***MAS DSE 201 Homework: 201Cats***

***Milestone III [due March 4 midnight]***

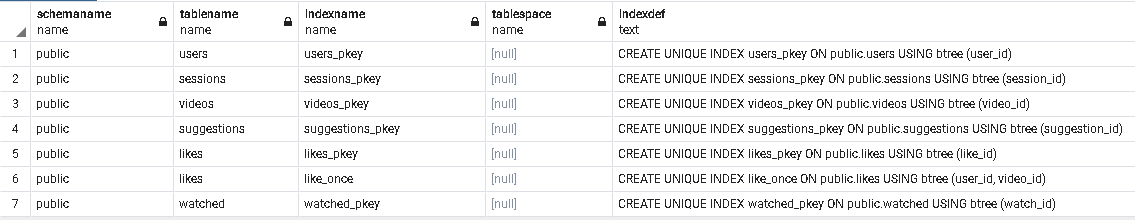
*Your next goal is to deliver top performance for the queries developed in Milestone II. Expect a significant amount of experimentation until you tune the performance.*

*In particular, create any indices that will be beneficially used by the queries. Do not create useless indices. Measure the performance of the queries on cold runs and report the performance improvements.*

*Submit your index choices (in the form of the script with the CREATE INDEX statements) and the measured query performance improvements.*

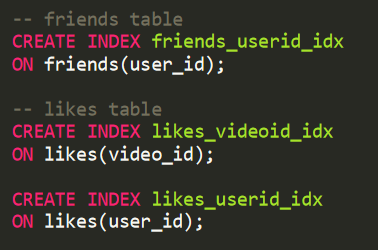
**Summary of Database Sizes**

|  |  |  |
| --- | --- | --- |
| **Table Name** | **Number of Tuples (Large Dataset)** | **Number of Tuples (Small Dataset)** |
| users | 10,000,000 | 10,000 |
| sessions (not used in queries) | 5 | 5 |
| videos | 2,500,000 | 25,000 |
| friends | 150,000,000 | 150,000 |
| suggestions (not used in queries) | 5 | 5 |
| likes | 100,000,000 | 100,000 |
| watched (not used in queries) | 5 | 5 |

**Indices Automatically Generated with Database 201 Cats Schema**

**Index Choices**

* Postgres creates indices by default for all primary keys in the data base. The indices below were created and tested in addition to the ones generated automatically (screenshot above).



**Summary of Performance**

* Table summary (queries were re-ran by adding all three index choices to the database)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Query** | **Run Time without Index Choices** | **Run Time with Index Choices** | **Improved Performance?** | **Dataset Size** |
| Overall Likes | 1 min 37 secs | 1 min 37 secs | No | Large |
| Friends Likes | 6 secs 121 msec | 45 msec | Yes | Large |
| Friends of Friends Likes | 21 secs 463 msec | 80 msec | Yes | Large |
| My Kind of Cats | 5 secs 958 msec | 106 msec | Yes | Large |
| My Kind of Cats with Preference | 3 min 36 secs | 3 min 52 secs | No | Small |

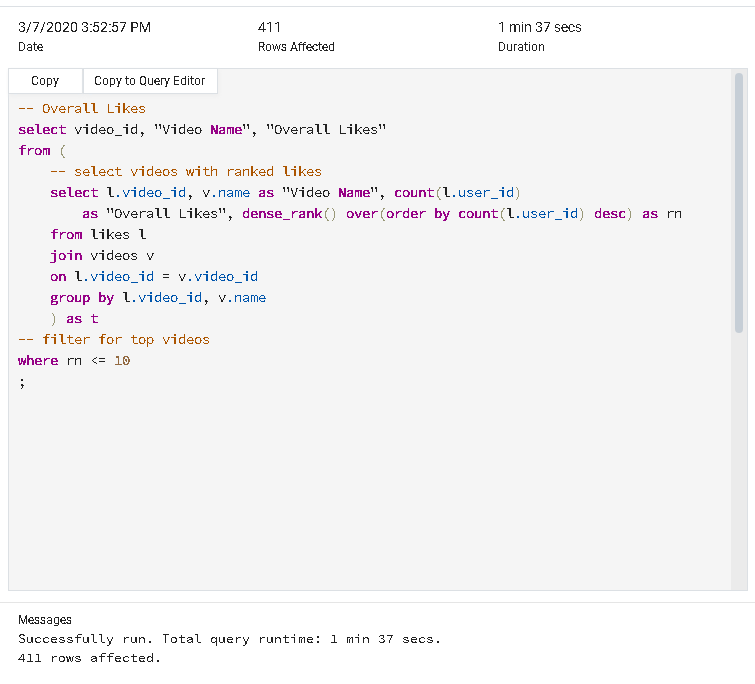
* Comments:
  + Indexing seems to be useful on queries where a small fraction of all tuples needs to be accessed. For the queries below, the most benefit was observed when filtering (with WHERE clause) for a specific user\_id and video\_id tuple as a foreign key index. Using indices on other clauses such as DISTINCT, GROUP BY, JOIN, and ORDER BY was not as effective.

