



# 华中科技大学

## 大数据管理概论实验报告

姓 名： 郑澍频  
学 院： 计算机科学与技术学院  
专 业： 计算机科学与技术  
班 级： CS2108  
学 号： I202120037

分数	
教师签名	

2024 年 12 月 19 日

## 教师评分页

子目标	子目标评分
1	
2	
3	
总分	

# 目 录

<b>1</b>	<b>课程任务概述 .....</b>	<b>1</b>
<b>2</b>	<b>MySQL for JSON 实验 .....</b>	<b>2</b>
2.1	任务要求 .....	2
2.2	完成过程 .....	2
2.3	任务小结 .....	9
<b>3</b>	<b>MongoDB 实验 .....</b>	<b>10</b>
3.1	任务要求 .....	10
3.2	完成过程 .....	10
3.3	任务小结 .....	14
<b>4</b>	<b>NEO4J .....</b>	<b>15</b>
4.1	任务要求 .....	15
4.2	完成过程 .....	15
4.3	任务小结 .....	18
<b>5</b>	<b>多数据库交互应用实验 .....</b>	<b>19</b>
5.1	任务要求 .....	19
5.2	完成过程 .....	19
5.3	任务小结 .....	23
<b>6</b>	<b>课程总结 .....</b>	<b>24</b>

# 1 课程任务概述

本实验将围绕大数据管理的基本原理与实践进行，主要使用三种数据库技术：MySQL、MongoDB 和 Neo4j，通过一系列实验任务，深入了解不同类型的数据库在实际应用中的操作与性能差异。实验任务分为五大模块，具体如下：

## 1. MySQL for JSON 实验

掌握如何在 MySQL 中使用 JSON 数据类型进行各种查询、增删改操作，并对 JSON 数据进行聚合、处理和优化。任务内容包括对 JSON 数据的提取、修改、聚合、以及性能优化的实验，旨在提高对 MySQL JSON 功能的掌握，并了解其执行计划和索引优化方法。

## 2. MongoDB 实验

主要围绕 MongoDB 进行条件查询、聚合查询、执行计划优化等操作。通过对 Yelp 数据集的操作，深入了解 MongoDB 的查询语言、索引机制、聚合管道及 MapReduce 方法。同时，通过比较不同查询方式的效率，学习如何优化查询性能。

## 3. Neo4j 实验

使用图数据库 Neo4j 进行图数据的查询与分析。任务内容包括简单的节点和关系查询、条件过滤、数据聚合以及执行计划分析。通过这些实验理解图数据库在处理复杂关系查询中的优势，并能够根据需求构建高效的查询语句。

## 4. 多数据库交互应用实验

结合 MySQL、MongoDB 和 Neo4j 的多数据库交互应用，展示如何在不同数据库之间进行数据迁移、去重与聚合。实验旨在提升多数据库环境下的协作能力，以及处理跨数据库的数据操作的技巧。

## 5. 不同类型数据库 MVCC 多版本并发控制对比实验

通过 MySQL 和 MongoDB 对 MVCC（多版本并发控制）进行对比，帮助同学们理解事务隔离性、并发控制机制及其在不同数据库中的实现。通过构造多用户并发操作场景，分析不同数据库在 MVCC 支持下的行为和性能差异。

## 2 MySQL for JSON 实验

### 2.1 任务要求

掌握如何在 MySQL 中使用 JSON 数据类型进行各种查询、增删改操作，并对 JSON 数据进行聚合、处理和优化。任务内容包括对 JSON 数据的提取、修改、聚合、以及性能优化的实验，旨在提高对 MySQL JSON 功能的掌握，并了解其执行计划和索引优化方法。

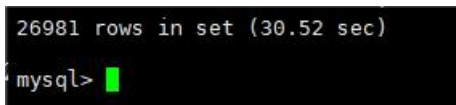
### 2.2 完成过程

启动华为云服务器，启动 XShell 连接服务器，导入所需数据库，启动 mysql，调用 test 数据库，开始任务。

#### 2.2.1 JSON 基本查询

使用 explain 查看 `select * from user where user_info->'$.cool' > 200` 的执行计划，其中执行计划按 JSON 格式输出；并且实际执行一次该查询，请注意观察语句消耗的时间并与 MongoDB 的查询方式进行对比（MongoDB 要执行此查询要求，相应的语句是什么？执行计划是怎样的？并给出查询效率对比）。最后，在 MySQL 中为 `user_info` 的字段加索引来优化提高查询效率，对比一下 MySQL 加索引查询前后的查询效率，分析加索引前后的执行计划。

（1）首先执行 `SELECT * FROM USER WHERE user_info->'$.cool'>200` 语句，并且记录运行时间，如下图所示：



```
26981 rows in set (30.52 sec)
mysql>
```

图 2.1 MySQL 的执行时间（无索引）

（2）接着使用 EXPLAIN 查看语句的执行计划，由图 2.2 可得知本次查询使用全表查询（ALL）的方式对用户表（user）进行查询，共扫描了 1352687 行数据。cost\_info 部分则是显示了本次查询的开销。

执行语句为：

```
EXPLAIN FORMAT=JSON
SELECT *
FROM user
WHERE JSON_UNQUOTE(JSON_EXTRACT(user_info, '$.cool')) > 200;
```

```

{
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "457883.70"
    },
    "table": {
      "table name": "user",
      "access_type": "ALL",
      "rows_examined_per_scan": 1352607,
      "rows_produced_per_join": 1352607,
      "filtered": "100.00",
      "cost_info": {
        "read_cost": "322623.00",
        "eval_cost": "135260.70",
        "prefix_cost": "457883.70",
        "data_read_per_join": "154M"
      },
      "used_columns": [
        "user_id",
        "user_info"
      ],
      "attached_condition": "(json_unquote(json_extract(`test`.`user`.`user_info`, '$.cool')) > 200)"
    }
  }
}

```

图 2.2 MySQL 查询的执行计划

(3) 在 MongoDB 的数据库下进行类似的查询，查询语句为：

```
db.user.find({ "user_info.cool": { $gt: 200 } });
```

查询语句的执行计划语句：

```
db.user.find({ "user_info.cool": { $gt: 200 } }).explain("executionStats");
```

```

> db.user.find({ "user_info.cool": { $gt: 200 } }).explain("executionStats");
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "yelp.user",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "user_info.cool" : {
        "$gt" : 200
      }
    },
    "winningPlan" : {
      "stage" : "COLLSCAN",
      "filter" : {
        "user_info.cool" : {
          "$gt" : 200
        }
      },
      "direction" : "forward"
    },
    "rejectedPlans" : [ ]
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 0,
    "executionTimeMillis" : 11843,
    "totalKeysExamined" : 0,
    "totalDocsExamined" : 1637141,
    "executionStages" : {
      "stage" : "COLLSCAN",
      "filter" : {
        "user_info.cool" : {
          "$gt" : 200
        }
      },
      "nReturned" : 0,
      "executionTimeMillisEstimate" : 9425,
      "works" : 1637143,
      "advanced" : 0,
      "needTime" : 1637142,
      "needYield" : 0,
      "saveState" : 1689,
      "restoreState" : 1689,
      "isEOF" : 1,
      "direction" : "forward",
      "docsExamined" : 1637141
    }
  },
  "serverInfo" : {
    "host" : "sping02",
    "port" : 27017,
    "version" : "4.4.29",
    "gitVersion" : "f4dda329a99811c707eb06d05ad023599f9be263"
  },
  "ok" : 1
}

