

目 录

1	教师评分页	1
2	课程任务概述	2
3	实验任务一 (Mysql for Json 实验)	3
3.1	任务要求一	3
3.2	完成过程	3
3.2.1	题目 12	3
3.2.2	题目 13	4
3.3	任务小结	5
3.3.1	题目 12	5
3.3.2	题目 13	6
4	实验任务二 (MongoDB 实验)	8
4.1	任务要求	8
4.2	完成过程	8
4.3	思路简述和感想	8
4.4	任务小结	9
5	实验任务三 (Neo4j 实验)	11
5.1	任务要求	11
5.2	完成过程	11
5.3	任务小结	11
6	实验任务四 (多数据库交互应用实验)	13
6.1	任务要求	13
6.2	完成过程	13
6.3	任务小结	14
7	实验任务五(不同类型数据库 MVCC 多版本并发控制对比实验)	18
7.1	任务要求	18
7.2	完成过程	18
7.3	任务小结	21
8	课程总结	22

1 教师评分页

子目标	子目标评分
1	
2	
3	
4	
5	
6	
7	
总分	

2 课程任务概述

在云平台部署图数据库 Neo4j，关系数据库 MySQL，文档数据库 MongoDB，完成对应数据集的导入和初始化，然后根据课程任务编写一系列增删改查的任务，按照课程要求同时关注不同数据库的使用体验和性能差异以及在不同场景下的适用性。体会不同数据库的特点，优势和适用范围。

该实验报告使用 `typst` 编辑完成。`Typst` 是一个排版文档的标记语言。它被设计为易于学习，速度快，用途广泛。`Typst` 获取带有标记的文本文件并输出为 PDF 文件。`Typst` 适合撰写长篇文本（如论文、文章、科学论文、书籍、报告和家庭作业）。此外，`Typst` 非常适合于编写任何包含数学符号的文档，例如在数学、物理和工程领域的论文。最后，由于其强大的风格化和自动化功能，它是任何一组具有共同风格的文件的绝佳选择，例如丛书。

3 实验任务一（Mysql for Json 实验）

3.1 任务要求一

在华为云服务器上安装 MySQL 并且导入 test 数据库, 然后完成诸个实验任务。
由于页数要求, 挑选以下典型题目进行叙述:

①JSON 聚合与索引的题目 12

②JSON 聚合与索引的题目 13。

12 题实验要求如下: 在 tip 表中找到被提建议最多的商户和提出建议最多的用户, 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。

13 题要求如下: 查询被评论数前三的商户, 使用 JSON_TABLE() 导出他们的名字, 被评论数, 是否在星期二营业(即"hours"中有"Tuesday"的键值对, 是返回 1, 不是返回 0), 和一周所有的营业时段(不考虑顺序, 一个时段就对应一行, 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

3.2 完成过程

配置 mysql 环境以及导入数据库的过程不再赘述。以下是代码以及思路:

3.2.1 题目 12

```
1 SELECT u.user_id,  
2        b.business_id,  
3        JSON_MERGE_PATCH(b.business_info, u.user_info)  
4        AS merged_info  
5 FROM (  
6     SELECT t.business_id  
7     FROM (  
8         SELECT business_id, COUNT(*) AS tip_count  
9         FROM tip  
10        GROUP BY business_id  
11        ORDER BY tip_count DESC  
12        LIMIT 1  
13     ) AS t  
14 ) AS bt  
15 JOIN business b ON bt.business_id = b.business_id  
16 JOIN (  
17     SELECT t.user_id  
18     FROM (  
19         SELECT user_id, COUNT(*) AS tip_count  
20         FROM tip  
21         GROUP BY user_id  
22         ORDER BY tip_count DESC
```

```

23         LIMIT 1
24     ) AS t
25 ) AS ut
26 JOIN user u ON ut.user_id = u.user_id;

```

使用聚合查找首先找到 tip 表中被提建议最多的商家 ID，然后找出提出建议最多的用户 ID，将这两个表连接以后使用 JSON_MERGE_PATCH 函数将商户信息和用户信息合并为一个 JSON 文档。

这里使用的 JSON_MERGE_PATCH 使用函数是用于合并两个 JSON 对象的 MySQL 函数。它接受两个 JSON 参数，并返回一个新的 JSON 对象，其中包含了两个输入对象的合并结果。这道题目需要的正是返回 JSON 格式文档，这个函数完美符合了题目的需求。

3.2.2 题目 13

```

1  JSON_UNQUOTE(JSON_EXTRACT(business_info, '$.name')) AS name,
2  JSON_UNQUOTE(JSON_EXTRACT(business_info, '$.review_count')) AS
   review_count,
3  IF(tuesday IS NULL, 0, 1) AS is_open_on_tuesday,
4  ROW_NUMBER() OVER () AS num,
5  monday, tuesday, wednesday, thursday, friday, saturday, sunday
6  FROM
7  (
8      SELECT
9          business_id,
10         business_info
11     FROM
12         business
13     ORDER BY
14         JSON_EXTRACT(business_info, '$.review_count') DESC
15     LIMIT 3
16 ) AS top_3_businesses,
17 JSON_TABLE(
18     business_info, '$.hours' COLUMNS (
19         monday VARCHAR(20) PATH '$.Monday',
20         tuesday VARCHAR(20) PATH '$.Tuesday',
21         wednesday VARCHAR(20) PATH '$.Wednesday',
22         thursday VARCHAR(20) PATH '$.Thursday',
23         friday VARCHAR(20) PATH '$.Friday',
24         saturday VARCHAR(20) PATH '$.Saturday',
25         sunday VARCHAR(20) PATH '$.Sunday'
26     )
27 ) AS operating_hours
28 ORDER BY
29     name ASC;

```

13 题难度较高，查询较为复杂。首先需要定位所需资源的位置。

mysql 的 business 表有两个属性分别是 business_id, business_info。后者是一个 json 对象。

详细解析 business_info,'\$.name'字段是商家的名字, '\$.review_count'是评论数, '\$.hours.Tuesday'是星期二的营业时间, 其他运营时间以此类推,这里通过简单的条件判断把是否在周二营业可以转换成 0 和 1 的形势。

