



华中科技大学

大数据管理概论实验报告

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

分数	
教师签名	

2024 年 12 月 29 日

教师评分页

子目标	子目标评分
1	
2	
3	
总分	

目 录

1 课程任务概述	1
1.1 实验软件和数据	1
1.2 软件安装	1
1.3 实验任务	1
2 MySQL for JSON 实验	3
2.1 任务要求	3
2.2 完成过程	3
2.3 任务小结	9
3 MongoDB 实验	10
3.1 任务要求	10
3.2 完成过程	10
3.3 任务小结	14
4 Neo4j 实验	16
4.1 任务要求	16
4.2 完成过程	16
4.3 任务小结	21
5 课程总结	22
5.1 主要工作	22
5.2 心得体会	22
5.3 未来展望	22

1 课程任务概述

1.1 实验软件和数据

1. 数据库：图数据库 Neo4j，关系数据库 MySQL，文档数据库 MongoDB。
2. 编程语言：java
3. 数据集：[Yelp Dataset](#)

1.2 软件安装

参照华为平台部署手册及实验指导书。

1.3 实验任务

本实验围绕大数据管理中的核心内容展开，包括关系数据库（MySQL）、文档数据库（MongoDB）、图数据库（Neo4j）的基本操作、性能优化、以及多数数据库交互和并发控制实验。

1.3.1 任务 1: MySQL for JSON 实验

a) JSON 基本查询

包括简单查询、简直操作、类型检测、复杂条件查询、执行计划分析等相关知识，要求会使用基础 JSON 函数。

b) JSON 增删改

通过 JSON_SET 增加键值对，新增和更新 JSON 数据。插入新记录后进行 JSON 修改（如删除特定键）。

b) JSON 聚合

理解聚合操作的概念，运用 JSON_ARRAYAGG 函数聚合内容，并返回 JSON 格式数据，实现按洲聚合、用户点评聚合等操作。

c) JSON 实用函数的使用

使用 JSON_OVERLAPS 判断是否满足条件。通过 JSON_ARRAY 生成数组结构数据。对两个 JSON 文档合并，保留重复键值对。使用 JSON_TABLE 将 JSON 数据导出为表格形式。

1.3.2 任务 2: MongoDB 实验

a) 条件查询与执行计划

通过基本条件过滤（如 skip、limit、\$in 条件查询），多条件组合查询（如 and、

or 及范围过滤操作），并结合 explain 进行执行计划分析和优化查询。

b) 聚合与索引

掌握统计分析的概念，按洲统计商店数量，按评论内容关键词查询；运用地理位置索引（如 2D Sphere 索引）查询附近商家；创建全文索引和其他索引以提升查询效率。

c) MapReduce 的使用

使用 MapReduce 对商家评分进行统计（如平均分、总分）。

1.3.3 任务 3：Neo4j 实验

通过基本节点查询获取用户或商家节点的基本信息，掌握多关系查询（如用户评价的商家、商家类别等），结合 where 和 toInteger 进行类型转换和条件过滤，利用 PROFILE 分析查询效率并进行优化，同时创建索引后测试其对插入和删除性能的影响，全面掌握 Neo4j 数据库的使用技巧和优化方法。

1.3.4 任务 4：多数据库交互应用实验

Neo4j 数据查询：找出被超过五个用户评论的商家。

MongoDB 数据导入：将查询结果导入 MongoDB 并基于数据进行处理。

MySQL 与 Neo4j 联动：使用 MySQL 聚合数据，在 Neo4j 中展示关系。

1.3.5 任务 5：MVCC 多版本并发控制对比实验

对比 MySQL 和 MongoDB 的多版本并发控制策略及性能差异。

模拟不同的并发读写场景，分析事务冲突解决机制和性能指标。

完成的实验任务有任务 1 的 1-13 题，任务 2 的 1-14 题以及任务 3 的 1-15、18 题，共计 80 颗星。未选择任务 4 和任务 5，故报告中无对应章节内容。

2 MySQL for JSON 实验

2.1 任务要求

a) JSON 基本查询

简单查询：按条件（如 `state = CA`）和字段提取信息（如商户评分）。

键值操作：使用 `JSON_KEYS`、`JSON_LENGTH` 提取键值对信息。

类型检测：查询 JSON 数据类型（如 `JSON_TYPE`）。

复杂条件查询：通过嵌套 JSON 条件（如 `HasTV=True`）并排序筛选记录。

执行计划分析：对 JSON 查询加索引并与 MongoDB 查询效率对比。

b) JSON 增删改

新增和更新 JSON 数据：通过 `JSON_SET` 增加键值对。

插入与删除操作：插入新记录后进行 JSON 修改（如删除特定键）。

c) JSON 聚合

按州聚合：返回州-城市统计的 JSON 格式数据。

用户点评聚合：通过 `JSON_ARRAYAGG` 聚合点评内容。

d) JSON 实用函数的使用

条件匹配：使用 `JSON_OVERLAPS` 判断是否满足条件。

数组操作：通过 `JSON_ARRAY` 生成数组结构数据。

文档合并：对两个 JSON 文档合并，保留重复键值对。

JSON 表格化：使用 `JSON_TABLE` 将 JSON 数据导出为表格形式。

2.2 完成过程

2.2.1 执行计划及索引优化（T5）

1. MySQL 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。

```
explain format=json
select *
from user
where user_info->'$.cool' > 200;
```

执行计划如图 2.1 所示。

实际执行一次该查询,查询语句为：

```
select *
from user
where user_info->'$.cool' > 200;
```

最终查询到 26981 个结果，用时 34.13 秒（图 2.2）。

```
{
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "489778.94"
    },
    "table": {
      "table_name": "user",
      "access_type": "ALL",
      "rows_examined_per_scan": 1662608,
      "rows_produced_per_join": 1662608,
      "filtered": "100.00",
      "cost_info": {
        "read_cost": "323518.14",
        "eval_cost": "166260.80",
        "prefix_cost": "489778.94",
        "data_read_per_join": "190M"
      },
      "used_columns": [
        "user_id",
        "user_info"
      ],
      "attached_condition": "(json_extract(`test`.`user`.`user_info`, '$.cool') > 200)"
    }
  }
}
```

图 2.1 MySQL 执行计划

```
-----+
26981 rows in set (34.13 sec)

mysql> █
ssh://root@113.44.87.186:22 SSH2
```

