大数据管理实验代码

实验一

1. ```
   select *
   from business
   where business_info->'$.city' = 'Tampa'
   order by business_info->'$.review_count' desc
   limit 10;
   ```

2. ```
   select
   JSON_KEYS(business_info),
   JSON_LENGTH(business_info),
   JSON_KEYS(business_info->'$.attributes') as attri_keys,
   JSON_LENGTH(business_info->'$.attributes') as attri_count
   from business
   limit 5;
   ```

3. ```
   select
   business_info->'$.name' as name,
   JSON_TYPE(business_info->'$.name') as name_type,
   business_info->'$.stars' as stars,
   JSON_TYPE(business_info->'$.stars') as stars_type,
   business_info->'$.attributes' as attributes,
   JSON_TYPE(business_info->'$.attributes') as attributes_type
   from business
   limit 5;
   ```

4. ```
   select
   JSON_UNQUOTE(business_info->'$.name') as name,
   business_info->'$.attributes' as attri,
   business_info->'$.hours' as hours
   from business
   where business_info->'$.attributes.HasTV' = 'True' and
   (JSON_VALUE(business_info, '$.hours.Sunday') is NULL or
   JSON_VALUE(business_info, '$.hours') is NULL)
   order by name asc
   limit 10;
   ```

5. ```
   explain format=json select * from user where user_info->'$.name'='Wanda';

   db.user.find({'name': 'Wanda'})
   db.user.find({'name': 'Wanda'}).explain("executionStats")
   ```
   ```

6.
```
select
JSON_PRETTY(business_info)
from business
where business_id = '4r3Ck65DCG1T6gpWodPyrg';

update business
set business_info = JSON_SET(
    business_info,
    '$.hours.Tuesday', '16:00-23:00',
    '$.rating', 4.5,
    '$.attributes.WiFi', 'Free'
)
where business_id = '4r3Ck65DCG1T6gpWodPyrg';
```

7.
```
insert into business (business_id, business_info)
select 'aaaaaabbbbbbccccccc2023', business_info
from business
where business_id = '5d-fkQteaqO6CSCqS5q4rw';

update business
set business_info = JSON_REMOVE(business_info, '$.name')
where business_id = 'aaaaaabbbbbbccccccc2023';

select business_id, JSON_PRETTY(business_info) from business
where business_id = 'aaaaaabbbbbbccccccc2023';
```

8.
```
select
state,
JSON_OBJECTAGG(city, city_count) as city_occ_num
from (
    select
    business_info->>'$.state' as state,
    business_info->>'$.city' as city,
    count(*) as city_count
    from business
    group by business_info->>'$.state', business_info->>'$.city'
) subquery
group by subquery.state
order by subquery.state;
```

9.
```
select
u.user_id as user_id,
u.name as name,
json_arrayagg(tip_info->'$.text') as text_array
from (
    select user_id, user_info->>'$.name' as name
    from user
    where REGEXP_LIKE(user_info->'$.friends', '__1cb6cwl3uAbMTK3xaGbg')
) as u
left join tip on tip.user_id = u.user_id
group by user_id, name
order by name asc;
```

10.
```
select
a.name as name1, a.city as city1,
b.name as name2, b.city as city2,
a.h as hours1, b.h as hours2,
JSON_OVERLAPS(a.h, b.h) as has_same_opentime
from (
    select
    business_info->'$.name' as name,
    business_info->'$.city' as city,
    business_info->'$.hours' as h
    from business
    where business_info->'$.city' = 'EdMonton'
) a, (
    select
    business_info->'$.name' as name,
    business_info->'$.city' as city,
    business_info->'$.hours' as h
    from business
    where business_info->'$.city' = 'Elsmere'
) b;
```

11.
```
select
user_info->'$.name',
user_info->'$.average_stars',
JSON_ARRAY(user_info->'$.funny', user_info->'$.useful', user_info->'$.cool',
user_info->'$.funny' + user_info->'$.useful' + user_info->'$.cool') as
'[funny, useful, cool, sum]'
from user
where user_info->'$.funny' > 2000 and user_info->'$.average_stars' > 4.0
# order by user_info->'$.average_stars' desc
limit 10;
```

12.
```
select
JSON_PRETTY(JSON_MERGE(b.business_info, u.user_info))
from (
    select business_info
    from business
    where business_id = (
        select business_id
        from tip
        group by business_id
        having count(*) >= all (
            select count(*)
            from tip
            group by business_id
        )
    )
) b, (
    select
    user_info
    from user
    where user_id = (
        select user_id
        from tip
```

```
        group by user_id
        having count(*) >= all (
            select count(*)
            from tip
            group by user_id
        )
    )
) u;
# in要比=慢很多!
```

