356 Lab 2

Part 2: Baseball query evaluation using SQL explain statement and optimization using index

a)

Query:

```
SELECT count(*) AS unkown_birthdate_count
FROM Master
WHERE birthYear = 0
  OR birthMonth = 0
  OR birthDay = 0;
```

Explain Result:

```
+----+-------------+--------+------+---------------+------+---------+------+-------+-------------+
| id | select_type | table  | type | possible_keys | key  | key_len | ref  | rows  | Extra       |
+----+-------------+--------+------+---------------+------+---------+------+-------+-------------+
|  1 | SIMPLE      | Master | ALL  | NULL          | NULL | NULL    | NULL | 19087 | Using where |
+----+-------------+--------+------+---------------+------+---------+------+-------+-------------+
1 row in set (0.21 sec)
```

From the explain result, the type is “ALL”, which means the entire table is scanned to find matching rows. Therefore, we need to add appropriate indexes on the table.
From the query, we can see that it is searching using three attributes birthYear or birthMonth or birthDay. Therefore, we can should separate indexes for those three attributes.

Add Index:

```
CREATE INDEX birthYear_index ON Master (birthYear) USING BTREE;
CREATE INDEX birthMonth_index ON Master (birthMonth) USING BTREE;
CREATE INDEX birthDay_index ON Master (birthDay) USING BTREE;
```

After adding the index, let’s check the explain result again:

```
mysql> EXPLAIN SELECT count(*) AS unkown_birthdate_count FROM Master WHERE birthYear = 0   OR birthMonth = 0   OR birthDay = 0;
+----+-------------+--------+------------+-----------------------------+-----------------------------+---------+------+------+----------------------------------------------------------------+
| id | select_type | table  | type       | possible_keys               | key                         | key_len | ref  | rows | Extra                                                          |
+----+-------------+--------+------------+-----------------------------+-----------------------------+---------+------+------+----------------------------------------------------------------+
|  1 | SIMPLE      | Master | index_merge | birthYear,birthMonth,birthDay | birthYear,birthMonth,birthDay | 5,5,5   | NULL |  882 | Using union(birthYear,birthMonth,birthDay); Using where |
+----+-------------+--------+------------+-----------------------------+-----------------------------+---------+------+------+----------------------------------------------------------------+
1 row in set (0.11 sec)
```

From the explain result, it is obvious that the number of rows will be examined reduces from 19087 to 882 and the type changes from ALL to index_merge. Therefore, the query performance has been improved.

b)

Query:

```
SELECT
(
    (SELECT COUNT(distinct HallOfFame.playerID) FROM HallOfFame inner join Master on
HallOfFame.playerID = Master.playerID where Master.deathYear = 0 )
-
    (SELECT COUNT(distinct HallOfFame.playerID) FROM HallOfFame inner join Master on
HallOfFame.playerID = Master.playerID where Master.deathYear > 0)
)
AS 'Alive - Dead';
```

Explain Result:

```
+----+-------------+------------+--------+---------------+---------+---------+-----------------------------------+------+----------------+
| id | select_type | table      | type   | possible_keys | key     | key_len | ref                               | rows | Extra          |
+----+-------------+------------+--------+---------------+---------+---------+-----------------------------------+------+----------------+
|  1 | PRIMARY     | NULL       | NULL   | NULL          | NULL    | NULL    | NULL                              | NULL | No tables used |
|  3 | SUBQUERY    | HallOfFame | index  | PRIMARY       | PRIMARY | 1538    | NULL                              | 4155 | Using index    |
|  3 | SUBQUERY    | Master     | eq_ref | PRIMARY       | PRIMARY | 767     | db356_z498zhan.HallOfFame.playerID |    1 | Using where    |
|  2 | SUBQUERY    | HallOfFame | index  | PRIMARY       | PRIMARY | 1538    | NULL                              | 4155 | Using index    |
|  2 | SUBQUERY    | Master     | eq_ref | PRIMARY       | PRIMARY | 767     | db356_z498zhan.HallOfFame.playerID |    1 | Using where    |
+----+-------------+------------+--------+---------------+---------+---------+-----------------------------------+------+----------------+
5 rows in set (0.00 sec)
```

Based on the query, we can see that there is a where clause using Master.deathYear. Therefore, we should add index on attribute deathYear in Master table.
We can also see that HallOfFame and Master inner join on “playerID”. Since playerID is a primary key in Master table, we should add index on playerID in HallofFame table.