**Overall Likes – Query (Large Dataset)**

*A screenshot of a cell phone

Description automatically generated*

**Overall Likes - Total Query Runtime without Index Choices**



**Overall Likes – Observations**

* No improvement in performance observed, run time without index choices is 1 min 37 secs.
* Summary of indices attempted:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** | **Run Time** | **Comments** |
| videos | name | GROUP BY | CREATE INDEX videos\_name\_idx ON videos(name); | 1 min 42 secs | Inner join generates new tuples on the fly, is likely that the index is not useful for this reason. |
| likes | video\_id (Foreign Key) | JOIN  GROUP BY | CREATE INDEX likes\_videoid\_idx ON likes(video\_id); | 1 min 36 secs | There are 2.5 million videos on my videos table and all of them have at least one like, therefore, it seems that indexing the foreign key video\_id is not benefiting the optimizer as it still must scan every entry. |
| likes | user\_id  (Foreign Key) | ORDER BY | CREATE INDEX likes\_userid\_idx  ON likes(user\_id); | 1 min 37 secs | Is possible that the index is not valuable because new data is generated with the COUNT clause and then the WHERE clause is used on a new variable ‘rn’. |

**Friend Likes – Query (Large Dataset)**

*A screenshot of a cell phone

Description automatically generated*

**Friend Likes - Total Query Runtime without Index Choices**

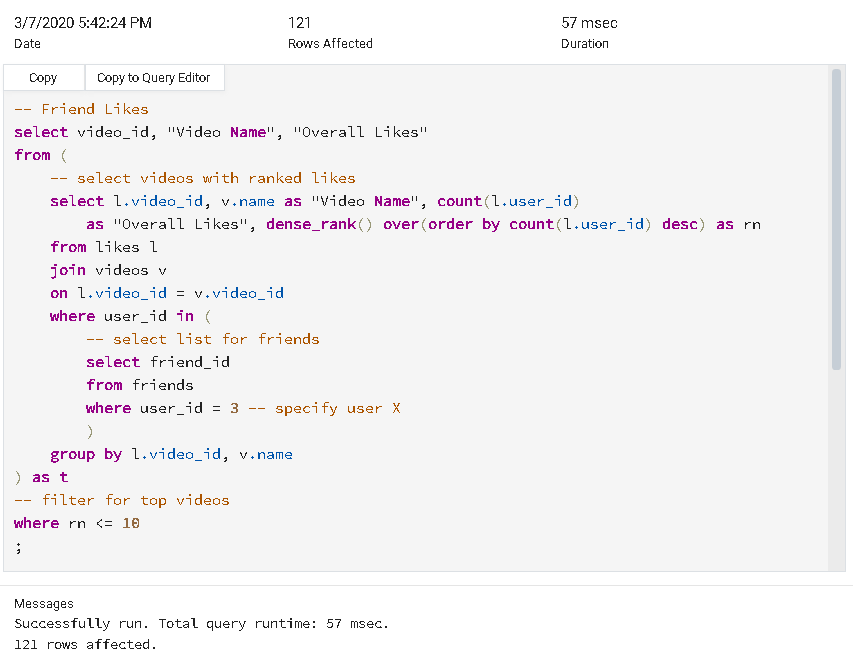


**Friend Likes – Observations**

* Improvement in performance observed, run time without index choices is 6 secs 121 msec. Using an index on ‘friends’ table for ‘user\_id’ column the query run time was reduced to 57 msec.
* Summary of indices attempted:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** | **Run Time** | **Comments** |
| friends | user\_id | WHERE | CREATE INDEX friends\_userid\_idx  ON friends(user\_id); | 57 msec | This index yields a significant improvement in run time as the query specifies a filter for a single user X. |
| videos | name | GROUP BY | CREATE INDEX videos\_name\_idx ON videos(name); | 6 secs 94 msec | Inner join generates new tuples on the fly, is likely that the index is not useful for this reason. |
