "legtype" : legtype,
        "lwalk_training_duration" : lwalk_training_duration,
        "lwalk_distance" : lwalk_distance,
        "step_datas" : step_datas
    })
```

Tramite il metodo queryInserisciDati è stato possibile andare a popolare il database, passando i valori presi dal file json attraverso un parser.

- Main

```
listaDataSet = ['ds10.json','ds100.json','ds1000.json','ds10000.json','ds100000.json']
#listaDataSet = ['ds10.json']

directory = "result"
if not os.path.exists(directory):
    os.makedirs(directory)

listaQuery = [primaQuery, secondaQuery, terzaQuery]
tempototaleIniziale = time.time()
#Eseguo il tutto per ogni dataset
for dataset in listaDataSet:
    queryEliminaCollection(db)
    data = utility.caricaDatiDaJson("../../../dataset_lokomat/" + dataset)
    #Inserisco dataSet
    for i in data:
        queryInserisciDati(db,i['id'],i['name'],i['width'],i['height'],i['l_shank'],i['l_thigh'],i['lokomat_shank'],i['lokomat_thigh']
                           ,i['lokomat_recorded'],i['version'],i['legtype'],i['lwalk_training_duration'],i['lwalk_distance']
                           ,i['step_datas'])
    #Eseguo query
    i=0
    for query in listaQuery:
        i = i + 1
        tempi = []
        #Stessa query 31 volte
        for _ in range(0, 31):
            tempo = time.time()
            query(db)
