# neo4j图数据库

## neo4j配置

> 详细配置见**neo4j.conf**文件中

## 建表语句

### Movies表

```
LOAD CSV WITH HEADERS FROM 'file:///Movie_Table.csv' AS row
CREATE (:Movies {
  Movie_ID: toInteger(row.New_Productid),
  Movie_Title: trim(row.Movie_Title),
  Average_Score: toFloat(row.Average_Score),
  Day: toInteger(row.Day),
  Month: toInteger(row.Month),
  Year: toInteger(row.Year),
  Reviews_Count: toInteger(row.Sample_Count)
})
```

### Actors表

```
LOAD CSV WITH HEADERS FROM 'file:///Actor_Mapping_Table1.csv' AS
row
CREATE (:Actors {
  Actor_ID: toInteger(row.New_Actor_ID),
  Actor_Name: trim(row.Actor_Name)
})
```

### Directors表

```
LOAD CSV WITH HEADERS FROM 'file:///Directors_Mapping_Table.csv'
AS row
CREATE (:Directors {
  Director_ID: toInteger(row.Director_ID_New),
  Director_Name: trim(row.Director_Name)
})
```

### Formats表

```
LOAD CSV WITH HEADERS FROM 'file:///Format_Table.csv' AS row
CREATE (:Formats {
  Format_ID: trim(row.Format_ID),
  Format: trim(row.Format)
})
```

### Genres表

```
LOAD CSV WITH HEADERS FROM 'file:///Genre_Table.csv' AS row
CREATE (:Genres {
  Genre_ID: trim(row.Genre_ID),
  Genre: trim(row.Genre)
})
```

### Reviews表

```
LOAD CSV WITH HEADERS FROM 'file:///Review_36.csv' AS row
CREATE (:Reviews {
  Review_ID: toInteger(row.reviewId),
  Score: toFloat(row.score)
})
```

## 建立关联关系语句

### HAS_FORMAT

```
LOAD CSV WITH HEADERS FROM 'file:///Movie_Format_Relationship.csv'
AS row
MATCH (m:Movies {Movie_ID: toInteger(row.New_Movie_ID)})
MATCH (f:Formats {Format_ID: trim(row.Format_ID)})
MERGE (m)-[:HAS_FORMAT]->(f);
```

## HAS_GENRE

```
LOAD CSV WITH HEADERS FROM 'file:///Movie_Genre_Relationship.csv'
AS row
MATCH (m:Movies {Movie_ID: toInteger(row.New_Movie_ID)})
MATCH (g:Genres {Genre_ID: trim(row.Genre_ID)})
MERGE (m)-[:HAS_GENRE]->(g);
```

## ACT

```
LOAD CSV WITH HEADERS FROM 'file:///All_Actor_Relationship8.csv'
AS row
MATCH (a:Actors {Actor_ID: toInteger(row.New_Actor_ID)})
MATCH (m:Movies {Movie_ID: toInteger(row.New_Productid)})
MERGE (a)-[:ACT]->(m);
```

## STAR

```
LOAD CSV WITH HEADERS FROM
'file:///Starring_Actor_Relationship.csv' AS row
MATCH (a:Actors {Actor_ID: toInteger(row.New_Actor_ID)})
MATCH (m:Movies {Movie_ID: toInteger(row.New_Productid)})
MERGE (a)-[:STAR]->(m);
```

## DIRECT

```
LOAD CSV WITH HEADERS FROM 'file:///Direct_Movie_Relationship.csv'
AS row
MATCH (d:Directors {Director_ID: toInteger(row.Director_ID)})
MATCH (m:Movies {Movie_ID: toInteger(row.Movie_ID)})
MERGE (d)-[:DIRECT]->(m);
```

## HAS_REVIEW

```
LOAD CSV WITH HEADERS FROM 'file:///Review.csv' AS row
MATCH (m:Movies {Movie_ID: toInteger(row.productId)})
MATCH (r:Reviews {Review_ID: toInteger(row.reviewId)})
MERGE (m)-[:HAS_REVIEW]->(r);
```

# 查询语句

## 按照时间进行查询及统计

1. XX年有多少电影

```
MATCH (m:Movies)
WHERE m.Year = 2020
RETURN count(m) AS movies_in_2020
```

2. XX年XX月有多少电影

```
MATCH (m:Movies)
WHERE m.Year = 2020 AND m.Month = 5
RETURN count(m) AS movies_in_may_2020
```

3. XX年XX季度有多少电影

```
MATCH (m:Movies)
WHERE m.Year = 2020 AND m.Month >= 1 AND m.Month <= 3
RETURN count(m) AS movies_in_q1_2020
```

4. 周二新增多少电影

```
MATCH (m:Movies)
WHERE m.Year = 2004 AND m.Month = 1 AND m.Day = 20
RETURN count(m) AS movies_in_Tues_2004
```