```

图 2.3 MongoDB 查询的执行计划

由图 2.1 和图 2.3 的可知，MySQL 使用全表查询的方式查询所需的时间是 30.52 秒，而在 MongoDB 中使用 COLLSCAN 方式查询相同的数据只需 11.8 秒。

(4) 在 MySQL 中为 user\_info 的字段加索引来优化提高查询效率

为了提高查询效率，可以在 user\_info 字段的 cool 部分添加一个虚拟列，并为该虚拟列添加索引。这样可以避免全表扫描。

```
//添加虚拟列
ALTER TABLE user ADD COLUMN cool INT GENERATED ALWAYS AS
(JSON_UNQUOTE(JSON_EXTRACT(user_info, '$.cool'))) STORED;

//为虚拟列添加索引
CREATE INDEX idx_cool ON user(cool);

//执行查询
SELECT *
FROM user
WHERE cool > 200;
```

图 2.4 是添加虚拟列后，执行查询所需的时间。图 2.5 则是其执行计划。

```
-----+-----+
26981 rows in set (1.42 sec)
```

图 2.4 MySQL 查询的执行时间（有索引）

```
{
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "66377.47"
    },
    "table": {
      "table_name": "user",
      "access_type": "range",
      "possible_keys": [
        "idx_cool"
      ],
      "key": "idx_cool",
      "used_key_parts": [
        "cool"
      ],
      "key_length": "5",
      "rows_examined_per_scan": 55674,
      "rows_produced_per_join": 55674,
      "filtered": "100.00",
      "index_condition": "(`test`.`user`.`cool` > 200)",
      "cost_info": {
        "read_cost": "60810.07",
        "eval_cost": "5567.40",
        "prefix_cost": "66377.47",
        "data_read_per_join": "6M"
      },
      "used_columns": [
        "user_id",
        "user_info",
        "cool"
      ]
    },
    "rows": 26981
  }
}
```

图 2.5 MySQL 查询的执行计划（有索引）

由上述实验可得，在没有索引的情况下，MySQL 执行全表扫描，执行计划显示使用了 ALL 访问类型，表明全表扫描，没有使用任何索引。执行查询所需时间为 30.52 秒。在没有索引的情况下，MongoDB 也进行了全表扫描，执行时间为 11.8s，但仍高于加索引后的 MySQL 查询。通过为 cool 字段添加索引，MySQL 执行计划显示使用了索引扫描，避免了全表扫描。执行查询所需时间也降到了 1.42 秒，大大的提高了查询的效率。

### 2.2.2 JSON 增删改:

在 business 表中,查询 id 为--eBbs3HpZYIym5pEw8Qdw 的商户 business\_info, 这里对 info 列的显示需要使用 JSON\_PRETTY(business\_info)让可读性更高,然后在它的 attributes 中新增"BikeParking":"True"的键值对,并将其评论数量改为 42,属性的 'WiFi' 对应的值置为 "Paid", 返回其 business\_info, 同样用 JSON\_PRETTY(business\_info)展示在修改前后的差异.

(1) 首先，对 id 为 eBbs3HpZYIym5pEw8Qdw 的商户进行查询。查询语句如下：

```
SELECT JSON_PRETTY(business_info) AS business_info
FROM business
WHERE business_id = '--eBbs3HpZYIym5pEw8Qdw';
```

```
| {
  "city": "Tampa",
  "name": "Holiday Inn Express & Suites Tampa-Fairgrounds-Casino",
  "hours": {
    "Friday": "0:0-0:0",
    "Monday": "0:0-0:0",
    "Sunday": "0:0-0:0",
    "Tuesday": "0:0-0:0",
    "Saturday": "0:0-0:0",
    "Thursday": "0:0-0:0",
    "Wednesday": "0:0-0:0"
  },
  "stars": 2.5,
  "state": "FL",
  "address": "8610 Elm Fair Blvd",
  "is_open": 1,
  "latitude": 27.9911714,
  "longitude": -82.3578649,
  "attributes": {
    "WiFi": "u'free'",
    "RestaurantsPriceRange2": "2",
    "BusinessAcceptsCreditCards": "True"
  },
  "categories": "Venues & Event Spaces, Event Planning & Services, Hotels, Hotels & Travel",
  "postal_code": "33610",
  "review_count": 24
} |
```

图 2.6 商户信息查询（修改前）



(2) 对商户信息进行增加与修改

```
UPDATE business
SET business_info = JSON_SET(
    business_info,
    '$.attributes.BikeParking', 'True',
    '$.review_count', 42,
    '$.attributes.WiFi', 'Paid'
)WHERE business_id = '--eBbs3HpZYIym5pEw8Qdw';
```

```
| {
  "city": "Tampa",
  "name": "Holiday Inn Express & Suites Tampa-Fairgrounds-Casino",
  "hours": {
    "Friday": "0:0-0:0",
    "Monday": "0:0-0:0",
    "Sunday": "0:0-0:0",
    "Tuesday": "0:0-0:0",
    "Saturday": "0:0-0:0",
    "Thursday": "0:0-0:0",
    "Wednesday": "0:0-0:0"
  },
  "stars": 2.5,
  "state": "FL",
  "address": "8610 Elm Fair Blvd",
  "is_open": 1,
  "latitude": 27.9911714,
  "longitude": -82.3578649,
  "attributes": {
    "WiFi": "Paid",
    "BikeParking": "True",
    "RestaurantsPriceRange2": "2",
    "BusinessAcceptsCreditCards": "True"
  },
  "categories": "Venues & Event Spaces, Event Planning & Services, Hotels, Hotels & Travel",
  "postal_code": "33610",
  "review_count": 42
}
```

图 2.7 商户信息查询（修改后）

向 user 表插入一个 id 是 'change' 的商户,其商户信息与 id 为 '--agAy0vRYwG6WqbInorfg' 的商户完全一样,插入完成之后,将这个新记录的 user\_info 中的 fans 以及 useful 键值对删去,为其增加一个 'city': 'New York' 的键值对,向其中最后查询 'change' 的所有信息。

(1) 从 user 表插入一条与现有商户信息相同的新记录,其 user\_id 为 'change',并且信息与 user\_id 为 '--agAy0vRYwG6WqbInorfg' 的商户完全一致。

```
INSERT INTO user (user_id, user_info)
SELECT 'change', user_info
FROM user
WHERE user_id = '--agAy0vRYwG6WqbInorfg';
```

(2) 删除新记录中的 fans 和 useful 键值对,并在 user\_info 中增加新的键值对 'city': 'New York'。

```
UPDATE user
SET user_info = JSON_REMOVE(user_info, '$.fans', '$.useful'),
    user_info = JSON_SET(user_info, '$.city', 'New York')
WHERE user_id = 'change';
```

