**Sales Queries**

To run the sales queries, I created a database with the following characteristics:

* 50,000 customers in all 50 states (1,000 per state)
* 10,000 products in 100 categories (100 per category)
* Only 50 customers appear in sales table
* Only 10 products appear in sales table
* All 50 states appear in sales table
* 15 million sales

**Sales Q1**

Prior to index:

Explain Analyze execution time: 29556.379 ms

Direct run execution time: 24 secs

Index addition(s):

create index on sale (customer\_id);

After index:

Explain Analyze execution time: 18616.243 ms

Direct run execution time: 14 secs

Q1 Notes:

This was performed on the query that grouped by customer\_id only, and not the “more refined” query that grouped on customer\_id and product\_id.

**Sales Q2**

Prior to index:

Explain Analyze execution time: 28201.943 ms

Direct run execution time: 23 secs

Index addition(s):

create index on sale (customer\_id);

create index on customer (state\_id);

After index:

Explain Analyze execution time: 16522.478 ms

Direct run execution time: 13 secs

Q2 Notes:

The index on customer (state\_id) was not utilized by the query processor, but it would be useful in some situations. In my data, there are only 50 states, and sales occur in all states in roughly equal proportion. If there were a sufficiently large number of states, such that the number of sales within any given state only made up a small fraction of the total number of sales, then the customer (state\_id) index would have been used.

**Sales Q3**

Prior to index:

Explain Analyze execution time: 1648.078 ms

Direct run execution time: 1 sec

Index addition(s):

create index on sale (customer\_id);

create index on sale (product\_id);

After index:

Explain Analyze execution time: 396.834 ms

Direct run execution time: 337 ms

Q3 Notes:

The index on sale (product\_id) was not used, but could have been used by the query optimizer to aggregate sales by product\_id if there were a sufficiently large number of products in the sales table, and the number of sales involving any single product were a small fraction of the total number of sales.

**Sales Q4**

The original database size resulted in prohibitively long query run times for queries 4, 5, and 6, so for Q4, Q5, and Q6, I scaled the database size down to the following:

* 1,000 customers in all 50 states (20 per state)
* 1,000 products in 100 categories (10 per category)
* Only 10 customers appear in sales table
* Only 10 products appear in sales table
* Only 10 states appear in sales table
* 5 million sales

Prior to index:

Explain Analyze execution time: 16761.794 ms

Direct run execution time: 47 secs

Index addition(s):

Create index on sale (customer\_id);

Create index on sale (product\_id);

After index:

Explain Analyze execution time: 16548.227 ms

Direct run execution time: 45 secs

Q4 Notes:

Neither of the added indices were utilized by the query processor, but I think they would be useful if any particular customer or product was a rare occurrence within the sales table, and the sales table was large enough. In that case, accessing particular customers or products within the sales table via an index sale (customer\_id) or sale (product\_id) would be more efficient than sequentially scanning the sales table.

**Sales Q5**

Prior to index:

Explain Analyze execution time: 18082.772 ms

Direct run execution time: 16 secs

Index addition(s):

create index on sale (customer\_id);

create index on sale (product\_id);

create index on customer (state\_id);

create index on product (category\_id);

After index:

Explain Analyze execution time: 18106.196 ms

Direct run execution time: 16 secs

Q5 Notes:

None of the added indices were utilized by the query processor, but they would be useful in some situations.

Because Q5 reuses Q4, indices on sale (customer\_id) and sale (product\_id) would be useful in the situation described in Q4 notes.

Similarly, as was the case for Q2, if there were a large enough number of states such that the number of sales within any given state only made up a sufficiently small fraction of overall number of sales, then the customer (state\_id) index would have been used.

Likewise, if there were a large enough number of categories such that the number of sales within any given category were small enough compared to the total number of sales, then the product (category\_id) index would have been used.