            tempoFinale = time.time() - tempo
            tempi.append(tempoFinale)
        plt.plot(tempi)
        plt.savefig('result/' + dataset + 'Query' + str(i) + '.png')
        plt.clf()

tempototaleFinale = time.time()
print str("Il tempo totale e : ") + str(tempototaleFinale-tempototaleIniziale)
```

E’ possibile iniziare la simulazione lanciando direttamente il Main.

Query utilizzate:

1) Selezionare tutti i pazienti

```
def primaQuery(db):
    collection = db.Patients
    patients = collection.find({})
    return patients
```

2) Selezionare tutti i pazienti con uno specifico nome

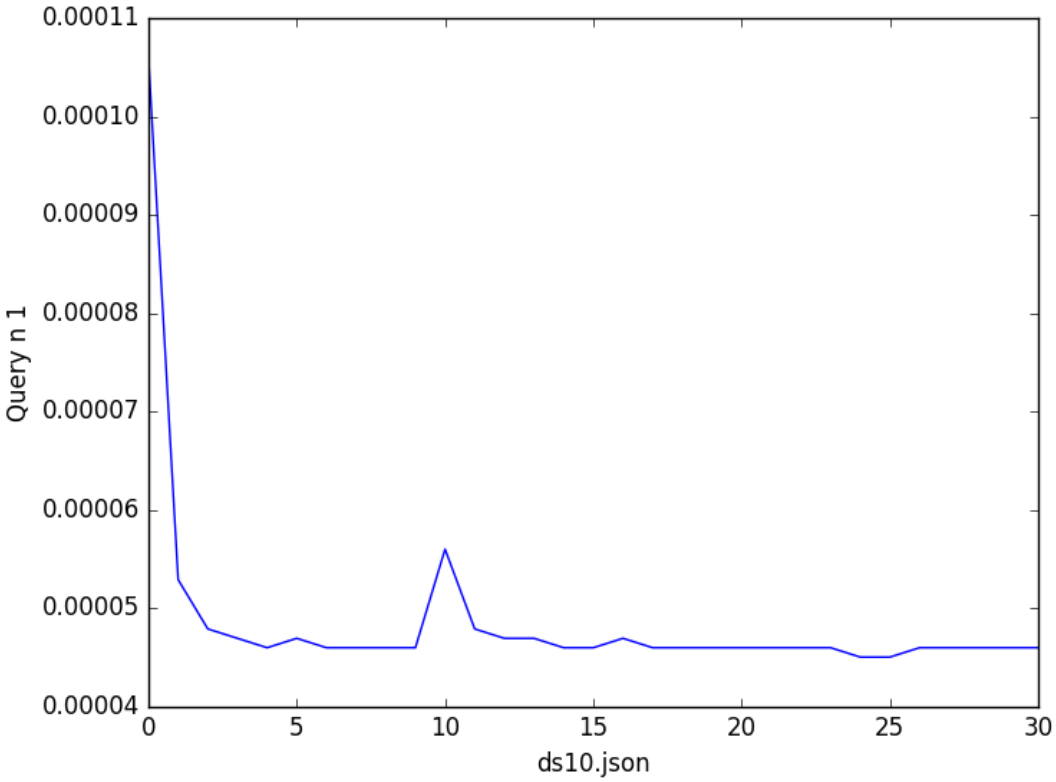
```
def primaQuery(db):
    collection = db.Patients
    patients = collection.find({})
    return patients
```

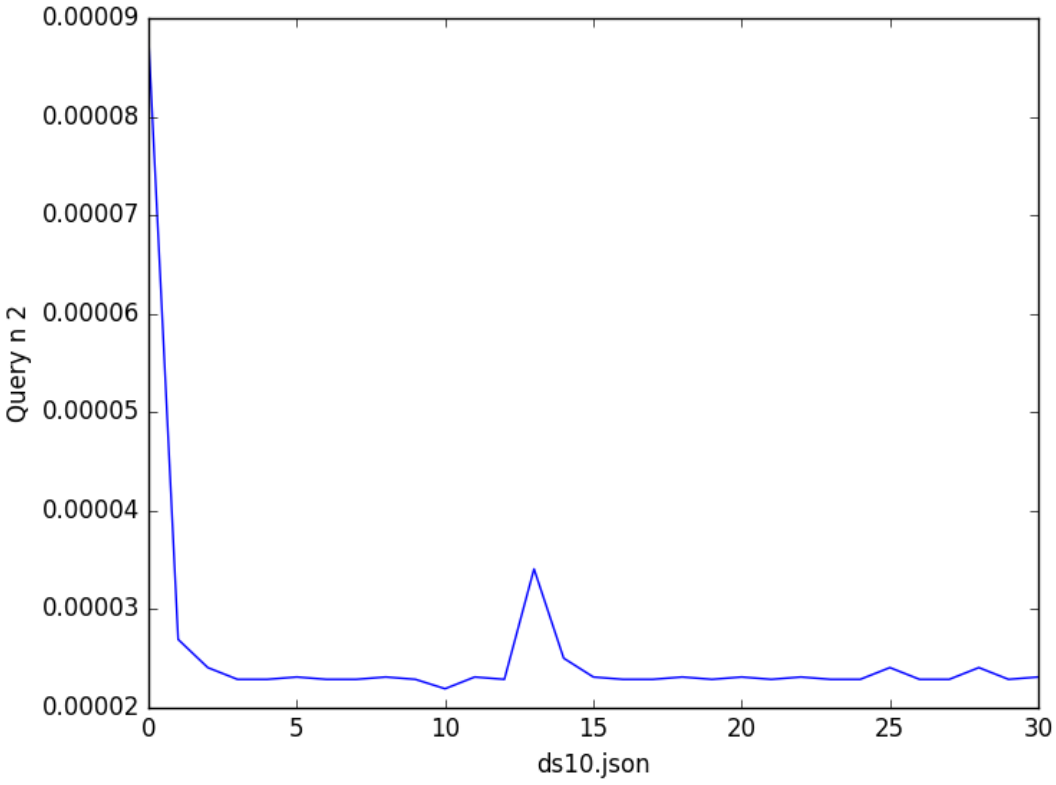
3) Selezionare tutti i pazienti con almeno 5 misurazioni che hanno un training_duration < x e width != y

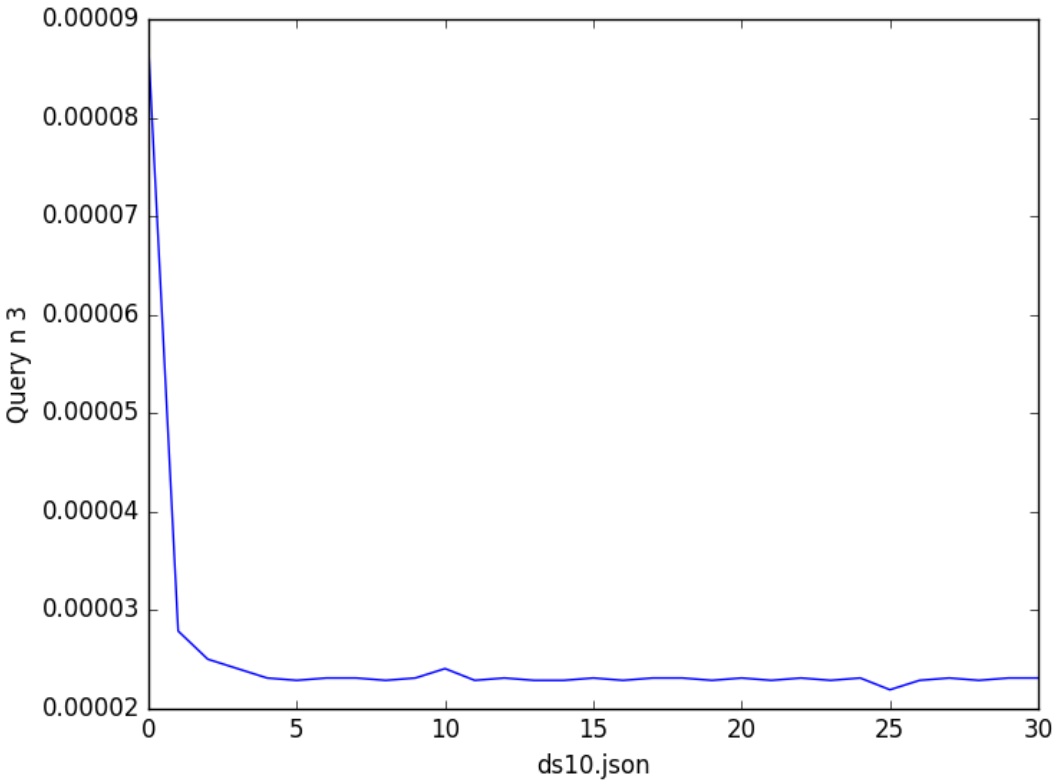
