Spatial Database Comparing SQL Server and MongoDB

(TECHNICAL REPORT)

Hsien-Yi Liu
Department of Computer Science
Stony Brook University
NY, United States
hsliu@cs.stonybrook.edu

Advisor - Dejun Teng
Professor - Fusheng Wang
Stony Brook University
NY, United States
dteng@cs.stonybrook.ed

***Abstract*— Comparing the performance between different databases.**

Keywords*— *SQL, SQL Server, MongoDB, Spatial Databases, Oracle Database

I. INTRODUCTION

This semester, our plan is to compare different spatial databases. In our research, we will demonstrate the difference between all the databases and the result of the same sql within different databases.

II. INSTALLATION

A. *MongoDB*

Prerequisites of installing MongoDB, you will have to use Homebrew to install. After checking your installation of Xcode and Homebrew, we could start to install MongoDB.

Our first step would be **`brew tap mongodb/brew.`**

```
==> Tapping mongodb/brew
Cloning into '/usr/local/Homebrew/Library/Taps/mongodb/homebrew-brew'...
remote: Enumerating objects: 97, done.
remote: Counting objects: 100% (97/97), done.
remote: Compressing objects: 100% (80/80), done.
remote: Total 462 (delta 38), reused 42 (delta 17), pack-reused 365
Receiving objects: 100% (462/462), 101.47 KiB | 2.47 MiB/s, done.
Resolving deltas: 100% (207/207), done.
Tapped 11 formulae (38 files, 171KB).
```

And then we could use the following command **`brew install mongodb-community@4.4`** to install the latest MongoDB.

```
==> Summary
🍺  /usr/local/Cellar/mongodb-community/4.4.1: 11 files, 136.8MB, built in 4 seconds
==> `brew cleanup` has not been run in 30 days, running now...
Removing: /Users/shane/Library/Caches/Homebrew/unixodbc--2.3.7.catalina.bottle.1.tar.gz... (564.9KB
Removing: /Users/shane/Library/Logs/Homebrew/msodbcsql... (3 files, 536B)
Removing: /Users/shane/Library/Logs/Homebrew/libtool... (64B)
Removing: /Users/shane/Library/Logs/Homebrew/unixodbc... (64B)
Removing: /Users/shane/Library/Logs/Homebrew/mssql-tools... (131B)
Removing: /Users/shane/Library/Logs/Homebrew/msodbcsql17... (3 files, 538B)
Pruned 2 symbolic links from /usr/local
==> Caveats
==> mongodb-community
To have launchd start mongodb/brew/mongodb-community now and restart at login:
  brew services start mongodb/brew/mongodb-community
Or, if you don't want/need a background service you can just run:
  mongod --config /usr/local/etc/mongod.conf
```

After finishing the installation, we will use the following steps to run MongoDB Community Edition. These instructions assume that you are using the default settings. You can run MongoDB as a macOS service using brew, or you can run MongoDB manually as a background process. It is recommended to run MongoDB as a macOS service, as doing so sets the correct system ulimit values automatically. To run MongoDB as a macOS service, issue the following: **`brew services start mongodb-community@4.4.`** To stop a mongodb running as a macOS service, use the following command as needed: **`brew services stop mongodb-community@4.4.`** To run MongoDB manually as a background process, issue the following: **`mongod --config etc/mongod.conf --fork`** To stop a mongod running as a background process, connect to the mongod from the mongo shell, and issue the shutdown command as needed.

B. SQL Server

It's easy to use SQL Server on Mac. First, pull the docker image file and run with the command. However, for importing data and other files. It will become very time consuming. Therefore, my suggestion would be to use windows to create your experiment environment.

C. *Oracle Spatial*

Comparing the difficulty of installing SQL server on mac. Oracle Database is friendly for the mac user. It provides the Oracle SQL Developer which could be easily connected to your docker container Oracle Database and you could create SQL and run your test.