图 2.2 MySQL 用时

2. MongoDB 查询

执行计划：

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

```
> db.user.find({ "compliment_cool": { $gt: 200 } }).explain("executionStats");
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "yelp.user",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "compliment_cool" : {
        "$gt" : 200
      }
    },
    "winningPlan" : {
      "stage" : "COLLSCAN",
      "filter" : {
        "compliment_cool" : {
          "$gt" : 200
        }
      },
      "direction" : "forward"
    },
    "rejectedPlans" : [ ]
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 3872,
    "executionTimeMillis" : 10450,
    "totalKeysExamined" : 0,
    "totalDocsExamined" : 1637141,
    "executionStages" : {
      "stage" : "COLLSCAN",
      "filter" : {
        "compliment_cool" : {
          "$gt" : 200
        }
      },
      "nReturned" : 3872,
      "executionTimeMillisEstimate" : 8123,
      "works" : 1637143,
      "advanced" : 3872,
      "needTime" : 1633270,
      "needYield" : 0,
      "saveState" : 1690,
      "restoreState" : 1690,
      "isEOF" : 1,
      "direction" : "forward",
      "docsExamined" : 1637141
    }
  },
  "serverInfo" : {
    "host" : "ecs-db8e-frq",
    "port" : 27017,
    "version" : "4.4.29",
    "gitVersion" : "f4dda329a99811c707eb06d05ad023599f9be263"
  },
  "ok" : 1
}
```

图 2.3 MongoDB 执行计划

3. MongoDB 与 MySQL 查询效率对比

(1) MySQL 查询分析

- access_type: ALL，说明 MySQL 执行的是全表扫描，因为 user_info->'\$.cool' 是 JSON 提取操作，MySQL 无法直接利用索引。
- rows_examined_per_scan: 1662608 扫描了整个表中的 1662608 行。
- cost_info: read_cost: 323518.14
eval_cost: 166260.80

没有索引，由于查询基于 JSON 提取，无法直接利用索引优化查询。查询的开销很高，因为需要逐行扫描表并解析 JSON 数据。存在性能瓶颈，表越大，查询时间越长。

(2) MongoDB 查询分析

- stage: COLLSCAN 执行了全表扫描（COLLSCAN），没有索引支持。
- executionTimeMillis: 10450 毫秒，查询耗时约 10.45 秒。
- docsExamined: 1637141，扫描了 1637141 行文档。
- nReturned: 3872，查询返回了 3872 条文档。

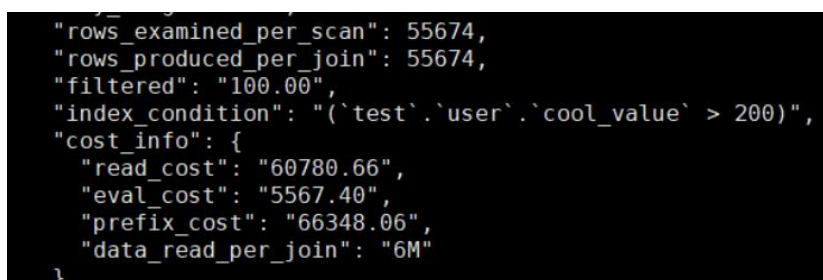
JSON 结构的原生支持 MongoDB 不需要额外的字段解析，直接查询嵌套字段，故用时相对 MySQL 大幅减少。

4. MySQL 索引创建

最后，在 MySQL 中为 user_info 的字段加索引来优化提高查询效率：

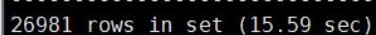
```
ALTER TABLE user
ADD COLUMN cool_value INT AS (JSON_UNQUOTE(user_info->'$.cool'))
STORED;
CREATE INDEX idx_cool_value ON user (cool_value);
```

优化后的执行计划、执行时间如图 2.4、图 2.5。



```
"rows_examined_per_scan": 55674,
"rows_produced_per_join": 55674,
"filtered": "100.00",
"index_condition": "(`test`.`user`.`cool_value` > 200)",
"cost_info": {
  "read_cost": "60780.66",
  "eval_cost": "5567.40",
  "prefix_cost": "66348.06",
  "data_read_per_join": "6M"
}
```

图 2.4 优化后执行计划



```
-----
26981 rows in set (15.59 sec)
```

图 2.5 优化后用时

MySQL 添加索引前后查询效率对比如表 2.1。

表 2.1 MySQL 添加索引前后查询效率对比

对比维度	优化前（未加索引）	优化后（加索引）
访问类型	ALL - 全表扫描	range - 范围扫描
查询成本	489,778.94	66,348.06
索引	无	idx_cool_value
扫描行数	1,662,608	55,674
读取成本	323,518.14	60,780.66
数据读取量	190M	6M
实际用时	34.13s	15.59s
索引条件	无	cool_value > 200
优缺点	全表扫描导致高成本和低效率，处理大量无关数据。	索引加速，显著减少扫描行数和读取数据量，性能大幅提升。

2.2.2 JSON_ARRAY (T11)

任务：在 user 表中，查询 useful 大于 1000 的用户，返回非 json 数组下他们的 name, funny, cool, useful 以及按 json 数组形式表示的 funny, useful, cool，以及三者的和，限制 10 条。

首先，使用 JSON_EXTRACT() 提取 user_info 字段中的 name、funny、cool 和 useful 值，通过 JSON_UNQUOTE() 去掉 name 字段值的引号：

SELECT

JSON_UNQUOTE(JSON_EXTRACT(user_info, '\$.name')) AS name,

JSON_EXTRACT(user_info, '\$.funny') AS funny,

JSON_EXTRACT(user_info, '\$.cool') AS cool,

JSON_EXTRACT(user_info, '\$.useful') AS useful,

使用 JSON_ARRAY() 将 funny、useful 和 cool 的值组合成一 JSON 数组：

JSON_ARRAY(

JSON_EXTRACT(user_info, '\$.funny'),

JSON_EXTRACT(user_info, '\$.useful'),

JSON_EXTRACT(user_info, '\$.cool')

) AS funny_useful_cool,

计算这三个值的总和：

(JSON_EXTRACT(user_info, '\$.funny') + JSON_EXTRACT(user_info, '\$.useful') + JSON_EXTRACT(user_info, '\$.cool')) AS total