**Sales Q6**

Prior to index:

Explain Analyze execution time: 37757.274 ms

Direct run execution time: 32 secs

Index addition(s):

create index on sale (customer\_id);

create index on sale (product\_id);

create index on customer (state\_id);

create index on product (category\_id);

After index:

Explain Analyze execution time: 37288.943 ms

Direct run execution time: 33 secs

Q6 Notes:

None of the added indices were utilized by the query processor, but they would be useful in the instances described for Q1 and Q5, as Q6 reuses those queries.

**Cats Queries**

To run the cats queries, I created a database with the following characteristics:

* 50,000 user table rows
* 30,000 video table rows
* 20,000 friend table rows
* 100,000 likes table rows
* 200,000 watch table rows
* 20,000 login table rows
* 5,000 suggestion table rows

**Cats Q1 (Overall Likes)**

Prior to index:

Explain Analyze execution time: 181.977 ms

Direct run execution time: 269 msec

Index addition(s):

create index on likes (user\_id);

create index on watch (user\_id);

create index on likes (video id);

create index on watch (video\_id);

After index:

Explain Analyze execution time: 139.864 ms

Direct run execution time: 198 msec

Q1 Notes:

The “Overall\_best” query was used here.

Indices for watch (user\_id) and likes (user\_id) were used.

Indices for watch (video\_id) and likes (video\_id) were not used in this case, but would be used if the watch and likes tables were sufficiently large, and if the appearance of videos was sufficiently infrequent in the likes and watch tables. If such were the case, it would be more efficient to search for the top 10 videos by index rather than by sequentially scanning the watch and likes tables.

**Cats Q2 (Friend Likes)**

Prior to index:

Explain Analyze execution time: 110.451 ms

Direct run execution time: 194 msec

Index addition(s):

create index on likes (user\_id);

create index on likes (video\_id);

create index on friend (user\_id);

create index on watch (user\_id);

create index on watch (video\_id);

After index:

Explain Analyze execution time: 45.919 ms

Direct run execution time: 152 msec

Q2 Notes:

Indices for watch (user\_id), likes (user\_id), and friend (user\_id) were used.

The indices for likes (video\_id) and watch (video\_id) would have been used in the scenario described in Q1 notes.

**Cats Q3 (Friend of Friend Likes)**

Prior to index:

Explain Analyze execution time: 188.285 ms

Direct run execution time: 265 msec

Index addition(s):

create index on likes (user\_id);

create index on likes (video\_id);

create index on friend (user\_id);

create index on watch (user\_id);

create index on watch (video\_id);

After index:

Explain Analyze execution time: 119.011 ms

Direct run execution time: 187 msec

Q3 Notes:

Indices for watch (user\_id), likes (user\_id), and friend (user\_id) were used.

The indices for likes (video\_id) and watch (video\_id) would have been used in the scenario described in Q1 notes.

**Cats Q4 (My Kind of Cats)**

Prior to index:

Explain Analyze execution time: 105.546 ms

Direct run execution time: 231 msec

Index addition(s):

create index on likes (user\_id);

create index on likes (video\_id);

create index on watch (user\_id);

create index on watch (video\_id);

After index:

Explain Analyze execution time: 38.754 ms

Direct run execution time: 203 msec

Q4 Notes:

Indices for watch (user\_id), likes (user\_id), and likes (video\_id) were used.

The index for watch (video\_id) would have been used in the scenario described in Q1 notes.

**Cats Q5 (Weighted Likes)**

Prior to index:

Explain Analyze execution time: 259.393 ms

Direct run execution time: 367 msec

Index addition(s):

create index on likes (user\_id);

create index on likes (video\_id);

create index on watch (user\_id);

create index on watch (video\_id);

After index:

Explain Analyze execution time: 214.842 ms

Direct run execution time: 317 msec

Q5 Notes:

Indices for watch (user\_id), likes (user\_id), and likes (video\_id) were used.

The index for watch (video\_id) would have been used in the scenario described in Q1 notes.