

Relazione tirocinio interno

Santoro Matteo 3CFU

Obbiettivi

Poter capire attraverso la sperimentazione quale modellazione dei dati risulti piu' efficiente per poter immagazzinare una grande quantita' di dati per poi poter effettuare delle interrogazioni e valutare le performance.

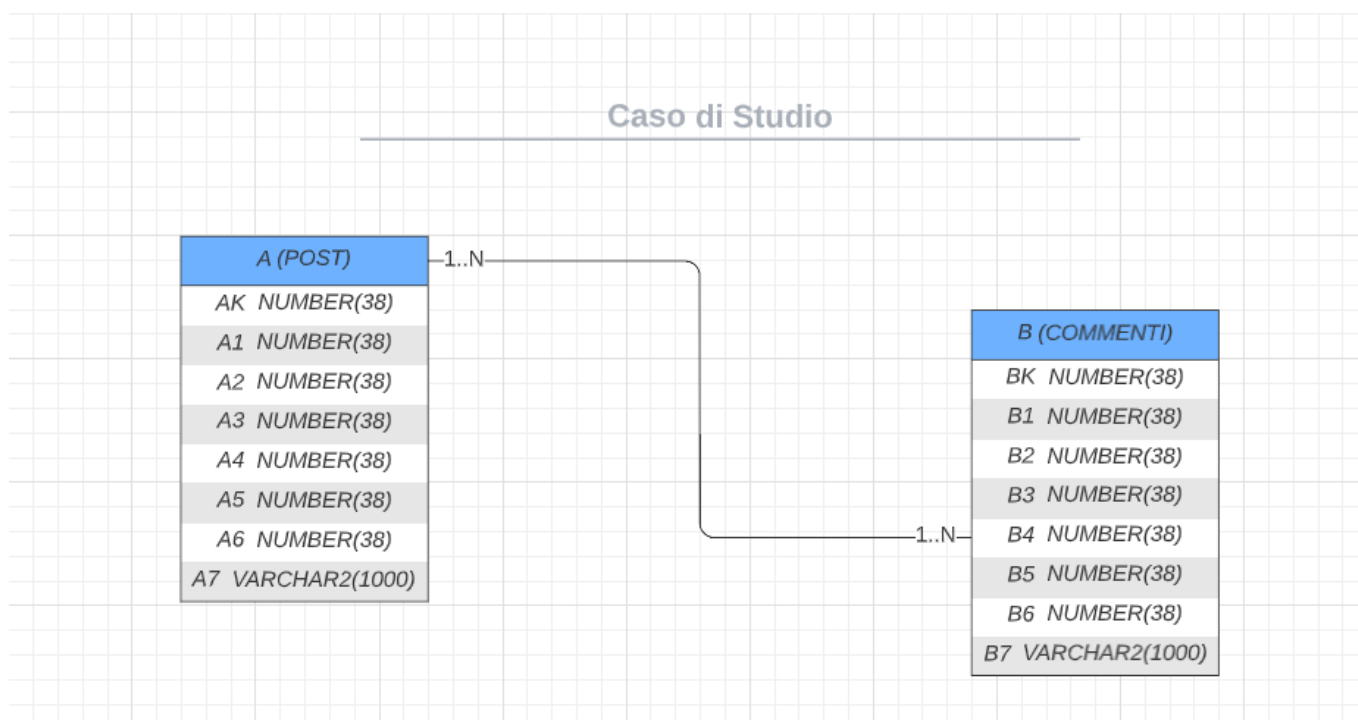
I sistemi database sulla quale ci siamo basati sono MongoDB, CouchDB per i sistemi non relazionali e Oracle per un database di tipo relazionale.

Creazione dataset

Il nostro caso di studio è quello di una relazione 1,N - 1,N tra due entità, chiamate comunemente A e B, ma che per comprensione abbiamo ipotizzato essere Post e Commenti all'interno del database di un social network, ogni Post contiene 10 Commenti.

La modellazione delle entita' e' stata fatta creando 2 triplette di attributi uguali, sulla seconda tripletta al contrario della prima sono stati costruiti degli indici su ogni attributo, questi attributi generici hanno una propria selettivita':