查询的筛选条件为 useful 字段值大于 1000，最终返回的结果限制为前 10

条记录:

```
FROM user
WHERE JSON_EXTRACT(user_info, '$.useful') > 1000
LIMIT 10;
得到结果如下图:
```

name	funny	cool	useful	funny_cool_useful	sum
Chantelle	900	1949	2751	[900, 2751, 1949]	5600
Harald	173096	199878	205765	[173096, 205765, 199878]	578739
Jen	3565	4529	4981	[3565, 4981, 4529]	13075
Amber	302	468	1153	[302, 1153, 468]	1923
Susie	2042	4450	5393	[2042, 5393, 4450]	11885
Akansha	514	678	1335	[514, 1335, 678]	2527
Christine	6166	7181	10765	[6166, 10765, 7181]	24112
Jim	2739	3839	4857	[2739, 4857, 3839]	11435
Akemi	475	724	1369	[475, 1369, 724]	2568
Debbie	927	1229	1632	[927, 1632, 1229]	3788

10 rows in set (0.00 sec)

图 2.6 T11 查询结果

2.2.3 JSON_TABLE 导出 (T13)

任务: 查询被评论数前 3 的商户,使用 JSON_TABLE()可以将 json 型数据转换为关系型表格,请使用 JSON_TABLE()将商户的 name, HasTV, 和所有的 attributes(不考虑顺序,一个属性就对应一行,对每个商户,从 1 开始对这些时段递增编号),最后按商户名字升序排序。

首先,需要查询 business 表中的商户基本信息,包括商户名称、是否有电视(HasTV)等属性,同时提取商户的属性数据。为了按商户的评论数排序,在查询中添加了排序条件,按照评论数降序排列,并限制返回前三个商户。

```
SELECT
    business_id,
    JSON_UNQUOTE(JSON_EXTRACT(business_info, '$.name')) AS name,
    JSON_UNQUOTE(JSON_EXTRACT(business_info, '$.attributes.HasTV'))
AS has_tv,
    JSON_EXTRACT(business_info, '$.attributes') AS attributes
FROM
    business
ORDER BY
    CAST(JSON_EXTRACT(business_info, '$.review_count') AS UNSIGNED)
DESC
LIMIT 3;
```

接下来，使用 JSON_TABLE 函数展开商户的属性数据（attributes）。该函数可以将 attributes 中的每个字段（如 WiFi、HasTV）转换为多行数据，每行代表一个属性的键值对。在展开属性时，将每个属性的键（attribute_key）和对应的值（attribute_value）提取出来，便于后续操作和分析，如下图所示。

```
JSON_TABLE( -- 使用 JSON_TABLE 函数展开商户的属性数据 (attributes)
  b.attributes, -- 传入刚才提取的商户属性
  '$.*' COLUMNS ( -- 遍历 attributes 中的每个字段 (即属性)
    attribute_key VARCHAR(255) PATH '$.key', -- 将属性的键提取为 attribute_key 列
    attribute_value VARCHAR(255) PATH '$.value' -- 将属性的值提取为 attribute_value 列
  )
)
```

图 2.7 JSON_TABLE

在查询的过程中，内层查询和外层查询结合起来。内层查询负责提取商户的基本信息和属性数据，而外层查询则使用 JSON_TABLE 展开商户的属性字段。通过这种方式，可以得到每个商户的属性信息，并根据商户的名称和属性键进行排序。为了为每个商户的属性生成递增编号，使用了 ROW_NUMBER() 函数，并将商户名称作为分区依据，确保每个商户的属性都有一个独立的编号，并升序排序。

```
SELECT
  b.name,
  b.has_tv,
  a.attribute_key, -- 选择每个商户属性的键
  a.attribute_value, -- 选择每个商户属性的值
  ROW_NUMBER() OVER (PARTITION BY b.name ORDER BY a.attribute_key) AS attribute_index
```

图 2.8 总查询

部分查询结果如下图所示。

name	has_tv	attribute_key	attribute_value	attribute_index
Acme Oyster House	True	NULL	NULL	2
Acme Oyster House	True	NULL	NULL	1
Acme Oyster House	True	NULL	NULL	24
Acme Oyster House	True	NULL	NULL	23
Acme Oyster House	True	NULL	NULL	22
Acme Oyster House	True	NULL	NULL	21
Acme Oyster House	True	NULL	NULL	20
Acme Oyster House	True	NULL	NULL	19
Acme Oyster House	True	NULL	NULL	18
Acme Oyster House	True	NULL	NULL	17
Acme Oyster House	True	NULL	NULL	16
Acme Oyster House	True	NULL	NULL	15
Acme Oyster House	True	NULL	NULL	14
Acme Oyster House	True	NULL	NULL	12
Acme Oyster House	True	NULL	NULL	13
Acme Oyster House	True	NULL	NULL	3
Acme Oyster House	True	NULL	NULL	4
Acme Oyster House	True	NULL	NULL	5
Acme Oyster House	True	NULL	NULL	6
Acme Oyster House	True	NULL	NULL	7
Acme Oyster House	True	NULL	NULL	8
Acme Oyster House	True	NULL	NULL	9
Acme Oyster House	True	NULL	NULL	10
Acme Oyster House	True	NULL	NULL	11
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	13
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	14
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	15
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	16
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	17
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	18
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	19
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	20
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	21
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	22
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	23
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	24
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	11
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	12
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	1
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	2
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	3
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	4
Hattie B's Hot Chicken - Nashville	True	NULL	NULL	5

图 2.9 T13 查询结果

2.3 任务小结

在 MySQL for JSON 实验中，我深入研究了 JSON 数据在关系数据库中的处理和优化方法。

实验中，我面临的一个困难是如何在没有直接索引支持的情况下，优化基于 JSON 字段的查询。为了解决这个问题，我学习了如何通过创建虚拟列和索引来提高查询效率。例如，在处理 `user_info->'$.cool' > 200` 的查询时，我添加了 `cool_value` 作为虚拟列，并为其创建了索引，显著提高了查询性能。

此外，我还学习了如何使用 `JSON_TABLE` 函数将 JSON 数据导出为表格形式。在尝试将商户的属性数据转换为关系型表格时，我最初对如何正确使用 `JSON_TABLE` 函数感到困惑。通过查阅文档和多次尝试，我逐渐理解了如何将 JSON 数据中的每个属性转换为表格中的一行，并且学会了如何为每个商户的属性生成递增编号。这个过程不仅提高了我对 `JSON_TABLE` 函数的掌握，也加深了我对 JSON 数据操作的理解。