| likes | video\_id (Foreign Key) | JOIN  GROUP BY | CREATE INDEX likes\_videoid\_idx ON likes(video\_id); | 6 secs 123 msec | There are 2.5 million videos on my videos table and all of them have at least one like, therefore, it seems that indexing the foreign key video\_id is not benefiting the optimizer as it still must scan every entry. |
| likes | user\_id  (Foreign Key) | ORDER BY | CREATE INDEX likes\_userid\_idx  ON likes(user\_id); | 6 secs 92 msec | Is possible that the index is not valuable because new data is generated with the COUNT clause and then the WHERE clause is used on a new variable ‘rn’. |

* Screenshot for improved query, friends - user\_id



**Friends of Friends Likes – Query (Large Dataset)**

*A screenshot of a cell phone

Description automatically generated*

**Friends of Friends Likes - Total Query Runtime without Index Choices**

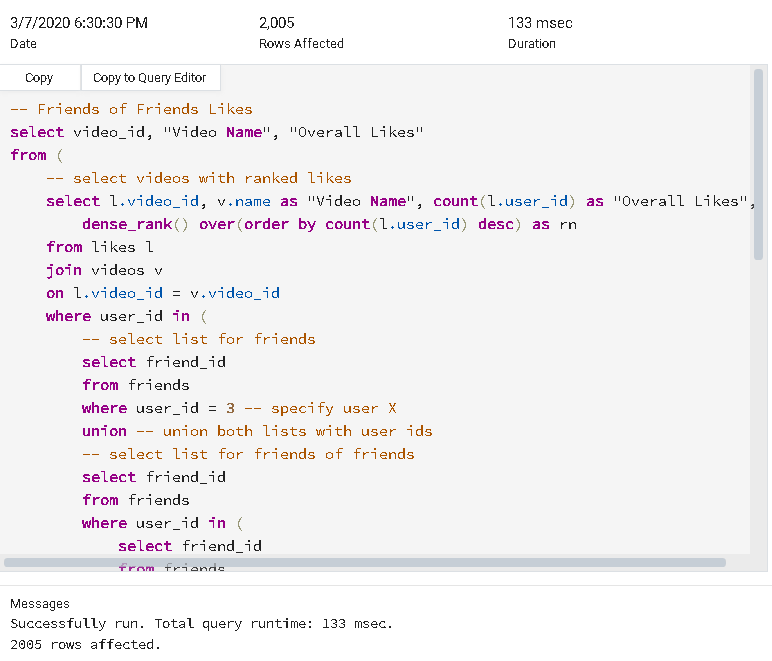


**Friends of Friends Likes - Observations**

* Improvement in performance observed, run time without index choices is 21 secs 463 msec. Using an index on ‘friends’ table for ‘user\_id’ column the query run time was reduced to 133 msec.
* Summary of indices attempted:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** | **Run Time** | **Comments** |
| friends | user\_id | WHERE | CREATE INDEX friends\_userid\_idx  ON friends(user\_id); | 133 msec | This index yields a significant improvement in run time as the query specifies a filter for a single user X. |
| videos | name | GROUP BY | CREATE INDEX videos\_name\_idx ON videos(name); | 21 secs 595 msec | Inner join generates new tuples on the fly, is likely that the index is not useful for this reason. |
| likes | video\_id (Foreign Key) | JOIN  GROUP BY | CREATE INDEX likes\_videoid\_idx ON likes(video\_id); | 22 secs 147 msec | There are 2.5 million videos on my videos table and all of them have at least one like, therefore, it seems that indexing the foreign key video\_id is not benefiting the optimizer as it still must scan every entry. |
| likes | user\_id  (Foreign Key) | ORDER BY | CREATE INDEX likes\_userid\_idx  ON likes(user\_id); | 21 secs 603 msec | Is possible that the index is not valuable because new data is generated with the COUNT clause and then the WHERE clause is used on a new variable ‘rn’. |