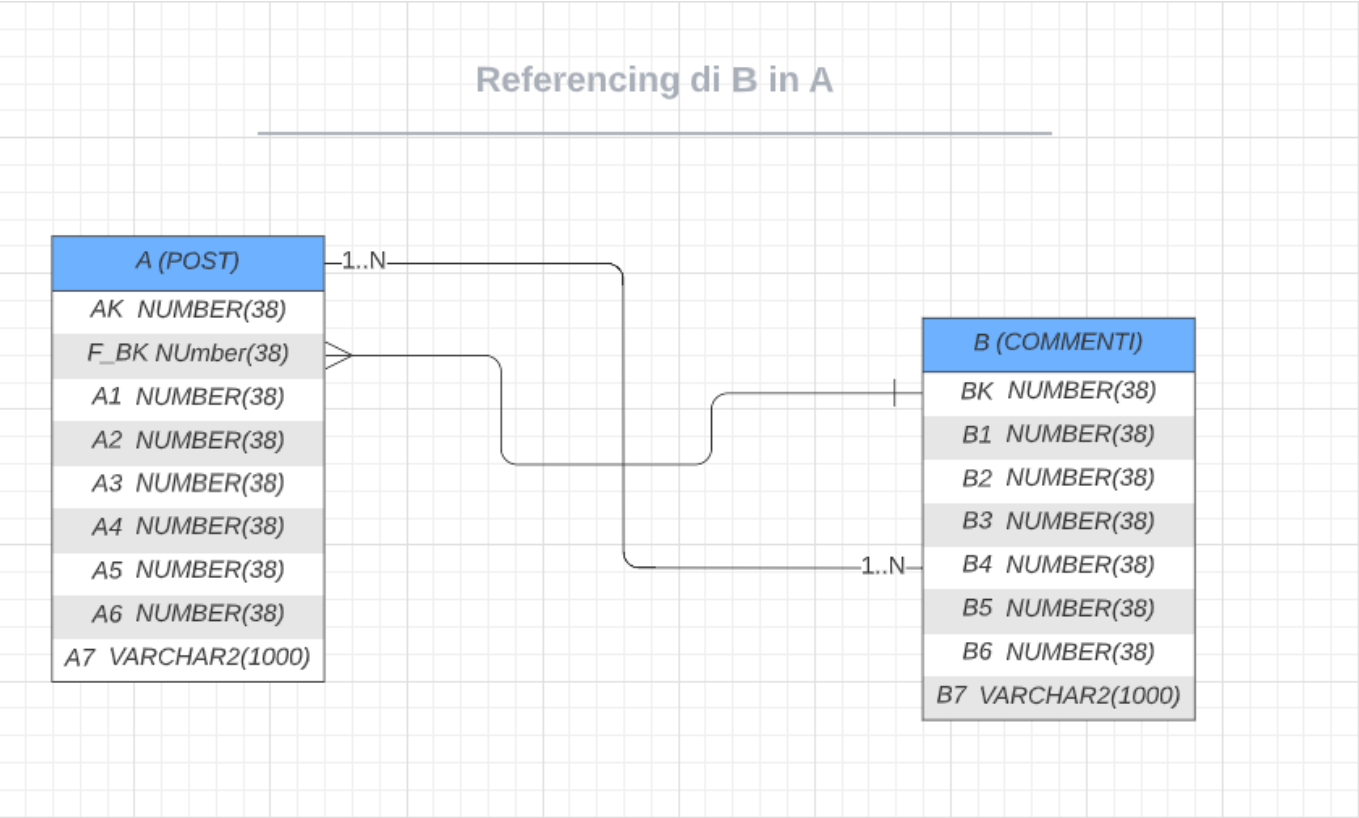
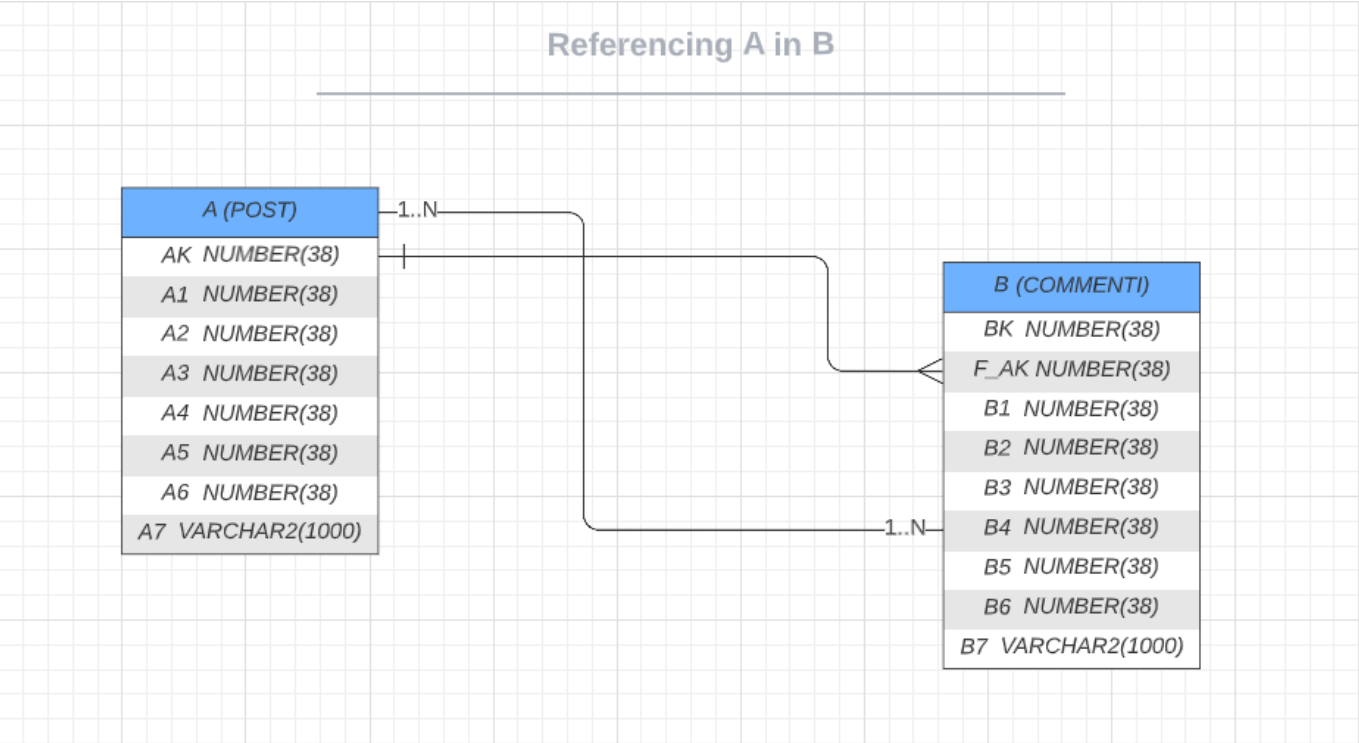
- $sel(x1, x4) = 1/10.000$
- $sel(x2, x5) = 1/1000$
- $sel(x3, x6) = 1/100$