接下来简单查询获取被评论数前三的商户, 使用 JSON_EXTRACT 降序提取出被评论数最多的商家然后取前三即可。

然后需要把营业时间的信息提取出来, 这里使用了 JSON_TABLE 函数, 它可以把 json 对象转换成表的形式, 我用它把周一到周日的营业时间转换成了表。JSON_TABLE 函数的第一个参数是 json 对象, 第二个参数是 json 对象的路径, 第三个参数是列名和路径的对应关系。这里需要注意的是, json 对象的路径是从根节点开始的, 而不是从当前节点开始的。因此这里的路径是 \$.hours。这样就把营业时间段提取出来了。

最后使用了 union 函数把每个商家的营业时间按照顺序编号, 完成查询。

3.3 任务小结

3.3.1 题目 12

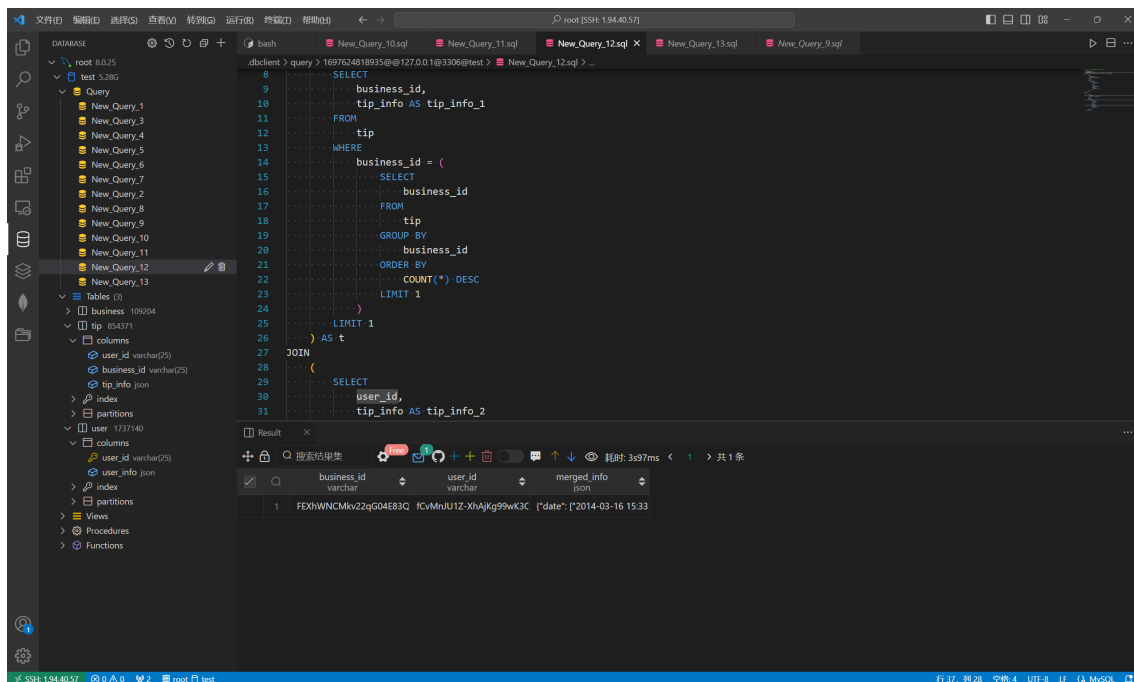


图 1: JSON_MERGE_PATCH 函数的使用的返回结果

12.在Subreview集合中统计评价中useful、funny和cool都大于6的商家,家id及平均打星,并按商家id降序排列。

共1558条记录,下面截取了前5条

```
{
  "stars_avg": 2,
  "business_id": "zvQIEpJUmLLmMMffNntHXQ"
}
{
  "stars_avg": 4.5,
  "business_id": "zrYpLdnGKA_EmOhgRCy_vg"
}
{
  "stars_avg": 4,
  "business_id": "znWHLWlpt19Hzw1VY6KfCA"
}
{
  "stars_avg": 3,
  "business_id": "znBAFVFRMvdUVJ7eJY_vjA"
}
```

```
{
  "stars_avg": 4,
  "business_id": "zitXLajbETOuHQJfok4a4g"
}
```

图 2: 实验结果参考

经过结果参考表可知,结果正确。

3.3.2 题目 13

经过结果参考表可知,结果正确。但是我发现保存下来的是错误的图片,服务器又因为钱花完已经释放掉了,图片仅供参考。

The screenshot displays a MySQL IDE interface with a query editor and a results pane. The query is as follows:

```

1  -- Active: 1697624818935@127.0.0.1@3306@test MySQL
2  > Execute
3  SELECT
4  ... JSON_UNQUOTE(JSON_EXTRACT(business_info, '$.name')) AS name,
5  ... JSON_UNQUOTE(JSON_EXTRACT(business_info, '$.review_count')) AS review_count,
6  ... IF(JSON_SEARCH(JSON_EXTRACT(business_info, '$.hours'), 'one', '%Tuesday%') IS NULL, 0, 1) AS is_open_on_tuesday,
7  ... ROW_NUMBER() OVER () AS period_number
8  FROM
9  (
10     ... SELECT
11     ... business_id,
12     ... business_info
13     FROM
14     ... business
15     ORDER BY
16     ... JSON_UNQUOTE(JSON_EXTRACT(business_info, '$.review_count')) DESC
17     LIMIT 3
18     ) AS top_3_businesses,
19  ... JSON_TABLE(
20  ... [{"day": "Monday", "open": "07:00", "close": "18:00"},
21  ... {"day": "Tuesday", "open": "09:00", "close": "17:00"},
22  ... {"day": "Wednesday", "open": "08:30", "close": "19:00"},
23  ... {"day": "Thursday", "open": "08:00", "close": "20:00"},
24  ... {"day": "Friday", "open": "07:30", "close": "18:30"}],

```

The results pane shows the following data:

	name	review_count	is_open_on_tuesday	period_number
	blob	blob	int	bigint
1	Bittercreek Alehouse	998	0	8
2	Bittercreek Alehouse	998	0	9
3	Bittercreek Alehouse	998	0	10
4	Bittercreek Alehouse	998	0	11
5	Bittercreek Alehouse	998	0	12
6	Bittercreek Alehouse	998	0	13
7	Bittercreek Alehouse	998	0	14
8	Crema Coffee Roasters	998	0	21

图 3: Mysql-Json 复杂查询的返回结果

4 实验任务二 (MongoDB 实验)