III. INDEX

A. MongoDB

Index Type	Description
Single Field	MongoDB supports the creation of user-defined ascending/descending indexes on a single field of a document.
Compound Index	MongoDB also supports user-defined indexes on multiple fields
Multikey Index	MongoDB uses multikey indexes to index the content stored in arrays.
Geospatial Index	To support efficient queries of geospatial coordinate data, MongoDB provides two special indexes: 2d indexes and 2dsphere indexes
Text Indexes	MongoDB provides a text index type that supports searching for string content in a collection.
Hash Indexes	The unique property for an index causes MongoDB to reject duplicate values for the indexed field.

Index Properties	Description
Unique Indexes	The unique property for an index causes MongoDB to reject duplicate values for the indexed field.
Partial Indexes	Partial indexes only index the documents in a collection that meet a specified filter expression.
Sparse Indexes	The sparse property of an index ensures that the index only contains entries for documents that have the indexed field.
TTL Indexes	TTL indexes are special indexes that MongoDB can use to automatically remove documents from a collection after a certain amount of time.
Hidden Indexes	Hidden indexes are not visible to the query planner
	and cannot be used to support a query.

For the example, we will using following command to insert several position in our MongoDB

```
> db.imooc_collection.insert({name:"圆明园",pos:[90,70]})
WriteResult({ "nInserted" : 1 })
> db.imooc_collection.insert({name:"故宫",pos:[100,80]})
WriteResult({ "nInserted" : 1 })
> db.imooc_collection.insert({name:"SBU",pos:[90,30]})
WriteResult({ "nInserted" : 1 })
```

Check the result using the find method.

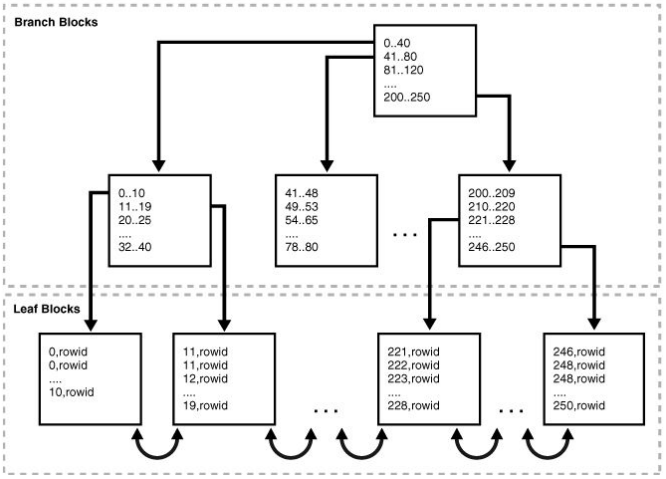
```
> db.imooc_collection.find()
{ "_id" : ObjectId("5f962e460bd9ac8e69fc2d78"), "name" : "圆明园", "pos" : [ 90, 70 ] }
{ "_id" : ObjectId("5f962e500bd9ac8e69fc2d79"), "name" : "故宫", "pos" : [ 100, 80 ] }
{ "_id" : ObjectId("5f962e610bd9ac8e69fc2d7a"), "name" : "SBU", "pos" : [ 90, 30 ] }
```

Quick tutorial: (1) Load the json file: `mongoimport <restaurants.json> -c=restaurants`
Because this data is geographical, (2) create a 2dsphere index on each collection using the mongo shell: `db.restaurants.createIndex({ location: "2dsphere" })` (3) Exploring the Data- Inspect an entry in the newly-created restaurants collection from within the mongo shell: `db.restaurants.findOne()`

```
> db.restaurants.findOne()
{
        "_id" : ObjectId("55cba2476c522cafdb053add"),
        "location" : {
                "coordinates" : [
                        -73.856077,
                        40.848447
                ],
                "type" : "Point"
        },
        "name" : "Morris Park Bake Shop"
}
```

B. Oracle Database

Index Type	Description
B-tree indexes	These indexes are the standard index type. They are excellent for primary key and highly-selective indexes. Used as concatenated indexes, B-tree indexes can retrieve data sorted by the indexed columns.
Bitmap Indexes	In a bitmap index, the database stores a bitmap for each index key. In a conventional B-tree index, one index entry points to a single row. In a bitmap index, each index key stores pointers to multiple rows.
Function-based indexes	This type of index includes columns that are either transformed by a function, such as the UPPER function, or included in an expression. B-tree or bitmap indexes can be function-based.
Application domain indexes	This type of index is created by a user for data in an application-specific domain. The physical index need not use a traditional index structure and can be stored either in the Oracle database as tables or externally as a file.