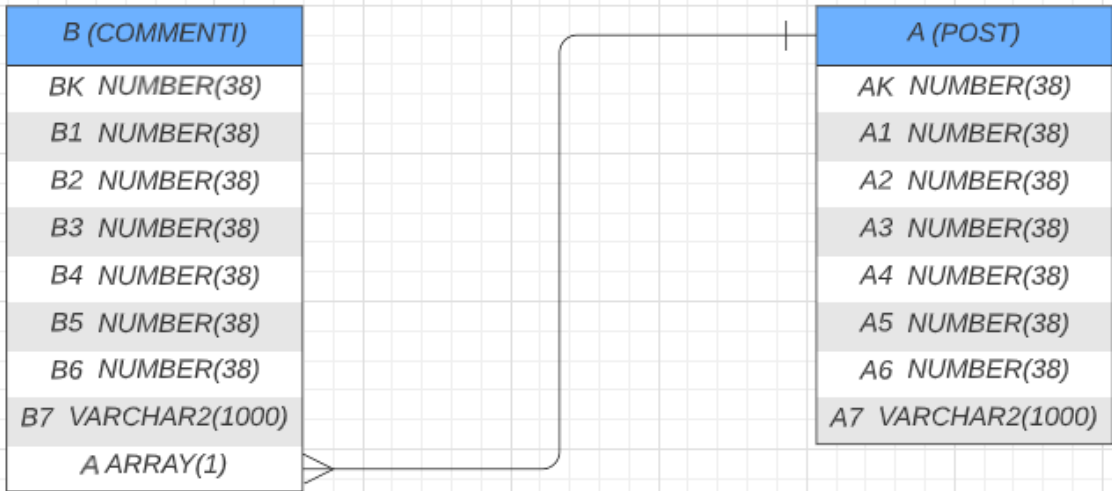
Modellazione

Per la modellazione dei dati abbiamo usato il referencing e l'embedding di documenti come segue:

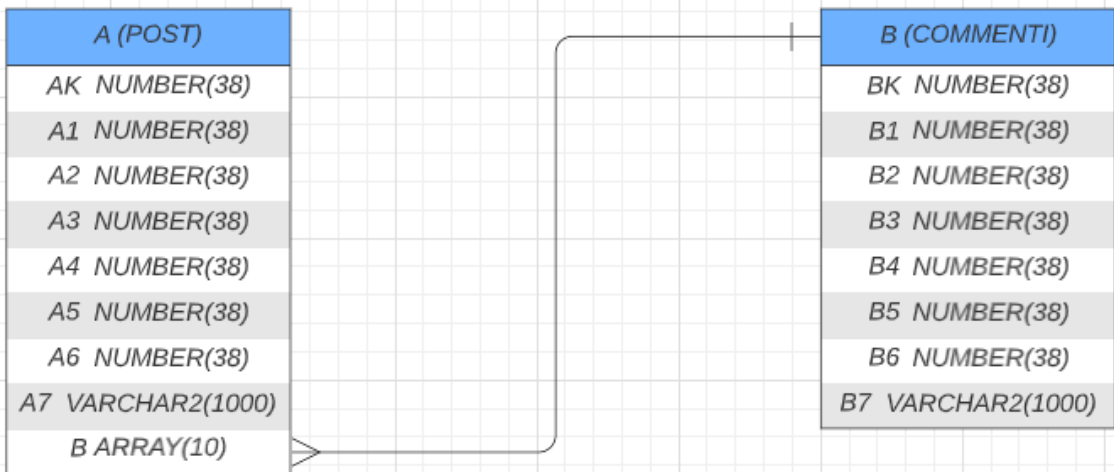
Come carico massimo abbiamo testato il database con |Post| = 100.000 e quindi |Commenti| = 1.000.000