```
def terzaQuery(db):
    collection = db.Patients
    patients = collection.find({'lwalk_training_duration' : { '$lt' : 5000} , 'width' : { '$ne' : 0.7830777515902633 }, 'step_datas' : { '$size' : 2} })
    return patients
```

Statistiche:

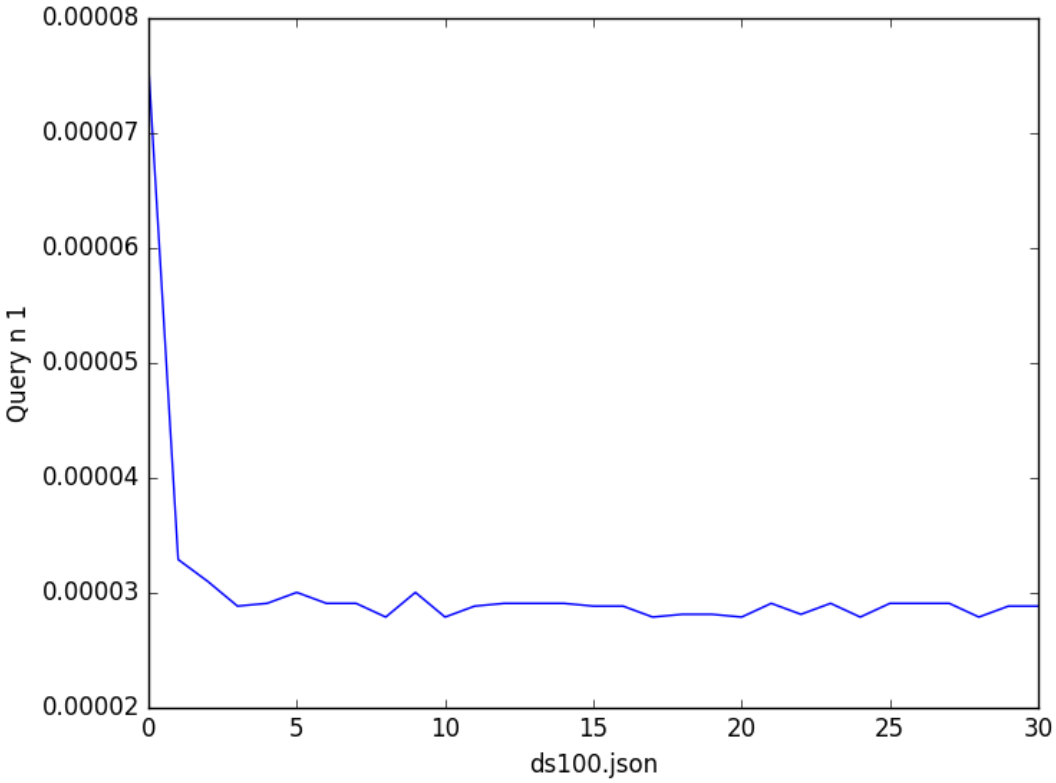
- File Json: ds10

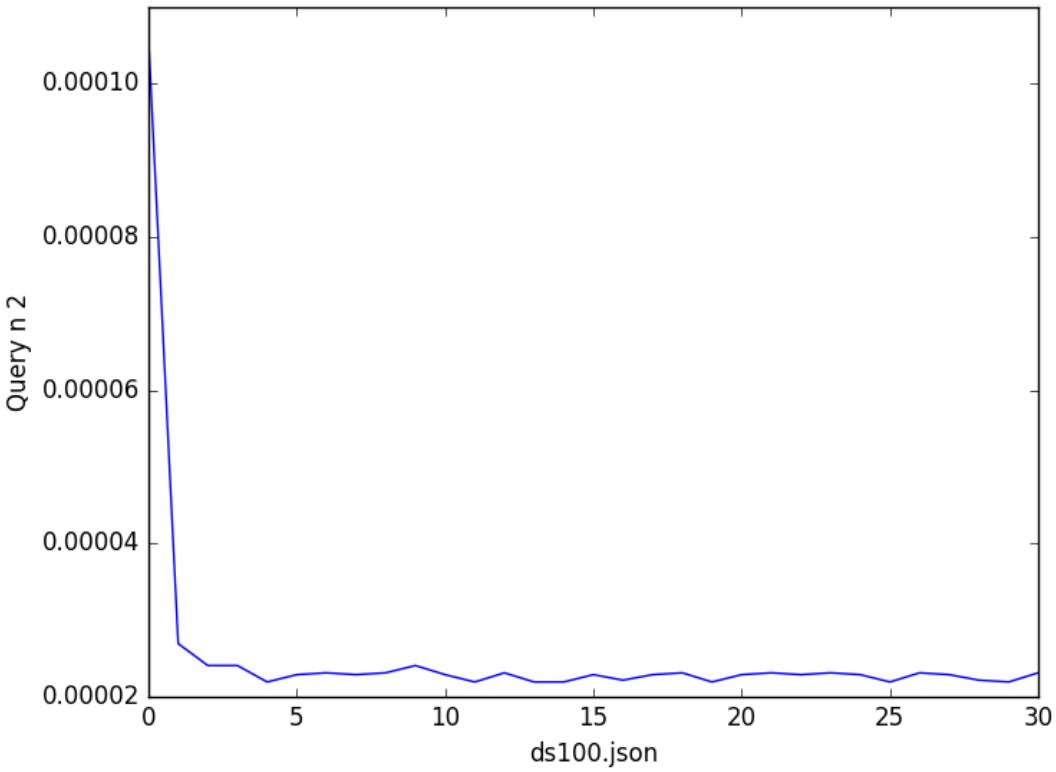


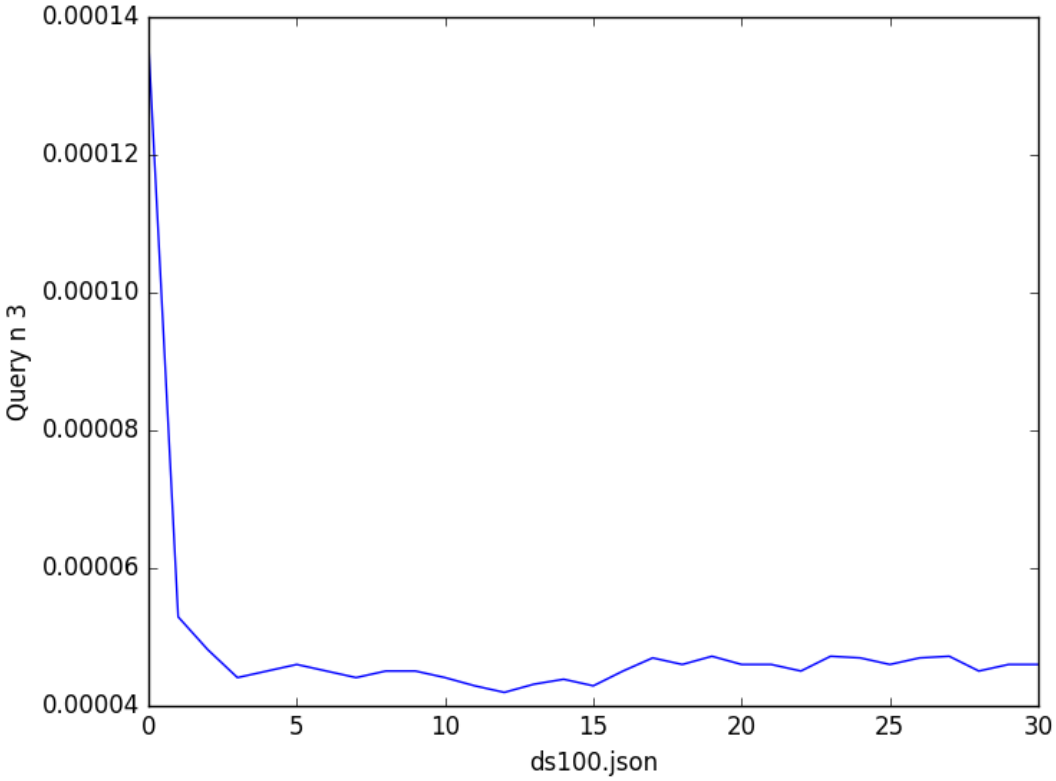




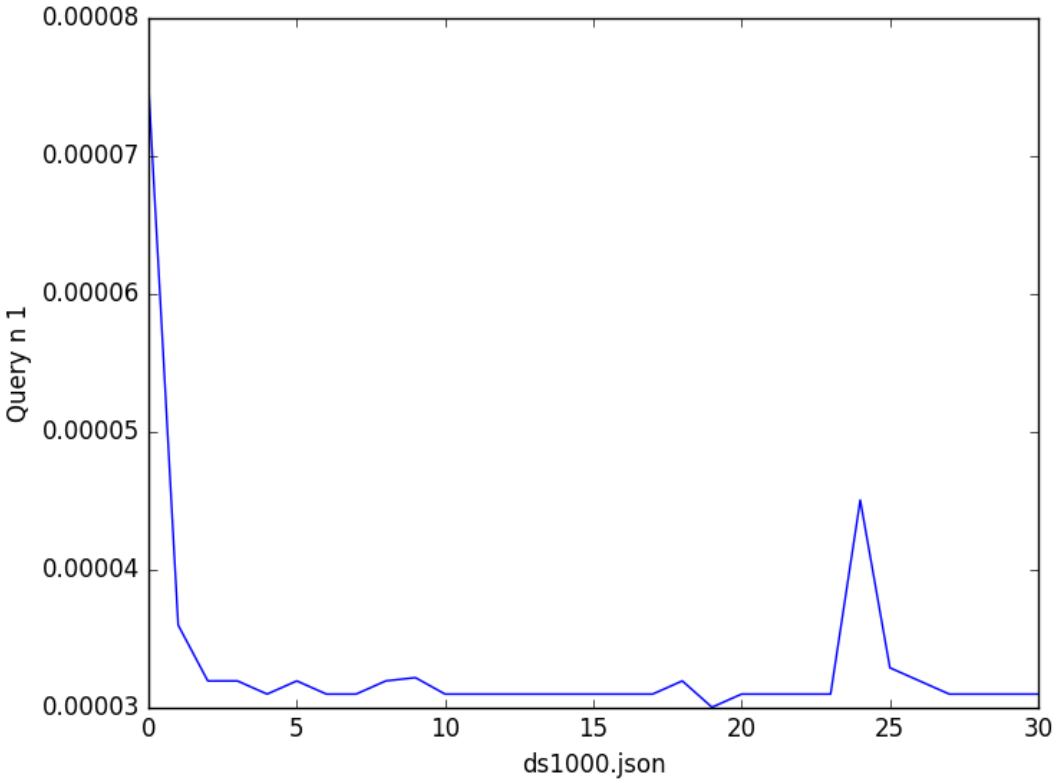