通过本实验，我对 JSON 数据在 MySQL 中的存储、查询和优化有了全面的认识。

3 MongoDB 实验

3.1 任务要求

a) 条件查询与执行计划

基本条件过滤: skip、limit、\$in 条件查询。

多条件组合查询: and、or 及范围过滤操作。

执行计划分析: 通过 explain 优化查询。

b) 聚合与索引

统计分析: 按州统计商店数量, 按评论内容关键词查询。

地理位置索引: 通过 2D Sphere 索引查询附近商家。

索引优化: 创建全文索引和其他索引以提升查询效率。

c) MapReduce 的使用

使用 MapReduce 对商家评分进行统计 (如平均分、总分)。

3.2 完成过程

3.2.1 子集合的创建、创建索引、索引查询 (T11)

任务: 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。

创建子集合:

```
db.review.aggregate([
    { $limit: 500000 }, // 取前五十万条数据
    { $out: "Subreview" } // 输出到新集合 Subreview
])
```

创建索引:

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

```
> db.Subreview.createIndex({ text: "text" })
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

图 3.1 索引 1

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



```
> db.Subreview.createIndex({ useful: 1 })
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}
```

图 3.2 索引 2

查询内容包含关键词并排序：

```
db.Subreview.find(
  {
    $text: { $search: "delicious" }, // 全文搜索关键词
    useful: { $gte: 50 } // useful 字段大于等于 50
  },
  { review_id: 1, text: 1, useful: 1 } // 返回指定字段
).sort({ review_id: 1 }).limit(5); // 限制返回 5 条
```

查询结果如下图。

```
mation:\ndonnasbakerycafe.com" }
{ "_id" : ObjectId("600d7ea5f5e9bd91d7c36643"), "review_id" : "CjP8CnbNLFau0iIVRfPudw", "useful" : 71, "text" : "I'm so happy The Corndog Company is here in Vegas now! Hubby and I have been fans of this truck for several years. When we are in St George, UT we always look to see where the truck will be. Now we can check their schedule to see where it will be here in Vegas! The Corndog Company LV makes its way all around town. I have been going to the Food Truck Party near Total Wine on Centennial Center and when I noticed on social media that this truck would be there I had to stop by. They had a long line and you need to be prepared to wait 5-10 minutes after ordering because each corndog is hand-dipped to order and fried to perfection. The corndogs come in a couple sizes and range in price from $4-$6. Not every time, but sometimes the truck does have fried cheese sticks and you'll want to try these if they have them when you visit. The fried cheese sticks have the same delicious crispy batter as the regular corndogs, but are filled with either pepper jack or american cheeses. The corndogs themselves are really tasty. I always order mine with honey and suggest you do also. The honey is drizzled on after the corndog comes out of the fryer. I also add ketchup and mustard that's usually in coolers right outside the truck and you can add as much as you want. The people working the truck are very friendly. This truck is worth checking out and you can see their schedule on social media and sometimes they have daily specials posted there also." }
{ "_id" : ObjectId("600d7eb4f5e9bd91d7c905c7"), "review_id" : "EluNPWga0_kYeVHj9W7pdw", "useful" : 98, "text" : "After reading the Yelp reviews for the Yardbird Southern Table and Bar I decided to make a reservation. The reviews were spot on. This is a great southern fare restaurant with a warm friendly staff. \n\nOur waiter recommended the Chicken-N-Watermelon-Waffles. ($36.00) my husband and I added buttermilk biscuits ($7.00) and crispy Brussel Sprouts ($10.00) to our order. The fried chicken is served with spiced watermelon and a Vermont cheddar cheese waffle. They kick it up a notch by adding a side of honey hot sauce and bourbon maple syrup. The flavor combination of this meal is incredibly delicious. The chicken is 100 percent natural with no steroids or hormones and brined for 27 hours before serving. I loved the biscuits and the brussel sprouts did not disappoint." }
>
```

图 3.3 T11 查询结果

3.2.2 聚合运算 (T12)

任务：在 Subreview 集合中统计评价中，找出 useful 大于 50 的所有评论，并返回 business_id，以及平均打星，按照商家 id 排序，限制返回 20 条记录。

聚合运算是 MongoDB 中用于对数据进行复杂计算和转化的强大工具。它允许我们按指定的条件对数据进行分组、筛选、排序、聚合等操作。在聚合操作中，常用的阶段有：

- **\$match**：用于筛选符合条件的文档，相当于 SQL 中的 WHERE 子句。

- **\$group**: 按指定字段对文档进行分组, 并可以进行统计运算 (如求和、平均数等), 相当于 SQL 中的 GROUP BY 子句。
- **\$avg**: 用于计算指定字段的平均值。
- **\$sort**: 对查询结果进行排序, 相当于 SQL 中的 ORDER BY。
- **\$limit**: 限制返回的文档数量。

聚合运算实现:

\$match: 首先, 使用 \$match 阶段筛选出 useful 大于 50 的评论。

\$group: 然后, 使用 \$group 阶段按照 business_id 对文档进行分组, 并计算每个分组的平均打星数。使用了 \$avg 聚合运算符来计算每个商家的平均打星。

\$sort: 接下来, 使用 \$sort 阶段按商家的 business_id 对结果进行升序排序。

\$limit: 最后, 使用 \$limit 阶段限制返回的结果为 20 条记录。

```
db.Subreview.aggregate([
  { $match: { useful: { $gt: 50 } } },
  { $group: { _id: "$business_id",          // 按 business_id 分组
              avg_stars: { $avg: "$stars" } } },
  { $sort: { _id: 1 } },
  { $limit: 20 }
])
```

查询结果如下图。