## 按照电影名称进行查询及统计

1. XX电影共有多少版本等

```
MATCH (m:Movies)-[:HAS_FORMAT]->(f:Formats)
WHERE m.Movie_Title = "Marvel's The Avengers"
RETURN count(DISTINCT f) AS versions_of_MarvelsTheAvengers
```

## 按照导演进行查询及统计

1. XX导演共有多少电影

```
MATCH (d:Directors)-[:DIRECT]->(m:Movies)
WHERE d.Director_Name = "Christopher Nolan"
RETURN count(m) AS movies_directed_by_Nolan
```

## 按照演员进行查询及统计

1. XX演员主演多少电影

```
MATCH (a:Actors)-[:STAR]->(m:Movies)
WHERE a.Actor_Name = "Andrew Garfield"
RETURN count(DISTINCT m) AS movies_stared_by_Andrew_Garfield
```

2. XX演员参演多少电影

```
MATCH (a:Actors)-[:ACT]->(m:Movies)
WHERE a.Actor_Name = "Andrew Garfield"
RETURN count(DISTINCT m) AS movies_stared_by_Andrew_Garfield
```

## 按照电影类别进行查询及统计

1. Action 电影共有多少

```
MATCH (m:Movies)-[:HAS_GENRE]->(g:Genres)
WHERE g.Genre = "Action"
RETURN count(m) AS action_movies_count
```

2. Adventure 电影共有多少

```
MATCH (m:Movies)-[:HAS_GENRE]->(g:Genres)
WHERE g.Genre = "Adventure"
RETURN count(m) AS action_movies_count
```

## 按照用户评价进行查询及统计

1. 用户评分3分以上的电影有哪些

```
MATCH (m:Movies)
WHERE m.Average_Score > 3
RETURN m.Movie_Title AS movies_reviews_above_3
```

2. 用户评价中有正面评价的电影有哪些

```
MATCH (m:Movies)-[:HAS_REVIEW]->(r:Reviews)
WHERE r.Score = 5
WITH m.Movie_Title AS title
RETURN DISTINCT title
ORDER BY title
```

Started streaming 66563 records after 10 ms and completed after 1315 ms

## 按照演员、导演之间的关系进行查询及统计

1. 经常合作的演员有哪些

   为合作创建关联：

```
CALL apoc.periodic.iterate(
"MATCH (a1:Actors)-[:ACT]->(m:Movies)<-[:ACT]-(a2:Actors)
RETURN a1, a2, COUNT(m) AS collaborations",
"MERGE (a1)-[r:COOPERATED]->(a2)
ON CREATE SET r.count = collaborations
ON MATCH SET r.count = r.count + collaborations",
{batchSize: 50, retries: 20, parallel: true}
);
```

```
MATCH (a1:Actors)-[r:COOPERATED]-(a2:Actors)
WHERE r.count > 20
RETURN a1.Actor_Name, a2.Actor_Name, r.count
ORDER BY r.count DESC;
```

2. 经常合作的导演和演员有哪些

```
MATCH (a:Actors)-[r:COOPERATED_WITH_DIRECTOR]->(d:Directors)
WHERE r.count > 20
RETURN a.Actor_Name AS Actor, d.Director_Name AS Director,
r.count AS Collaborations
ORDER BY r.count DESC;
```

3. 如果要拍一部XXX类型的电影，最受关注（评论最多）的演员组合（2人）是什么

```
MATCH (m:Movies)-[:HAS_GENRE]->(g:Genres {Genre: 'Action'})
WITH m
MATCH (m)<-[:ACT]-(a:Actors)
WITH m, COUNT(a) AS actor_count
WHERE actor_count >= 2
SET m:FilteredMovie;

MATCH (m:FilteredMovie)<-[:ACT]-(a1:Actors)
MATCH (m)<-[:ACT]-(a2:Actors)
WHERE a1.Actor_ID < a2.Actor_ID
WITH a1, a2, SUM(m.Reviews_Count) AS total_reviews
RETURN a1.Actor_Name AS Actor1, a2.Actor_Name AS Actor2,
total_reviews
```

```
ORDER BY total_reviews DESC
LIMIT 1;
```

4. 如果要拍一部XXX类型的电影，最受关注（评论最多）的演员组合（3人）是什么

```
MATCH (m:Movies)-[:HAS_GENRE]->(g:Genres {Genre: 'Action'})
WITH m
MATCH (m)<-[:ACT]-(a:Actors)
WITH m, COUNT(a) AS actor_count
WHERE actor_count >= 3
SET m:FilteredMovie;

MATCH (m:FilteredMovie)<-[:ACT]-(a:Actors)
WITH m, COLLECT(a.Actor_Name) AS actor_names, m.Reviews_Count
AS reviews
WHERE SIZE(actor_names) >= 3
UNWIND apoc.coll.combinations(actor_names, 3) AS trio
RETURN trio, SUM(reviews) AS total_reviews
ORDER BY total_reviews DESC
LIMIT 1;
```

