Cats: Milestone III

*Let me begin by saying I had significant challenges trying to clear the cache to get “cold runs.” My process consisted of disconnecting from the data-base, potentially reloading some tables, querying large other tables, etc. Considering that this is production technology and optimizing storage of cache for future utility helps to improve the product as a whole, I found that it was very hard to use alternative measures. Regardless, I believe I was able to gain an understanding of the material that was taught and apply it in this homework.*

Query 1: “Overall Likes”

Without Index: Limit (cost=4914.00..4914.03 rows=10 width=4) (actual time=198.240..198.253 rows=10 loops=1)

Believed to be optimal query performs pretty well on data-sets videos (10K records), Likes (50K records), and watch (100K records).

**CREATE INDEX** watch\_u\_id\_index **on** watch(user\_id);

Limit (cost=3620.88..3620.91 rows=10 width=4) (actual time=204.316..204.329 rows=10 loops=1)

Although the query processing time increased by 3.6%, the cost of the query was reduced substantially by incorporating the index (which was used in the query plan). My guess is while it suffered a minor performance decrease using the current data-sets, it will perform better with larger data. **Recommendation:** currently, do not use index in isolation.

**CREATE INDEX** likes\_u\_id\_index **on** likes(user\_id);

Limit (cost=4174.06..4174.08 rows=10 width=4) (actual time=186.291..186.305 rows=10 loops=1)

In this case, the query processing time decreased by 6% , and the cost of the query was reduced by incorporating the index (which was used in the query plan). I believe that while this appears to be a win-win scenario, the prior index will have better performance on larger data-sets. **Recommendation:** use this index.

Using both indexes: Limit (cost=2933.57..2933.59 rows=10 width=4) (actual time=189.914..189.940 rows=10 loops=1)

Using both substantially reduced the cost of execution, and also decreased the overall run-time by 5%. Based on the execution plan, the query used both indexes and I believe that this will be the best long-term solution. The query only suffers marginally in execution time to the one index solution. **Final Recommendation:** this is the best long-term solution, use both.

Query 2: “Friend Likes”

Without Index: Limit (cost=5128.85..5128.87 rows=10 width=4) (actual time=131.179..131.192 rows=10 loops=1)

Believed to be optimal query performs pretty well on data-sets videos (10K records), Likes (50K records) friends (24K), and watch (100K records).

**CREATE INDEX** friend\_u\_id\_index **on** friend(user\_id);

Limit (cost=4854.40..4854.42 rows=10 width=4) (actual time=122.161..122.175 rows=10 loops=1)

Since I was running a distinct on friend.user\_id, figured would get a large lift from indexing on it. Cost had a minimal reduction, yet query processing time decreased by 7.5% when using the index. I believe that there could be an opportunity to leverage a combination of indexes to improve performance, since I was surprised that the cost did not decrease as much as I anticipated. **Recommendation:** use this index, or some combination of this index.

**CREATE INDEX** watch\_u\_id\_index **on** watch(user\_id);

Limit (cost=3902.90..3902.93 rows=10 width=4) (actual time=108.645..108.657 rows=10 loops=1)

This index was better than the previous index in both cost and processing time, in the latter it decreased by 17%. Believing now that the heuristic I am learning is to use the index on the largest able, that requires unique tuple look-ups. **Recommendation:** definitely use this index, potentially in combination with other indexes.

**CREATE INDEX** likes\_u\_id\_index **on** likes(user\_id);

Limit (cost=3840.31..3840.34 rows=10 width=4) (actual time=68.100..68.114 rows=10 loops=1)

Considering that this index was iterated on twice, I was able to get the best possible performance from it – in both processing time and cost. Processing time was down 48% and I believe a combination of watch and likes should give the best possible results. **Recommendation:** definitely use this index, in a combination with other indexes.

Using both likes and watch user\_id indexes: Limit (cost=2599.82..2599.85 rows=10 width=4) (actual time=41.930..41.941 rows=10 loops=1)

This yields my final recommendation, as cost and processing time were vastly improved. With even larger data-sets, this benefit should serve as a huge lift to ongoing database performance. **Final Recommendation:** use both watch and like user\_id indexes.