```
{ "_id" : "0crFnrlWPlldjkmVOMWPmTQ", "avg_stars" : 5 }
{ "_id" : "19hxp08h1K9wvnMAlIoJqw", "avg_stars" : 4 }
{ "_id" : "19r0pwKG8aeE0_hjDtDouw", "avg_stars" : 4 }
{ "_id" : "1XWpUatzGk0Cgn6C6VDaWg", "avg_stars" : 5 }
{ "_id" : "3KG13qMCNegGT9oIxWAAWw", "avg_stars" : 3 }
{ "_id" : "3fdtp-bzoE4ZgTakkeEBzQ", "avg_stars" : 4 }
{ "_id" : "3saG51hM2M14G5f-QwhGhg", "avg_stars" : 4 }
{ "_id" : "4H_MjEd3amnThjyVCSTcUQ", "avg_stars" : 4 }
{ "_id" : "4X9tVVQHlIewliXqaancfQA", "avg_stars" : 2 }
{ "_id" : "4buRElPC3-ka-s2q55VVFw", "avg_stars" : 1 }
{ "_id" : "4tUaY3gPYm2KS1L6lTEjvQ", "avg_stars" : 1 }
{ "_id" : "5dy80iWJty9AuxBcBzeBZQ", "avg_stars" : 3 }
{ "_id" : "5mTOBaAdm2gf0_jR8PVkTA", "avg_stars" : 4 }
{ "_id" : "5qao1bKTbJ0-knt0gxg0Pg", "avg_stars" : 4 }
{ "_id" : "65GMvjrrsvyR9pXXAys7qg", "avg_stars" : 4 }
{ "_id" : "74C1z6cqEdvJVgETTcEodw", "avg_stars" : 4 }
{ "_id" : "77h11eWv6HKJAgOjLx8G4w", "avg_stars" : 5 }
{ "_id" : "7uaVNa67K5qwk9ZrbHeZ0Q", "avg_stars" : 3 }
{ "_id" : "89F4IA9HIn6RDER09C4QZw", "avg_stars" : 1 }
{ "_id" : "8E2Uc0Mn4cn_yEY0xFv-9A", "avg_stars" : 2 }
>
```

图 3.4 T12 查询结果

3.2.3 地图索引 (T13)

任务: 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. 先查看 business 的结构：

```
> db.business.find().limit(10);
{ "_id" : ObjectId("6016c6b4af81085b0f2183c1"), "business_id" : "1SWh84yJXfytoVILX0AQ", "name" : "Arizona Biltmore Golf Club", "address" : "2818 E Camino Acequia Drive", "city" : "Phoenix", "state" : "AZ", "postal_code" : "85016", "latitude" : 33.5221425, "longitude" : -112.0184807, "stars" : 3, "review_count" : 5, "is_open" : 0, "attributes" : { "GoodForKids" : "False" }, "categories" : [ "Golf", "Active Life" ], "hours" : null, "loc" : { "type" : "Point", "coordinates" : [ -112.0184807, 33.5221425 ] } }
```

图 3.5 business 结构

由上图可知，商家地址 loc 为：

```
"loc" : {
  "type" : "Point",
  "coordinates" : [ -112.0184807, 33.5221425 ]
}
```

这里的 loc 对象包含两个属性：

type: 这指定了几何对象的类型。在这个例子中，type 是 "Point"，意味着这是一个点的地理位置。

coordinates: 这是一个数组，包含了两个数字，分别代表经度和纬度。在这个例子中，coordinates 数组的值是 [-112.0184807, 33.5221425]。

2. 为 location 字段创建 2dsphere 索引：

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

获取目标商家的地理坐标：

```
db.business.find(
  { business_id: "smkZUv_IeYYj_BA6-Po7oQ" },
  { "loc.coordinates": 1, _id: 0 }
)
```

创建好后输出如下图。

```
> db.business.createIndex({ loc: "2dsphere" });
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}
> db.business.find(
...   { business_id: "smkZUv_IeYYj_BA6-Po7oQ" },
...   { "loc.coordinates": 1, _id: 0 }
... )
{ "loc" : { "coordinates" : [ -79.5617757296, 43.7675451093 ] } }
>
```

图 3.6 创建索引

3. 编写代码

使用 \$geoNear 阶段来查找与指定地理位置（经纬度为[-79.5617757296,

43.7675451093]) 距离 2000 米以内的文档，并将结果按照距离这个点的远近排序在 `dist.calculated` 字段中。接着使用 `$sort` 阶段按星级 `stars` 降序排列这些结果，然后通过 `$project` 阶段选择输出文档中的 `name`（名称）、`address`（地址）和 `stars`（星级）字段，同时排除 `_id` 字段。`$limit` 阶段限制结果集只包含前 20 个文档。整个查询的目的是获取指定地点附近、评价最高的 20 个商业实体的信息。

具体代码如下：

```
db.business.aggregate([
  { $geoNear:
    {
      near: { type: "Point", coordinates: [-79.5617757296, 43.7675451093] },
      distanceField: "dist.calculated",
      maxDistance: 2000,
      spherical: true, key: "loc"
    }
  },
  { $sort: { stars: -1 } },
  { $project: { name: 1, address: 1, stars: 1, _id: 0 } },
  { $limit: 20 }
])
```

查询结果如下图。