4.1 任务要求

在华为云远程服务器上配置 MongoDB 环境，导入数据集，完成诸个实验任务。
由于页数要求，挑选典型题目：题目 12 作为叙述对象。题目详细要求如下：

在 Subreview 集合中统计评价中 useful、funny 和 cool 都大于 6 的商家返回商家 id 及平均打星，并按商家 id 降序排列。

4.2 完成过程

详细代码如下：

```
1 const database = 'yelp';
2 const collection = 'NEW_COLLECTION_NAME';
3
4 // The current database to use.
5 use(database);
6
7 // Create a new collection.
8 // 题目要求
9 // 题目要求
10 db.Subreview.aggregate([
11     {$match: {useful: {$gt: 6}, funny: {$gt: 6}, cool: {$gt: 6}}},
12     {$group: {_id: "$business_id", average_stars: {$avg: "$stars"}}},
13     {$sort: {_id: -1}}
14 ])
```

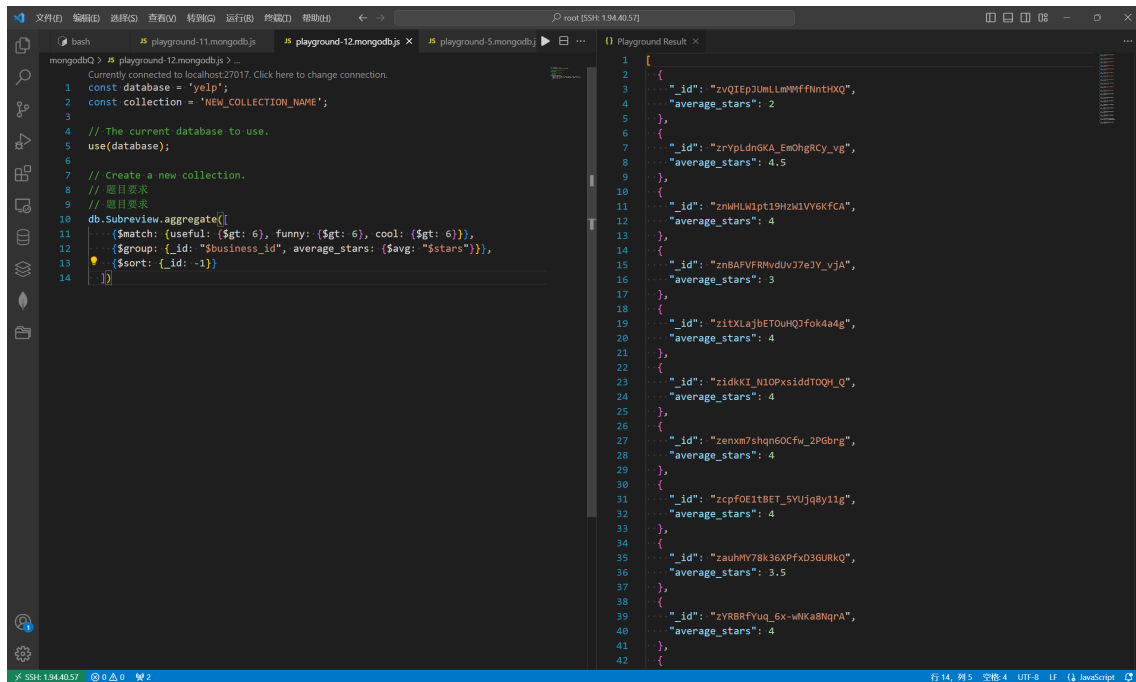
4.3 思路简述和感想

mongoDB 是快速上手的文档数据库，操作和 web 的开发语言 javascript 十分相似，上手难度不大。mongo 的数据层级是 database-collection，基本查询操作就是选择数据库然后对 db 对象的对应集合进行操作。

在这道题中，首先需要我了解并且使用 aggregate。aggregate 是 MongoDB 中用于数据聚合的强大工具。aggregate 操作可用于对文档进行多个阶段的处理，每个阶段的输出是下一个阶段的输入。

因此按照题意，我在 aggregate 里首先使用 match 筛选出符合要求大于 6 的数据，然后使用 group 按照商家 id 进行分组，并且把商家的平均打分作为聚合值返回，最后对商家 id 进行降序排列。

4.4 任务小结



The screenshot shows the MongoDB Playground interface. On the left, a code editor displays a MongoDB aggregate query. The query uses a match stage to filter documents where 'useful' is greater than 6 and 'funny' is greater than 6, and a group stage to calculate the average number of stars for each business. The results are sorted by the average stars in descending order. On the right, the 'Playground Result' tab shows the output of the query, which is a JSON array of documents. Each document contains an '_id' field representing the business ID and an 'average_stars' field representing the calculated average. The results are sorted by 'average_stars' in descending order.

```
1 const database = 'yelp';
2 const collection = 'NEW_COLLECTION_NAME';
3
4 // The current database to use.
5 use(database);
6
7 // Create a new collection.
8 // 题目要求
9 // 题目要求
10 db.Subreview.aggregate([
11   {$match: {useful: {$gt: 6}, funny: {$gt: 6}, cool: {$gt: 6}}},
12   {$group: {_id: "$business_id", average_stars: {$avg: "$stars"}}},
13   {$sort: {_id: -1}}
14 ])
```

```
1 {
2   "_id": "zvQIEpJUmLLeMffnHXQ",
3   "average_stars": 2
4 }
5 {
6   "_id": "zrvpLdnGKA_EmOhgRCy_vg",
7   "average_stars": 4.5
8 }
9 {
10  "_id": "zmMHLwipt19HZwIVV6KfCA",
11  "average_stars": 4
12 }
13 {
14  "_id": "mBAFvFRMydUvJ7e3Y_vjA",
15  "average_stars": 3
16 }
17 {
18  "_id": "zitXLajbETOuHQJfok4a4g",
19  "average_stars": 4
20 }
21 {
22  "_id": "zidkKI_N1OPxiddTOQH_Q",
23  "average_stars": 4
24 }
25 {
26  "_id": "tenxm7shqn6OCfw_2PGbrg",
27  "average_stars": 4
28 }
29 {
30  "_id": "zcpFOEitBET_5VUjq8y1lg",
31  "average_stars": 4
32 }
33 {
34  "_id": "zauhMY78k36XPfxD3GURkQ",
35  "average_stars": 3.5
36 }
37 {
38  "_id": "zyRBRFVuq_6x-mKa8NqA",
39  "average_stars": 4
40 }
41 {
42  ...
43 }
```

