MySQL优化

查询优化

查询经常合作的演员

```
SELECT A1.Actor_Name AS Actor1, A2.Actor_Name AS Actor2,
c.Cooperation_Count
FROM (
    SELECT AA1.Actor_ID AS Actor1_ID, AA2.Actor_ID AS Actor2_ID,
COUNT(*) AS Cooperation_Count
    FROM All_Actor AA1
    JOIN All_Actor AA2 ON AA1.Movie_ID = AA2.Movie_ID
                      AND AA1.Actor_ID < AA2.Actor_ID
    GROUP BY AA1.Actor_ID, AA2.Actor_ID
    HAVING COUNT(*) > 20
) c
JOIN Actor A1 ON c.Actor1_ID = A1.Actor_ID
JOIN Actor A2 ON c.Actor2_ID = A2.Actor_ID
ORDER BY c.Cooperation_Count DESC;
```

查询经常合作的演员和导演

```
SELECT D.Director_Name, A.Actor_Name, T.Cooperation_Count
FROM (
    SELECT MD.Director_ID, AA.Actor_ID, COUNT(*) AS
Cooperation_Count
    FROM Direct_Movie AS MD
    JOIN All_Actor AS AA ON MD.Movie_ID = AA.Movie_ID
    GROUP BY MD.Director_ID, AA.Actor_ID
    HAVING COUNT(*) > 10
) AS T
JOIN Director AS D ON T.Director_ID = D.Director_ID
JOIN Actor AS A ON T.Actor_ID = A.Actor_ID
ORDER BY T.Cooperation_Count DESC;
```

以上两个查询通过分步处理（先过滤、后连接），减少了主查询中的数据量和连接操作的复杂度。

存储优化

拆分大表Actor

```
--将 All_Actor 分成 4 个子表
CREATE TABLE All_Actor_Movie_1 (
    Actor_ID INT,
    Movie_ID INT,
    PRIMARY KEY (Movie_ID, Actor_ID),
    FOREIGN KEY (Actor_ID) REFERENCES Actor(Actor_ID),
    FOREIGN KEY (Movie_ID) REFERENCES Movie(Movie_ID)
);

CREATE TABLE All_Actor_Movie_2 LIKE All_Actor_Movie_1;
CREATE TABLE All_Actor_Movie_3 LIKE All_Actor_Movie_1;
CREATE TABLE All_Actor_Movie_4 LIKE All_Actor_Movie_1;

INSERT INTO All_Actor_Movie_1 (Actor_ID, Movie_ID)
SELECT Actor_ID, Movie_ID FROM All_Actor WHERE MOD(Movie_ID, 4) =
1;

INSERT INTO All_Actor_Movie_2 (Actor_ID, Movie_ID)
SELECT Actor_ID, Movie_ID FROM All_Actor WHERE MOD(Movie_ID, 4) =
2;

INSERT INTO All_Actor_Movie_3 (Actor_ID, Movie_ID)
SELECT Actor_ID, Movie_ID FROM All_Actor WHERE MOD(Movie_ID, 4) =
3;

INSERT INTO All_Actor_Movie_4 (Actor_ID, Movie_ID)
SELECT Actor_ID, Movie_ID FROM All_Actor WHERE MOD(Movie_ID, 4) =
0;
```

将Review表分区

```
CREATE TABLE Review_Partitioned (
    Movie_ID INT,
    Review_ID INT,
    Score INT,
    PRIMARY KEY (Review_ID, Score), -- 包含分区键 Score
    INDEX idx_movie_id (Movie_ID)
) PARTITION BY RANGE (Score) (
    PARTITION p_low VALUES LESS THAN (5),
    PARTITION p_high VALUES LESS THAN (10)
);
```

Denormalization优化

查询演员与演员，演员与导演，以及最受欢迎的演员都需要大规模的join操作，无论如何优化都还要几分钟的时间（取决于硬件）。现在通过去范式化，建立冗余信息表，可最快速度地提高查询效率。