```
{ db.business.aggregate([ { $geoNear: { near: { type: "Point", coordinates: [-79.5617757296, 43.7675451093] }, distanceField: "dist.calculated", maxDistance: 2000, spherical: true, key: "loc" } }, { $sort: { stars: -1 } }, { $project: { name: 1, address: 1, stars: 1, _id: 0 } }, { $limit: 20 } ])
{ "name": "Auto Trax", "address": "5235 Steeles Avenue W, Unit 2 & 3", "stars": 5 }
{ "name": "Revival Bridal Boutique", "address": "7007 Islington Avenue, Unit 3", "stars": 5 }
{ "name": "Acores Rotisserie Chicken", "address": "7611 Pine Valley Drive, Unit 27", "stars": 5 }
{ "name": "Raid Fighting Systems", "address": "3625 Weston Road", "stars": 5 }
{ "name": "La Fonte Trattoria", "address": "4370 Steeles Avenue W, Unit 27", "stars": 5 }
{ "name": "Steeles Paint & Decorating", "address": "4190 Steeles Avenue W", "stars": 5 }
{ "name": "Stars Hair Salon", "address": "7611 Pine Valley Drive, Unit 26", "stars": 5 }
{ "name": "Freshhouse Juice Bar", "address": "2 Tall Grass Trail, Unit 1", "stars": 5 }
{ "name": "Hanlan Automotive Parts Distributors", "address": "320 Hanlan Road, Unit 4", "stars": 5 }
{ "name": "Cannoli Queens", "address": "200 Marycroft Avenue", "stars": 5 }
{ "name": "Shine Autos", "address": "5153 Steeles Avenue W", "stars": 5 }
{ "name": "Peppers", "address": "3027 Islington Avenue", "stars": 4.5 }
{ "name": "Union Rim Repair", "address": "80 Millwick Drive", "stars": 4.5 }
{ "name": "Dolce Bombe", "address": "7611 Pine Valley Drive, Unit 4", "stars": 4.5 }
{ "name": "That's Italian Ristorante", "address": "2 Tall Grass Trail, Unit 4", "stars": 4.5 }
{ "name": "Investments Hardware", "address": "250 Rowntree Dairy Road", "stars": 4.5 }
{ "name": "Paramount Conference and Event Centre", "address": "222 Rowntree Dairy Road", "stars": 4.5 }
{ "name": "Dp Lazer Maze", "address": "31 Gaudaur Road", "stars": 4.5 }
{ "name": "Aida's Pine Valley Bakery", "address": "830 Rowntree Dairy Road, Unit 8", "stars": 4.5 }
{ "name": "Pho Soho", "address": "4000 Steeles Ave W, 25", "stars": 4.5 }
```

图 3.7 T13 查询结果

3.3 任务小结

在 MongoDB 实验中，我体验了文档数据库在处理半结构化数据时的优势。

实验中，我遇到的一个挑战是如何有效地创建和管理索引以优化查询性能。为了解决这个问题，我学习了如何使用 `explain` 命令来分析查询，并根据分析结果创建了合适的索引。例如，在处理全文搜索时，我为文本字段创建了全文索引，从而加快了搜索速度。另一个挑战是如何设计 MapReduce 作业以处理大规模数

据集。通过实践，我学会了如何编写 MapReduce 脚本，并对商家评分进行了有效的统计分析。这些经验让我对 MongoDB 的数据处理能力有了深刻的理解。

地图索引部分也具有挑战性。查询距离特定商家 2 公里以内的所有商家，并返回商家名字、地址和星级时，我刚开始没有弄清 `business` 中表示位置的数据时如何定义的，后来仔细研究了 `loc` 的内容并梳理清楚，才继续进行实验。另一个困难是如何正确使用 `2dsphere` 索引和 `$geoNear` 聚合阶段来实现这一功能。通过学习 MongoDB 的地理空间查询功能，我掌握了如何创建 `2dsphere` 索引，以及如何使用 `$geoNear` 来根据地理位置进行查询。这些技能对于处理地理位置数据至关重要。

4 Neo4j 实验

4.1 任务要求

基本节点查询：查询用户或商家节点的基本信息。

多关系查询：如用户评价的商家、商家类别等。

复杂条件查询：结合 `where` 和 `toInteger` 进行类型转换和过滤。

执行计划分析：通过 `PROFILE` 分析查询效率并优化。

索引的性能影响：创建索引后测试插入、删除性能。

4.2 完成过程

启动 Neo4j 后，在浏览器输入相应地址进入 Neo4j。

开始实验前，先点开图形界面查看各数据关系，如图 4.1，上下分别是 HasFriend 和 Reviewed 关系，注意箭头的方向，这对之后的代码编写至关重要。

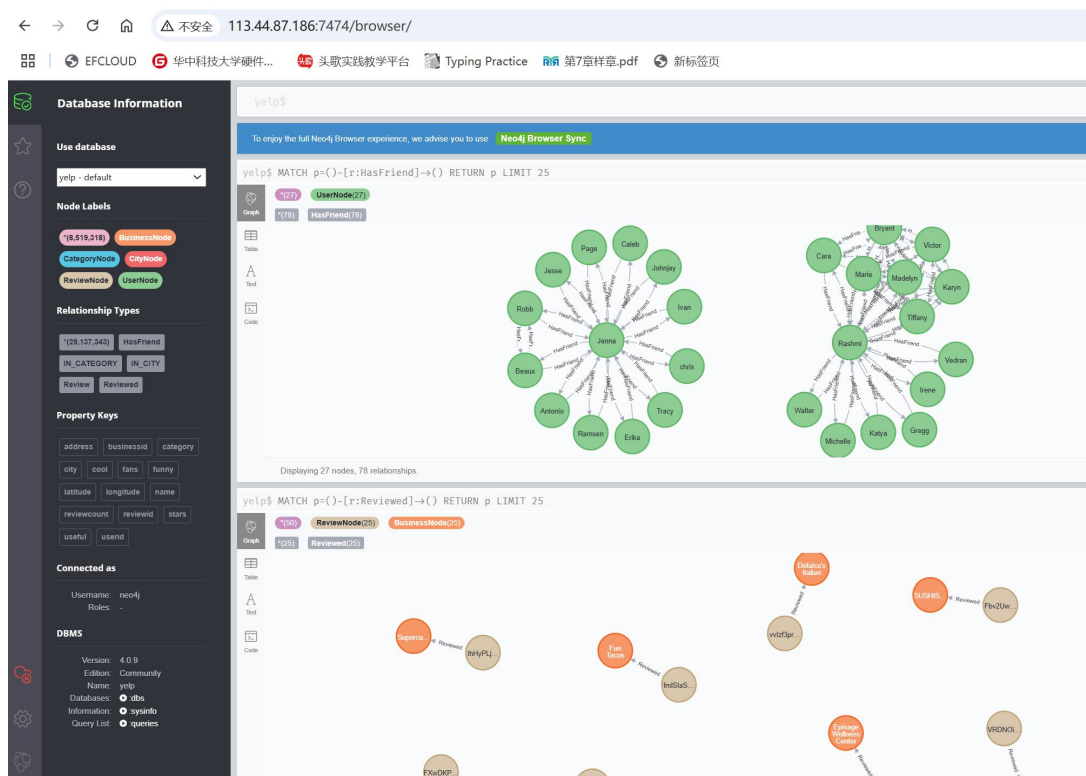


图 4.1 Neo4j 关系图

4.2.1 查询时简单运算 (T13)

任务：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。

(1) 确定目标关系：从用户节点 (UserNode) 出发，通过 `Review` 关系找

到与指定商户相关的 `ReviewNode`，再通过 `Reviewed` 关系找到特定的商户节点 (`BusinessNode`)。

(2) 数据转换与计算：提取用户的 `useful`、`funny` 和 `cool` 属性，将其从字符串转换为整数 (使用 `toInteger` 函数)，并计算它们的总和。

(3) 排序与返回：根据总和 (`total_score`) 降序排列，返回用户名称 (`name`) 及其得分。

使用 `MATCH` 来建立节点间的关系，`WITH` 用于中间计算，`ORDER BY` 用于排序，最终返回用户的得分结果。

具体代码如下：