- File Json: ds100

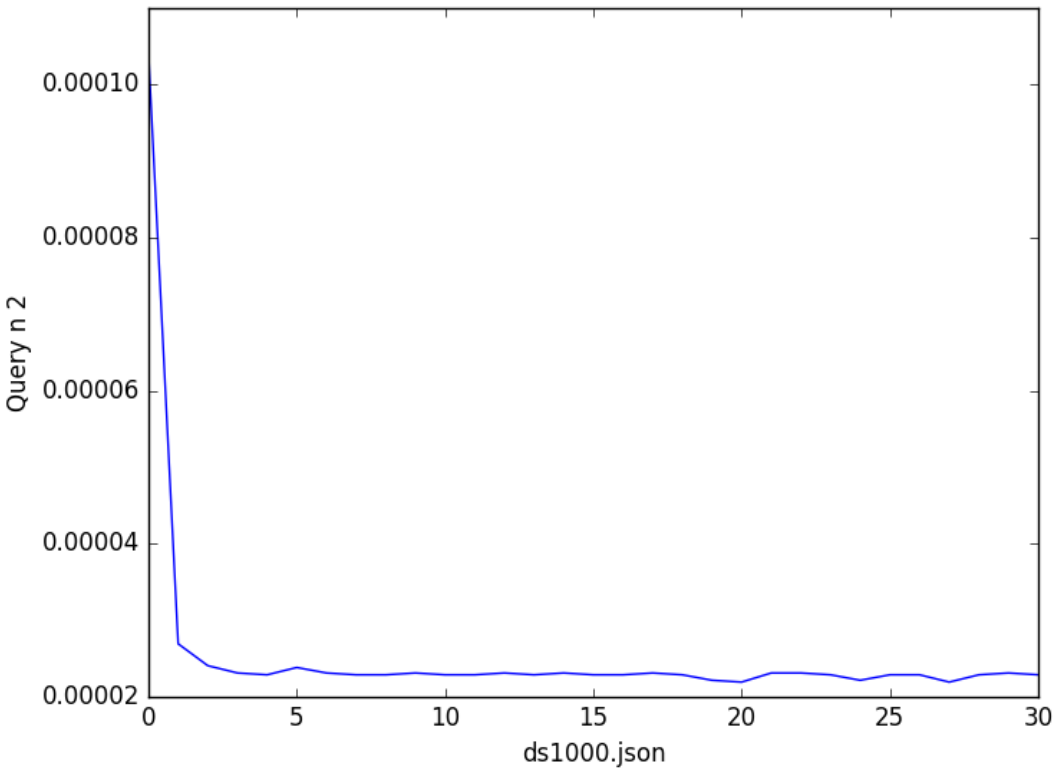


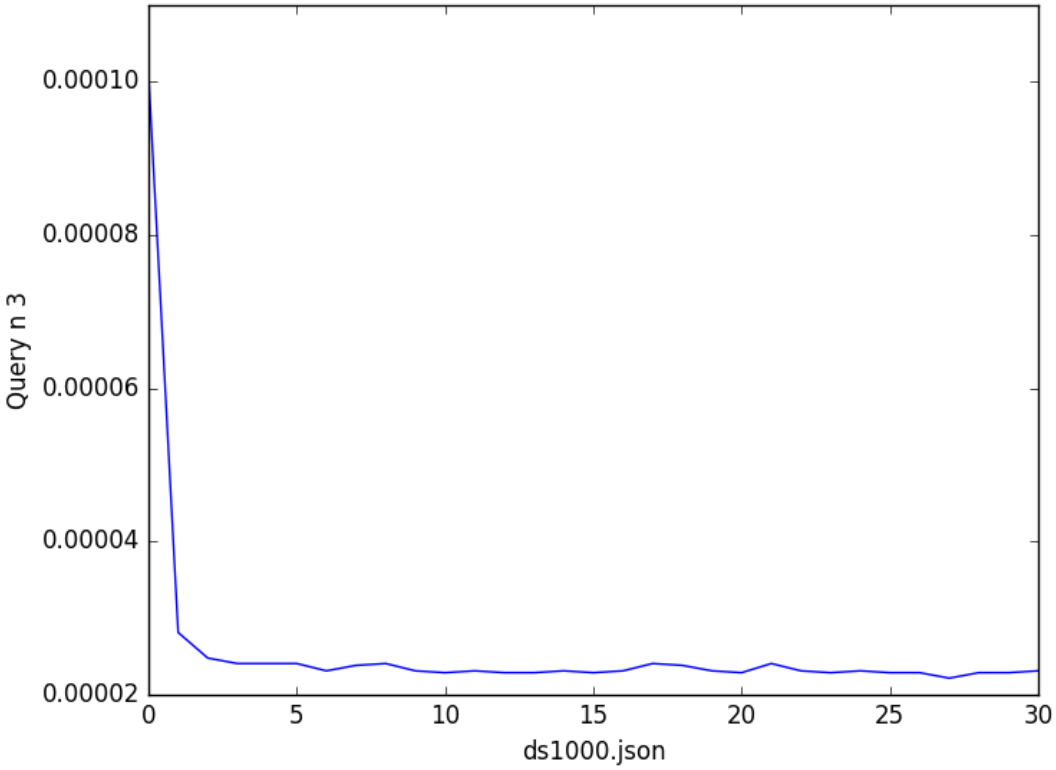

0.00010
0.00008
0.00006
0.00004
0.00002
Query n 2
0
5
10
15
20
25
30
ds100.json


0.00014
0.00012
0.00010
0.00008
0.00006
0.00004
Query n 3
0
5
10
15
20
25
30
ds100.json

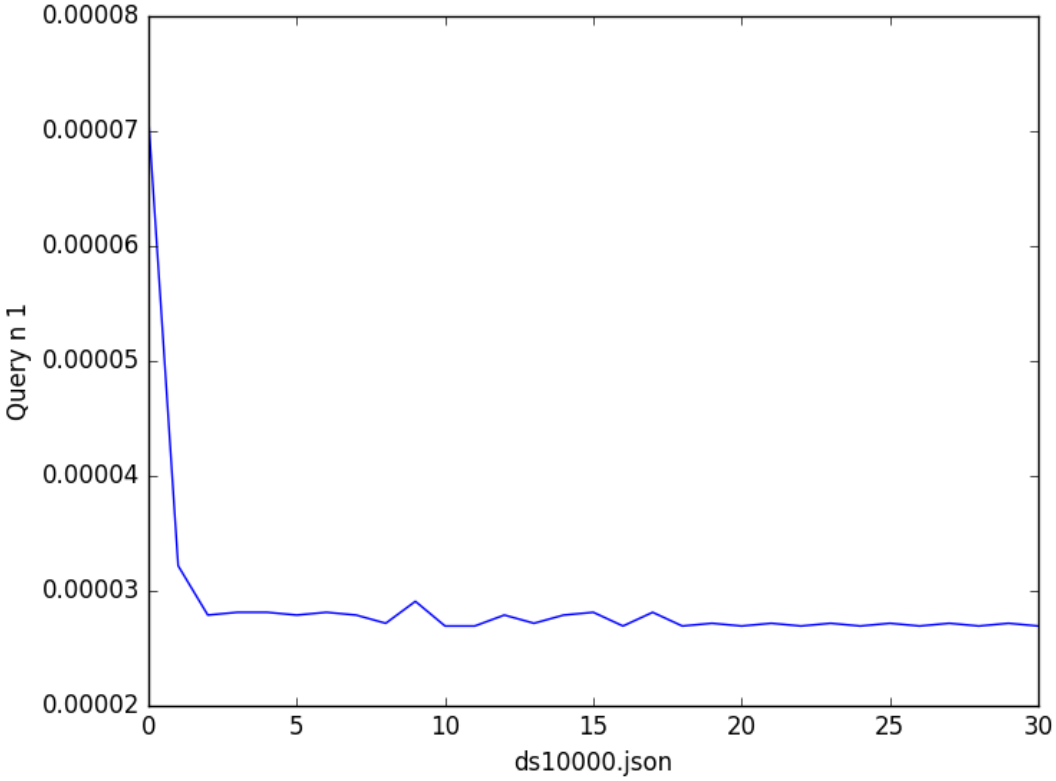
- File Json: ds1000

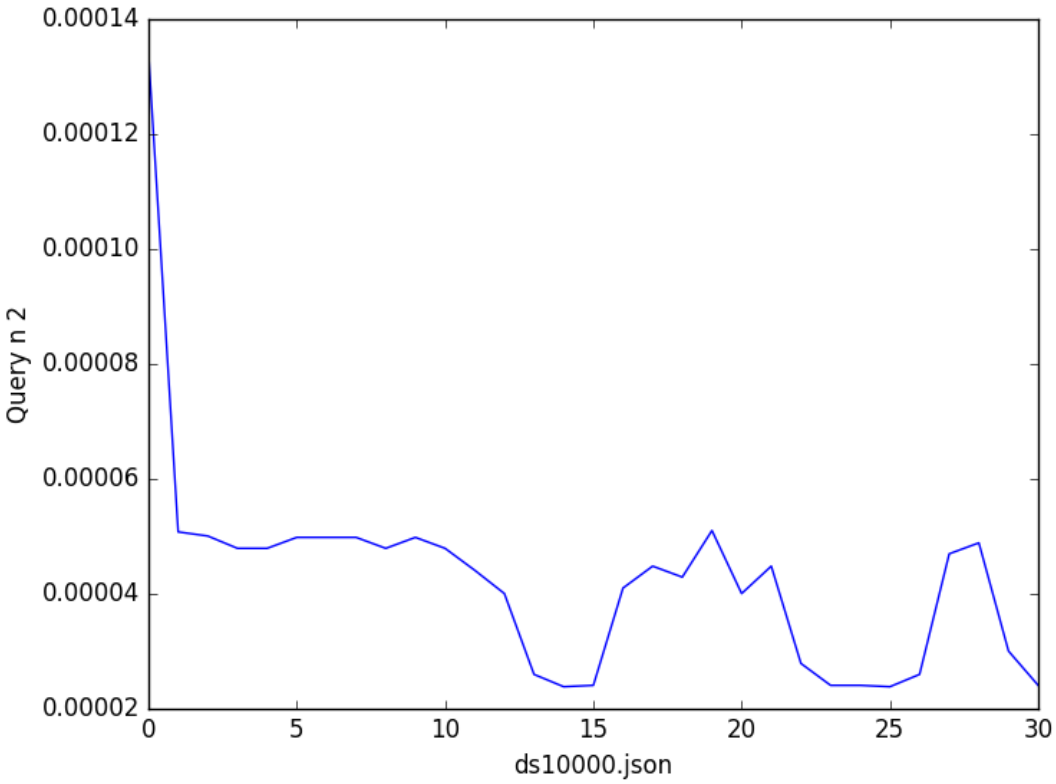


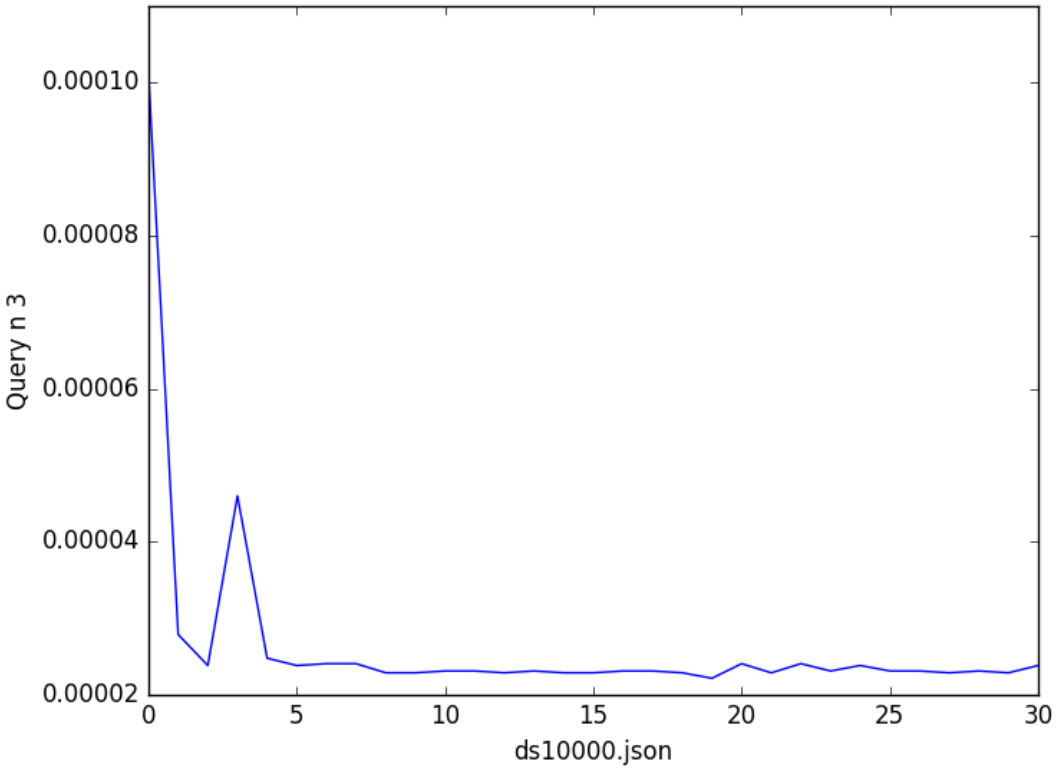




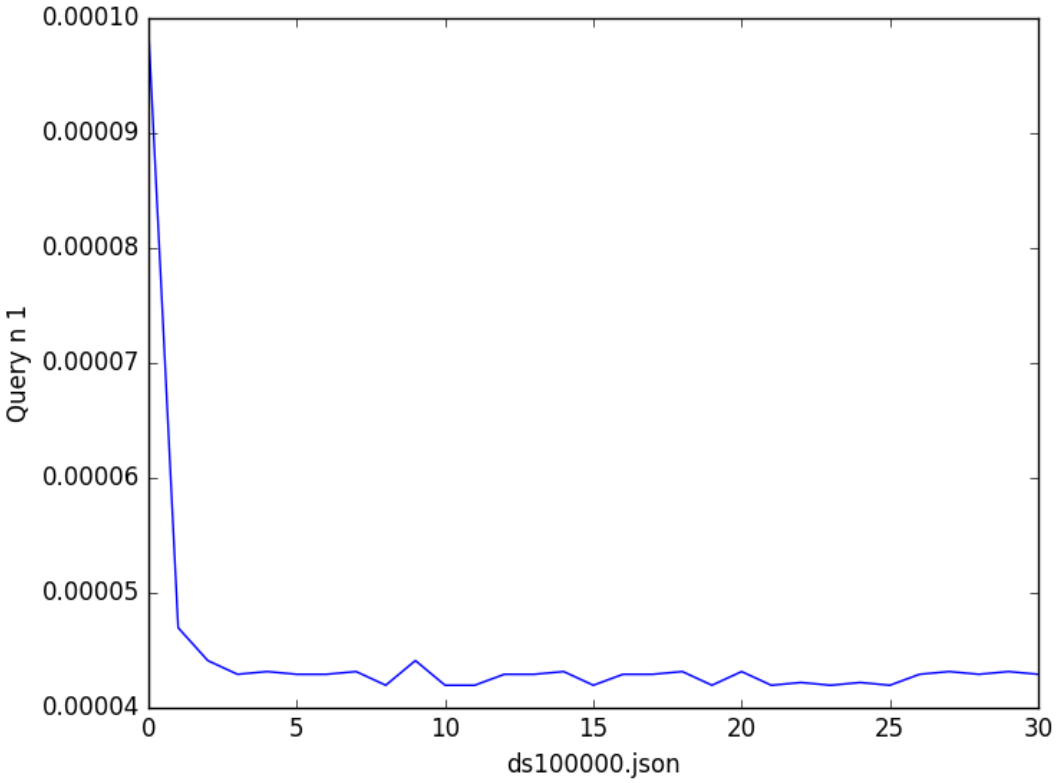
