MYSQL optimization

Concerning changing the indexing:

In each case, as you can see below , I considered adding indices, but could not due to lack of uniqueness, on column-sets that did not already contain the primary key (ie a superset of the primary key). I also considered taking away the indices on the primary key in the ‘before’ sql schema, and then adding them back in in the ‘after schema’, but on **multiple** joins on tables with 50,000 tuples (it took a long time on much smaller tables), this took an extraordinary amount of time. So yes, while I could certainly show that the addition of indices, improves the query time, The difference is very much offscale, relative to the changes due to changes in parentheses shown below.

Q1:

We’re doing a point select on a column that has already been designated as a primary key. The primary key already has a BTREE index by default. Query1 could be sped up (on very large tables) by switching the index to a HASH map. However, we don’t switch because of query2, see below.

Q2:

We’re doing a range select on a column that has already been designated as a primary key. This primary key already uses BTREE indexing (by default), which is already optimized for range queries.

Q3:

* We’re doing 2 equijoins over three tables: course.crsCode = transcript.crsCode and student.id = transcript.studId. In order to potentially speed this up, I add could add indexes on transcript.crsCode and transcript.studId, but neither of these are unique.
* Here, I used parentheses in the ‘before’ queries to slow down execution. Basically I try to enforce joining large tables, before selecting out the small number of tuples. This is in opposition to the ‘shrink the size of the query fast’ strategy used by the internal optimizer.

Q4:

* Again, I could add an index on teaching.profId to optimize the join of teaching and professor, except that it isn’t unique.
* Again, I used parentheses in the ‘before’ queries to slow down execution. Basically I try to enforce joining large tables, before selecting out the small number of tuples. This is in opposition to the ‘shrink the size of the query fast’ strategy used by the internal optimizer.

Q5:

* I could add an index on course.deptId to optimize the select, but again deptId won’t be unique so the index won’t work.
* Again, I used parentheses in the ‘before’ queries to slow down execution. Basically I try to enforce joining large tables, before selecting out the small number of tuples. This is in opposition to the ‘shrink the size of the query fast’ strategy used by the internal optimizer.

Q6:

* After putting the above into place, query 6 has all the indices that are allowable given the data model. Parentheses around the second select query allow optimal speed of this joint-select statement.
* Again, I used parentheses in the ‘before’ queries to slow down execution. Basically I try to enforce joining large tables, before selecting out the small number of tuples. This is in opposition to the ‘shrink the size of the query fast’ strategy used by the internal optimizer.

**Speed changes conclusions:**

(See the excel sheet for results) As you can see, I succeeded in optimizing the last 4 queries, by using parenthesize, to force them to do efficient/inefficient joins. This decreased the query time by a factor of 2-3. As you can see, though, the first two queries were so simple, that there wasn’t much to change/optimize, so they perform similarly in both ‘before’ and ‘after’.