(3) 查询新插入的记录,即 user\_id = 'change' 的所有信息。

```
SELECT user_id, JSON_PRETTY(user_info) AS user_info
FROM user
WHERE user_id = 'change';
```

```
{
  "change": {
    "city": "New York",
    "cool": 3,
    "name": "Noah",
    "elite": "",
    "funny": 4,
    "friends": "xN8yzZ6qLLYgvgl-7uA_mA, FvMtFly9q074hvxhkYL1aA, NSnRzqDfmltTzbfKbltNng, I4TegAino0tpeRkYK
uRYWQ, xt4C_64SX5p1mJw2ZeU0vQ, nYhA-DYkzdbnnPCf0gtpag, WqAu_9V8a6n_nVGfQnCTg, ltpH0xg87UTHmZxwAN4tW,
Yh0qSVy-i_0KCPHn4Z6Clew, Wq22ugrg-YeVhU_6150tIg, nY5E5E-eeM-511287fReFA, Zf3tkoJezL-uKvzcFyc110, og-v09p
lC5tiniV74dhndg, AK_nre1V03v0jcrnuWk80, HoVrasA1Fic-qiyy0s3W, 40szAwPkW0LRVFE17Uf1Fg, nYb8za1Yvj_SuA
QqC1Aapp, G3an0iobFa-wt_LladWew, n3kbbko17ZC8Rhfhw4n7W, Eb9LLVbetjF61W0YpB0ZCg, TTmLlf_YA61ouaFbRWMLf
Q, 391u6C9WRoq2R1qVsH0wuiA, -KUaFuBrqRbInA7nr6X5Xg, KykJnvANS5A54jk00-Drr-0, 9Dgn90Ng58CAU9_xHHKw, RqMP
Wo_x5MnzEwP33nKN00, sa9VkvvFSvbW10B-p7592Q, 7a8vMXFmzH5ff4EYm35XyQ, HdV7BlwDVuCY_9PPI-oiNA, ppd0aa_7lwj
HT6_zTt901g, Ue2Kcx6CMmrYws9RkaIaQ, KPJx8CCn8g-02zGDCwvW-g, h57RNCwyrY1oujpj8fLXw, FYSR-S4wfxix0YY1ad
qU4A, 1j3EFSy2JAA2vXsID28S9w, ys4gdg3jCbUECsU9ouMKuQ, Kkj_yeyfLPM8step35n0A, 3a8Zm5XGk7UoFnashAfuJQ, -
Y_bMstrtVZT6Sc12a6WdQ, uIMTy106kMufc227FEALjA, o5wVRsswBaVLE10jTmIDzw, HtopFK4zgVArCjXuZDPhHw, saS_anPG
9NrycMU0QuGlg, j54Ioru-24vMN8vJ2kGLLQ, 4Ee9j6yHnty2aNd-M9m3Q, UD8pkL7fo-fRnPGZ2B04_Q, NA2-Yq4cusSVRhs
js-JziQ, lUbsEyNeBdETGbuSYi8kBW, wjiBMc0SI7hvfX-g5-oDuw",
    "review_count": 5,
    "average_stars": 4.0,
    "yelping_since": "2010-07-17 17:11:53",
    "compliment_hot": 0,
    "compliment_cool": 0,
    "compliment_cute": 0,
    "compliment_list": 0,
    "compliment_more": 0,
    "compliment_note": 0,
    "compliment_funny": 0,
    "compliment_plain": 0,
    "compliment_photos": 0,
    "compliment_writer": 0,
    "compliment_profile": 0
  }
}
```

图 2.8 用户信息增改结果

### 2.2.3 JSON 聚合

从 tip 表中, 选择所有用户的 user\_id, 以及与每个用户关联的 business\_id 和对应的 tip\_info, 使用 JSON\_ARRAYAGG 函数将每个用户的 tips 聚合成一个 JSON 数组, 限制返回 5 行。

(1) 在 JSON\_ARRAYAGG 中使用 JSON\_OBJECT 函数将每条 tip 的 business\_id 和 tip\_info 结合成一个 JSON 对象。JSON\_ARRAYAGG 将每个用户的所有 tip 聚合成一个 JSON 数组。再用 GROUP BY user\_id: 按照 user\_id 聚合数据, 这样每个用户对应一个聚合的 tips 数组。

```
SELECT
    user_id,
    JSON_ARRAYAGG(
        JSON_OBJECT('business_id', business_id, 'tip_info', tip_info)
    ) AS tips
FROM tip
GROUP BY user_id
LIMIT 5;
```

## 2.2.4 JSON 实用函数的使用

在 tip 表中找到 business\_id 为 -1b2kNOowsPrPpBOK4lNkQ 的商户和 user\_id 为 --7XOV5T9yZR5w1DIy\_Dog 的用户, 合并二者的 info 列的 JSON 文档为一个文档显示, 对于 JSON 文档中相同的 key 值, 应该保留二者的 value 值。

(1) 使用 JSON\_MERGE\_PRESERVE() 函数将两个 JSON 文档合并为一个。与普通的 JSON\_MERGE() 不同, JSON\_MERGE\_PRESERVE() 会保留两个文档中相同 key 的所有值, 而不是覆盖其中一个。

(2) 第一个子查询从 tip 表中获取 business\_id = '-1b2kNOowsPrPpBOK4lNkQ' 的商户的 tip\_info 列的 JSON 文档。第二个子查询从 user 表中获取 user\_id = '--7XOV5T9yZR5w1DIy\_Dog' 的用户的 user\_info 列的 JSON 文档。

```
SELECT JSON_MERGE_PRESERVE(  
  (SELECT tip_info FROM tip  
   WHERE business_id = '-1b2kNOowsPrPpBOK4lNkQ' LIMIT 1),  
  (SELECT user_info FROM user  
   WHERE user_id = '--7XOV5T9yZR5w1DIy_Dog' LIMIT 1)  
) AS merged_info;
```

```
---+  
| merged_info  
+---+  
|  
+-----+  
| { "cool": 7, "date": "2018-05-10 16:20:46", "fans": 0, "name": "Angela", "text": "The huevos rancheros and the peri  
cos are worth trying. Also the pan de bono as well", "elite": "", "funny": 2, "useful": 37, "friends": "None", "revi  
ew count": 25, "average stars": 4.31, "yelping since": "2018-05-10 20:02:01", "compliment_hot": 0, "compliment_cool"  
: 0, "compliment_cute": 0, "compliment_list": 0, "compliment_more": 1, "compliment_note": 0, "compliment_count": 0,  
"compliment_funny": 0, "compliment_plain": 0, "compliment_photos": 0, "compliment_writer": 0, "compliment_profile":  
0 } |  
+-----+  
+---+  
1 row in set (0.00 sec)
```