Internal Structure of a B-tree Index

C. SQL Server

Index Type	Description
Hash indexes	With a hash index, data is accessed through an in-memory hash table. Hash indexes consume a fixed amount of memory, which is a function of the bucket count.
memory-optimized Nonclustered Indexes	For memory-optimized nonclustered indexes, memory consumption is a function of the row count and the size of the index key columns
Clustered indexes	A clustered index sorts and stores the data rows of the table or view in order based on the clustered index key. The clustered index is implemented as a B-tree index structure that supports fast retrieval of the rows, based on their clustered index key values.
Nonclustered Indexes	A nonclustered index can be defined on a table or view with a clustered index or on a heap
Unique Indexes	A unique index ensures that the index key contains no duplicate values and therefore every row in the table or view is in some way unique.
Columnstore Indexes	An in-memory columnstore index stores and manages data by using column-based data storage and column-based query processing.
Filtered Indexes	An optimized nonclustered index, especially suited to cover queries that select from a well-defined subset of data. It uses a filter predicate to index a portion of rows in the table.
Spatial Indexes	A spatial index provides the ability to perform certain operations more efficiently on spatial objects (spatial data) in a column of the geometry data type. The spatial index reduces the number of objects on which relatively costly spatial operations need to be applied.

IV. DATA

During out research, we used several datasets.

Earthquake

Yellow Stone Park Boundry

Restaurant example

Neighborhood example

Open Address

Dataset	File Name	points	polygon	type
Sample - 1	1. Restaurant 2. Neighbor Json File	25360	195	Json
Sample - 2	1. Earthquake 2. Yellowstone Park Boundry Geojson	5299	1	Geojson
Sample - 3	1. Open Address 2. Connecticut boundry Json	1015947	1	Json

V. SQL Server

1. Import Data

1.1 SQL Server

```
DECLARE @GeoJSON nvarchar(max) = N'{...}' -- Put the GeoJSON here

INSERT INTO dbo.EarthquakeData (EventDate,Magnitude,Place,Coordinates,Long,Lat)
SELECT
    -- Convert unix epoch time to datetime
    -- We also know the source is in UTC, so we specify that using AT TIME ZONE
    DATEADD(second,cast(UnixMillisecondsSinceEpoch/1000 as int)
    ,'1970-01-01 00:00:00') AT TIME ZONE 'UTC' AS EventDate,
    Magnitude,
    Place,
    -- Build our geography datatype
    geography::STPointFromText('POINT ('+Long + ' ' + Lat + ')', 4326) AS Geogr
    Long,
    Lat
FROM
    OPENJSON(@GeoJSON, '$.features')
        WITH (
            UnixMillisecondsSinceEpoch bigint '$.properties.time',
            Magnitude float '$.properties.mag',
            Place varchar(300) '$.properties.place',
            Long varchar(100) '$.geometry.coordinates[0]',
            Lat varchar(100) '$.geometry.coordinates[1]'
        )
```

1.2 SQL Server - Load with file

```
DECLARE @JSON VARCHAR(MAX)

SELECT @JSON = BulkColumn
FROM OPENROWSET
(BULK 'C:\file-location\my-data.json', SINGLE_CLOB)
AS j
```