图 4: aggregate 的使用的返回结果

12.在Subreview集合中统计评价中useful、funny和cool都大于6的商家，返回商家id及平均打星，并按商家id降序排列。

共1558条记录，下面截取了前5条

```
{
  "stars_avg": 2,
  "business_id": "zvQIEpJUmLLmMMffNntHXQ"
}
{
  "stars_avg": 4.5,
  "business_id": "zrYpLdnGKA_EmOhgRCy_vg"
}
{
  "stars_avg": 4,
  "business_id": "znWHLWlpt19Hzw1VY6KfCA"
}
{
  "stars_avg": 3,
  "business_id": "znBAFVFRMvdUVJ7eJY_vjA"
}
```

```
{
  "stars_avg": 4,
  "business_id": "zitXLajbETOuHQJfok4a4g"
}
```

图 5: 实验结果参考

经过结果参考表可知，结果正确。

5 实验任务三 (Neo4j 实验)

5.1 任务要求

在华为云远程服务器上安装并且配置 Neo4j 环境并且完成一系列图查询实验。

neo4j 是一个流行的图数据库，它的数据结构是图，图是由节点和关系组成的。neo4j 的查询语言是 cypher，它的语法和 sql 有很大的不同，但是也有很多相似的地方。cypher 的查询语句是由 match, where, return 组成的。match 用于匹配节点和关系，where 用于筛选，return 用于返回结果。以上就是 neo4j 的简要介绍

Neo4j 实验挑选题目 10 作为典型题目。下面给出详细题目和解题过程：

10. 查询 Allison 的朋友（直接相邻）分别有多少位朋友。（考察：使用 with 传递查询结果到后续的处理）

5.2 完成过程

首先给出实验代码：

```
1 MATCH(:UserNode{name:'Allison'})-[:HasFriend]->(friend)
2 WITH friend.name as friendsList,
3 size((friend)-[:HasFriend]-()) as number0fFoFs
4 RETURN friendsList,number0fFoFs
```

图查询的重点就是提取题目的所需节点和节点与节点之间的关系。

对于这道题，首先 Allison 的属性是 UserNode，想要查询他的朋友们就需要通过朋友这个关系来获取，因此是 MATCH(:UserNode{name:'Allison'})-[:HasFriend]->(friend)。

再然后，题目已经提醒我们通过 with 来传递查询结果到后续的处理。因此对于每个朋友，都应该使用和上一步类似的方法查询与他们存在朋友关系的节点，使用 size() 聚合函数以单个 Allison 的朋友作为对象返回这些朋友节点的数量，加上朋友的名字，就是这道题目的答案了。

5.3 任务小结

返回结果与参考答案一致，因此结果正确。

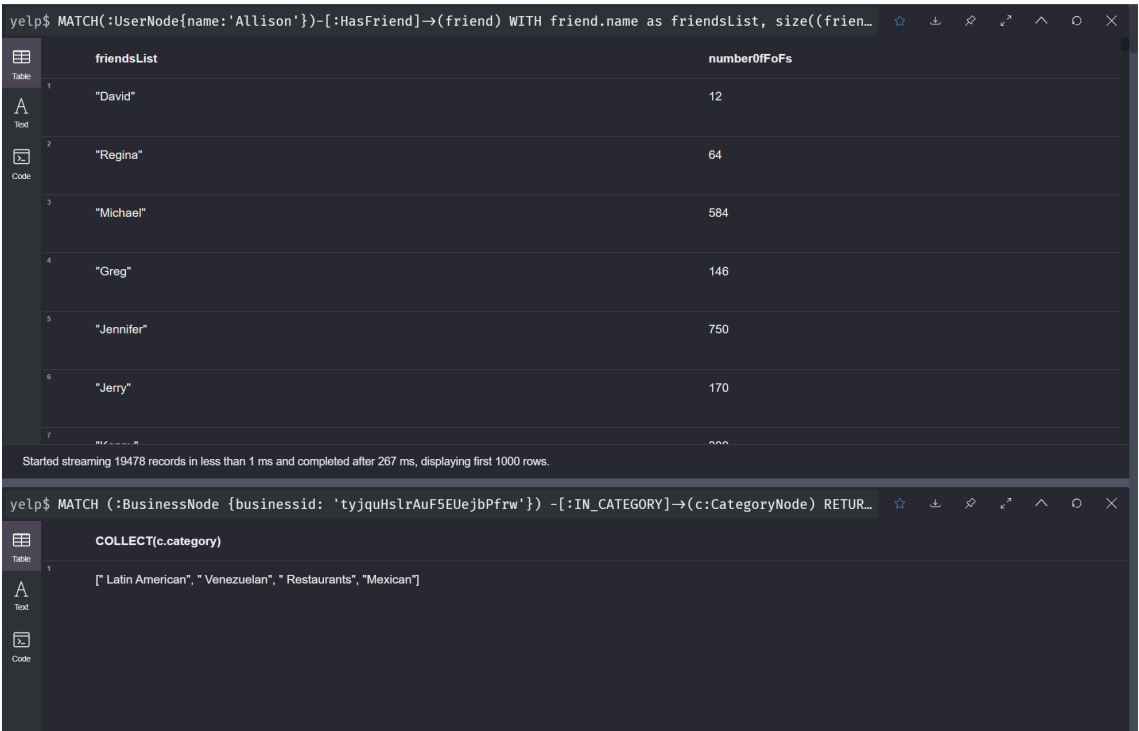


图 6: neo4j 查询

10.查询Allison的朋友（直接相邻）分别有多少位朋友。（考察：使用with传递查询结果到后续的处理）

共16836条记录,此处截取前5条

"friendsList"	"numberOfFoFs"
["David"]	12
["Regina"]	64
["Michael"]	584
["Greg"]	146
["Jennifer"]	750

图 7: 实验结果参考

6 实验任务四 (多数据库交互应用实验)

6.1 任务要求

多数据库交互应用实验具体就是使用 neo4j, mongoDB 执行依次的任务, 前者的输入是后者的输出, 通过这个实验可以体会到不同数据库的优势和适用范围。题目 2 的输入是题目 1 的输出, 因此下面将是题目 1 和题目 2 两个实验共同完成。以下是具体的题目要求:

题目 1: 使用 Neo4j 查找: 找出评论过超过 5 家不同商户的用户, 并在 Neo4j 以表格形式输出满足以上条件的每个用户的信息: name, funny, fans。题目 2: 将 1 得到的结果导入 MongoDB, 并使用该表格数据, 统计其中所有出现的用户名及该用户名对应的出现次数, 并按照出现次数降序排序, 使用 aggregate 实现

6.2 完成过程

首先依次给出题目 1 和题目 2 的代码:

```
1 MATCH      (user:UserNode)-[:Review]->(review:ReviewNode)-[:Reviewed]-
2 >(business:BusinessNode)
3 WITH user,count(distinct(business)) AS reviewCount
4 WHERE reviewCount > 5
5 RETURN user.name,user.funny,user.fans
```

题目 1

```
1 mongoimport --db Mydb --collection collection41 --file ~/resulttask4/
  records2.json --jsonArray
```

命令行代码, 以上为一行命令

```
1 use('Mydb');
2
3 db.collection43.aggregate([
4   { $group: { _id: '$name', count: { $sum: 1 } } },
5   { $sort: { count: -1 } }
6 ]);
7 db.collection43.mapReduce(
8   function() {
9     emit(this.name, 1);
10  },
11  function(key, values) {
12    return Array.sum(values);
13  },
```

```
14    {
15        out: { inline: 1 },
16        finalize: function(key, reducedValue) {
17            return { count: reducedValue };
18        }
19    }
20 )
```

题目 2

题目 1 思路：

1、找到评论用户：使用 Cypher 查询语句，匹配评论节点到用户节点的关系，统计每个用户评论过的不同商户数量。用 MATCH 和 COUNT 来实现。

2、过滤用户：在查询中添加条件，仅选择评论过不同商户数量大于 5 的用户。用 WHERE 子句进行过滤。

3、获取用户信息：使用之前筛选出的用户节点，进一步查询用户的信息，如用户名（name）、有趣指数（funny）、粉丝数量（fans）等。

4、以表格形式输出：最终结果通过 RETURN 子句指定要输出的用户信息，并在 Neo4j 中以表格形式显示然后下载导出结果为 JSON 格式数据。

题目 2 思路：

1、首先使用上方给出的命令行代码把数据导入到 MongoDB 中。2、使用 aggregate 函数依次执行 group 和 sort 操作，group 操作是按照 name 分组，然后使用 count 函数统计每个 name 出现的次数，sort 操作是按照 count 降序排列。

6.3 任务小结

```

1 MATCH (user:UserNode)-[:Review]-(review:ReviewNode)
2 WITH user,count(review) AS reviewCount
3 WHERE reviewCount > 5
4 RETURN user.name,user.funny,user.fans
5
yelp$ MATCH (user:UserNode)-[:Review]-(review:ReviewNode) WITH user,count(review) AS reviewCount WHERE re...

```

	user.name	user.funny	user.fans
1	"Rashmi"	"17"	"5"
2	"Jenna"	"22"	"4"
3	"David"	"8"	"0"
4	"Nancy"	"279"	"39"
5	"Keane"	"19356"	"696"
6	"Andre"	"0"	"0"

Started streaming 231177 records after 1 ms and completed after 34 ms, displaying first 1000 rows.

```

yelp$ MATCH (user:UserNode)-[:Review]-(review:ReviewNode) WITH user,count(review) AS reviewCount RETURN u...

```

	user.username	reviewCount
1	null	7
2	null	21

图 8: 题目 4-1 结果

```

1 use('mydb');
2
3 db.collection43.aggregate([
4   { $group: { _id: '$name', count: { $sum: 1 } } },
5   { $sort: { count: -1 } } ],
6   {});
7
8 db.collection43.mapReduce(
9   function() {
10     emit(this.name, 1);
11   },
12   function(key, values) {
13     return Array.sum(values);
14   },
15   {
16     out: { inline: 1 },
17     finalize: function(key, reducedValue) {
18       return { count: reducedValue };
19     }
20   }
21 );

```

```

1 {
2   "results": [
3     {
4       "_id": "Mad Man Army Surplus & Hip Hop Sho",
5       "value": {
6         "count": 8
7       }
8     },
9     {
10      "_id": "Valentine Sales & Management",
11      "value": {
12        "count": 5
13      }
14    },
15    {
16      "_id": "Bella Pizza",
17      "value": {
18        "count": 9
19      }
20    },
21    {
22      "_id": "Yonge Karaoke",
23      "value": {
24        "count": 2
25      }
26    },
27    {
28      "_id": "Sixteen Mile Sports Complex",
29      "value": {
30        "count": 2
31      }
32    },
33    {
34      "_id": "Godfather Shawarma, kabab Grill an",
35      "value": {
36        "count": 6
37      }
38    },
39    {
40      "_id": "Safari Landscaping",
41      "value": {
42        "count": 8
43      }
44    }
45  ]
46 }

```

图 9: 题目 4-2 结果

1.使用Neo4j查找：找出评论过超过5家不同商户的用户，并在Neo4j以表格形式输出满足以上条件的每个用户的信息：name,funny,fans。

134658条记录，只截取了10条，查询耗时长，需要等待几分钟

"name"	"funny"	"fans"
"Janet"	"0"	"0"
"Eve"	"28"	"4"
"Bernie"	"886"	"57"
"Curt"	"40"	"2"
"Filip"	"22"	"1"
"Merrill"	"3"	"0"
"Soleil"	"18286"	"278"
"Wojtek"	"19"	"1"
"Melissa"	"1"	"0"
"Beverly"	"12"	"12"

图 10: 题目 4-1 参考结果

2.将1得到的结果导入MongoDB，并使用该表格数据，统计其中所有出现的用户名及该用户名对应的出现次数，并按照出现次数降序排序,使用aggregate实现.