Embedding di A in B



Embedding di B in A



Docker

Ho fatto ampio utilizzo di docker, per poter creare dei container all'interno dei quali fossi più libero di muovermi e per poter limitare l'uso di alcune risorse di sistema, in questo progetto vogliamo arrivare a

determinare una stima dei costi di DB non relazionali che facciano il minor uso di cache se non addirittura 0. Ho trovato alcune immagini su dockerhub che sono state un ottimo punto di partenza, alle quali ho poi aggiunto una serie di caratteristiche adatte al nostro caso. Ho creato dei file docker-compose.yml ed una serie di script per poter caricare comodamente i dati non appena il container viene creato

```
# sharing di directory
volumes:
  - mongodb-data:/data/db
  - ../../dataJSON:/home/data
# ma soprattutto limitazione della memoria RAM e CPU
mem_limit: 512m
cpu_count: 1
```

Script per dati

```
mongoimport --collection embedding_A_in_B --db tirocinio --username
$MONGO_INITDB_ROOT_USERNAME --password $MONGO_INITDB_ROOT_PASSWORD --
host=localhost:27017 --authenticationDatabase admin
/home/data/embedding_A_in_B.json

mongoimport --collection embedding_B_in_A --db tirocinio --username
$MONGO_INITDB_ROOT_USERNAME --password $MONGO_INITDB_ROOT_PASSWORD --
host=localhost:27017 --authenticationDatabase admin
/home/data/embedding_B_in_A.json

mongoimport --collection referencing_A_in_B --db tirocinio --username
$MONGO_INITDB_ROOT_USERNAME --password $MONGO_INITDB_ROOT_PASSWORD --
host=localhost:27017 --authenticationDatabase admin
/home/data/referencing_A_in_B.json

mongoimport --collection referencing_B_in_A --db tirocinio --username
$MONGO_INITDB_ROOT_USERNAME --password $MONGO_INITDB_ROOT_PASSWORD --
host=localhost:27017 --authenticationDatabase admin
/home/data/referencing_B_in_A.json
```

MongoDB

Collections				Collection Name	+ Create collection
View	Export	[JSON]	Import	A	Del
View	Export	[JSON]	Import	B	Del
View	Export	[JSON]	Import	embedding_A_in_B	Del
View	Export	[JSON]	Import	embedding_B_in_A	Del
View	Export	[JSON]	Import	referencing_A_in_B	Del
View	Export	[JSON]	Import	referencing_B_in_A	Del

Database Stats	
Collections (incl. system.namespaces)	6
Data Size	5.75 GB
Storage Size	614 MB
Avg Obj Size #	1.74 KB
Objects #	3300000
Indexes #	30
Index Size	142 MB

crezione collezioni

```
// referencing_A_in_B

db.B.aggregate([
  {
    $project: {
      "_id" : "$BK",
      "BK" : "$BK",
      "AK" : "$FAK",
      "B1" : "$B1",
      "B2" : "$B2",
      "B3" : "$B3",
      "B4" : "$B4",
      "B5" : "$B5",
      "B6" : "$B6",
      "B7" : "$B7"
    }
  }, {
    $out : "referencing_A_in_B"
  }
])
```

```
// referencing_B_in_A

db.B.aggregate(
[
{
  $group: {
    _id: {"AK" : "$FAK"}, "BK": {$addToSet : "$BK"}
  }
},
{
  $lookup: {
    from: 'A',
    localField: '_id.AK',
    foreignField: 'AK',
    as: 'AK'
  }
},
{
  $project : {"_id" : 0,"BK.FAK" : 0}
},
{
  $unwind: {
    path: "$AK",
  }
},
{
  $project : {
    "_id" : "$AK.AK",
    "AK" : "$AK.AK",
    "A1" : "$AK.A1",
    "A2" : "$AK.A2",
    "A3" : "$AK.A3",
    "A4" : "$AK.A4",
    "A5" : "$AK.A5",
    "A6" : "$AK.A6",
    "A7" : "$AK.A7",
    "B" : "$BK"} // li voglio in quest ordine
  },
{
  $out : "referencing_B_in_A"
}
],{allowDiskUse:true}
)
```

```
// embedding_B_in_A

db.B.aggregate(
[
{
  $group: {
    _id: {"AK" : "$FAK"}, "BK": {$addToSet : "$$ROOT"}
  }
},
{
  $lookup: {
    from: 'A',
```

```

        localField: '_id.AK',
        foreignField: 'AK',
        as: 'AK'
    }
}, {
    $project : { "_id" : 0, "BK.FAK" : 0 }
}, {
    $unwind: {
        path: "$AK",
    }, {
        $project : {
            "_id" : "$AK.AK",
            "AK" : "$AK.AK",
            "A1" : "$AK.A1",
            "A2" : "$AK.A2",
            "A3" : "$AK.A3",
            "A4" : "$AK.A4",
            "A5" : "$AK.A5",
            "A6" : "$AK.A6",
            "A7" : "$AK.A7",
            "B" : "$BK" } // li voglio in quest ordine
        }, {
            $project : { "A._id" : 0, "B._id" : 0 }
        }, {
            $out : "embedding_B_in_A"
        }
    ], {allowDiskUse:true}
)