```
MATCH
(u:UserNode)-[:Review]->(r:ReviewNode)-[:Reviewed]->(b:BusinessNode
{businessid: "nh_kQ16QAoXWwqZ05MPfBQ"})
WITH u, (toInteger(u.useful) + toInteger(u.funny) + toInteger(u.cool)) AS
total_score
RETURN u.name AS user_name, total_score
ORDER BY total_score DESC
查询结果如图 4.2 所示。
```

yelp\$ MATCH (u:UserNode)-[:Review]->(r:ReviewNode)-[:Reviewed]->(b:BusinessNode {businessid: "nh_kQ16QAoXWwqZ05MPfBQ"})

"user_name"	"total_score"
"Michael"	380
"Delilah"	291
"Lisa"	57
"Dawn"	20
"Amanda"	18
"Riley"	14
"Marisol"	13
"Erika"	11
"alistair"	9
"Sal"	8
"Tauny"	8
"Ellen"	6
"Rachel"	6
"Jeff"	6
"Jenna"	5
"Jane"	4
"Amanda"	3
"Ashton"	1
"Kailani"	0
"Christy"	0
"RaAnaa"	0

图 4.2 T13 查询结果

4.2.2 多关系联合查询（T14）

任务：查询具有评分为 5.0 的 Propane 类别的商铺的名字和所在的城市以及地址。

(1) 分类关系匹配：通过 IN_CATEGORY 关系找到属于 Propane 类别的商铺（BusinessNode）。

(2) 评分关系匹配：匹配与该商铺相关联的评分节点（ReviewNode），筛选评分为 5.0 的商铺。

(3) 结果过滤与返回：筛选结果中唯一的商铺信息（去重），并返回商铺名称、城市和地址。

利用 MATCH 建立多关系匹配，WHERE 用于筛选评分条件，DISTINCT 确保结果唯一性。

具体代码如下：

```
MATCH (b:BusinessNode)-[:IN_CATEGORY]->(c:CategoryNode {category: 'Propane'})
MATCH (b)<-[:Reviewed]-(r:ReviewNode)
WHERE r.stars = '5.0'
RETURN DISTINCT b.name AS businessName, b.city AS businessCity,
b.address AS businessAddress;
```

部分查询结果：

yelp\$ MATCH (b:BusinessNode)-[:IN_CATEGORY]->(c:CategoryNode {category: 'Propane'}) MATCH (b)<-[:Reviewed]-(r:ReviewNode) WHERE r.stars = '5.0' RETURN DISTINCT b.name AS businessName, b.city AS businessCity, b.address AS businessAddress;

"businessName"	"businessCity"	"businessAddress"
"Canyon State Propane"	"Phoenix"	"5700 W Buckeye Rd"
"E&H Ace Hardware"	"Avon Lake"	"375 Lear Rd"
"AZ Propane Express"	"Chandler"	"123 W Chandler Heights Rd"
"U-Haul Moving & Storage at Boulder Hwy"	"Las Vegas"	"5316 Boulder Hwy"
"U-Haul Moving & Storage at Downtown Campus"	"Madison"	"602 W Washington Ave"
"F1 Food And Fuel"	"Phoenix"	"8941 N 12th St"
"Trop Stop Gas & Car Wash"	"Las Vegas"	"4885 W Tropicana Ave"
"U-Haul Moving & Storage of Warner Park"	"Madison"	"2701 Packers Ave"
"Speedee Mart #123"	"Las Vegas"	"390 E Silverado Ranch Blvd"
"Acorn Propane"	"Phoenix"	"908 S 27th Ave"
"U-Haul Moving & Storage At 24th & McDowell"	"Phoenix"	"2345 E McDowell Rd"

图 4.3 T14 查询结果

4.2.3 条件查询（T15）

任务：统计每个用户评价过多少个不同的商家，按照此数量降序排列，返回

name, fans, useful 以及评价过的不同商家数 结果限制 20 条记录。

(1) 关系匹配：通过 Review 和 Reviewed 关系连接用户、评论和商铺。

(2) 统计不同商铺数量：对每个用户，统计其评论过的不同商铺数，命名为 unique_business_count。

(3) 结果排序与返回：按不同商铺数量降序排列，并返回用户的相关信息及统计结果。

通过 count(DISTINCT ...) 统计不同商铺数，结合 WITH 和 ORDER BY 实现分组和排序。

具体代码如下：

```
MATCH (u:UserNode)-[:Review]->(r:ReviewNode)-[:Reviewed]->(b:BusinessNode)
    // 查找用户、评论和商户之间的关系

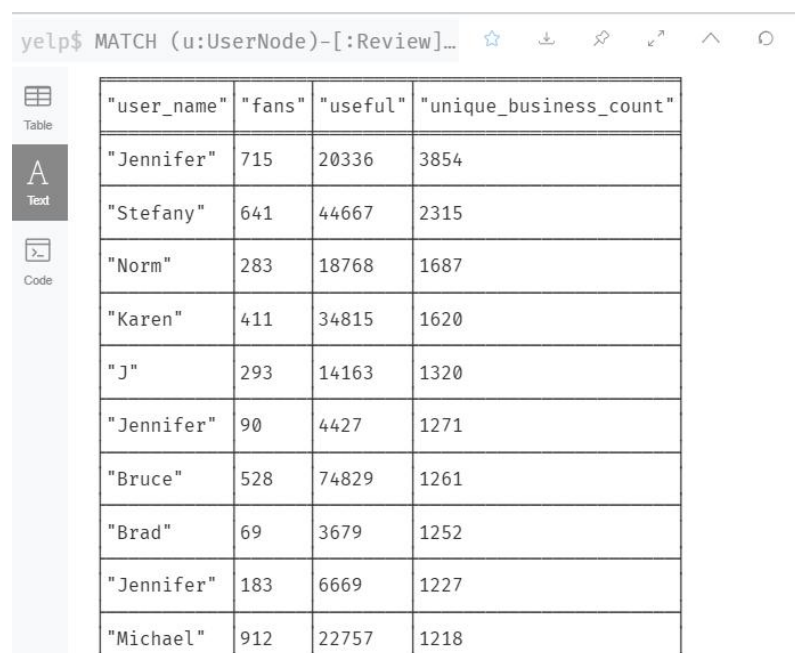
WITH u, count(DISTINCT b) AS unique_business_count
    // 统计每个用户评论过的不同商户数，并命名为 unique_business_count

RETURN u.name AS user_name, toInteger(u.fans) AS fans, toInteger(u.useful)
AS useful, unique_business_count
    // 返回用户的名称、粉丝数、"有用"评价数，以及评论过的不同商户数