1.3 SQL - 1 find the restaurant in the neighborhood

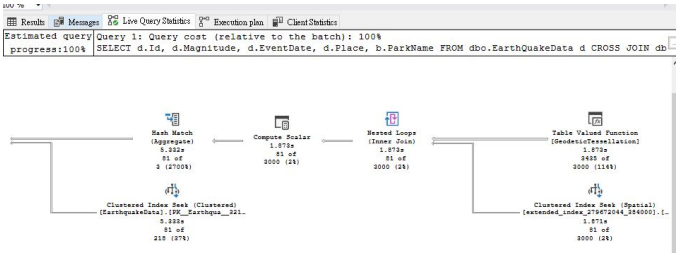
```
DECLARE @g1 geometry;
SET @g1 = geometry::STPolyFromText(
'POLYGON((
 -73.94732672160579 40.62916656720943 , -73.94687439946738 40.626773668128486, -73.94
 -73.9460669961243 40.61710373918014 , -73.94576359491893 40.61550406422122, -73.9457
 -73.95026514882962 40.613924351797365 , -73.94999598103774 40.612491461663225 ,-73.9
 -73.95258786948972 40.61111223958901 , -73.95359888852907 40.610949865606365 ,-73.95
 -73.95952069567825 40.610302052836765 , -73.96044361881539 40.61020150680253 ,-73.96
 -73.96381145170662 40.61758940154457 , -73.96388935578909 40.61800054988432 , -73.96
 -73.96757137736036 40.62206027486021 , -73.96864510622689 40.621941695814826 , -73.9
 -73.97086150876679 40.626237183335725 , -73.97092126268777 40.626569985505505 , -73.
 -73.96707656865328 40.62943468164793 , -73.96608191359866 40.62954270682626 , -73.96
 -73.96136921593188 40.630058388262405 , -73.96046280639705 40.630155661219426 , -73.
 -73.95488499156933 40.6283355483374 , -73.95395983418409 40.62843938961405 , -73.953
))', 4326 );



select count(*) as Countain_Points
from res
where points.STWithin(@g1) = 1
```

Data: Polygon 195/ Points 25360
Time Cose: 2 sec

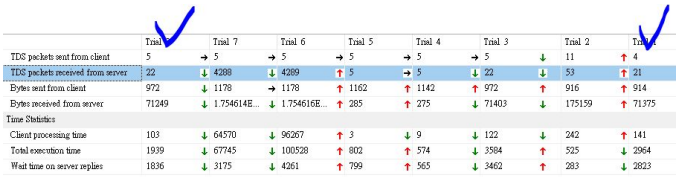
2. SQL - 2 find the earthquake in the boundary

```
SELECT
    d.Id,
    d.Magnitude,
    d.EventDate,
    d.Place,
    b.ParkName
FROM
    dbo.EarthQuakeData d
    CROSS JOIN dbo.ParkBoundaries b
WHERE
    Coordinates.STIntersects(ParkBoundary) =1
    AND b.ParkName = 'Yellowstone National Park'
ORDER BY
    Magnitude DESC
```

*2.1 Adding the **index** to see the execution plan.*



After setting the max worker size and using admin connect the database.



	Trial 8	Trial 7	Trial 6	Trial 5	Trial 4	Trial 3	Trial 2	Trial 1
TDS packets sent from client	5	5	5	5	5	5	11	4
TDS packets received from server	22	4288	4289	5	5	22	53	21
Bytes sent from client	972	1178	1178	1162	1142	972	916	914
Bytes received from server	71249	1.754614E...	1.754616E...	285	275	71403	175159	71375
Time Statistics								
Client processing time	103	64570	96267	3	9	122	242	141
Total execution time	1939	67745	100528	802	574	3584	525	2964
Wait time on server replies	1836	3175	4261	799	565	3462	283	2823

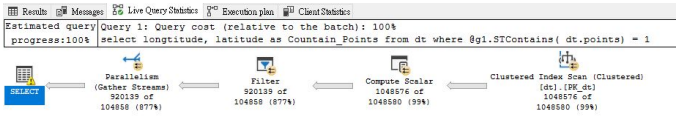
Data: Polygon 1/ Points 5299
Time cose: 1.934 sec