```

```

// embedding_A_in_B
db.B.aggregate(
[
    {
        $lookup: {
            from: 'A',
            localField: 'FAK',
            foreignField: 'AK',
            as: 'A'
        }
    }, {
        $project: {
            "FAK" : 0,
            "_id" : 0,
            "A._id" : 0
        }
    }
], {
    $project : {
        "_id" : "$BK",
        "B1" : 1,
    }
}
)

```

```

        "B2" : 1,
        "B3" : 1,
        "B4" : 1,
        "B5" : 1,
        "B6" : 1,
        "B7" : 1,
        "A" : 1
    }
}, {
    $out : "embedding_A_in_B"
}
]
)

```

collection : "referencing_B_in_A"

```

# select A.*
# from A
# where Ax='val'

db.getCollection('referencing_B_in_A').aggregate(
    [
        {
            '$match' : { ind : val}
        }
    ]
)

# select A.*, B.*
# from A join B on (A.AK=B.AK)
# where Ax='val'

db.getCollection('referencing_B_in_A').aggregate([{'$match' : { ind : val} },
{'$unwind' : { 'path': "$B"}}, {
    '$lookup': {
        'from': 'B',
        'localField': 'B',
        'foreignField': 'BK',
        'as': 'B'
    }}, {'$unwind' : {'path' : "$B"}}, {'$project' : {"B._id" : 0}}]])

# select B.*
# from B
# where Bx='val'
db.getCollection('B').aggregate(
    [
        {
            '$match' : { ind : val}
        }
    ]
)

```



```

# select A.*, B.*
# from A join B on (A.AK=B.BK)
# where Bx='val'
db.getCollection('B').aggregate([
  {'$match': {
    ind : val
  }}, {
  {'$lookup': {
    'from' : 'referencing_B_in_A',
    'localField' : 'FAK',
    'foreignField' : 'AK',
    'as' : 'A'
  }}, {
  {'$project': {
    "_id" : 0,
    "FAK" : 0,
    "A.B" : 0
  }}, {'$unwind' : {'path' : "$A"}}])

```

collection : "referencing_A_in_B"

```

# select A.*
# from A
# where Ax='val'
db.getCollection('A').aggregate(
  [
    {
      '$match' : { ind : val}
    }
  ]
)
# select A.*, B.*
# from A join B on (A.AK=B.AK)
# where Ax='val'
db.getCollection('referencing_B_in_A').aggregate([{'$match' : { ind : val
} }, {'$unwind': { 'path': "$B"}}, {'$lookup': {'from':
'referencing_A_in_B', 'localField': 'B', 'foreignField': '_id', 'as': 'B'}}])
# select B.*
# from B
# where Bx='val'
db.getCollection('referencing_A_in_B').aggregate(
  [
    {
      '$match' : { ind : val}
    }
  ]
)

# select A.*, B.*

```

```

# from A join B on (A.AK=B.BK)
# where Bx='val'
db.getCollection('referencing_A_in_B').aggregate([{'$match' : {ind :
val}},{'
    '$lookup': {
        'from': 'A',
        'localField': 'AK',
        'foreignField': 'AK',
        'as': 'A'
    }},{
    '$project': {
        "A._id" : 0
    }},{ '$unwind' : {'path': "$A"}}])

```

collection : "embedding_B_in_A"

```

# select A.*
# from A
# where Ax='val'
db.getCollection('embedding_B_in_A').aggregate(
    [
        {
            '$match' : { ind : val}
        },{' $project' : { "B" : 0}}
    ]
)

# select A.*, B.*
# from A join B on (A.AK=B.AK)
# where Ax='val'
query =(
    [
        {
            '$match' : { ind : val}
        }
    ]
)

# select B.*
# from B
# where Bx='val'
db.getCollection('embedding_B_in_A').aggregate([{'$match': {"B." + ind :
val}},{'$unwind': {'path' : "$B"}},{ '$match' : {"B." + ind : val}}, {'$project':
{"B" : 1, "_id" : 0}}])

# select A.*, B.*
# from A join B on (A.AK=B.BK)
# where Bx='val'
db.getCollection('embedding_B_in_A').aggregate([{'$match': {"B." + ind :
val}},{'$unwind': {'path' : "$B"}},{ '$match' : {"B." + ind : val}}])