提示：

- 1) 从Neo4j的查询中导出csv文件
- 2) 在mongodb新建集合from_neo4j，将csv文件导入集合

3) 统计其中所有出现的商家名及该商家名对应的出现次数，并按照出现次数降序排序。

共144427条记录，两种方式导出的集合可能排列顺序不同。

aggregate方式产生的记录的前5条：

<code>_id: "Jennifer"</code> <code>cnt: 527</code>
<code>_id: "John"</code> <code>cnt: 468</code>
<code>_id: "David"</code> <code>cnt: 458</code>
<code>_id: "Michael"</code> <code>cnt: 456</code>
<code>_id: "Chris"</code> <code>cnt: 441</code>

图 11: 题目 4-2 参考结果

经过结果参考表可知，结果正确。

7 实验任务五(不同类型数据库 MVCC 多版本并发控制对比实验)

7.1 任务要求

自行构造多用户同时对同一数据库对象的增删改查案例，实验对比 MySQL 和 MongoDB 数据库对 MVCC 多版本并发控制的支持。

1. 体验 MySQL 在 InnoDB 存储引擎下的 MVCC 多版本并发控制，实现的事务 ACID 特性。请注意 Mysql 需要选用什么事务隔离级来支持 MVCC？请构造 多用户多写多读案例来展现 MVCC 并发控制特性，解释各种结果产生的原因。
2. 体验 MongoDB 的 MVCC，数据集可自建或选用 yelp 数据集中的 test 集合中进行测试，测试方法同 MySQL。请对测试结果进行说明，并与 MySQL 的 MVCC 实验结果进行对比分析。建议创建 MongoDB 副本或分片集群，体验 MVCC 的不同效果（可选做其一）