图 2.9 JSON\_MERGE\_PRESERVE 结果

rating 保留了 tip\_info 和 user\_info 中的两个 rating 值, 并将它们作为数组保留。comment, location, name, friends: 这些键只有在其中一个 JSON 文档中存在时才会出现在合并结果中。

## 2.3 任务小结

MySQL 任务的难点在于各种连接以及 JSON 语句的掌握, 在熟练掌握数据存储的结构以及 JSON 的语句用法之后就会轻松不少。

## 3 MongoDB 实验

### 3.1 任务要求

主要围绕 MongoDB 进行条件查询、聚合查询、执行计划优化等操作。通过对 Yelp 数据集的操作，深入了解 MongoDB 的查询语言、索引机制、聚合管道及 MapReduce 方法。同时，通过比较不同查询方式的效率，学习如何优化查询性能。

### 3.2 完成过程

启动华为云服务器，启动 XShell 连接服务器，导入所需数据库，启动 MongoDB，调用 yelp 数据库，开始任务。

#### 3.2.1 条件查询与执行计划:

使用 explain 看 db.business.find({business\_id: "5JucpCfHZltJh5r1JabjDg"}) 的执行计划，了解该查询的执行计划及查询执行时间，并给出物理优化手段，以提高查询性能，通过优化前后的性能对比展现优化程度。

(1) 查询该语句执行计划的语句如下:

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

执行上述语句之后可以得到图 3.1 的结果。从结果中，我们可以得知整个查询的执行细节，其中包括查询方式，执行时间，执行开销等。图 3.1 的结果显示，该查询采用 COLLSCAN 方式扫描全表查询与该 business\_id 相关的元素，该查询执行成功且用时 65 毫秒，共扫描了 192609 个文件。

(2) 采用建立索引的方式来提高查询的性能，关键语句如下:

1. 确保查询字段有索引（不用全表扫描，使用索引扫描）

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

2. 覆盖索引（覆盖索引会使 MongoDB 直接从索引中返回结果，而不需要回表查询，这会大大提高查询效率。）

```
db.business.createIndex({ business_id: 1, name: 1, address: 1 });
```

```

> db.business.find({business_id: "5JucpCfHZltJh5r1JabjDg"}).explain("executionStats");
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "yelp.business",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "business_id" : {
        "$eq" : "5JucpCfHZltJh5r1JabjDg"
      }
    },
    "winningPlan" : {
      "stage" : "COLLSCAN",
      "filter" : {
        "business_id" : {
          "$eq" : "5JucpCfHZltJh5r1JabjDg"
        }
      },
      "direction" : "forward"
    },
    "rejectedPlans" : [ ]
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 1,
    "executionTimeMillis" : 65,
    "totalKeysExamined" : 0,
    "totalDocsExamined" : 192609,
    "executionStages" : {
      "stage" : "COLLSCAN",
      "filter" : {
        "business_id" : {
          "$eq" : "5JucpCfHZltJh5r1JabjDg"
        }
      },
      "nReturned" : 1,
      "executionTimeMillisEstimate" : 6,
      "works" : 192611,
      "advanced" : 1,
      "needTime" : 192609,
      "needYield" : 0,
      "saveState" : 192,
      "restoreState" : 192,
      "isEOF" : 1,
      "direction" : "forward",
      "docsExamined" : 192609
    }
  },
  "serverInfo" : {
    "host" : "sping02",
    "port" : 27017,
    "version" : "4.4.29",
    "gitVersion" : "f4dda329a99811c707eb06d05ad023599f9be263"
  },
  "ok" : 1
}
>

```

图 3.1 查询的执行计划

```

spring02 > db.business.createIndex({ business_id: 1 });
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
spring02 > db.business.createIndex({ business_id: 1, name: 1, address: 1 });
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}
spring02 > db.business.find({business_id: "5JucpCFH2ltjh5r1Jabj0g"}).explain("executionStats");
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "yelp.business",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "business_id" : {
        "seq" : "5JucpCFH2ltjh5r1Jabj0g"
      }
    },
    "winningPlan" : {
      "stage" : "FETCH",
      "inputStage" : {
        "stage" : "IXSCAN",
        "keyPattern" : {
          "business_id" : 1
        },
        "indexName" : "business_id_1",
        "isMultiKey" : false,
        "multiKeyPaths" : {
          "business_id" : [ ]
        },
        "isUnique" : false,
        "isSparse" : false,
        "isPartial" : false,
        "indexVersion" : 2,
        "direction" : "forward",
        "indexBounds" : {
          "business_id" : [
            ["5JucpCFH2ltjh5r1Jabj0g"], ["5JucpCFH2ltjh5r1Jabj0g"]
          ]
        }
      }
    },
    "rejectedPlans" : [
      {
        "stage" : "FETCH",
        "inputStage" : {
          "stage" : "IXSCAN",
          "keyPattern" : {
            "business_id" : 1,
            "name" : 1,
            "address" : 1
          },
          "indexName" : "business_id_1_name_1_address_1",
          "isMultiKey" : false,
          "multiKeyPaths" : {
            "business_id" : [ ],
            "name" : [ ],
            "address" : [ ]
          },
          "isUnique" : false,
          "isSparse" : false,
          "isPartial" : false,
          "indexVersion" : 2,
          "direction" : "forward",
          "indexBounds" : {
            "business_id" : [
              ["5JucpCFH2ltjh5r1Jabj0g"], ["5JucpCFH2ltjh5r1Jabj0g"]
            ],
            "name" : [
              ["MinKey", MaxKey]
            ],
            "address" : [
              ["MinKey", MaxKey]
            ]
          }
        }
      }
    ]
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 1,
    "executionTimeMillis" : 1,
    "totalKeysExamined" : 1,
    "totalDocsExamined" : 1,
    "executionStages" : [
      {
        "stage" : "FETCH",
        "nReturned" : 1,
        "executionTimeMillisEstimate" : 0,
        "works" : 3,
        "advanced" : 1,
        "needTime" : 0,
        "needYield" : 0,
        "saveState" : 0,
        "restoreState" : 0,
        "isEOF" : 1,
        "docsExamined" : 1,
        "alreadyHasObj" : 0,
        "inputStage" : {
          "stage" : "IXSCAN",
          "nReturned" : 1,
          "executionTimeMillisEstimate" : 0,
          "works" : 2,
          "advanced" : 1,
          "needTime" : 0,
          "needYield" : 0,
          "saveState" : 0,
          "restoreState" : 0,
          "isEOF" : 1,
          "keyPattern" : {
            "business_id" : 1
          },
          "indexName" : "business_id_1",
          "isMultiKey" : false,
          "multiKeyPaths" : {
            "business_id" : [ ]
          },
          "isUnique" : false,
          "isSparse" : false,
          "isPartial" : false,
          "indexVersion" : 2,
          "direction" : "forward",
          "indexBounds" : {
            "business_id" : [
              ["5JucpCFH2ltjh5r1Jabj0g"], ["5JucpCFH2ltjh5r1Jabj0g"]
            ]
          }
        },
        "keysExamined" : 1,
        "seeks" : 1,
        "dupsTested" : 0,
        "dupsDropped" : 0
      }
    ]
  },
  "serverInfo" : {
    "host" : "spring02",
    "port" : 27027,
    "version" : "4.4.20",
    "gitVersion" : "f4dda329a99811c707eb06d05ad023599f9be263"
  },
  "ok" : 1
}

```

图 3.2 优化后的执行计划

### 3.2.2 聚合与索引

创建一个review的子集合Subreview(取review的前五十万条数据), 分别对评论的内容建立全文索引, 对useful建立升序索引, 然后查询评价的内容中包含关键词delicious且useful大于等于50的评价,按照review\_id进行升序排序, 限制返回5条.