* Screenshot for improved query, friends - user\_id



**My Kind of Cats – Query (Large Dataset)**

*A screenshot of a cell phone

Description automatically generated*

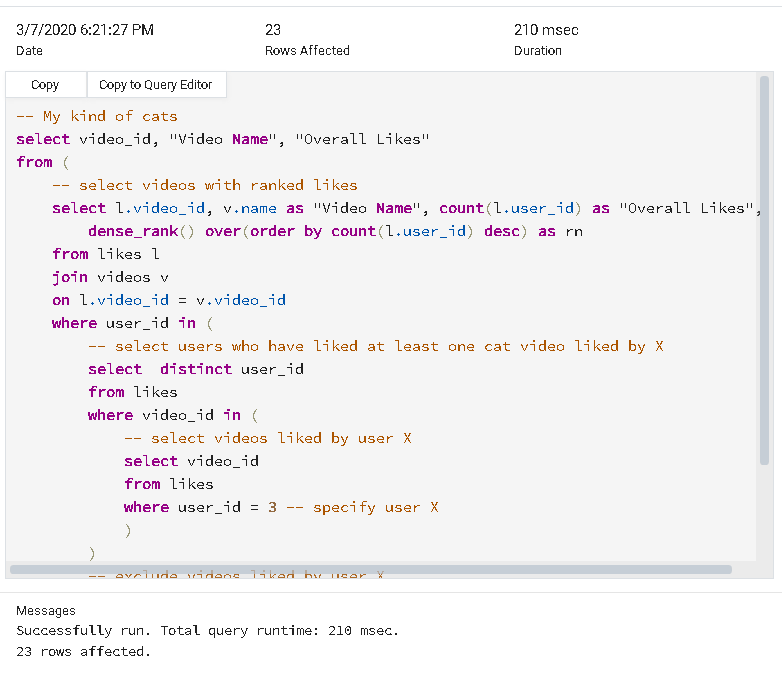
**My Kind of Cats - Total Query Runtime without Index Choices**

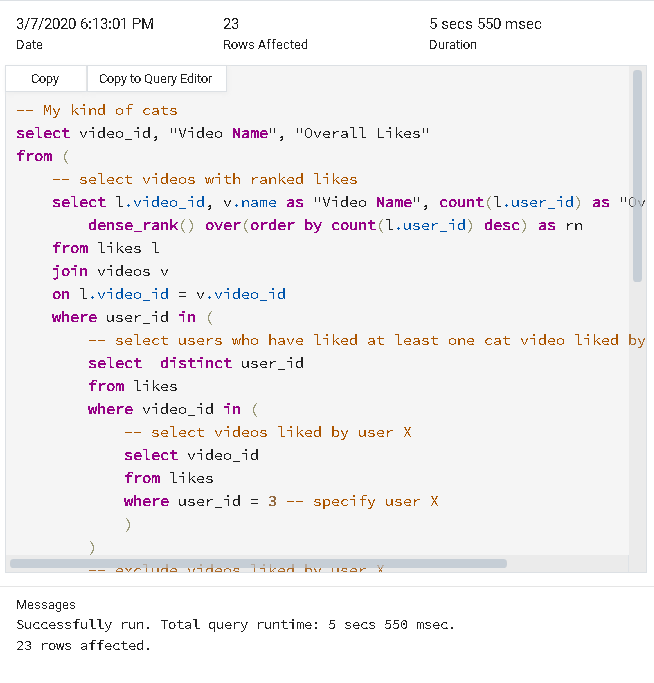


**My Kind of Cats – Observations**

* Improvement in performance observed, run time without index choices is 5 secs 958 msec. Using an index on ‘likes’ table for ‘video\_id’ column the query run time was reduced to 210 msec.
* Summary of indices attempted:

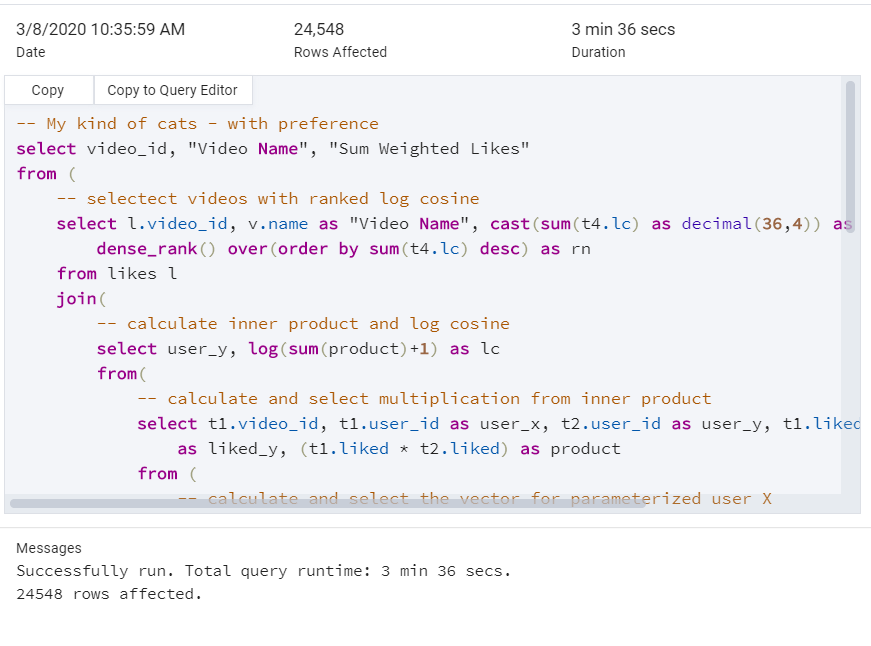
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** | **Run Time** | **Comments** |
| videos | name | GROUP BY | CREATE INDEX videos\_name\_idx ON videos(name); | 5 secs 862 msec | Inner join generates new tuples on the fly, is likely that the index is not useful for this reason. |
| likes | video\_id (Foreign Key) | JOIN  WHERE  GROUP BY | CREATE INDEX likes\_videoid\_idx ON likes(video\_id); | 210 msec | Significant performance improvement, the foreign key video\_id is used as a filter in a sub-query to create a list. |
| likes | user\_id  (Foreign Key) | DISTINCT  WHERE | CREATE INDEX likes\_userid\_idx  ON likes(user\_id); | 5 secs 550 msec | There is a small improvement of about 300 msec, it likely happens in both sub-queries where the distinct user\_id tuples are selected from the likes table, and on the filter where user X is specified. |

* Screenshot for improved query, likes - video\_id
* Screenshot for improved query, likes - user\_id



*A screenshot of a cell phone

Description automatically generated***My Kind of Cats with Preference – Query (Small Dataset)**

**My Kind of Cats with Preference - Total Query Runtime without Index Choices**

**My Kind of Cats with Preference - Observations**

* No improvement in performance observed, run time without index choices is 3 min 36 secs.
* Summary of indices attempted:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** | **Run Time** | **Comments** |
| videos | name | GROUP BY | CREATE INDEX videos\_name\_idx ON videos(name); | 3 min 42 secs | Inner join generates new tuples on the fly, is likely that the index is not useful for this reason. |
| likes | video\_id (Foreign Key) | LEFT JOIN  JOIN  GROUP BY | CREATE INDEX likes\_videoid\_idx ON likes(video\_id); | 3 min 45 secs | Indexing is not helping joins because the optimizer still must visit a large percentage of entries to merge both tables. |
| likes | user\_id  (Foreign Key) | LEFT JOIN  JOIN | CREATE INDEX likes\_userid\_idx  ON likes(user\_id); | 3 min 51 secs | Indexing is not helping joins because the optimizer still must visit a large percentage of entries to merge both tables. |

***MAS DSE 201 Homework: Sales Cube***

***Milestone III [due March 4 midnight]***

*Your next goal is to deliver top performance for the queries. Expect a significant amount of experimentation until you tune the performance.*

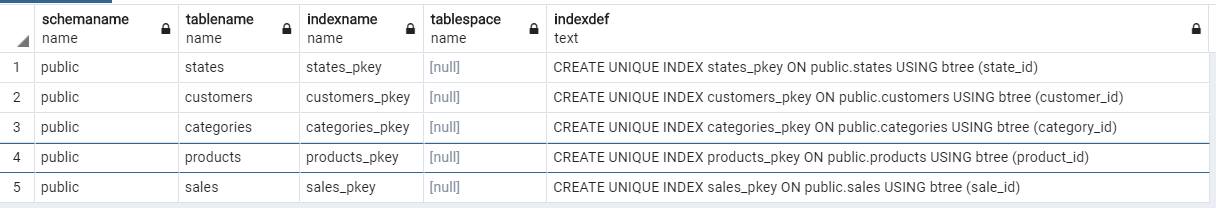
*In particular, create any indices that will be beneficially used by the queries. Do not create useless indices. Deploy your solution and measure the performance of the Milestone II queries on cold runs.*

*Submit your index choices (in the form of the script with the CREATE INDEX statements) and the measured query performance improvement.*

**Summary of Database Sizes**

|  |  |  |
| --- | --- | --- |
| **Table Name** | **Number of Tuples (Large Dataset)** | **Number of Tuples (Small Dataset)** |
| states | 400 | 25 |
| customers | 320,000 | 2,000 |
| categories | 160,000 | 1,000 |
| products | 240,000 | 1,500 |
| sales | 80,000 | 5,000 |