这里选择题目 1 作为典型案例，示范数据库如何实现 MVCC 的多版本并发支持控制。

7.2 完成过程

首先给出一个没有设置事务隔离级别的导致不可重复读的例子：

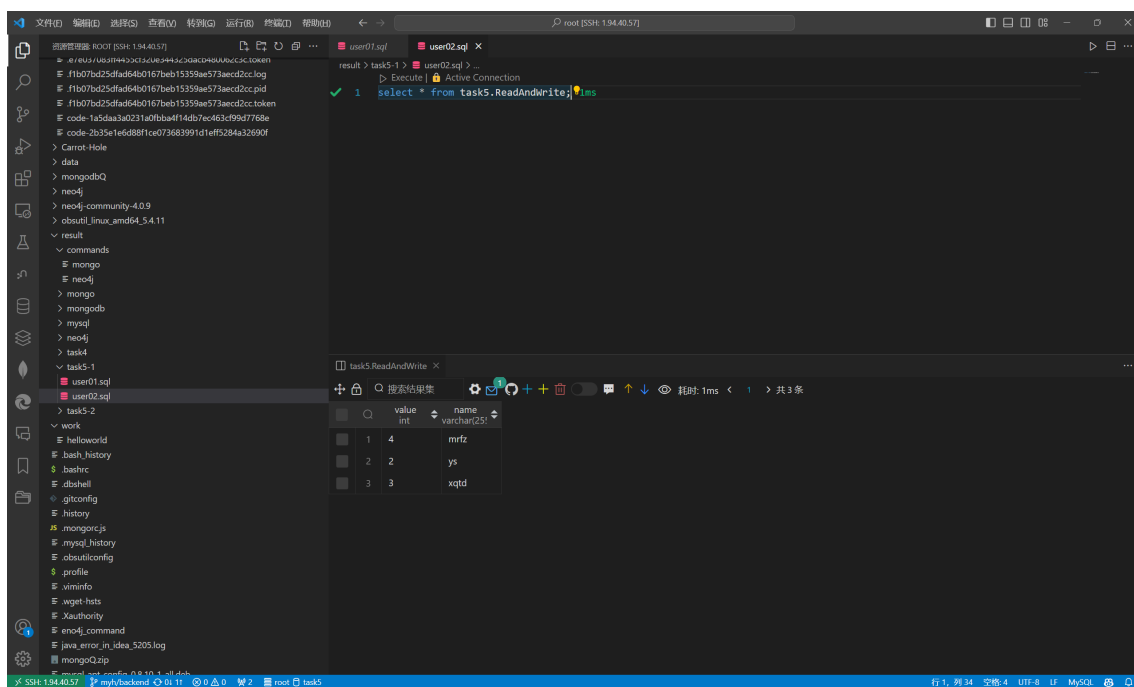


图 12: 事务 2

首先执行事务 2，可见 mrfz 的值为 4

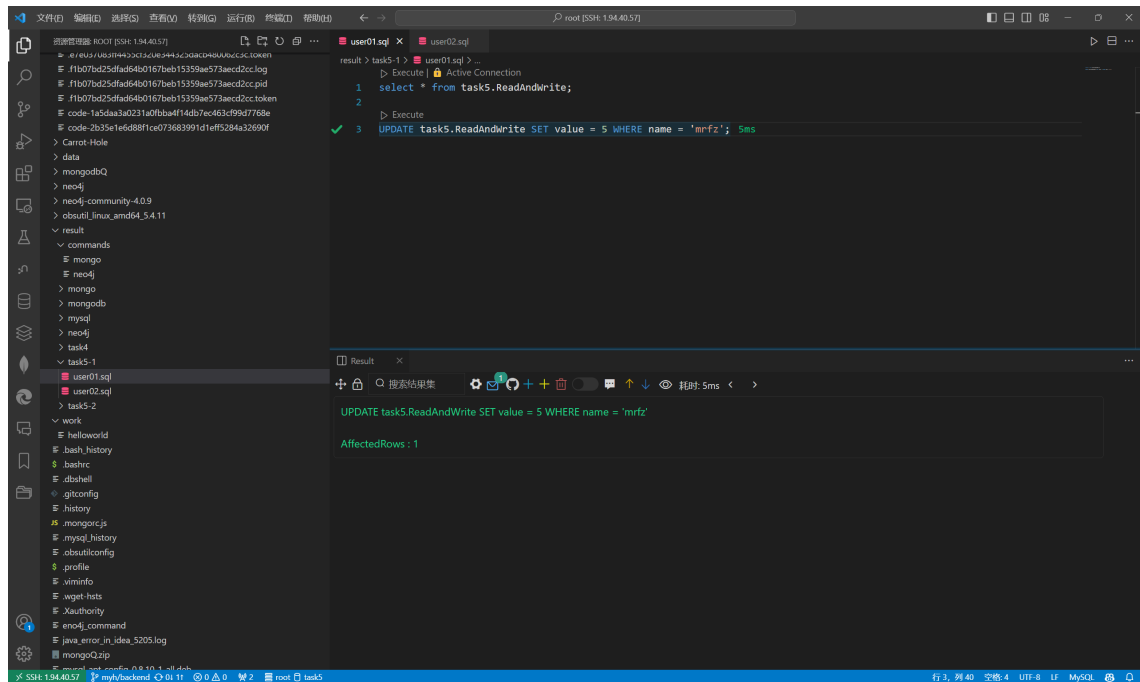


图 13: 事务 1

然后执行事务 1，更新 mrfz 的值为 5

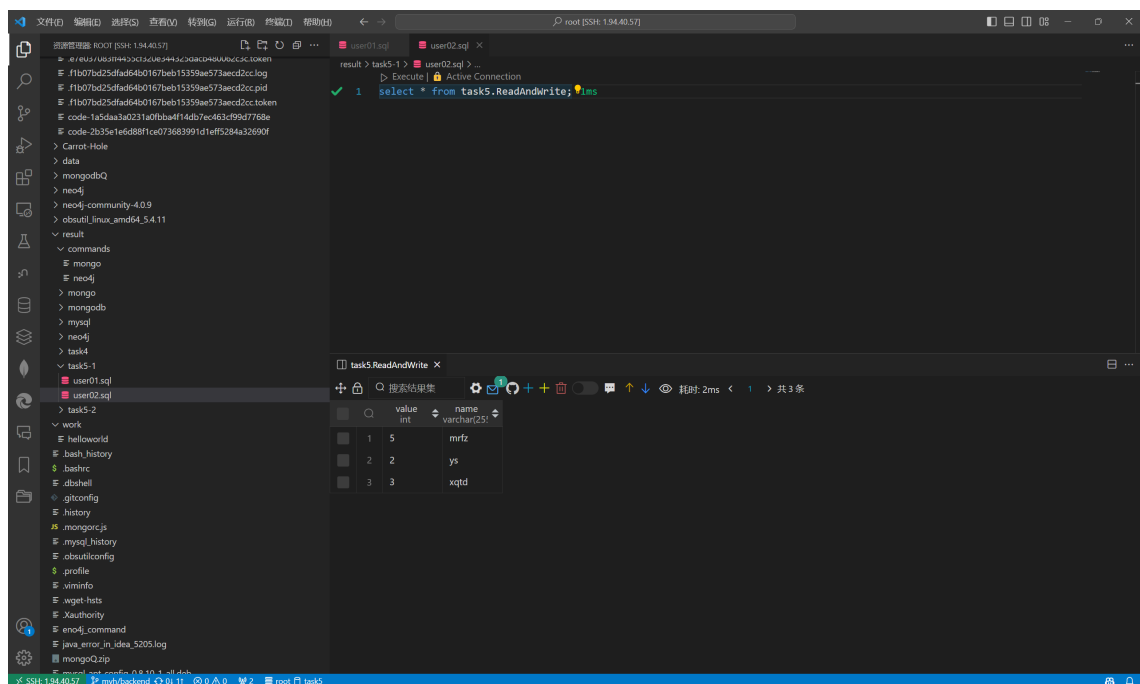


图 14: 事务 2

再次执行事务二，发现 mrfz 的值已经变成 5 了。

事务2两次执行没有改变mrfz的值但是读出来的值不同,这在生产环境中会导致错误和bug的产生,因此需要使用命令行把事务隔离级别设置为可重复读(REPEATABLE READ)。

以下是切换了事务隔离等级以后的情况。首先执行事务2:

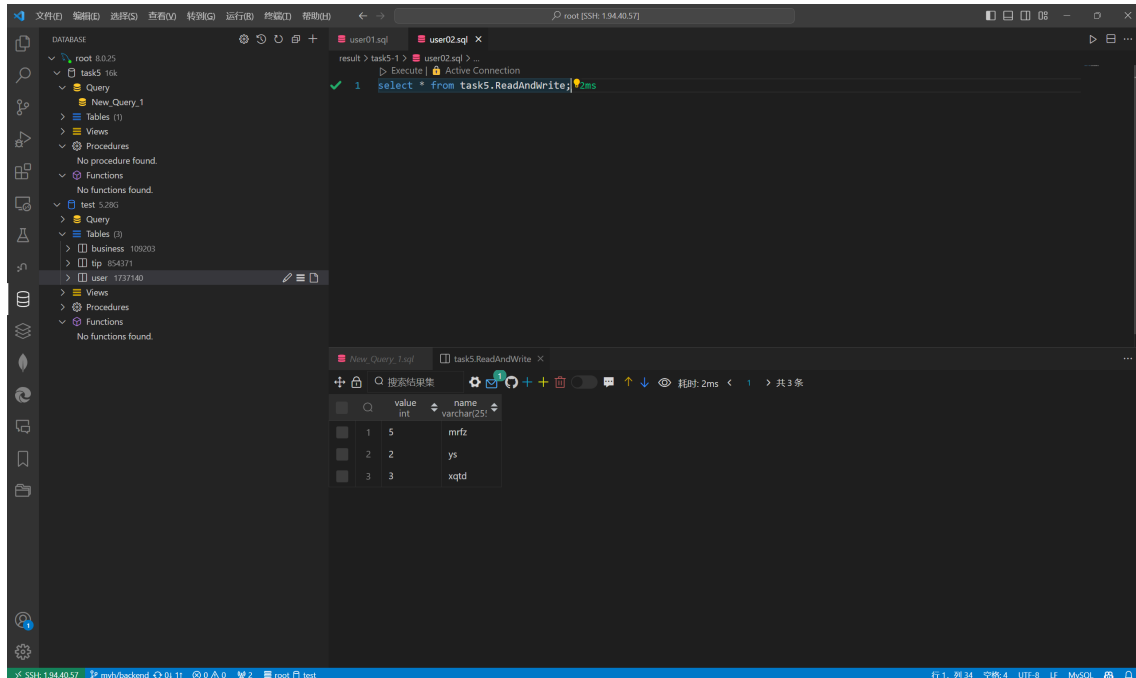


图 15: 事务 2

然后执行事务 1 发现 mrfz 的值为 6, 然后修改 mrfz 的值为 6.