13.
```
select
business_name,
business_review_count,
IF(business_info->'$.hours.Tuesday' is not null, 1, 0) AS
business_open_on_Tuesday,
day.num,
day.hours_in_a_week
from (
    select
    business_info->>'$.name' as business_name,
    business_info->'$.review_count' as business_review_count,
    business_info->'$.hours' as hours,
    business_info
    from business
    order by business_info->'$.review_count' desc
    limit 3
) as candidate join JSON_TABLE(candidate.hours, "$.*"
    COLUMNS (
        num FOR ORDINALITY,
        hours_in_a_week CHAR(32) PATH '$'
    )
)as day
order by business_name;
```

实验二

1.
```
db.review.find().limt(2).skip(6)
```

2.
```
db.business.find({ 'city':'Las Vegas' })
```

3.
```
db.user.find(
    { 'name':'Steve' },
    { _id: 1, useful: 1, cool: 1 }
).limit(10)

db.user.aggregate([
    { $match: { 'name': 'Steve' } },
    { $project: { _id: { 'oid': { $toString: '$_id' } }, useful: 1, cool: 1 }
},
    { $limit: 10 }
])
```

4.
```
db.user.find(
    { 'funny': { $in: [66, 67, 68] } },
    { name: 1, funny: 1}
).limit(20)

db.user.aggregate([
    { $match: { 'funny': {$in: [66, 67, 68]} } },
    { $project: { _id: { 'oid': { $toString: '$_id' } }, name: 1, funny: 1 }
},
    { $limit: 20 }
])
```

5.
```
db.user.find({ 'cool': { $gte: 15 }, 'cool': { $lt: 20 }, 'useful': { $gte:
50 } }).limit(10)
```

6.
```
db.business.estimatedDocumentCount()

db.business.find().explain("executionStats")
```

7.
```
db.business.find({ $or: [{ 'city': 'Westlake' }, { 'city': 'Calgary' }] })
```

8.
```
db.business.aggregate([
    { $match: { categories: {$size: 6}} },
    { $project: { _id: { 'oid': { $toString: '$_id' } }, categories: 1 } },
    { $limit: 10 }
])
```

9.
```
db.business.find({ business_id: "5JucpCfHZltJh5r1JabjDg"
}).explain("executionStats")

db.business.createIndex({ business_id: 1 })
```

10.
```
db.business.aggregate([
    { $group: { '_id': '$stars', cnt: { $sum: 1 } } },
    { $project: { _id: 0, cnt: 1, 'stars': '$_id' } },
    { $sort: { "stars": -1 } }
])
```

11.
```
 db.Subreview.insertMany(db.review.find().limit(500000).toArray())
 db.Subreview.createIndex({ text: "text" })
 db.Subreview.createIndex({ useful: 1 })
 db.Subreview.find({
     $text: { $search: "delicious" },
     useful: { $gt: 9 }
 })
 //db.Subreview.estimatedDocumentCount()
```

12.
```
db.Subreview.aggregate([
    { $match: { useful: { $gt: 6 }, funny: { $gt: 6 }, cool: { $gt: 6 } } },
    { $group: { '_id': '$business_id', star_avg: { $avg: '$stars' } } },
    { $project: { _id: 0, star_avg: 1, business_id: '$_id'} },
    { $sort: { business_id: -1 } }
])
```

13.
```
db.business.createIndex({ loc: '2dsphere' })
const targetBusiness = db.business.findOne({ business_id:
'xvX2CttrVhyG2z1dFg_0xw' });
const targetLocation = targetBusiness.loc;
db.business.find({loc: {
    $near: {
        $geometry: {
            type: 'Point',
            coordinates: targetLocation.coordinates
        },
        $maxDistance: 100
    }
} },
{ name: 1, address: 1, stars: 1, _id: 0})
```

14.
```
db.Subreview.createIndex({ date: 1 })
db.Subreview.createIndex({ user_id: 1 })
db.Subreview.aggregate([
    { $addFields: { parsedDate: { $toDate: "$date" } } },
    { $match: { parsedDate: { $gte: new Date("2017-01-01") } } },
    { $group: { _id: "$user_id", total: { $sum: 1 } } },
    { $project: { _id: 0, total: 1, 'user_id': '$_id' } },
    { $sort: { total: -1 } },
    { $limit: 20 }
])
```

15.
```
var mapFunction = function () {
   emit(this.business_id, { count: 1, total: this.stars });
};

var reduceFunction = function (key, values) {
    var reduced = { count: 0, total: 0 };
    values.forEach(function (value) {
    reduced.count += value.count;
    reduced.total += value.total;
   });
   return reduced;
};

var finalizeFunction = function (key, reduced) {
    return { count: reduced.count, stars: reduced.total , avg: reduced.total
/ reduced.count };
};

db.Subreview.mapReduce(
    mapFunction,
```

```
    reduceFunction,
    {
        out: "average_ratings",
        finalize: finalizeFunction
    }
);

db.average_ratings.find();
```