Add Index:

```
CREATE INDEX deathYear_index ON Master (deathYear) USING BTREE;
CREATE INDEX playerID_index ON HallOfFame (playerID) USING BTREE;
```

After adding the index, let’s check the explain result again:

```
+----+-------------+------------+--------+-------------------------+---------------+---------+-----------------------------------+------+----------------+
| id | select_type | table      | type   | possible_keys           | key           | key_len | ref                               | rows | Extra          |
+----+-------------+------------+--------+-------------------------+---------------+---------+-----------------------------------+------+----------------+
|  1 | PRIMARY     | NULL       | NULL   | NULL                    | NULL          | NULL    | NULL                              | NULL | No tables used |
|  3 | SUBQUERY    | HallOfFame | index  | PRIMARY,playerID_index  | playerID_index | 767     | NULL                              | 4155 | Using index    |
|  3 | SUBQUERY    | Master     | eq_ref | PRIMARY,deathYear_index | PRIMARY       | 767     | db356_z498zhan.HallOfFame.playerID |    1 | Using where    |
|  2 | SUBQUERY    | HallOfFame | index  | PRIMARY,playerID_index  | playerID_index | 767     | NULL                              | 4155 | Using index    |
|  2 | SUBQUERY    | Master     | eq_ref | PRIMARY,deathYear_index | PRIMARY       | 767     | db356_z498zhan.HallOfFame.playerID |    1 | Using where    |
+----+-------------+------------+--------+-------------------------+---------------+---------+-----------------------------------+------+----------------+
5 rows in set (0.03 sec)
```

The number of rows will be examined does not change. Because the count function and the where clause is after the inner join.
The key_len is reduced from 1538 to 767. Therefore, the query performance has been improved slightly.

C)

Query:

```
SELECT nameFirst,
       nameLast,
       temp.total_salary
FROM Master
INNER JOIN
  (SELECT playerID,
          sum(salary) AS total_salary
   FROM Salaries
   GROUP BY playerID) AS temp ON Master.playerID = temp.playerID
ORDER BY temp.total_salary DESC
LIMIT 1;
```

Explain Result:

```
+----+-------------+------------+--------+-----------------------------------------+-------------------+---------+--------------+-------+----------------+
| id | select_type | table      | type   | possible_keys                           | key               | key_len | ref          | rows  | Extra          |
+----+-------------+------------+--------+-----------------------------------------+-------------------+---------+--------------+-------+----------------+
|  1 | PRIMARY     | <derived2> | ALL    | NULL                                    | NULL              | NULL    | NULL         | 26110 | Using filesort |
|  1 | PRIMARY     | Master     | eq_ref | PRIMARY                                 | PRIMARY           | 767     | temp.playerID |     1 | NULL           |
|  2 | DERIVED     | Salaries   | index  | PRIMARY,fk_Salaries_Teams,fk_Salaries_Master | fk_Salaries_Master | 767     | NULL         | 26110 | NULL           |
+----+-------------+------------+--------+-----------------------------------------+-------------------+---------+--------------+-------+----------------+
3 rows in set (0.02 sec)
```

As we can see, for the derived table, we have the worst case of join type: ALL for this query.
However, since it is a derived table, we should only care about the performance of the base tables.
For Master table, the type is eq_ref, which means it is optimized by its primary key which we added in the first lab.
For the Salary table, it is already using the index. Which means there are no additional explicit indexes needed for this query.

D)

Query:

```
SELECT avg(total) AS avg_HR
FROM
  (SELECT sum(HR) AS total
   FROM Batting
   GROUP BY playerID) AS temp
```

Explain Result:

```
+----+-------------+------------+-------+-----------------------------------------+-------------------+---------+------+--------+-------+
| id | select_type | table      | type  | possible_keys                           | key               | key_len | ref  | rows   | Extra |
+----+-------------+------------+-------+-----------------------------------------+-------------------+---------+------+--------+-------+
|  1 | PRIMARY     | <derived2> | ALL   | NULL                                    | NULL              | NULL    | NULL | 102216 | NULL  |
|  2 | DERIVED     | Batting    | index | PRIMARY,fk_Batting_Teams,fk_Batting_Master | fk_Batting_Master | 767     | NULL | 102216 | NULL  |
+----+-------------+------------+-------+-----------------------------------------+-------------------+---------+------+--------+-------+
2 rows in set (0.00 sec)
```

As we can see, for the derived table, we have the worst case of join type: ALL for this query.
However, since it is a derived table, we should only care about the performance of the base tables.
In the query, we have “GROUP BY playerID” and playerID is a foreign key in Batting table, which means playerID is indexed already. Therefore, there are no additional explicit indexes needed for this query.