3. SQL - 3 All address within Connecticut

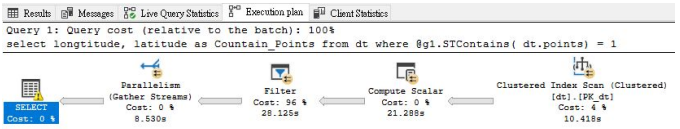
```
DECLARE @g1 geometry;
SET @g1 = geometry::STPolyFromText(
'POLYGON((-73.564453125 41.178653972331674,
-71.69128417968749 41.178653972331674,
-71.69128417968749 42.114523952464246,
-73.564453125 42.114523952464246,
-73.564453125 41.178653972331674
))', 4326 );

select longtitude, latitude as Countain_Points
from dt
where @g1.STContains( dt.points) = 1
```

Data: Polygon 1/ Points 1015947
Time Cose: 18.723 sec

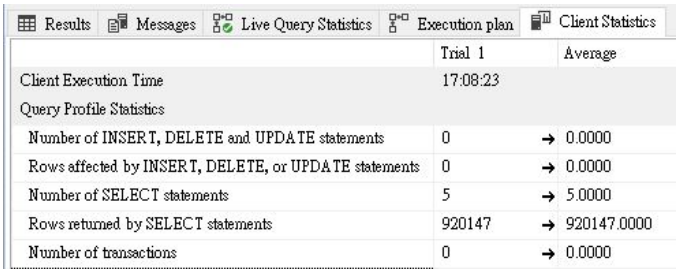
4. Execution Plan - Default
4.1 Live Query Static



4.2 Execution Plan



4.3 Clients Statics



Results | Messages | Live Query Statistics | Execution plan | Client Statistics

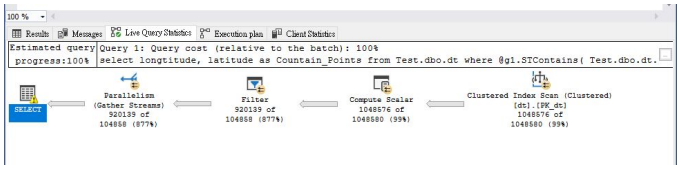
	Trial 1		Average
Client Execution Time	17:08:23		
Query Profile Statistics			
Number of INSERT, DELETE and UPDATE statements	0	→	0.0000
Rows affected by INSERT, DELETE, or UPDATE statements	0	→	0.0000
Number of SELECT statements	5	→	5.0000
Rows returned by SELECT statements	920147	→	920147.0000
Number of transactions	0	→	0.0000

5. Execute Plan - Multiple Threads

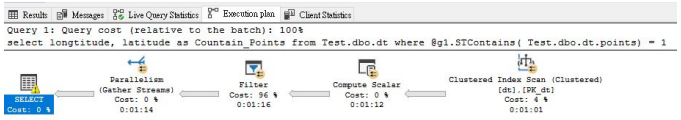
5.1 Update the setting

```
EXEC sp_configure 'show advanced options', 1;
GO
RECONFIGURE ;
GO
EXEC sp_configure 'max worker threads', 900 ;
GO
RECONFIGURE;
GO
```

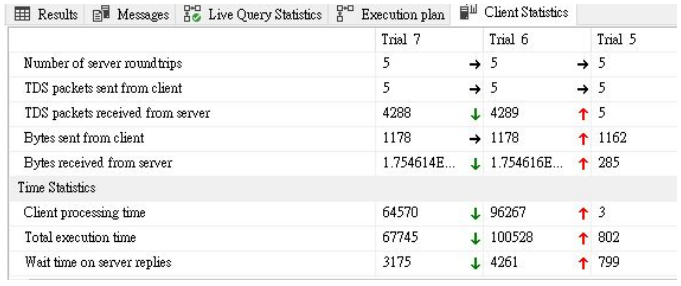
5.2 Live Query Static



5.3 Execution Plan



5.4 Clients Statics



	Trial 7	Trial 6	Trial 5
Number of server roundtrips	5 →	5 →	5
TDS packets sent from client	5 →	5 →	5
TDS packets received from server	4288 ↓	4289 ↑	5
Bytes sent from client	1178 →	1178 ↑	1162
Bytes received from server	1.754614E... ↓	1.754616E... ↑	285
Time Statistics			
Client processing time	64570 ↓	96267 ↑	3
Total execution time	67745 ↓	100528 ↑	802
Wait time on server replies	3175 ↓	4261 ↑	799