实验三

1. ```
   MATCH (n: CityNode)
     RETURN n
   LIMIT 10
   ```

2. ```
   MATCH (business: BusinessNode{ city: 'Ambridge' })
   RETURN business
   ```

3. ```
   MATCH (:ReviewNode{ reviewid: 'rEITo9OtpyKmEfNDp3Ou3A' })-[:Reviewed]->
   (business: BusinessNode)
   RETURN business
   ```

4. ```
   MATCH (u: UserNode)-[:Review]->(:ReviewNode)
   -[:Reviewed]->(:BusinessNode{ businessid: 'fyJAqmweGm8VXnpU4CWGNw' })
   RETURN u.name, u.fans
   ```

5. ```
   MATCH (:UserNode{ userid: 'TEtzbpgA2BFBrC0y0sCbfw' })
     -[:Review]->(:ReviewNode{ stars: '5.0' })
   -[:Reviewed]->(business: BusinessNode)
   RETURN business.name, business.address
   ```

6. ```
   MATCH (business: BusinessNode)
     RETURN business.name, business.stars, business.address
   ORDER BY business.stars DESC
   LIMIT 15
   ```

7. ```
   MATCH (u: UserNode)
     WHERE toInteger(u.fans) > 200
   RETURN u.name, u.fans
   LIMIT 10
   ```

8. ```
   MATCH (:BusinessNode{ businessid: 'tyjquHslrAuF5EUejbPfrw' })
     -[IN_CATEGORY]->(cate: CategoryNode)
   RETURN count(cate)

   PROFILE MATCH (:BusinessNode{ businessid: 'tyjquHslrAuF5EUejbPfrw' })
   -[IN_CATEGORY]->(cate: CategoryNode)
   RETURN count(cate)
   ```

9.
```
MATCH (:BusinessNode{ businessid: 'tyjquHslrAuF5EUejbPfrw' })-[IN_CATEGORY]->
(cate: CategoryNode)
  RETURN collect(cate.category)
```

10.
```
MATCH(:UserNode{ name: 'Allison' })
-[:HasFriend]->(friend)
WITH friend.name as friendsList,
size((friend)-[:HasFriend]->()) as numberOfFoFs
RETURN friendsList, numberOfFoFs
// 很慢↓
MATCH(:UserNode{ name: 'Allison' })
-[:HasFriend]->(friend)
WITH friend.name as friendsList,
(friend)-[:HasFriend]->(ff)
RETURN friendsList, count(ff) as numberOfFoFs
```

11.
```
MATCH (business: BusinessNode)
-[:IN_CATEGORY]->(:CategoryNode{ category: 'Salad' })
RETURN business.city as city, count(*) as cnt
ORDER by cnt DESC
LIMIT 5
```

12.
```
MATCH (business: BusinessNode)
WITH business.name as name, count(*) as cnt
RETURN name, cnt
ORDER BY cnt DESC
LIMIT 10
```

13.
```
MATCH (business: BusinessNode)
WITH count(DISTINCT business.name) as total
MATCH (business: BusinessNode)
WHERE toInteger(business.reviewcount) > 5000
WITH business, business.reviewcount as rcnt, count(business.name) as cnt,
total
RETURN (cnt * 1.0 / total) as rate, business.name, rcnt
ORDER BY rcnt DESC
```

14.
```
MATCH (:UserNode)
-[:Review]->(:ReviewNode{ stars: '5.0' })
-[:Reviewed]->(business: BusinessNode)
-[:IN_CATEGORY]->(:CategoryNode{ category: 'Zoos' })
RETURN DISTINCT business.city as city
```

15.
```
MATCH (u: UserNode)
-[:Review]->(:ReviewNode)
-[:Reviewed]->(business: BusinessNode)
WITH business.businessid as businessid, business.name as name, count(DISTINCT
u) as cnt
RETURN businessid, name, cnt
ORDER BY cnt DESC
LIMIT 10
```

16.
```
// 创建新属性
MATCH (user: UserNode)
WHERE toInteger(user.funny) > 450
SET user.test = user.funny

// 创建索引
CREATE INDEX FOR (user: UserNode) ON (user.test)

// 查询
MATCH (user: UserNode)
WHERE toInteger(user.test) > 4000
RETURN user

// 更新
MATCH (user: UserNode)
WHERE toInteger(user.test) > 4000
SET user.test = 8888

// 删除
MATCH (user: UserNode)
WHERE toInteger(user.test) > 450
REMOVE user.test
```