E)

Query:

```
SELECT avg(total) AS avg_HR
FROM
  (SELECT sum(HR) AS total
   FROM Batting
   GROUP BY playerID
   HAVING total >= 1) AS temp;
```

Explain Result:

```
+----+-------------+------------+-------+---------------------------------------------+------------------+---------+------+--------+-------+
| id | select_type | table      | type  | possible_keys                               | key              | key_len | ref  | rows   | Extra |
+----+-------------+------------+-------+---------------------------------------------+------------------+---------+------+--------+-------+
|  1 | PRIMARY     | <derived2> | ALL   | NULL                                        | NULL             | NULL    | NULL | 102356 | NULL  |
|  2 | DERIVED     | Batting    | index | PRIMARY,fk_Batting_Teams,fk_Batting_Master | fk_Batting_Master | 767     | NULL | 102356 | NULL  |
+----+-------------+------------+-------+---------------------------------------------+------------------+---------+------+--------+-------+
2 rows in set (0.01 sec)
```

From the query, we can see that the query uses “GROUP BY playerID” and “HAVING total >= 1”. Therefore, we decide to add index on attribute “playerID” and “HR” in Batting table.

```
CREATE INDEX HR_playerID_index ON Batting (HR, playerID) USING BTREE;
```

After adding the index, the explain result:

```
+----+-------------+------------+-------+-----------------------------------------------------+------------------+---------+------+--------+-------+
| id | select_type | table      | type  | possible_keys                                       | key              | key_len | ref  | rows   | Extra |
+----+-------------+------------+-------+-----------------------------------------------------+------------------+---------+------+--------+-------+
|  1 | PRIMARY     | <derived2> | ALL   | NULL                                                | NULL             | NULL    | NULL | 102356 | NULL  |
|  2 | DERIVED     | Batting    | index | PRIMARY,fk_Batting_Teams,fk_Batting_Master,HR_playerID_index | fk_Batting_Master | 767     | NULL | 102356 | NULL  |
+----+-------------+------------+-------+-----------------------------------------------------+------------------+---------+------+--------+-------+
2 rows in set (0.01 sec)
```

From the explain result, we can see that the number of rows remain unchanged, which means the index we added does not improve the query performance. Therefore, we do not need additional explicit indexes for this query

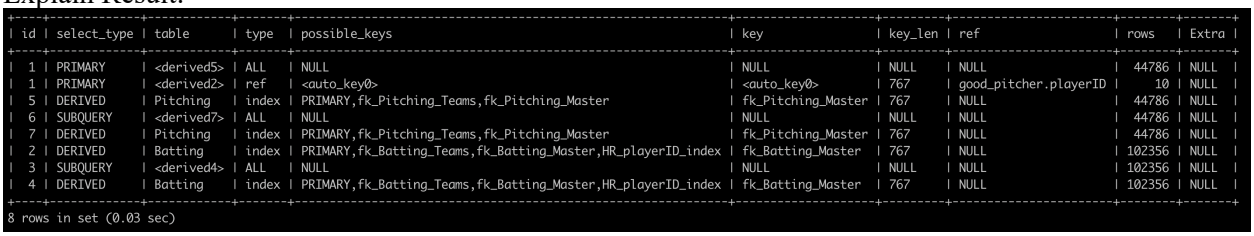
F)

Query:

```
SELECT count(*) AS good_player_count
FROM
  (SELECT playerID,
          sum(HR) AS total
   FROM Batting
   GROUP BY playerID
   HAVING total >
     (SELECT avg(total)
      FROM
        (SELECT sum(HR) AS total
         FROM Batting
         GROUP BY playerID) AS temp)) AS good_batter
INNER JOIN
  (SELECT playerID,
          sum(SHO) AS total
   FROM Pitching
   GROUP BY playerID
   HAVING total >
```

```
     (SELECT avg(total)
      FROM
        (SELECT sum(SHO) AS total
         FROM Pitching
         GROUP BY playerID) AS temp)) AS good_pitcher ON good_batter.playerID =
good_pitcher.playerID;
```