Query 3: “Friends of Friends” likes

Without index: Limit (cost=6435.87..6435.89 rows=10 width=4) (actual time=190.302..190.315 rows=10 loops=1)

Query takes a large amount of processing time and cost to execute. Presumably, since we are leveraging the friend table twice with unions, that creating an index on friend user id should increase some performance.

**CREATE INDEX** friend\_u\_id\_index **on** friend(user\_id);

Limit (cost=5428.44..5428.46 rows=10 width=4) (actual time=179.280..179.292 rows=10 loops=1)

This query iterated on the index 3 separate times and was able to achieve modest improvement with 6% less processing time and reduced cost. Considering I had success with watch and likes as well, I will attempt them next. **Recommendation:** implement index, potentially in a combination of other indexes.

**CREATE INDEX** watch\_u\_id\_index **on** watch(user\_id);

Limit (cost=5195.38..5195.41 rows=10 width=4) (actual time=181.073..181.086 rows=10 loops=1)

Watch user\_id index improves processing time, but not as much as the friend user\_id index considering that this was only iterated on once. The cost has still reduced, however, leading me to believe that this still could be valuable in large data sets. **Recommendation:** implement index if data-set increases in size, potentially use in a combination of other indices.

**CREATE INDEX** likes\_u\_id\_index **on** likes(user\_id);

Limit (cost=5815.59..5815.61 rows=10 width=4) (actual time=230.003..230.015 rows=10 loops=1)

This index marginally reduced the cost but increased the processing time. It is easy to eliminate using this index from contention. **Recommendation:** do not use index.

Combination of watch user\_id and friend user\_id: Limit (cost=4187.95..4187.97 rows=10 width=4) (actual time=153.914..153.926 rows=10 loops=1)

By implementing both watch user\_id and friend user\_id indexes, the query performs at its most optimal state by cutting both cost and processing time. By implementing both, the processing time was reduced by 19%. Hence, my belief that using both will have the best long-term success. **Final recommendation:** implement both friend user\_id and watch user\_id indexes.

Query 4: “My Kind of Cats”

Without index: Limit (cost=8227.72..8227.75 rows=10 width=4) (actual time=232.707..232.719 rows=10 loops=1)

Query takes a large amount of processing time and cost to execute. Presumably, since we are leveraging the likes table repeatedly based on sub-queries, that creating an index on likes user id should increase some performance.

**CREATE INDEX** likes\_u\_id\_index **on** likes(user\_id);

Limit (cost=6987.16..6987.18 rows=10 width=4) (actual time=230.121..230.133 rows=10 loops=1)

By implementing the likes user\_id index, there is no noticeable improvement in processing time but a decrease in cost to execute. This is not as large of a lift as I would anticipate, hence I am not sure it is worth it to use the index in this scenario. **Recommendation:** implement index if data size increases, otherwise no need for the index.

**CREATE INDEX** watch\_u\_id\_index **on** watch(user\_id);

Limit (cost=6987.23..6987.26 rows=10 width=4) (actual time=207.864..207.877 rows=10 loops=1)

Using the watch\_u\_id\_index, the query iterated over that range once and reduced its processing time substantially, while producing a cost reduction almost identical to the likes index. The final processing increase was 10% in time saved. Considering that the other index did not perform well, I believe this is the only index worth adding. **Final recommendation:** implement index.

Query 5: “My Kind of Cats – with preference”

Without index: Limit (cost=9837.88..9837.91 rows=10 width=12) (actual time=217.347..217.358 rows=10 loops=1)

Based on the query, I only perceive value in attempting to place an index on the user column in likes, since it is utilized in 3 sub-queries prior to the final query. Hopefully an index there will reduce both cost and processing time.

**CREATE INDEX** likes\_u\_id\_index **on** likes(user\_id);

Limit (cost=8597.32..8597.34 rows=10 width=12) (actual time=214.295..214.307 rows=10 loops=1)

After implementing this index, only appearing to see noticeable decreases in cost, and not processing time. The index does not appear to provide enough of a lift to consider to be used in production. **Final Recommendation:** do not implement index, unless data size substantially increases.