**Indices Automatically Generated with Database Sales Cube Schema**



**Index Choices**

* Postgres creates indices by default for all primary keys in the data base (screenshot above).
* No additional indices were selected for the queries 1-6 below as there was not an observable improvement in performance. Although several different indices were tested, and their implementation is summarized in detail.
* Is still unclear to me whether performance differences will be evident with larger amounts of data (i.e. 10^6 tuples, something my computer was not able to generate), in this exercise I tried my best to pinpoint the areas where an index might be useful when running my queries.

**Summary of Performance**

* Table summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Query Number** | **Run Time without Index Choices** | **Run Time with Index Choices** | **Improved Performance?** | **Dataset Size** |
| 1 | 1 secs 195 msec | No Indices Chosen | No | Large |
| 2 | 297 msec | No Indices Chosen | No | Large |
| 3 | 82 msec | No Indices Chosen | No | Large |
| 4 | 15 secs 774 msec | No Indices Chosen | No | Small |
| 5 | 6 secs 117 msec | No Indices Chosen | No | Small |
| 6 | 1 secs 106 msec | No Indices Chosen | No | Large |

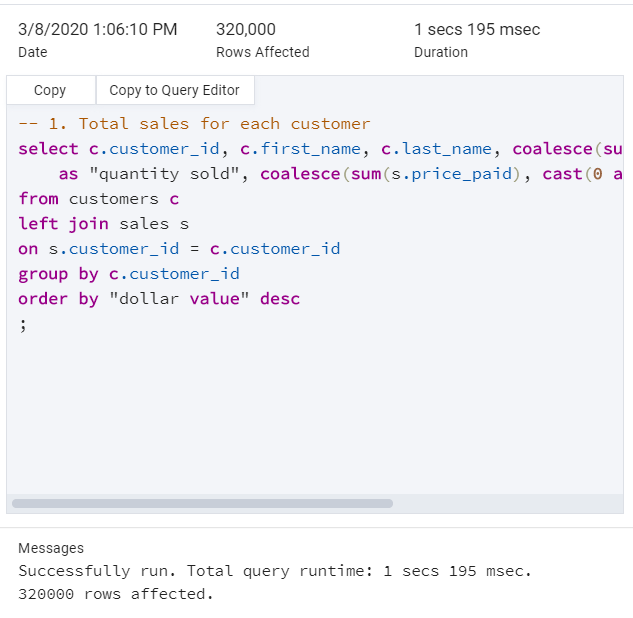
* Comments:
  + No significant improvement in performance was observed in any of the queries across the indices tested.

**1 – Query (Large Dataset)**

*A screenshot of a cell phone

Description automatically generated*

**1 - Total Query Runtime without Index Choices**



**1 - Observations**

* No improvement in performance observed, run time without index choices is 1 secs 195 msec.
* Summary of indices attempted:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** | **Run Time** | **Comments** |
| sales | customer\_id  (Foreign Key) | LEFT JOIN | CREATE INDEX sales\_customer\_id\_idx ON sales(customer\_id); | 1 secs 177 msec | Is likely that no significant difference in performance is observed during left join because the optimizer still must visit a high percentage of tuples, therefore the index is not very useful. |

**2 – Query (Large Dataset)**

*A screenshot of a cell phone

Description automatically generated*

**2 - Total Query Runtime without Index Choices**



**2 - Observations**

* No improvement in performance observed, run time without index choices is 297 msec.
* Summary of indices attempted:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** | **Run Time** | **Comments** |
| sales | customer\_id  (Foreign Key) | LEFT JOIN | CREATE INDEX sales\_customerid\_idx ON sales(customer\_id); | 312 msec | Is likely that no improvement in performance is observed during left join because the optimizer still must visit a high percentage of tuples, therefore the index is not very useful. |
| customers | state\_id  (Foreign Key) | RIGHT JOIN | CREATE INDEX customers\_stateid\_idx ON customers(state\_id); | 305 msec | The right join is not benefited from indexing. Again, is probable that the query involves comparing most of the tuples in both tables. |
| states | name | GROUP BY | CREATE INDEX states\_name\_idx ON states(name); | 284 msec | There is a very small, not sure if significant, improvement when indexing the string column ‘name’. We are also grouping by state\_id which can possibly be another factor to consider during performance. |