VI. MongoDB

1. Import Data

```
jq --compact-output ".features" input.geojson > output.geojson

mongoimport --db dbname -c collectionname --file "output.geojson" --jsonArray
```

1. 1 Result

```
mongodb mongoimport --collection address --file ct.json
2020-12-14T18:58:48.865+0800    connected to: mongodb://localhost/
2020-12-14T18:58:51.867+0800    [####....................] test.address 35.6MB/206MB (17.3%)
2020-12-14T18:58:54.869+0800    [########................] test.address 72.2MB/206MB (35.1%)
2020-12-14T18:58:57.868+0800    [############............] test.address 109MB/206MB (52.9%)
2020-12-14T18:59:00.869+0800    [################........] test.address 146MB/206MB (70.8%)
2020-12-14T18:59:03.866+0800    [####################....] test.address 181MB/206MB (87.9%)
2020-12-14T18:59:06.042+0800    [########################] test.address 206MB/206MB (100.0%)
2020-12-14T18:59:06.042+0800    1015947 document(s) imported successfully. 0 document(s) failed to import.
```

*1.2 Find one - (Select * like SQL)*


```
> db.address.findOne()
{
        "_id" : ObjectId("5fd74568b678f5c7ca972b79"),
        "state" : "CT",
        "postcode" : "06037",
        "street" : "Spruce Brook Rd",
        "district" : "",
        "unit" : "",
        "location" : {
                "type" : "Point",
                "coordinates" : [
                        -72.7482723,
                        41.5993596
                ]
        },
        "region" : "Hartford",
        "number" : "152",
        "city" : "Berlin"
}
```


1.3 SQL - 1 find the restaurant in the neighborhood

```
var neighborhood =
db.neighborhoods.findOne( { geometry:
{ $geoIntersects: { $geometry:
{ type: "Point", coordinates: [ -73.93414657, 40.82302903 ]}}}})

db.restaurants.find
( { location: { $geoWithin: { $geometry: neighborhood.geometry } } } ).count()
```

Data: Polygon 195/ Points 25360
Time Cose: 0 sec

2. SQL - 2 Find the earthquake in the boundary

```
var bound = db.boundry.findOne({
geometry: { $geoIntersects: { $geometry: { type: "Point",
coordinates: [ -111.0970801722779,44.487322588834374 ] } } } })

db.earthquake.find( { geometry: { $geoWithin: { $geometry: bound.geometry } }
```

Data: Polygon 1/ Points 5299
Time cose: 1.934 sec

3. SQL - 3 All address within Connecticut

```
var connecticut = db.address.find ({location:
     {$geoWithin:
       {$geometry: {
       type: "Polygon",
       coordinates:
  [[ [ -73.564453125, 41.178653972331674 ],
    [ -71.69128417968749, 41.178653972331674 ],
    [ -71.69128417968749, 42.114523952464246 ],
    [ -73.564453125, 42.114523952464246 ],
    [ -73.564453125, 41.178653972331674 ]
    ]]
}}}});

connecticut.explain("executionStats")
```

Data: Polygon 1/ Points 1015947
Time Cose: 0.5 sec

VII. Time Comparison

DataBase	SQL - 1	SQL - 2	SQL - 3
SQL Server	2 sec	1.934 sec	18.723sec
MongoDB	0 sec	0 sec	0.5 sec

Table 2. Time comparison SQL Server and MongoDB

VIII. Conclusion

After comparing different SQL, we could find the large difference between SQL Server and MongoDB. The overall performance from the SQL of spatial data, NoSQL - MongoDB provides more effective and shorter time to finish the task. On the aspect of importing the data, MongoDB is more flexible to import different types of files. CSV and Excel are the easy way for SQL Server to import the data.