```

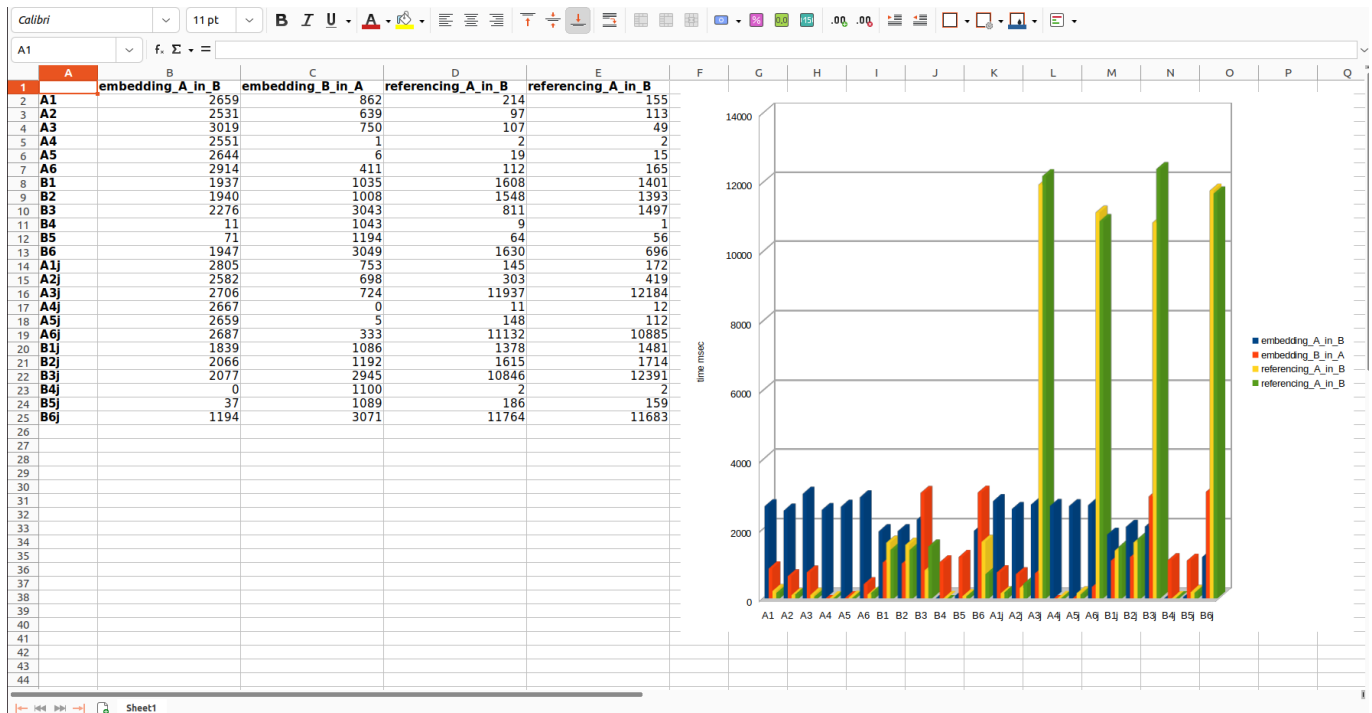
collection : "embedding_A_in_B"

```
# select A.*
# from A
# where Ax='val'
db.getCollection('embedding_A_in_B').aggregate([{'$match': {"A." + ind :
val}},{'$project': {"A" : 1, "_id" : 0}},{'$unwind' : {'path' : "$A"}}]])

# select A.*, B.*
# from A join B on (A.AK=B.AK)
# where Ax='val'
db.getCollection('embedding_A_in_B').aggregate([{'$match': {"A." + ind :
val}}]])

# select B.*
# from B
# where Bx='val'
query =(
    [
        {
            '$match' : { ind : val}
        },{'$project' : { "A" : 0}}
    ]
)

# select A.*, B.*
# from A join B on (A.AK=B.BK)
# where Bx='val'
db.getCollection('embedding_A_in_B').aggregate(
    [
        {
            '$match' : { ind : val}
        }
    ]
)
```



CouchDb

Per poter usare couchDB ho dovuto studiare il sistema API di couchDB, con il quale ho potuto effettuare le più semplici operazioni tramite programmi in python che effettuava PUT e POST con documenti all'interno del body. CouchDB utilizza un dialetto particolare chiamato MangoQuery, è molto simile a quello di Mongo ma la vera differenza sta nel fatto che non ha l'istruzione di lookup ossia di join.