## 按照上述条件的组合查询和统计

1. XX演员参演了几部XX类型的电影

```
MATCH (a:Actors {Actor_Name: "Andrew Garfield"})-[:ACT]->
(m:Movies)-[:HAS_GENRE]->(g:Genres {Genre: "Action"})
RETURN count(m);
```

2. XX演员在XX年参演了多少电影

```
MATCH (a:Actors {Actor_Name: "Andrew Garfield"})-[:ACT]->
(m:Movies {Year: 2012})
RETURN count(m);
```

# 图数据库查询优化

1. 为节点的ID创建索引

### Movie_ID、Movie_Title、Average_Score

```
CREATE INDEX FOR (m:Movies) ON (m.Movie_ID);
CREATE TEXT INDEX FOR (m:Movies) ON (m.Movie_Title);
CREATE INDEX FOR (m:Movies) ON (m.Average_Score);
```

### Format_ID

```
CREATE INDEX FOR (f:Formats) ON (f.Format_ID);
```

### Genre_ID、Genre

```
CREATE INDEX FOR (g:Genres) ON (g.Genre_ID);
CREATE TEXT INDEX FOR (g:Genres) ON (g.Genre);
```

### Actor_ID、Actor_Name

```
CREATE INDEX FOR (a:Actors) ON (a.Actor_ID);
CREATE TEXT INDEX FOR (a:Actors) ON (a.Actor_Name);
```

### Review_ID、Score

```
CREATE INDEX FOR (r:Reviews) ON (r.Review_ID);
CREATE INDEX FOR (r:Reviews) ON (r.Score);
```

### Director_ID、Director_Name

```
CREATE INDEX FOR (r:Directors) ON (r.Director_ID);
CREATE TEXT INDEX FOR (d:Directors) ON (d.Director_Name);
```

2. 修改图数据库配置
   - 指定 Neo4j 的 Java 虚拟机（JVM）堆内存的初始大小，设置为 **4096MB（4GB）**。

     ```
     server.memory.heap.initial_size=4096m
     ```

     > **作用**：初始堆内存大小影响 JVM 启动时分配的内存空间，设置一个较大的初始值可以减少动态内存调整的开销。

   - 指定 Neo4j 的 JVM 堆内存的最大大小，设置为 **8192MB（8GB）**。

```
server.memory.heap.max_size=8192m
```

> **作用**：限制堆内存的最大使用量，防止 JVM 使用过多内存导致系统资源耗尽。

- 指定 Neo4j 的事务内存池的最大值，设置为 **8192MB（8GB）**。

```
dbms.memory.transaction.total.max=8192m
```

> **作用**：控制事务处理所能使用的最大内存量，用于管理正在处理的事务数据。如果超出这个限制，Neo4j 会抛出内存不足的错误。

- 指定 Neo4j 的 **页面缓存（Page Cache）** 的大小，设置为 **6GB**。

```
server.memory.pagecache.size=6g
```

> **作用**：页面缓存是用来存储磁盘上节点和关系数据的缓冲区。较大的页面缓存有助于减少磁盘 I/O，从而提高数据库查询的性能。

3. 在合作查询中，使用**APOC**创建合作关系

- 演员与演员的合作，通过设置批次和重试次数提升关联关系创建的速率，最终失败的批次数为0。

```
CALL apoc.periodic.iterate(
"MATCH (a1:Actors)-[:ACT]->(m:Movies)<-[:ACT]-(a2:Actors)
RETURN a1, a2, COUNT(m) AS collaborations",
"MERGE (a1)-[r:COOPERATED]->(a2)
ON CREATE SET r.count = collaborations
ON MATCH SET r.count = r.count + collaborations",
{batchSize: 50, retries: 20, parallel: true}
);
```

- 演员与导演的合作，通过设置批次和重试次数提升关联关系创建的速率，最终失败的批次数为0。

```
CALL apoc.periodic.iterate(
"MATCH (a:Actors)-[:ACT]->(m:Movies)<-[:DIRECT]-
(d:Directors)
RETURN a, d, COUNT(m) AS collaborations",
"MERGE (a)-[r:COOPERATED_WITH_DIRECTOR]->(d)
ON CREATE SET r.count = collaborations
ON MATCH SET r.count = r.count + collaborations",
{batchSize: 50, retries: 20, parallel: true}
);
```

4. 批量筛选电影的预处理
    - 通过标签临时标记符合条件的电影

```
MATCH (m:Movies)-[:HAS_GENRE]->(g:Genres {Genre: 'XXX'})
WITH m
MATCH (m)<-[:ACT]-(a:Actors)
WITH m, COUNT(a) AS actor_count
WHERE actor_count >= 2
SET m:FilteredMovie;
```

```
MATCH (m:Movies)-[:HAS_GENRE]->(g:Genres {Genre: 'Action'})
WITH m
MATCH (m)<-[:ACT]-(a:Actors)
WITH m, COUNT(a) AS actor_count
WHERE actor_count >= 3
SET m:FilteredMovie;
```