(1) 创建Subreview (review的钱50w条数据)

```
db.review.aggregate([
  { $limit: 500000 }, // 限制取前 50 万条数据
  { $out: "Subreview" } // 将结果输出到 Subreview 集合中
]);
```

(2) 对评论的内容(text)建立全文索引

```
db.Subreview.createIndex({ text: "text" });
```

(3) 对useful建立升序索引

```
db.Subreview.createIndex({ useful: 1 });
```

(4) 查询评价的内容中包含delicious&useful>=50的评论, orderby review\_id asc, limit5

```
db.Subreview.find(
  { $text: { $search: "delicious" }, useful: { $gte: 50 } },
  { review_id: 1, text: 1, useful: 1 }
).sort({ review_id: 1 })
.limit(5);
```



在 business 表中, 查询距离商家 smkZUv\_IeYYj\_BA6-Po7oQ(business\_id) 2 公里以内的所有商家, 返回商家名字, 地址和星级, 按照星级降序排序, 限制返回 20 条.提示: 使用 2dsphere 建立索引、获取商家地理坐标、使用坐标进行查询

(1) 确保在 business 集合中对地理坐标 (latitude, longitude) 字段创建 2dsphere 索引。这个索引是用来进行地理空间查询的。

```
db.business.createIndex({ loc: "2dsphere" });
```

(2) 查询目标商家的 latitude 和 longitude 通过 \$geoNear 聚合操作符查询商家位置与给定目标位置的距离, 限制 2 公里以内的商家。

```
// 获取商家 smkZUv_IeYYj_BA6-Po7oQ 的地理坐标
const targetBusiness = db.business.findOne(
  { business_id: "smkZUv_IeYYj_BA6-Po7oQ" });
const targetCoordinates = targetBusiness.loc.coordinates; // [longitude, latitude]

// 执行 geoNear 查询
db.business.aggregate([
  {$geoNear: {
    near: { type: "Point", coordinates: targetCoordinates }, // 目标商家的经纬度
    distanceField: "distance", // 输出查询到的商家距离
    maxDistance: 2000, // 设置最大查询距离为 2 公里（单位：米）
    spherical: true, // 开启球面计算
  }},
  {$sort: { stars: -1 } },
  {$project: { // 只返回商家名称、地址和星级
    name: 1,
    address: 1,
    stars: 1,
    _id: 0}},
  {$limit: 20 // 限制返回 20 条记录
  }
]);
```

### 3.3 任务小结

在 MonGoDB 实验中, 个人认为困难的点在于各种函数的运用, 同时需要一点细心耐心, 这是因为经历了一个标点符号没写对导致半天没进展的经验。

## 4 NE04J

### 4.1 任务要求

使用图数据库 Neo4j 进行图数据的查询与分析。任务内容包括简单的节点和关系查询、条件过滤、数据聚合以及执行计划分析。通过这些实验理解图数据库在处理复杂关系查询中的优势，并能够根据需求构建高效的查询语句。

### 4.2 完成过程

启动华为云服务器，启动 XShell 连接服务器，导入所需数据库，启动 Neo4j，调用 yelp 数据库，开始任务。

#### 4.2.1 求和

统计评价过商户 id 为 nh\_kQ16QAoXWwqZ05MPfBQ 的用户的 name 以及 useful, funny, cool 三者的和，并按照该和降序排列。

实现语句如下：

```
MATCH
(u:UserNode)-[:Review]->(r:ReviewNode)-[:Reviewed]->(b:BusinessNode)
WHERE b.businessid = 'nh_kQ16QAoXWwqZ05MPfBQ'
WITH u.name AS user_name,
      toInteger(u.useful) AS useful,
      toInteger(u.funny) AS funny,
      toInteger(u.cool) AS cool,
      (toInteger(u.useful) + toInteger(u.funny) + toInteger(u.cool)) AS
total_compliments
RETURN user_name,useful,funny,cool, total_compliments
ORDER BY total_compliments DESC;
```

在本题中需要注意的是 useful, funny 和 cool 存储的时候存的并不是 int 类型，因此在求和的时候如果没有将其转换成 int 类型，得到的结果将会是三个数的排列组合。比如 useful=9, funny=3, cool=6，不转换类型得到的结果会是 936 而不是预期中的 18。

体会建立索引对查询带来的性能提升，但会导致插入，删除等操作变慢（需要额外维护索引代价）。

(1) 创建没有索引的数据库查询、执行查询，记录查询时间

```
MATCH
(u:UserNode)-[:Review]->(r:ReviewNode)-[:Reviewed]->(b:BusinessNode)
WHERE b.name = "Starbucks"
RETURN u.name AS user_name, b.name AS business_name;
```

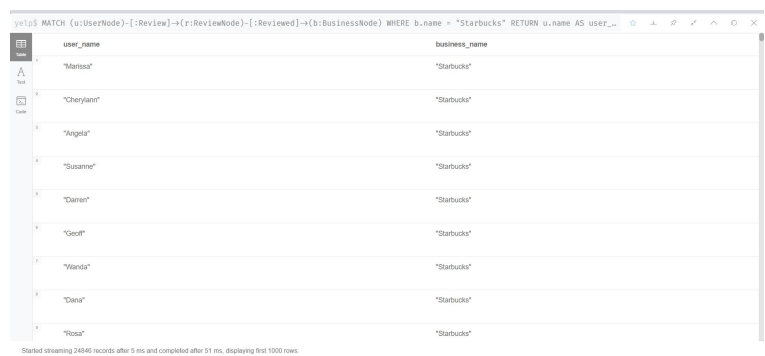


图 4.1 查询（无索引）



图 4.2 查询执行时间（无索引）

(2) 创建索引，user\_id， 进行查询， 比较两者查询时间

```
CREATE INDEX ON :UserNode(name);
CREATE INDEX ON :BusinessNode(name);
```

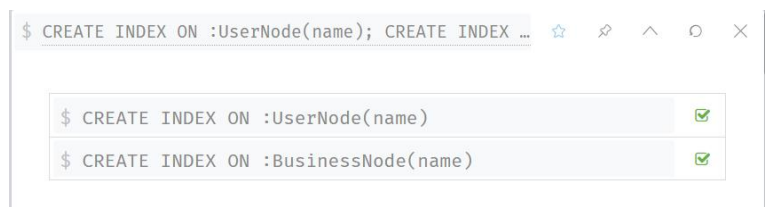


图 4.3 建立索引

PROFILE

MATCH

(u:UserNode)-[:Review]->(r:ReviewNode)-[:Reviewed]->(b:BusinessNode)