查询经常合作的演员

```

CREATE TABLE Actor_Cooperation_MV (
    Actor1_ID INT NOT NULL,
    Actor2_ID INT NOT NULL,
    Cooperation_Count INT NOT NULL,
    PRIMARY KEY (Actor1_ID, Actor2_ID)
);
INSERT INTO Actor_Cooperation_MV
SELECT
    AA1.Actor_ID AS Actor1_ID,
    AA2.Actor_ID AS Actor2_ID,
    COUNT(*) AS Cooperation_Count
FROM All_Actor AA1
JOIN All_Actor AA2 ON AA1.Movie_ID = AA2.Movie_ID AND AA1.Actor_ID
< AA2.Actor_ID
GROUP BY AA1.Actor_ID, AA2.Actor_ID;

CREATE INDEX idx_cooperation_count ON Actor_Cooperation_MV
(Cooperation_Count);

SELECT A1.Actor_Name AS Actor1, A2.Actor_Name AS Actor2,
MV.Cooperation_Count
FROM Actor_Cooperation_MV MV
JOIN Actor A1 ON MV.Actor1_ID = A1.Actor_ID
JOIN Actor A2 ON MV.Actor2_ID = A2.Actor_ID
```

```
WHERE MV.Cooperation_Count > 24
ORDER BY MV.Cooperation_Count DESC;
```

查询经常合作的演员和导演

```
CREATE TABLE Director_Actor_Cooperation AS
SELECT MD.Director_ID, AA.Actor_ID, COUNT(*) AS Cooperation_Count
FROM Direct_Movie MD
JOIN All_Actor AA ON MD.Movie_ID = AA.Movie_ID
GROUP BY MD.Director_ID, AA.Actor_ID;

SELECT D.Director_Name, A.Actor_Name, C.Cooperation_Count
FROM Director_Actor_Cooperation C
JOIN Director D ON C.Director_ID = D.Director_ID
JOIN Actor A ON C.Actor_ID = A.Actor_ID
WHERE C.Cooperation_Count > 10
ORDER BY C.Cooperation_Count DESC;
```

拍一部XX类型的电影，最受欢迎的演员组合

```
DROP TABLE IF EXISTS Movie_Comments;
CREATE TABLE Movie_Comments AS
SELECT Movie_ID, SUM(Comment_Num) AS Movie_Comment_Sum
FROM Movie
GROUP BY Movie_ID;

-- 为查询加速建立索引
CREATE INDEX idx_movie_comments_movieid ON
Movie_Comments(Movie_ID);

DROP TABLE IF EXISTS Movie_ActorPairs;
CREATE TABLE Movie_ActorPairs AS
SELECT
    AA1.Movie_ID,
    AA1.Actor_ID AS ActorID1,
    AA2.Actor_ID AS ActorID2
FROM All_Actor AA1
JOIN All_Actor AA2
   ON AA1.Movie_ID = AA2.Movie_ID
  AND AA1.Actor_ID < AA2.Actor_ID;

-- 创建索引加快后续连接
CREATE INDEX idx_movie_actorpairs_movieid ON
Movie_ActorPairs(Movie_ID, ActorID1, ActorID2);
```

```sql
DROP TABLE IF EXISTS ActorPair_GenreComments_ID;
CREATE TABLE ActorPair_GenreComments_ID AS
SELECT
    MP.ActorID1,
    MP.ActorID2,
    MG.Genre_ID,
    MC.Movie_Comment_Sum AS Total_Comments
FROM Movie_ActorPairs MP
JOIN Movie_Genre MG ON MP.Movie_ID = MG.Movie_ID
JOIN Movie_Comments MC ON MP.Movie_ID = MC.Movie_ID;

-- 加索引便于后续查询或关联
CREATE INDEX idx_actorpairgenrecommentsid_actorids ON
ActorPair_GenreComments_ID(ActorID1, ActorID2, Genre_ID);

CREATE INDEX idx_genre_totalcomments_desc
ON ActorPair_GenreComments_ID (Genre_ID, Total_Comments DESC);



SELECT
    (SELECT Actor_Name FROM Actor WHERE Actor_ID = APCI.ActorID1)
AS Actor1,
    (SELECT Actor_Name FROM Actor WHERE Actor_ID = APCI.ActorID2)
AS Actor2,
    (SELECT Genre FROM Genre WHERE Genre_ID = APCI.Genre_ID) AS
Genre,
    APCI.Total_Comments
FROM ActorPair_GenreComments_ID AS APCI
WHERE APCI.Genre_ID = (SELECT Genre_ID FROM Genre WHERE BINARY
Genre = 'Action')
ORDER BY APCI.Total_Comments DESC
LIMIT 1;
```

通过提前计算复杂查询的结果并存储到专门的表中，避免了每次查询时的大量 JOIN 和聚合操作，提高了查询速度。这种方式适用于频繁的、固定模式的查询场景。