Sono partito dai JSON generati e tramite sed li ho modificati per fare in modo che couchDB potesse accettarli creando indici come stringhe

Attraverso questa funzione ho eseguito ogni query di selezione.

```
def simpleSelection(collection, ind, value):
    url = "http://admin:admin@127.0.0.1:5984/" + collection + "/_find"

    time = 0

    payload = json.dumps({
        "selector": {
            ind : int(val)
        },
        "execution_stats": True
    })

    headers = {
        'Content-Type': 'application/json'
    }

    response = requests.request("POST", url, headers=headers, data=payload)

    resp = json.loads(response.text)
```

```

time += resp["execution_stats"]['execution_time_ms']
worksheet.write(get_rowNum[collection] + 1, get_colNum[ind] + 1, time)
print("sel time: " )
print(str(time) + " ms")

```

Per poter eseguire l'operazione di join ho dovuto provvedere manualmente, effettuando una selezione sulla tabella interna dei documenti o meglio chiavi dei documenti coinvolti e poi per ognuno effettuare una query di selezione nella tabella esterna

```

def joinQuery(to_col, from_col, ind, value):
    url = "http://admin:admin@127.0.0.1:5984/" + from_col + "/_find"

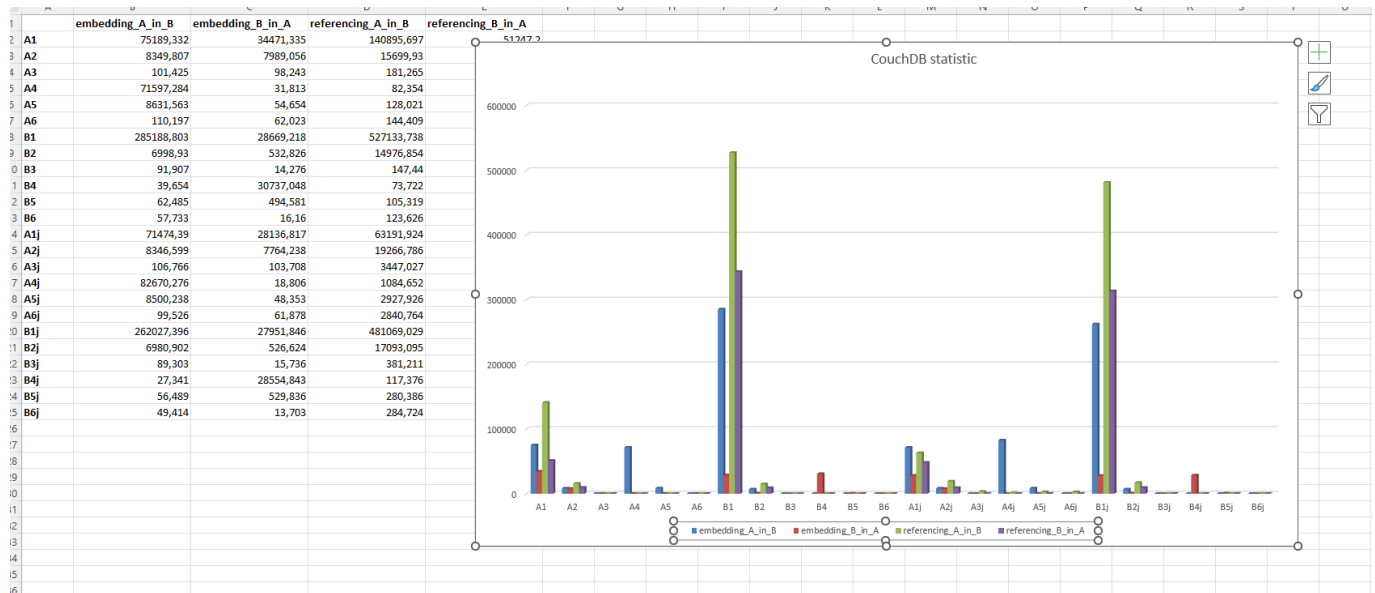
    payload = json.dumps({
        "selector": {
            ind : int(val)
        },
        "fields": [
            "AK"
        ],
        "execution_stats": True
    })

    headers = {
        'Content-Type': 'application/json'
    }

    time = 0
    response = requests.request("POST", url, headers=headers, data=payload)

    resp = json.loads(response.text)
    time += resp["execution_stats"]['execution_time_ms']
    foreign_keys = [list(x.values())[0] for x in resp["docs"]]
    url = "http://admin:admin@127.0.0.1:5984/" + to_col + "/_find"
    for key in foreign_keys:
        payload = json.dumps({
            "selector": {
                "AK" : key
            },
            "execution_stats": True
        })
        response = requests.request("POST", url, headers=headers, data=payload)
        resp = json.loads(response.text)
        time += resp["execution_stats"]['execution_time_ms']
        worksheet.write(get_rowNum[to_col] + 1, get_colNum[ind] + 1, time)
    print("join time: " )
    print(str(time) + " ms")