WHERE b.name = "Some Business"

RETURN u.name AS user\_name, b.name AS business\_name;

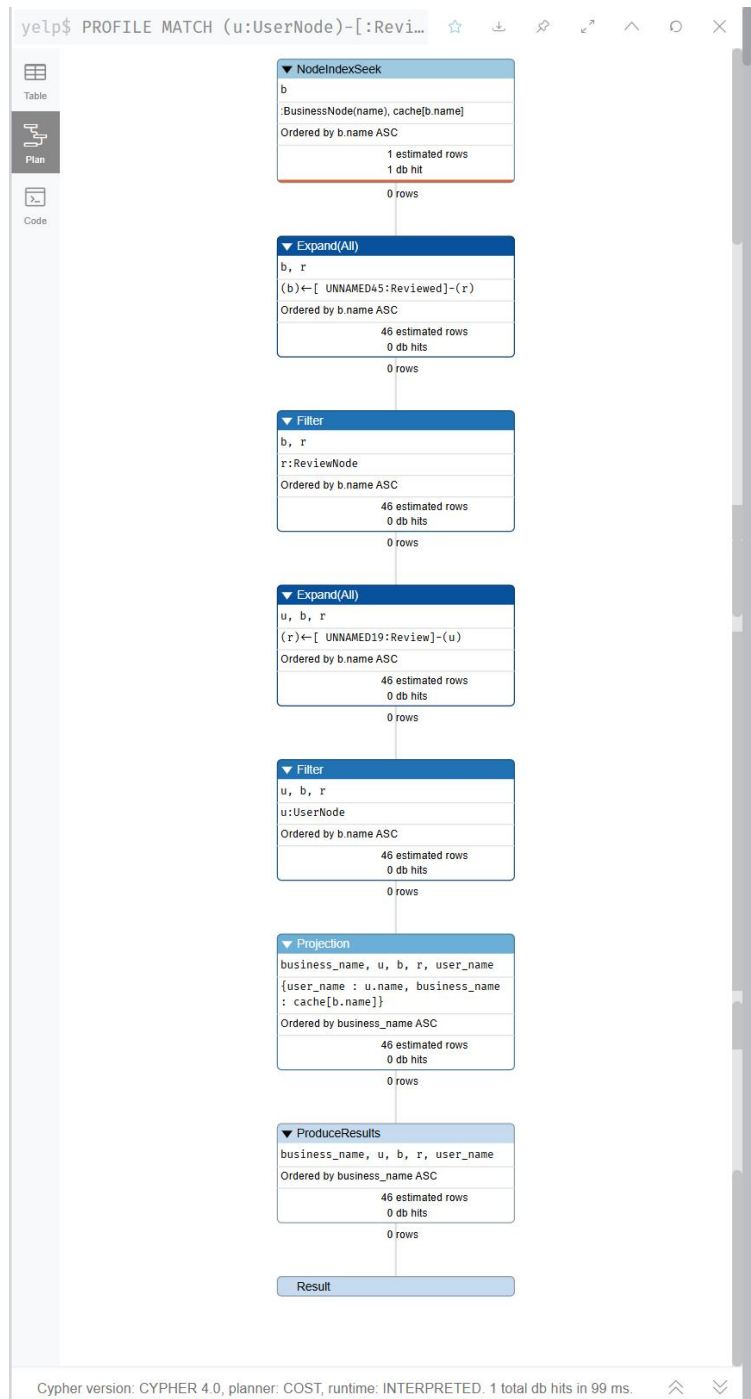


图 4.4 执行计划查询

```
CREATE (u:UserNode {name: "New User", fans: "100", useful: 50});
MATCH (u:UserNode {name: "New User"}) DELETE u;
```

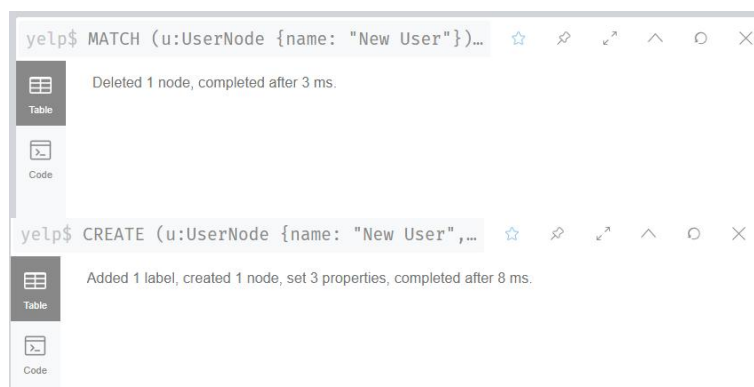


图 4.5 执行耗时

总结：

类别	查询	插入	删除
有索引	51	280	1186
无索引	99	8	3

表 4.1 索引对查询带来的性能提升总结

### 4.3 任务小结

在 Neo4J 实验中，最关键的还是各个图之间的联系，熟练掌握后可能会发现更简洁的查询方式，提高查询效率的同时也降低查询的开销。

## 5 多数据库交互应用实验

### 5.1 任务要求

结合 MySQL、MongoDB 和 Neo4j 的多数据库交互应用，展示如何在不同数据库之间进行数据迁移、去重与聚合。实验旨在提升多数据库环境下的协作能力，以及处理跨数据库的数据操作的技巧。