**3 – Query (Large Dataset)**

*A picture containing screen, indoor, monitor

Description automatically generated*

**3 - Total Query Runtime without Index Choices**



**3 - Observations**

* No improvement in performance observed, run time without index choices is 82 msec.
* Summary of indices attempted:

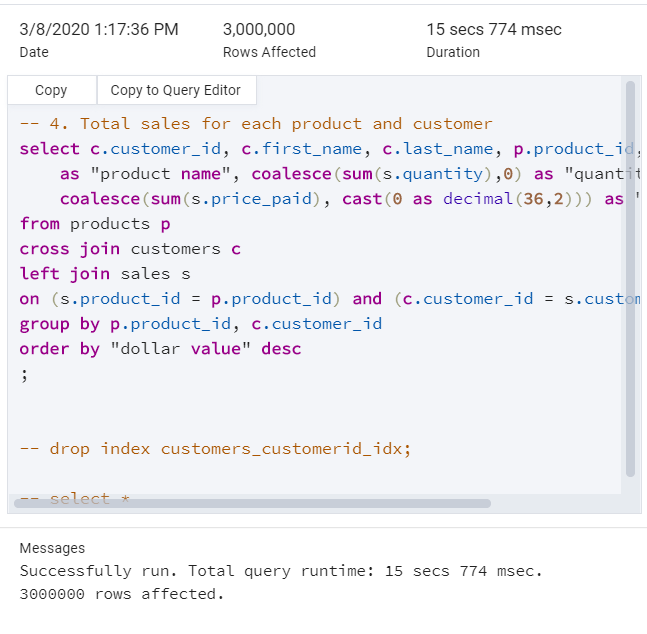
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** | **Run Time** | **Comments** |
| sales | customer\_id  (Foreign Key) | JOIN | CREATE INDEX sales\_customerid\_idx ON sales(customer\_id); | 70 msec | There seems to be a very small improvement during the inner join with customer\_id, although not convincing that is significant. |
| sales | product\_id  (Foreign Key) | JOIN | CREATE INDEX sales\_productid\_idx ON sales(product\_id); | 89 msec | Indexing product\_id appears to not be optimizing the inner join. Interesting observation – there are more tuples in customers (320k) than products (240k) table, the inner join for customer\_id (above) seems to be benefited more than product\_id with indexing, is possible that this observation may scale up with more tuples. |
| products | name | GROUP BY | CREATE INDEX products\_name\_idx ON products(name); | 72 msec | Slight reduction in run time by using an index on the string column ‘name’ from products table. |

**4 – Query (Small Dataset)**

*A screen shot of a computer

Description automatically generated*

**4 - Total Query Runtime without Index Choices**



**4 - Observations**

* No improvement in performance observed, run time without index choices is 15 secs 774 msec.
* Summary of indices attempted:

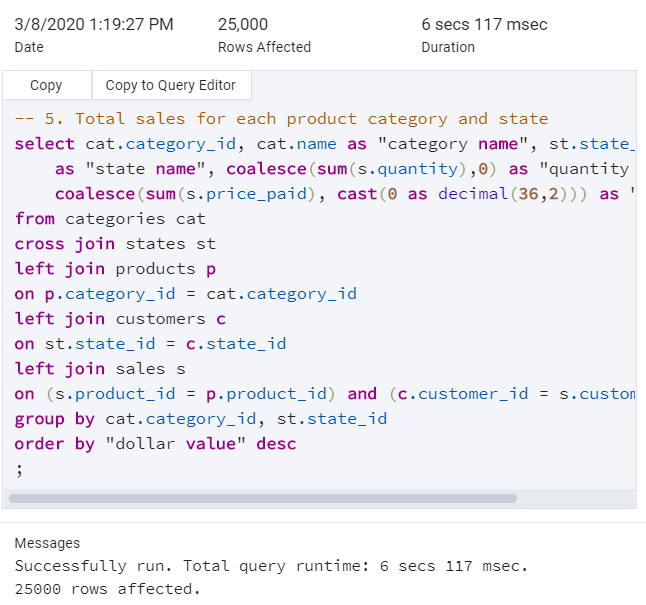
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** | **Run Time** | **Comments** |
| sales | customer\_id  (Foreign Key) | LEFT JOIN | CREATE INDEX sales\_customerid\_idx ON sales(customer\_id); | 15 secs 683 msec | There seems to be a very small improvement during the left join with customer\_id, still not convinced this is significant. |
| sales | product\_id  (Foreign Key) | LEFT JOIN | CREATE INDEX sales\_productid\_idx ON sales(product\_id); | 15 secs 663 msec | There seems to be a very small improvement during the left join with product\_id, still not convinced this is significant. |