```



Oracle

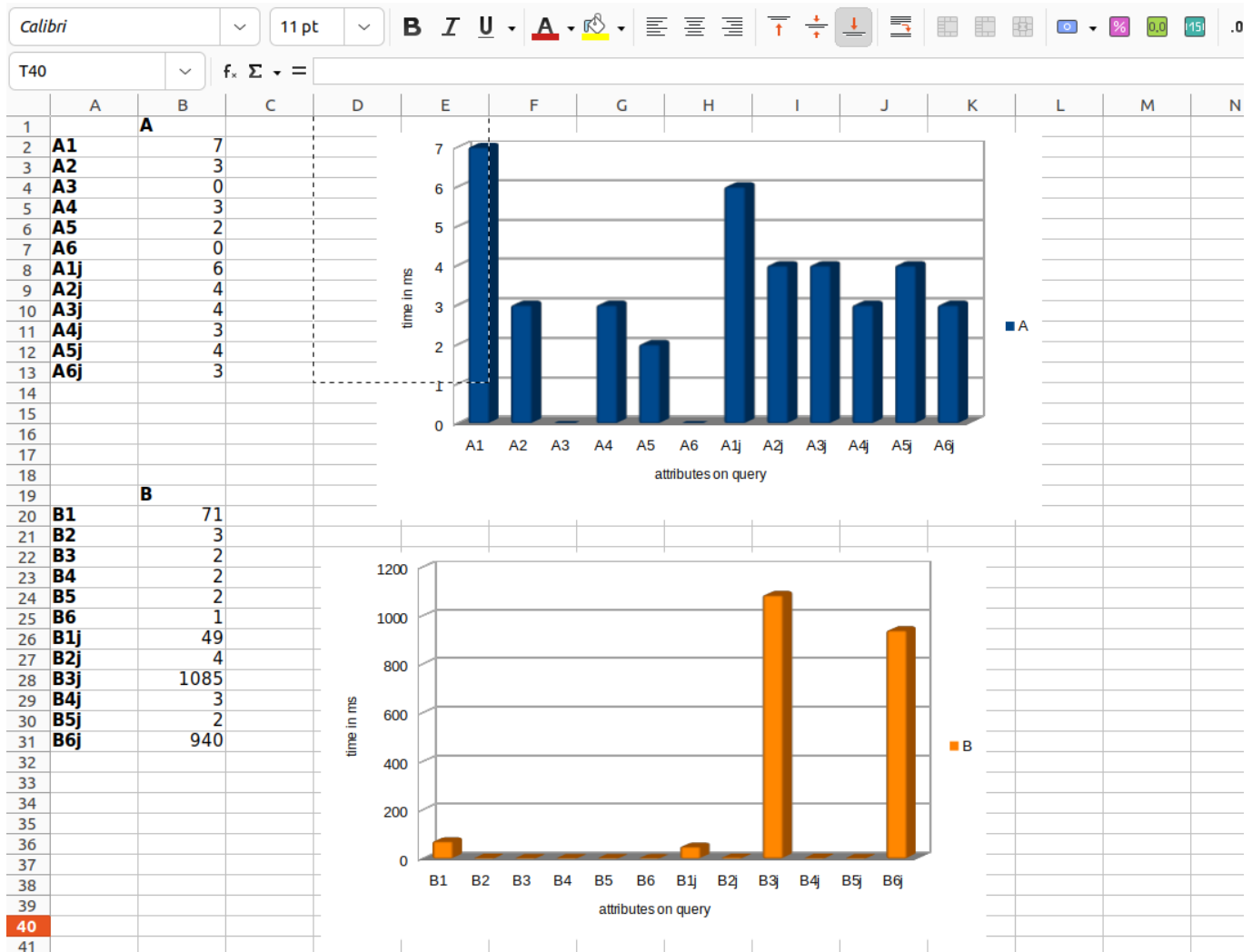
Per ogni indice presente nella collezione ho eseguito queste query dimostrative

```
select A.*
from A
where Ax='val'
```

```
select B.*
from B
where Bx='val'
```

```
select A.*, B.*
from A join B on (A.AK=B.AK)
where Ax='val'
```

```
select A.*, B.*
from A join B on (A.AK=B.BK)
where Bx='val'
```



Risultati

Fin ora abbiamo ottenuto risultati ottimi ma non ancora del tutto completi, questo progetto verrà ulteriormente approfondito nella mia tesi triennale. Al momento sembrerebbe che la modellazione ad embedding sia quella più efficace e completa, ci aspettiamo di poter migliorare ulteriormente le performance creando un modello ibrido tra embedding e referencing che però sia modellato su un caso reale con adeguato carico di lavoro, ad es. nell'esempio social network si potrebbero inserire nel referencing dei post anche i 10 commenti più recenti. Inoltre poi si devono valutare le altre operazioni del CRUD (delete, update) ed arrivare a stimare un effettiva stima dei costi.

Commenti

Questo tirocinio e' stato molto interessante, l'ho scelto perche' ho trovato altrettanto interessante il corso di basi di dati, tutto cio' che ho fatto fin ora verra' approfondito nella mia tesi triennale. Ho potuto lavorare in autonomia ma ad ogni problema o dubbio la prof.ssa Lumini e' sempre stata disponibile nel risolverlo insieme e nel correggermi dove potevo fare qualcosa in un modo migliore.