Explain Result:



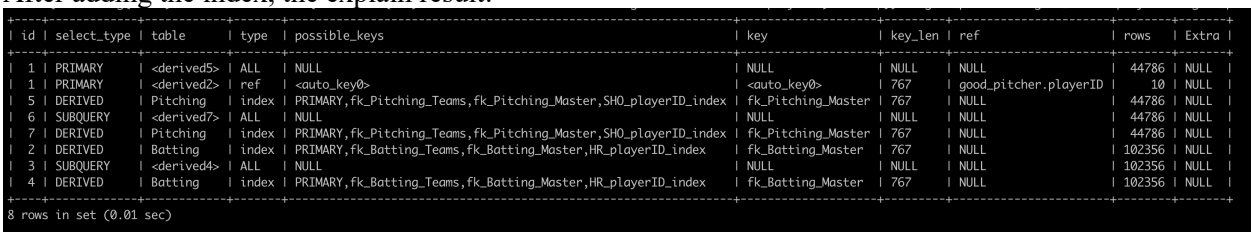
id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived5>	ALL	NULL	NULL	NULL	NULL	44786	NULL
1	PRIMARY	<derived2>	ref	<auto_key0>	<auto_key0>	767	good_pitcher.playerID	10	NULL
5	DERIVED	Pitching	index	PRIMARY,fk_Pitching_Teams,fk_Pitching_Master	fk_Pitching_Master	767	NULL	44786	NULL
6	SUBQUERY	<derived7>	ALL	NULL	NULL	NULL	NULL	44786	NULL
7	DERIVED	Pitching	index	PRIMARY,fk_Pitching_Teams,fk_Pitching_Master	fk_Pitching_Master	767	NULL	44786	NULL
2	DERIVED	Batting	index	PRIMARY,fk_Batting_Teams,fk_Batting_Master,HR_playerID_index	fk_Batting_Master	767	NULL	102356	NULL
3	SUBQUERY	<derived4>	ALL	NULL	NULL	NULL	NULL	102356	NULL
4	DERIVED	Batting	index	PRIMARY,fk_Batting_Teams,fk_Batting_Master,HR_playerID_index	fk_Batting_Master	767	NULL	102356	NULL

8 rows in set (0.03 sec)

From the query, it is obvious that playerID is used to join the good_pitcher table and good_batter table and used in “GROUP BY”. Therefore, playerID should be indexed in the Batting table and Pitching table. Besides,
Add Index:

```
CREATE INDEX HR_playerID_index ON Batting (HR, playerID) USING BTREE;
CREATE INDEX SHO_playerID_index ON Pitching (SHO, playerID) USING BTREE;
```

After adding the index, the explain result:



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived5>	ALL	NULL	NULL	NULL	NULL	44786	NULL
1	PRIMARY	<derived2>	ref	<auto_key0>	<auto_key0>	767	good_pitcher.playerID	10	NULL
5	DERIVED	Pitching	index	PRIMARY,fk_Pitching_Teams,fk_Pitching_Master,SHO_playerID_index	fk_Pitching_Master	767	NULL	44786	NULL
6	SUBQUERY	<derived7>	ALL	NULL	NULL	NULL	NULL	44786	NULL
7	DERIVED	Pitching	index	PRIMARY,fk_Pitching_Teams,fk_Pitching_Master,SHO_playerID_index	fk_Pitching_Master	767	NULL	44786	NULL
2	DERIVED	Batting	index	PRIMARY,fk_Batting_Teams,fk_Batting_Master,HR_playerID_index	fk_Batting_Master	767	NULL	102356	NULL
3	SUBQUERY	<derived4>	ALL	NULL	NULL	NULL	NULL	102356	NULL
4	DERIVED	Batting	index	PRIMARY,fk_Batting_Teams,fk_Batting_Master,HR_playerID_index	fk_Batting_Master	767	NULL	102356	NULL

8 rows in set (0.01 sec)

After adding the index, the type and rows number do not change in the explain result. This index does not improve the query performance. This is because playerID is already indexed as a foreign key in both Batting and Pitching table, and HR and SHO are only used in aggregation. Therefore, we do not need additional explicit indexes for this query.