Future Work

After struggling for tons of time in preprocessing the data, we finally imported different types to json file into SQL server and MongoDB. The future work would be to design more sql to test the performance on spatial data. Second, I will try to import the data to Oracle Database (I have stuck for a while with the docker version). The last thing would be comparing the SQL result with multiple thread setting.

Acknowledge

I would like to express the deepest appreciation to my advisor, PhD student Dejun Teng, his valuable and constructive suggestions during the planning and development of this research work. His willingness to give his time to teach the experience and knowledge so generously has been very much appreciated.

References

[1] Devogele, Thomas, Christine Parent, and Stefano Spaccapietra. "On spatial database integration." International Journal of Geographical Information Science 12.4 (1998): 335-352.

[2] Güting, Ralf Hartmut. "An introduction to spatial database systems." the VLDB Journal 3.4 (1994): 357-399.

[3] Hunter, Gary J., and Michael F. Goodchild. "Dealing with error in a spatial database: A simple case study." Photogrammetric Engineering and Remote Sensing 61.5 (1995): 529-537.

[4] Yeung, Albert KW, and G. Brent Hall. Spatial database systems: Design, implementation and project management. Vol. 87. Springer Science & Business Media, 2007.

[5] Makris, Antonios, et al. "Performance Evaluation of MongoDB and PostgreSQL for Spatio-temporal Data." EDBT/ICDT Workshops. 2019.

[6] Bartoszewski, Dominik, Adam Piorkowski, and Michal Lupa. "The comparison of processing efficiency of spatial data for PostGIS and MongoDB databases." International Conference: Beyond Databases, Architectures and Structures. Springer, Cham, 2019.

[7] Fang, Yi, et al. "Spatial indexing in microsoft SQL server 2008." Proceedings of the 2008 ACM SIGMOD international conference on Management of data. 2008.

[8] Aitchison, Alastair. Beginning spatial with SQL Server 2008. Apress, 2009.

[9] Yeung, Albert KW, and G. Brent Hall. Spatial database systems: Design, implementation and project management. Vol. 87. Springer Science & Business Media, 2007.

[10] https://www.mssqltips.com/sqlservertip/5295/different-ways-to-import-json-files-into-sql-server/

[11] https://kohera.be/blog/sql-server/how-to-import-a-json-file-into-a-slq-server-table/

[12] https://docs.exasol.com/sql_references/geospatialdata/import_geospatial_data_from_csv.htm

[13] https://www.bmc.com/blogs/mongodb-geolocation-query-examples/

[14] http://sharedderrick.blogspot.com/2017/12/sql-server-look-at-execution-plan-and.html

[15] https://docs.mongodb.com/manual/tutorial/geospatial-tutorial/

[16] https://bertwagner.com/posts/importing-geojson-earthquake-data-into-sql-server/

[17] https://kohera.be/blog/sql-server/how-to-import-a-json-file-into-a-slq-server-table/

[18] https://download.oracle.com/otndocs/products/spatial/pdf/biwa2018/BIWA18_Using_GeoJSON_in_Oracle_Database.pdf

[19] https://www.pauric.blog/How-To-Import-JSON-To-SQL-Server/

[20] https://www.sqlshack.com/import-json-data-into-sql-server/

[21] https://docs.microsoft.com/zh-tw/sql/t-sql/spatial-geography/point-geography-data-type?view=sql-server-ver15

[22] https://stackoverflow.com/questions/31379185/could-mongodb-send-multiply-update-commands-at-once

[23] https://www.mssqltips.com/sqlservertip/5404/parallelism-in-sql-server-execution-plan/

[24] https://docs.microsoft.com/zh-tw/sql/database-engine/configure-windows/configure-the-max-degree-of-parallelism-server-configuration-option?view=sql-server-ver15