ORDER BY unique_business_count DESC
    // 按照评论过的不同商户数降序排序，最多商户数的用户排前

LIMIT 20
    // 返回前 20 个用户

部分查询结果：
```



"user_name"	"fans"	"useful"	"unique_business_count"
"Jennifer"	715	20336	3854
"Stefany"	641	44667	2315
"Norm"	283	18768	1687
"Karen"	411	34815	1620
"J"	293	14163	1320
"Jennifer"	90	4427	1271
"Bruce"	528	74829	1261
"Brad"	69	3679	1252
"Jennifer"	183	6669	1227
"Michael"	912	22757	1218

图 4.4 T15 查询结果

4.2.4 多数据库的比较 (T18)

任务: 分别用 Neo4j 和 MongoDB 查询 review_id 为 TIYgnDzezfeEnVeu9jHeEw 对应的 business 信息, 比较两者查询时间, 指出 Neo4j 和 MongoDB 主要的适用场景。

1. Neo4j

(1) 查询评论节点: 通过 MATCH 匹配 ReviewNode, 筛选出 reviewid 为目标值的节点。

(2) 关系匹配商铺节点: 通过 Reviewed 关系找到关联的商铺节点 (BusinessNode)。

(3) 返回结果: 直接返回商铺信息。

MATCH

(r:ReviewNode

{reviewid: "TIYgnDzezfeEnVeu9jHeEw"})-[:Reviewed]->(b:BusinessNode)

RETURN b

查询结果如图 4.1 所示。

Started streaming 1 records in less than 1 ms and completed after 14993 ms.

图 4.5 Neo4j 查询结果

2. MongoDB

(1) 查找评论数据: 通过 findOne 查询 review 集合中目标 review_id 的评论记录。

(2) 获取商铺数据: 从 business 集合中查找与评论记录关联的 business_id, 并返回完整的商铺信息。

(3) 计算查询时间: 记录查询的开始和结束时间, 计算总耗时。

具体代码如下:

```
var startTime = new Date();
```

```
var review = db.review.findOne({ review_id: "TIYgnDzezfeEnVeu9jHeEw" });
```

```
var businessInfo = db.business.findOne({ business_id: review.business_id });
```

```
var endTime = new Date();
```

```
var elapsedTime = endTime - startTime;
```

```
print("Business Information:", JSON.stringify(businessInfo));
```

```
print("Query Time:", elapsedTime, "ms");
```

查询结果如下图所示。


```

> var startTime = new Date();
> var review = db.review.findOne({ review_id: "TIYgnDzezfeEnVeu9jHeEw" });
> var businessInfo = db.business.findOne({ business_id: review.business_id });
> var endTime = new Date();
> var elapsedTime = endTime - startTime;
> print("Business Information:", JSON.stringify(businessInfo));
Business Information: {"_id":{"_id":"6016c6b5af81085b0f21b190"},"business_id":"I3UkP4Mmp0cmfe3vTev0jw","name":"Rue Sherbrooke Est","city":"Montréal","state":"QC","postal_code":"H2L 1J9","latitude":45.517348,"longitude":-73.5677004,"review_count":39,"is_open":0,"attributes":{"RestaurantsGoodForGroups":"True","BikeParking":"False","NoiseLevel":"True","WiFi":"u'free","BusinessParking":{"garage":False,"street":True,"validated":False,"lot":False,"delivery":True},"RestaurantsReservations":"True","Alcohol":"beer_and_wine","HasTV":"True","RestaurantType":"False","RestaurantsTakeOut":"True","RestaurantsAttire":"u'casual","Ambience":{"romantic":False,"hipster":False,"divey":False,"touristy":False,"trendy":False,"upscale":False,"casual":True},"Japanese":"Restaurants"},"hours":null,"loc":{"type":"Point","coordinates":[-73.5677004,45.517348]}}
> print("Query Time:", elapsedTime, "ms");
Query Time: 38 ms

```

图 4.6 MongoDB 查询结果

4.3 任务小结

在 Neo4j 实验中，我专注于图数据库的特点和应用。通过基本节点查询和多关系查询，我学会了如何高效地获取和操作图数据。在复杂条件查询中，我掌握了结合 where 子句和类型转换的技巧，以实现精确的数据筛选。执行计划分析让我认识到了查询优化的重要性，而索引的性能影响实验则展示了索引在提升图数据库性能中的关键作用。

实验中，我遇到了对箭头指向关系的混淆，比如餐厅是被评论评价的，如果这里箭头写反了就会出现报错，这在编写 Cypher 查询时造成了困难。为了解决这个问题，我通过仔细分析数据库图形化的关系图，逐渐熟悉了 Neo4j 的图形模型和关系方向。

另一个挑战是在处理评分为 5.0 的查询时，我没有意识到 stars 字段是字符而不是数字，导致查询出错。通过错误排查，我学会了如何正确地使用 Cypher 查询语言进行类型转换和条件过滤。这些经验让我提高了查询效率，并加深了对图数据库的理解。

5 课程总结

5.1 主要工作

- (1) 学习数据库基础知识：掌握了关系数据库 MySQL、文档数据库 MongoDB 和图数据库 Neo4j 的基本概念和操作。
- (2) 实验软件和数据部署：成功在华为平台上部署了所需的三个数据库软件，并加载了 Yelp Dataset 数据集。
- (3) 在 MySQL for JSON 实验中，我学习了 JSON 数据的基本查询、增删改操作、聚合和实用函数的使用，并通过创建索引优化了查询性能。
- (4) 在 MongoDB 实验中，我探索了条件查询、聚合与索引的使用，以及 MapReduce 的实现，特别是在地图索引的创建和使用上获得了实践经验。
- (5) 在 Neo4j 实验中，我专注于图数据库的节点查询、多关系查询和复杂条件查询，以及索引对性能的影响。

5.2 心得体会

通过本次实验，我认识到了不同数据库系统有其独特的优势和适用场景。关系数据库在事务处理和结构化数据管理方面表现出色；文档数据库在处理半结构化数据和快速迭代开发中具有优势；图数据库在分析和查询复杂关系网络时展现出无与伦比的能力。

5.3 未来展望

由于时间能力有限，实验任务中四五涉及到多数据库交互应用以及 MVCC 多版本并发控制对比的部分我都没有完成，对不同数据库的交互特性和并发度的探索不够深入，对性能调优的实践还有待加强。在未来，我计划继续深化对数据库技术的理解，提升自己的技能水平，特别是在多数据库系统的协同工作方面。

课程设置方面，如果实验课的课时再设置多一些就更好了。