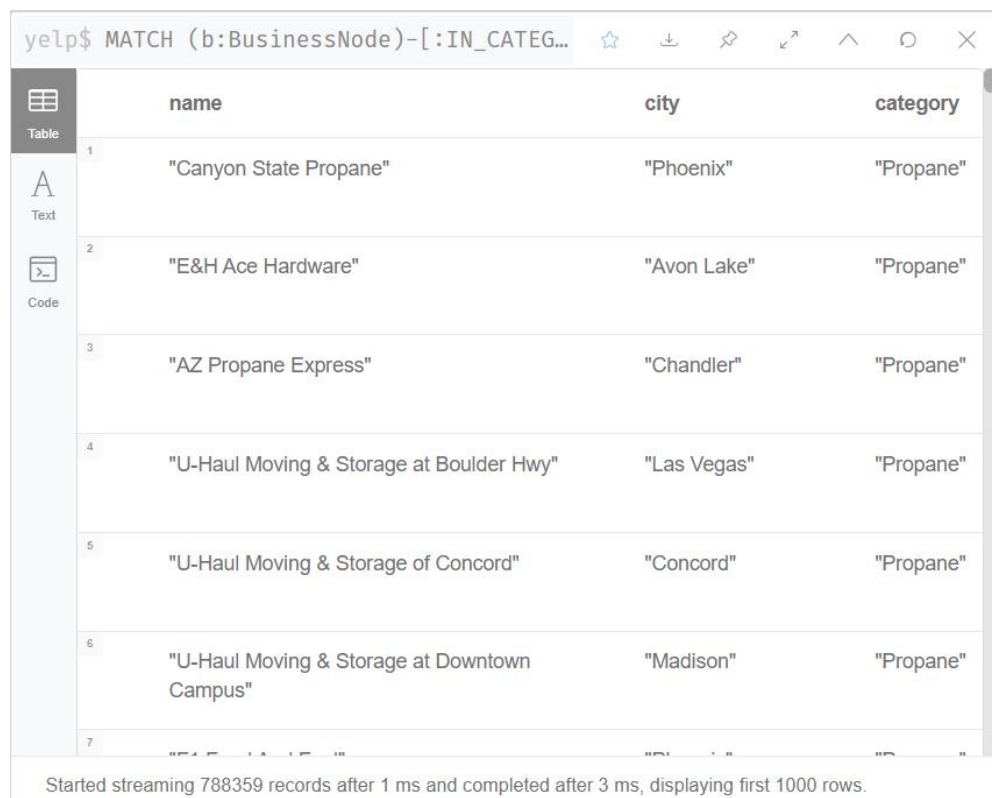
### 5.2 完成过程

在 Neo4j 中查找所有商家，要求返回商家的名字，所在城市、商铺类。

1. 将查找结果导入 MongoDB 中实现对数据的去重（提示：使用 aggregate，仅保留城市、商铺类型即可）
2. 将去重后的结果导入 Neo4j 中的新库 result 中，完成(City-[Has]->Category)图谱的构建。

(1) 在 Neo4j 中完成查找，查找语句如下：

```
MATCH (b:BusinessNode)-[:IN_CATEGORY]-(c:CategoryNode)
RETURN b.name AS name, b.city AS city, c.category AS category
```



The screenshot shows the Neo4j interface with a Cypher query executed. The results are displayed in a table view with columns: name, city, and category. The table contains 7 rows of data. Below the table, a status message indicates that 788359 records were streamed and the first 1000 rows are displayed.

	name	city	category
1	"Canyon State Propane"	"Phoenix"	"Propane"
2	"E&H Ace Hardware"	"Avon Lake"	"Propane"
3	"AZ Propane Express"	"Chandler"	"Propane"
4	"U-Haul Moving & Storage at Boulder Hwy"	"Las Vegas"	"Propane"
5	"U-Haul Moving & Storage of Concord"	"Concord"	"Propane"
6	"U-Haul Moving & Storage at Downtown Campus"	"Madison"	"Propane"
7	"U-Haul Moving & Storage at Downtown Campus"	"Madison"	"Propane"

Started streaming 788359 records after 1 ms and completed after 3 ms, displaying first 1000 rows.

图 5.1 Neo4j 查找商家结果（去重前）

- (2) 点击右上角下载图标，将数据导出成格式为 csv 的文件
- (3) 使用 WinSCP 将 task4\_3.csv 文件存到服务器上的 data\_for\_mongo 文件夹。
- (4) XShell 连接服务器，cd 到存放 task4\_3.csv 的目录中，执行以下语句，完成导入：

```
mongoimport --db yelp --collection task4 --type csv --file task4_2.csv
--headerline --authenticationDatabase admin -u admin -p (password)
```

- db : 目的 database 名称
- collection : 目的表名
- type : 导入文件的格式 (csv、json)
- file : 文件路径 (无前缀默认在当前路径中寻找该文件)
- headerline : 表示第一行为行头
- authenticationDatabase : 授权数据库名称
- u : 登入用户名
- p : 登入密码

```
root@sping02:~/data/data_for_mongo# mongoimport --db yelp --collection task4_3 --type csv --file task4_3.csv --headerline --authenticationDatabase admin -u admin -p @561432Rsp
connected to: mongodb://localhost/
2024-12-18T23:04:46.328+0800 [#####.....] yelp.task4_3 15.8MB/32.3MB (48.9%)
2024-12-18T23:04:49.329+0800 [#####.....] yelp.task4_3 30.8MB/32.3MB (95.4%)
2024-12-18T23:04:49.604+0800 [#####.....] yelp.task4_3 32.3MB/32.3MB (100.0%)
2024-12-18T23:04:49.604+0800 788359 document(s) imported successfully. 0 document(s) failed to import.
root@sping02:~/data/data_for_mongo#
```

图 5.2 Mongo 导入成功

- (5) 登入 MongoDB 检查导入是否正确

```
switched to db yelp
> db.task4_3.find()
{"_id" : ObjectId("6762e48ba2d0faf75d8febcd"), "name" : "Canyon State Propane", "city" : "Phoenix", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febce"), "name" : "E&H Ace Hardware", "city" : "Avon Lake", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febcb"), "name" : "AZ Propane Express", "city" : "Chandler", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febd0"), "name" : "U-Haul Moving & Storage at Boulder Hwy", "city" : "Las Vegas", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febd1"), "name" : "U-Haul Moving & Storage of Concord", "city" : "Concord", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febd2"), "name" : "U-Haul Moving & Storage at Downtown Campus", "city" : "Madison", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febd3"), "name" : "FI Food And Fuel", "city" : "Phoenix", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febd4"), "name" : "Trop Stop Gas & Car Wash", "city" : "Las Vegas", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febd5"), "name" : "U-Haul Moving & Storage of Warner Park", "city" : "Madison", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febd6"), "name" : "Speedee Mart #123", "city" : "Las Vegas", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febd7"), "name" : "Acorn Propane", "city" : "Phoenix", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febd8"), "name" : "U-Haul Moving & Storage At 24th & McDowell", "city" : "Phoenix", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febd9"), "name" : "Farrellgas", "city" : "Phoenix", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febdda"), "name" : "Vegas Propane", "city" : "North Las Vegas", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febddb"), "name" : "Walterville Propane", "city" : "Copley", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febddc"), "name" : "U-Haul Moving & Storage at Tropicana", "city" : "Las Vegas", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febdd"), "name" : "U-Haul Moving & Storage at Arrowhead Towne Center", "city" : "Peoria", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febbe"), "name" : "U-Haul at Elliot Rd", "city" : "Tempe", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febedf"), "name" : "Waldo's Nursery", "city" : "North Olmsted", "category" : "Propane"}
{"_id" : ObjectId("6762e48ba2d0faf75d8febe0"), "name" : "Panera Bread", "city" : "Warrensville Heights", "category" : "Salad"}
Type "it" for more
```