- File Json: ds10000

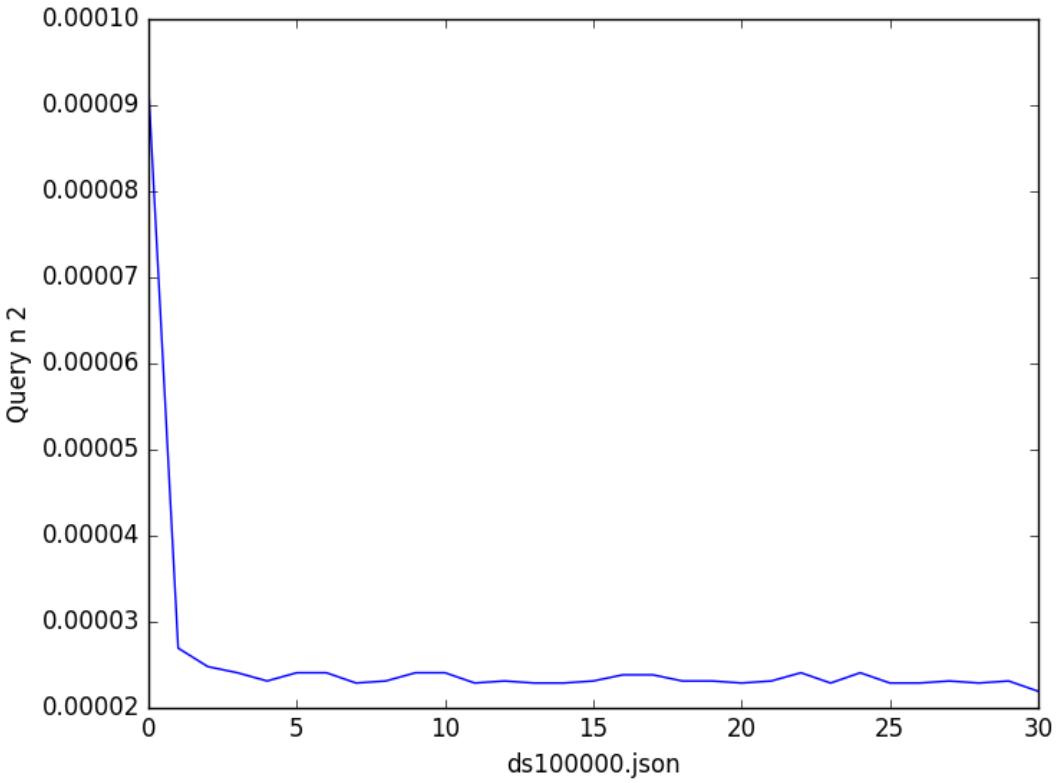


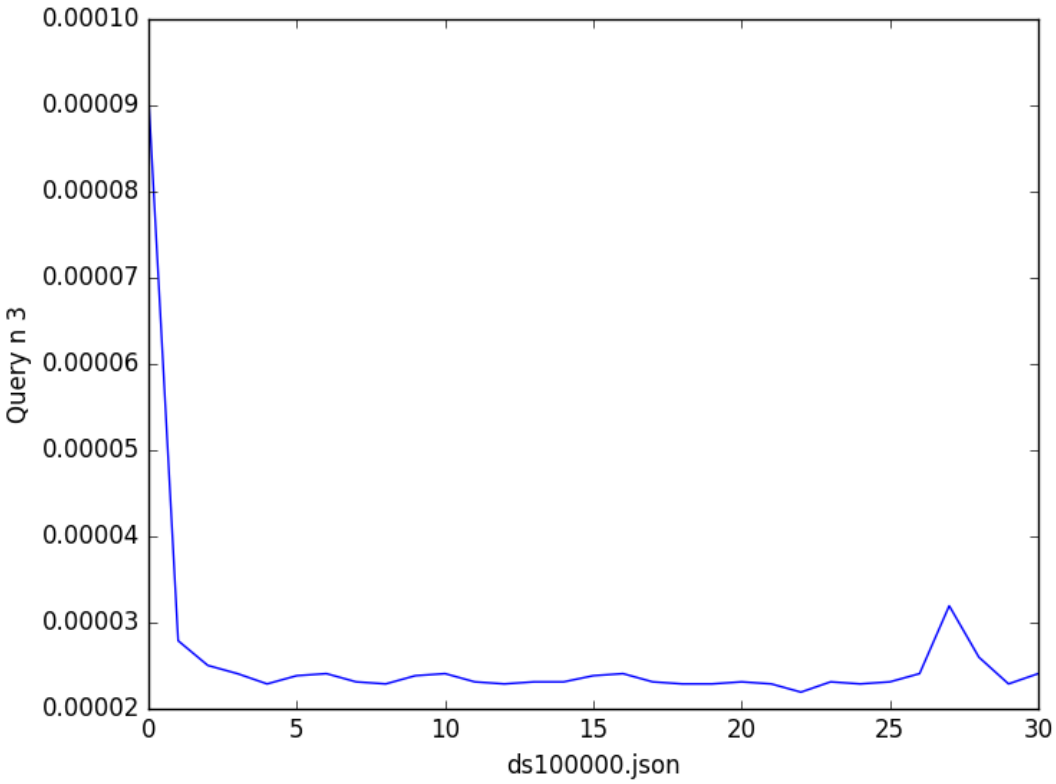

0.00014
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
Query n 2
0
5
10
15
20
25
30
ds10000.json


0.00010
0.00008
0.00006
0.00004
0.00002
Query n 3
0
5
10
15
20
25
30
ds10000.json

- File Json: ds100000






0.00010
0.00009
0.00008
0.00007
0.00006
0.00005
0.00004
0.00003
0.00002
Query n 3
0
5
10
15
20
25
30
ds100000.json