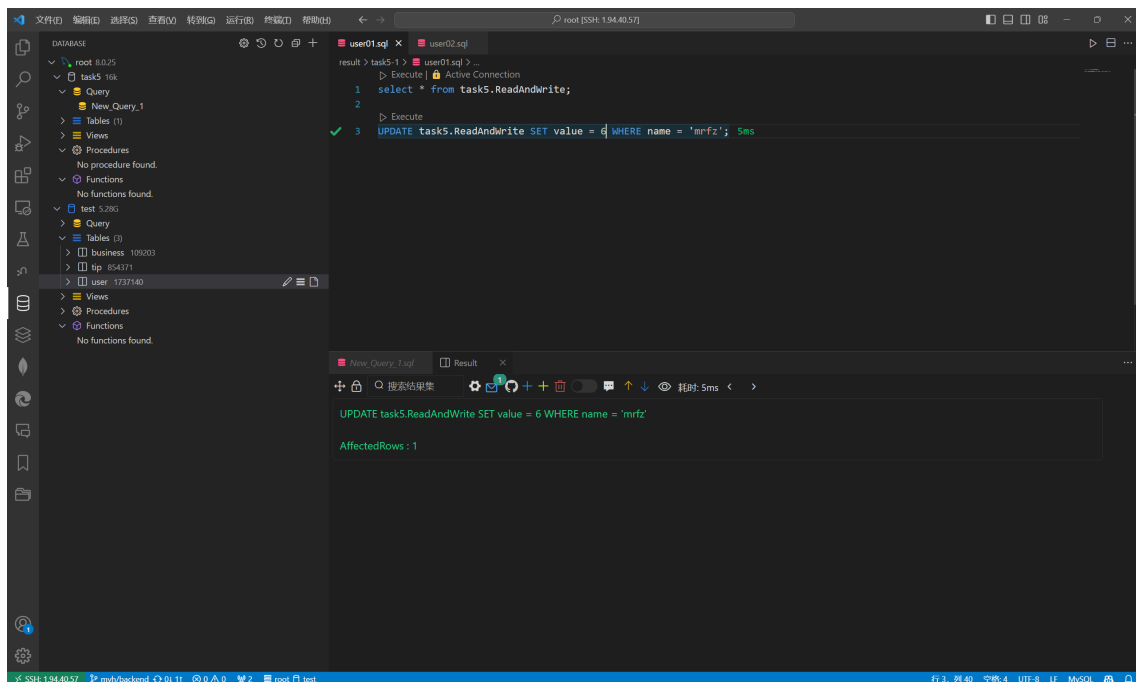


图 16: 事务 1

最后使用事务 2 读取, 发现 mrfz 的值仍然是 5:

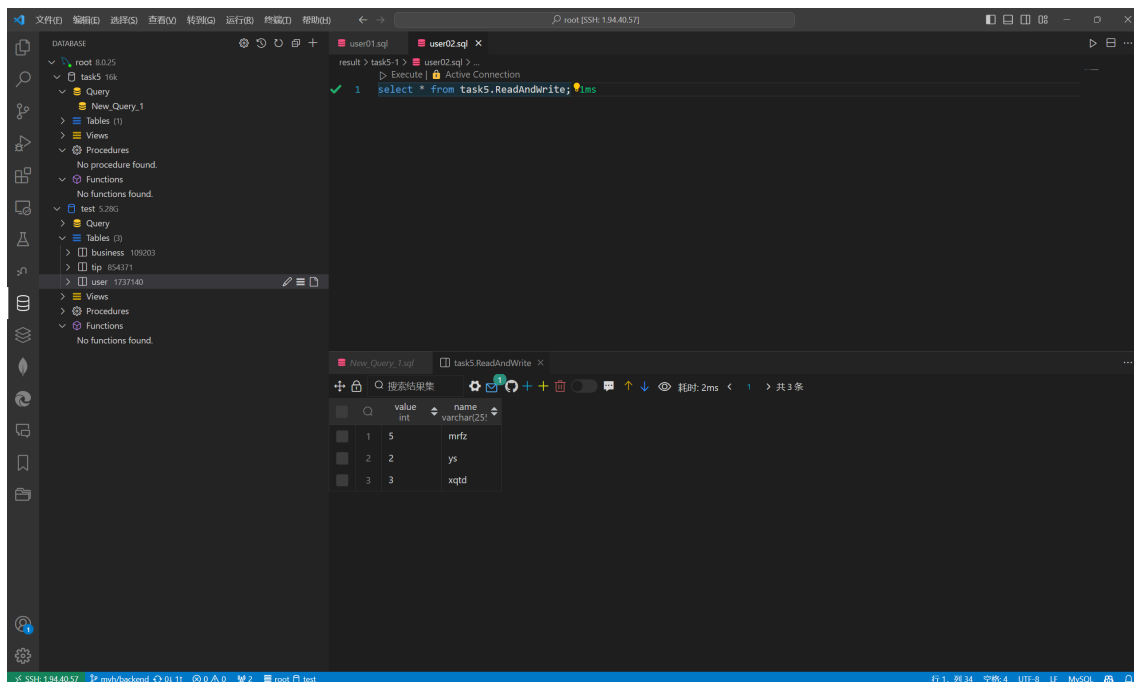


图 17: 事务 2

由此可见设置可重复读的事务隔离级别以后，事务 2 读取的值不会随着事务 1 的修改而改变，这就是 MVCC 的多版本并发控制。可以解决不可重复读问题。

同理，写写问题就是事务 2 改变了某个值，而事务 1 又改变了这个值，因此覆盖了事务 2 的修改，这里可以通过对数据加锁来完成。

7.3 任务小结

原理：在“可重复读”事务隔离级别下，MySQL 使用 MVCC 来处理并发事务。MVCC 通过创建数据的快照视图（snapshot view）来实现事务的隔离性。每个事务在开始时创建一个一致性的快照视图，该视图会记录事务开始时数据库中的数据状态。事务执行期间，其他事务对数据的修改不会对当前事务的查询结果产生影响。“可重复读”事务隔离级别确保了每个事务在整个过程中看到的数据是一致的，即使其他事务对数据进行了修改。这种隔离级别适用于多用户并发访问数据库的场景，并保证了事务的 ACID 特性。通过选择“可重复读”事务隔离级别，MySQL 能够支持 MVCC 并发控制，保证了事务的一致性和隔离性。

8 课程总结

本次实验通过数十个不同的题目，配置并且使用了不同种类的数据库，例如文档数据库，图数据库，关系型数据库等。

通过这些题目，我对不同种类的数据库有了更深入的了解，也对数据库的使用有了更多的经验。更加了解不同数据库的适应范围和优势领域，以及为什么我们需要不同的数据库：因为大数据时代的到来，数据量越来越大，数据的类型也越来越多，适用范围单一的关系型数据库以及越来越不能满足性能和更多需求的要求了。

通过大数据管理实验这门课，我对数据库领域的知识和能力都获得了提高和加强，很满意。