17.
```
CREATE INDEX FOR (u: UserNode) ON (u.userid)
CREATE INDEX FOR (business: BusinessNode) ON (business.businessid)

MATCH (user1: UserNode{ userid: 'tvZKPah2u9G9dFBg5GTOeg' })
-[:Review]->(:ReviewNode)
-[Reviewed]->(business: BusinessNode)
WITH user1, collect(DISTINCT business.businessid) as user1_list
MATCH (user2: UserNode)
-[:Review]->(:ReviewNode)
-[Reviewed]->(business: BusinessNode)
WHERE not (user1)-[:HasFriend]->(user2) and business.businessid in user1_list
RETURN user1.name, user2.name, count(business) as cnt
ORDER BY cnt DESC
```

18.
```
MATCH (:ReviewNode{reviewid: 'TIYgnDzezfeEnVeu9jHeEw'})
-[:Reviewed]->(business: BusinessNode)
RETURN business

const certain = db.review.findOne({'review_id':
'TIYgnDzezfeEnVeu9jHeEw'}).business_id
db.business.find({"business_id": certain}).explain()
```

实验四

1.
```
MATCH (u: UserNode)
-[:Review]->(:ReviewNode)
-[:Reviewed]->(business: BusinessNode)
WITH DISTINCT u, COUNT(DISTINCT business) as cnt
WHERE cnt > 5
RETURN u.name, u.funny, u.fans

scp 3.1.csv root@1.94.53.234:/root/data/

db.createCollection("userReview")
mongoimport -d=yelp -c=userReview --type=csv --headerline ./data/3.1.csv
```

2.
```
db.userReview.aggregate([
{ $group: { '_id': '$u.name', name_sum: { $sum: 1 } } },
{ $project: { _id: 0, name: '$_id', name_sum: 1} },
{ $sort: { name_sum: -1 } }
])
```

3.
```
MATCH (business: BusinessNode)
-[:IN_CATEGORY]->(cate: CategoryNode)
WITH business, cate.category as category
RETURN business.name as name, category, business.city as city
// 上传文件到服务器
scp 3.2.csv root@1.94.53.234:/root/data/

db.createCollection("businessInfo")
// 导入MongoDB
mongoimport -d=yelp -c=businessInfo --type=csv --headerline ./data/3.2.csv
// 查看是否导入成功
db.businessInfo.find().limit(1)

db.createCollection("businessInfoDistinct")
db.businessInfo.aggregate([
{ $group: { _id: { city: '$city', category: '$category' } } }
]).forEach((item) => { db.businessInfoDistinct.insert( item._id ) } )

db.businessInfoDistinct.count()
db.businessInfoDistinct.find().limit(1)

mongoexport -d yelp -c businessInfoDistinct --type=csv --fields city,category
--out data/res.csv
cd ~/neo4j-community-4.0.9/import
cp /root/data/res.csv ./

LOAD CSV WITH HEADERS FROM "file:///res.csv" AS f
MERGE (c:CityNode {city: COALESCE(f.city, "")})
MERGE (cate: CategoryNode {category: COALESCE(f.category, "")})
CREATE (c) -[:Has]-> (cate)
```

实验五

```
create database mvcc_test

create table movie (
    id int primary key,
    title varchar(20),
    time varchar(50)
) engine = InnoDB;

insert into movie (id, title, time)
values
(1, "movie1", "2023/11/17"),
(2, "movie2", "2023/11/18"),
(3, "movie3", "2023/11/19"),
(4, "movie4", "2023/11/20");

set session transaction isolation level repeatable read;

insert into movie (id, title, time)
values
(5, "movie5", "2023/11/21");

update movie
set title = "hhh"
where id = 5;
```

```
use testdb
db.movie.insertOne({ "title": "movie1", "time": "2023/11/17" })
db.movie.insertMany([
    {"title": "movie2", "time": "2023/11/18"},
    {"title": "movie3", "time": "2023/11/19"},
    {"title": "movie4", "time": "2023/11/20"}
])
db.movie.find()

const session = db.getMongo().startSession();
session.startTransaction();

session.getDatabase("testdb").getCollection("movie").insertOne(
    { "title": "movie1001", "time": "2023/11/17" }, { session }
);

session.getDatabase("testdb").getCollection("movie").find();

session.commitTransaction();
session.endSession();

session.abortTransaction();

db.getDatabase("testdb").getCollection("movie").find();
db.getSiblingDB("testdb").getCollection("movie").find({});
```

```
db.getSiblingDB("testdb").getCollection("movie")
.deleteOne({ _id:ObjectId("6566e7d4cfdc26c107b5ac27") })
```