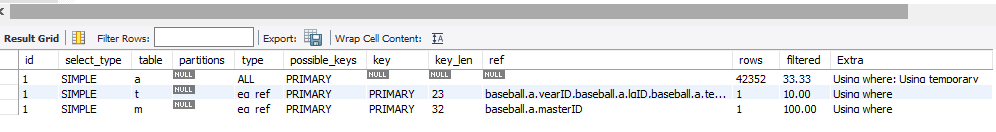
Daniel Oliveros – A02093272

CS 5800 – More SQLing

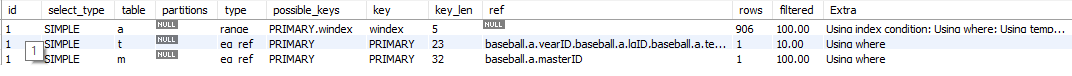
Optimizing queries

**Query 1:**

1. Originally it took 42354 rows to evaluate the query

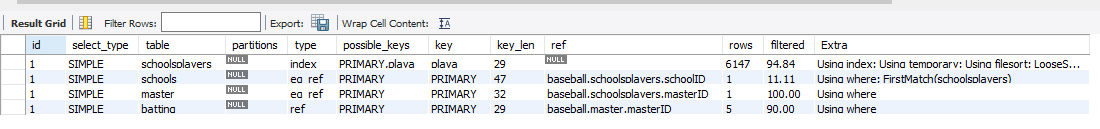


1. Created an index for wins on pitching
2. After optimizing, it only took 908 rows to evaluate the query

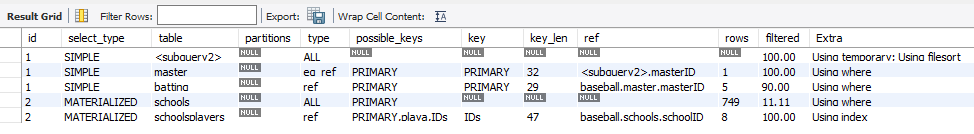


**Query 2:**

1. Originally took 6152 row lookups

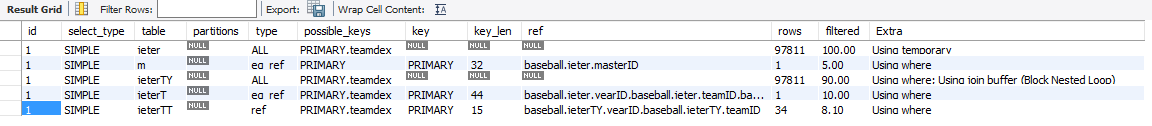


1. Created an index on schoolID on schoolsplayers
2. Row lookups went down to 763



**Query 3:**

1. Originally, it took 195658 row lookups to evaluate the query



1. The reason why this query cannot be optimized is because any way of ordering the data through an index is nullified by the query. The query first needs to find all the masterID’s of players that have played with Derek Jeter. Since we have no way way of knowing which players those are we have to go through all the masterIDs in the appearances table. We then have to find all the players that have played with the previously found masterID’s. Once again we are left with no way of knowing who those players are so we can’t use an index to optimize. This is why the number of rows is approximately twice the number of rows in the appearances table..

**Query 4:**

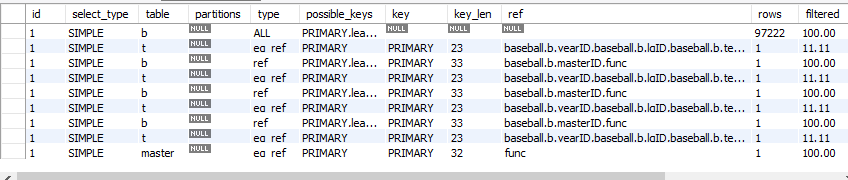
1. This query is originally very nice, it uses the yearID index in order to look through data nicely. It takes 2764 rows to execute, which means that it performs a full table scan at some point
2. We tried making indexes everywhere, we made one on teams based on wins, we made a combined one on years and wins, we even tried forcing these indices into our query by using the force index command, but in the end, the query defeated us.
3. The query ran just as fast after our attempts at optimizing it.

**Query 5:**

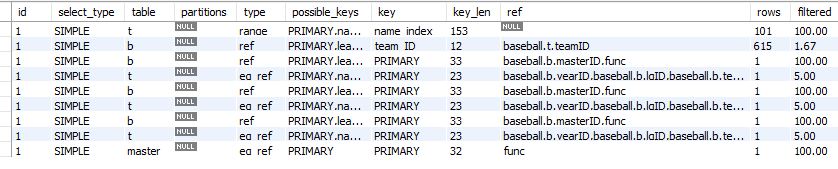
1. Due to how long the query took to execute, we had to change how long the server would run before timing out. This query would take 385797 rows to process, which takes an extremely long amount of time.
2. We created an index on names for the teams table. At some point in our testing, the query started executing in 2 seconds, we are genuinely not sure how it got to this point. B would also, for some reason, crash MySQL. So, we have accepted this query is haunted, and bow down to the two second runtime as the optimization to rule them all.

**Query 6:**

1. 97,230 rows



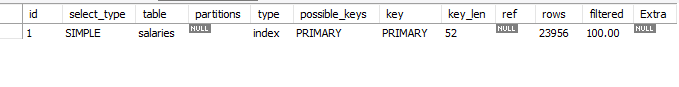
1. Added an index to the names on the teams table
2. 723 rows



**Query 7:**

1. 71,897 rows
2. Created a table for the inner queries since they were the same query and sql was recognizing it as such. Then created an index on the lgID and teamID
3. 24,808 rows

Cost to make the table



Cost after table was made