**5 – Query (Small Dataset)**

*A flat screen tv

Description automatically generated*

**5 - Total Query Runtime without Index Choices**



**5 - Observations**

* No improvement in performance observed, run time without index choices is 6 secs 117 msec.
* Summary of indices attempted:

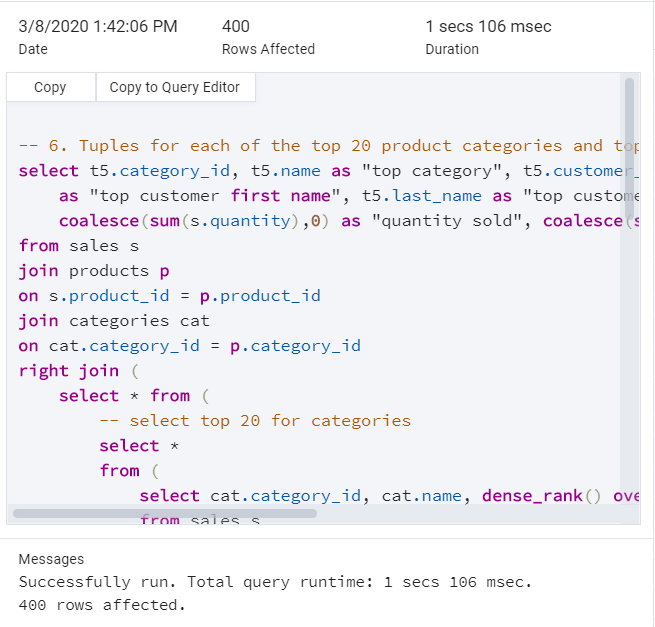
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** | **Run Time** | **Comments** |
| sales | customer\_id  (Foreign Key) | LEFT JOIN | CREATE INDEX sales\_customerid\_idx ON sales(customer\_id); | 6 secs 115 msec | No performance improvement observed during left join. There is a possibility that more data is needed to make a conclusion, something in the order of 10^6 tuples. |
| sales | product\_id  (Foreign Key) | LEFT JOIN | CREATE INDEX sales\_productid\_idx ON sales(product\_id); | 6 secs 480 msec | No performance improvement observed during left join. There is a possibility that more data is needed to make a conclusion, something in the order of 10^6 tuples. |
| customers | state\_id  (Foreign Key) | LEFT JOIN | CREATE INDEX customers\_stateid\_idx ON customers(state\_id); | 6 secs 182 msec | No performance improvement observed during left join. There is a possibility that more data is needed to make a conclusion, something in the order of 10^6 tuples. |
| products | category\_id  (Foreign Key) | LEFT JOIN | CREATE INDEX products\_categoryid\_idx ON products(category\_id); | 6 secs 358 msec | No performance improvement observed during left join. There is a possibility that more data is needed to make a conclusion, something in the order of 10^6 tuples. |

**6 – Query (Large Dataset)**

***A screenshot of a cell phone

Description automatically generated***

**6 - Total Query Runtime without Index Choices**



**6 - Observations**

* No improvement in performance observed, run time without index choices is 1 secs 106 msec.
* Summary of indices attempted:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** | **Run Time** | **Comments** |
| sales | customer\_id  (Foreign Key) | JOIN | CREATE INDEX sales\_customerid\_idx ON sales(customer\_id); | 1 secs 132 msec | No performance improvement observed during join. Optimizer must be accessing a high percentage of tuples. |
| sales | product\_id  (Foreign Key) | JOIN | CREATE INDEX sales\_productid\_idx ON sales(product\_id); | 1 secs 188 msec | No performance improvement observed during join. Optimizer must be accessing a high percentage of tuples. |
| products | category\_id  (Foreign Key) | JOIN | CREATE INDEX products\_categoryid\_idx ON products(category\_id); | 1 secs 270 msec | No performance improvement observed during join. Optimizer must be accessing a high percentage of tuples. |
| categories | name | GROUP BY | CREATE INDEX categories\_name\_idx ON categories(name); | 1 secs 193 msec | No performance improvement observed during group by. The other column ‘category\_id’ in the same ‘group by’ clause has a primary key index, is probable that this one plays a factor during the aggregation. |