图 5.3 Mongo 导入正确

(6) 用导入的表 (task4\_3) 实现数据去重

```
db.task4_3.aggregate([
  {$group: { _id: { city: "$city", category: "$category" }, }},
  {$project: { _id: 0, city: "$_id.city", category: "$_id.category" }}
])
```

```
> db.task4_3.aggregate([
... {
...   $group: {
...     _id: { city: "$city", category: "$category" },
...   },
... },
... {
...   $project: {
...     _id: 0,
...     city: "$_id.city",
...     category: "$_id.category"
...   }
... }
... ])
{ "city" : "Aurora", "category" : "Hair Removal" }
{ "city" : "Savoy", "category" : "Automotive" }
{ "city" : "Parma Heights", "category" : "Restaurants" }
{ "city" : "Montreal", "category" : "Gift Shops" }
{ "city" : "Queen Creek", "category" : "Food" }
{ "city" : "Queen Creek", "category" : "Veterinarians" }
{ "city" : "New Kensington", "category" : "Mexican" }
{ "city" : "Scottsdale", "category" : "Taxidermy" }
{ "city" : "N E Las Vegas", "category" : "Hardware Stores" }
{ "city" : "Champaign", "category" : "Local Services" }
{ "city" : "Las Vegas", "category" : "Music & Video" }
{ "city" : "Oakville", "category" : "Dentists" }
{ "city" : "Middleton", "category" : "Financial Services" }
{ "city" : "Savoy", "category" : "Colleges & Universities" }
{ "city" : "Pepper Pike", "category" : "Makeup Artists" }
{ "city" : "Dorval", "category" : "Financial Services" }
{ "city" : "Kent", "category" : "Piercing" }
{ "city" : "Cuyahoga Falls", "category" : "Seafood" }
{ "city" : "Indian Trail", "category" : "Comfort Food" }
{ "city" : "McFarland", "category" : "Child Care & Day Care" }
Type "it" for more
>
```

图 5.4 Mongo 中实现 task4\_3 数据去重的结果

(7) 检查数据去重后的数据量，结果显示为 67536 条。

```
db.businesses.aggregate([
  {$group: { _id: { city: "$city", category: "$category" } }},
  {$count: "uniqueCount"}
])
```

```
> db.task4_3.aggregate([
... {
...   $group: {
...     _id: { city: "$city", category: "$category" }
...   },
... },
... {
...   $count: "uniqueCount"
... }
... ])
{ "uniqueCount" : 67536 }
```

图 5.5 实现数据去重后的数据量



(8) 聚合去重后的数据后完成导出

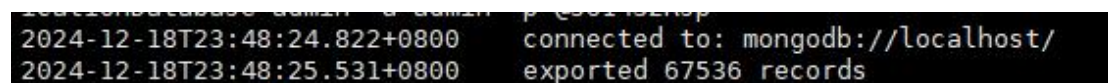
聚合数据语句如下：

```
db.task4_3.aggregate([
  {$group: {_id: { city: "$city", category: "$category" }}}},
  {$project: {_id: 0,city: "$_id.city",category: "$_id.category"}},
  {$out: "task4_3_after"}
])
```

导出语句如下：

```
mongoexport --db yelp --collection task4_3_after --type=csv
--out data/task4_3_after.csv --fields city,category
--authenticationDatabase admin -u admin -p (password)
```

--db	: database 名称
--collection	: 表名（这里使用聚合后的表名确保导出的是去重后的数据）
--type	: 导出文件的格式（csv、json）
--out	: 文件导出目的地址
--fields	: 导出元素
--authenticationDatabase	: 授权数据库名称
-u	: 登入用户名
-p	: 登入密码



2024-12-18T23:48:24.822+0800 connected to: mongodb://localhost/
2024-12-18T23:48:25.531+0800 exported 67536 records

图 5.6 MongoDB 导出成功（去重后）

(9) 使用 WinSCP 将 mongoDB 导出的 task4\_3\_after.csv 文件存到 neo-community-4.0.9/import 路径中

(10) 登入 Neo4j，将去重后的数据文件导入

```
LOAD CSV WITH HEADERS FROM 'file:///task4_3_after.csv' AS after
MERGE (c:CityNode {name: after.city})
MERGE (cat:CategoryNode {name: after.category})
MERGE (c)-[:HAS]->(cat)
```

'file:///task4\_3\_after.csv' 中的 file 是 neo4j 默认的协议，///是为了避免符号在执行过程中被误判为执行语句，task4\_3\_after.csv 是等待导入的文件



图 5.7 执行报错导入的元素值不可以为空

如图 5.7 所示，在执行（10）所示语句时遇到了一个问题，我们所导出的数据文件中存在一个 city 值为空的数据，而 Neo4j csv 导入无法处理该空值。因此，我选择将空值替换成 null 之后再导入。具体语句如下：

```
LOAD CSV WITH HEADERS FROM 'file:///task4_3_after.csv' AS row
MERGE (c:CityNode {name: COALESCE(row.city, 'null')})
MERGE (cat:CategoryNode {name: COALESCE(row.category, 'null')})
MERGE (c)-[:HAS]->(cat)
```

使用 COALESCE 将遇到的空值补充为 null，以便顺利完成导入。



图 5.8 修改后导入成功

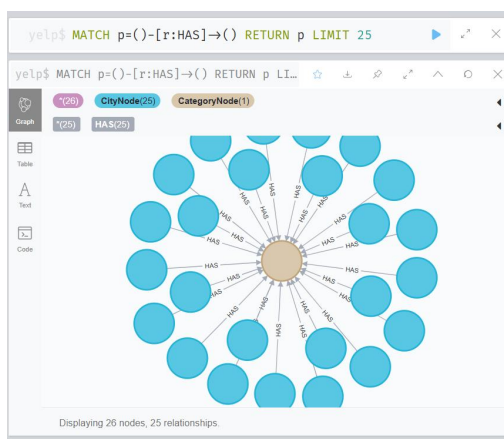


图 5.9 HAS 关系图谱

## 5.3 任务小结

在多数数据交互实验中，最关键的还是各个数据库、服务器与本机之间的联系，熟练掌握三者的关系后，导入导出的语句和操作就会变得简洁明了。

## 6 课程总结

在本次课程实践中，我们完成了多个数据库操作和查询任务，涵盖了关系型数据库（MySQL）、文档型数据库（MongoDB）、图数据库（Neo4j）的使用与实践以及多数据库交互应用实验。

在 MySQL 实验中，我熟练掌握 JSON\_TABLE() 等高级功能，将嵌套 JSON 数据转换为关系型结构，并完成复杂的多条件筛选与聚合操作，也探索了索引的优化作用，提升了大数据查询的效率。

在 MongoDB 实验中，我实现了多表数据导入、去重、索引创建等操作。另外，也学习及使用 \$lookup 和 \$aggregate 处理跨集合查询，展示了 MongoDB 在大数据场景下的灵活性。

在 Neo4j 实验中，我学会了运用图数据库的关系查询能力完成多层关系分析，包括用户间的朋友关系和商户的多级分类。除此之外，我也学会了利用 WITH 和聚合函数，探索了图数据的深度查询与性能优化。

在本次实验中，我个人收获最大的是学到了工具选择的重要性。同数据库在特定场景下各有优势。MySQL 在结构化数据处理上表现优异，而 MongoDB 对半结构化数据更加友好，Neo4j 则在复杂关系的分析中具有显著优势。除此之外，我也学会通过索引创建和执行计划分析，深刻理解了数据库性能优化对大数据查询的重要性。

我希望未来可以尝试在一个应用场景中综合使用 MySQL、MongoDB 和 Neo4j，解决更复杂的数据存储和分析问题。通过本次课程实践，我对三种主流数据库的应用能力得到了提升，为未来的实际项目开发打下了坚实基础。