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

***Milestone IV [due March 18 midnight]***

*Your next goal will be to improve the performance of the “My Kind of cats – with preference” option by appropriate precomputations. It is understood that the maintenance of the precomputed tables will lead to some slowdown while viewers interact. Your precomputation choices must be such that the precomputations that you introduce collectively save more time to this option than they cost to maintain. Precomputed tables will also benefit from creating certain indices on them. Build any beneficial indices. Calibrate your solution against cold runs of the activity script provided.*

*Submit the following.*

* *The precomputed tables you created: CREATE TABLE statement and query that initially loaded it.*
* *Your new “My Kind of cats – with preference” query, which makes the best use of the precomputed tables you choose.*
* *The indices you created (CREATE INDEX scripts)*

**201 Cats - Summary of Database Size**

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

**201 Cats - Pre-Computed Tables Created**

|  |  |  |  |
| --- | --- | --- | --- |
| **Approach** | **Table Name** | **Run Time (without Indices)** | **Run Time (with Indices)** |
| #1 | calculate\_userX | 124 msec | 107 msec |
| calculate\_userY | 10 min 40 secs | 12 min 17 secs |
| #2 | calculate\_logcosine | 4 min 38 secs | 4 min 13 secs |

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedScreenshots for Approach #1

**A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated**Run Time for *calculate\_userX -* without Indices (left) and with indices (right)

**A screenshot of a cell phone

Description automatically generated**A screenshot of a social media post

Description automatically generatedRun Time for *calculate\_userY -* without Indices (left) and with indices (right)

Screenshots for Approach #2

A screenshot of a cell phone

Description automatically generated

**A screenshot of a social media post

Description automatically generatedA screenshot of a social media post

Description automatically generated**Run Time for *calculate\_logcosine* – without Indices (left) and with indices (right)

*A screenshot of a cell phone

Description automatically generated***201 Cats - New “My Kind of cats – with preference” queries**

Approach #1

|  |  |  |
| --- | --- | --- |
| **Query Type** | **Run Time (with indices)** | **Run Time (without indices)** |
| Original | 4 min 19 secs | 4 min 19 secs |
| With Pre-Computed Tables (Approach #1) | 2 min 24 secs | 2 min 23 secs |
| With Pre-Computed Tables (Approach #2) | 361 msec | 351 msec |

Original

*A screenshot of a cell phone

Description automatically generated*

**A screenshot of a cell phone

Description automatically generated**

Approach #2

A screenshot of a cell phone

Description automatically generated

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generateds**

**201 Cats - Indices Created**

|  |  |  |  |
| --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** |
| videos | name | GROUP BY | CREATE INDEX videos\_name\_idx ON videos(name); |
| likes | video\_id (Foreign Key) | LEFT JOIN  JOIN  GROUP BY | CREATE INDEX likes\_videoid\_idx ON likes(video\_id); |
| likes | user\_id  (Foreign Key) | LEFT JOIN  JOIN | CREATE INDEX likes\_userid\_idx ON likes(user\_id); |

**201 Cats – Summary of Observations**

* Two approaches were taken to implement pre-computed tables:
  + For approach #1: a table is created for each vector X and Y, where X is the user specified and Y are all the other users. This approach is useful because it takes care of all the expensive computations happening during the cross join between tables "users" and "likes", however, generating the pre-computed table "calculate\_userY" requires a significant amount of memory and run time with 249,975,000 tuples and more than 10 minutes to execute (in this database tested).
  + For approach #2: a table is created to take care of similar computations mentioned on approach #1 but to also to handle joining the tables for vectors X and Y, plus performing the vector inner product and the log cosine calculation.
* Indices
  + On pre-computed tables: Indices seem to be benefiting tables "calculate\_userX" and "calculate\_logcosine", but affecting table "calculate\_userY" from 10min 40sec to 12min 17sec in run time. As mentioned before, table "calculate\_userY" is quite large and is possible that memory access and writing makes the run time slower.
  + On queries: Using indices to run the queries, for both pre-computed tables and without, does not reflect an impact in run time; on milestone 3 I previously showed the same result for the query without pre-computed tables. However, what I believe is interesting on this homework is that indices are probably not helping the query as a whole but when you create a pre-computed table on a smaller segment this one can reflect an improvement in run time.
* The Best Approach/Conclusion
  + Approach #2 is definitely performing better than #1. Creating the table "calculate\_logcosine" will take less than half of the time needed to create "calculate\_userY" alone (for the reasons mentioned previously). Another step that is very expensive on approach #1 is the left join between "calculate\_userX" and "calculate\_userY", considering that the latter table will always be quite large in reference to the former.
  + Approach #2 is also a better option than the original query, when comparing the best scenarios running it collectively (pre-computed table + query) takes less than 4 min 14 sec while the original query takes 4 min 19 sec.

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

***Milestone IV [due March 18 midnight]***

*Your next goal will be to improve the performance of Query 6 by appropriate precomputations. It is understood that the maintenance of the precomputed table(s) will lead to some slowdown while viewers interact. Your precomputation choices must be such that the precomputations that you introduce collectively save more time to this option than they cost to maintain. Precomputed tables will also benefit from creating certain indices on them. Build any beneficial indices. Calibrate your solution against cold runs of the activity script.*

*Submit the following.*

* *The precomputed tables you created: CREATE TABLE statement and query that initially loaded it.*
* *Your new Query 6, which makes the best use of the precomputed tables you chose.*
* *The indices you created (CREATE INDEX scripts)*

**Sales Cube - Summary of Database Size**

|  |  |
| --- | --- |
| **Table Name** | **Number of Tuples** |
| states | 5,000 |
| customers | 700,000 |
| categories | 900,000 |
| products | 800,000 |
| sales | 1,000,000 |

**Sales Cube - Pre-Computed Tables Created**

|  |  |  |
| --- | --- | --- |
| **Table Name** | **Run Time (without Indices)** | **Run Time (with Indices)** |
| top20\_customers | 5 secs 765 msec | 6 secs 420 msec |
| top20\_categories | 4 secs 666 msec | 4 secs 676 msec |
| sales\_customerandcategories | 8 secs 803 msec | 6 secs 875 msec |

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedRun Time for *top20\_customers* – without Indices (left) and with indices (right)

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

**A screenshot of a cell phone

Description automatically generated**Run Time for *top20\_categories* – without Indices (left) and with indices (right)

**A screenshot of a cell phone

Description automatically generated**

A picture containing screen, monitor, table, black

Description automatically generated

**A screenshot of a cell phone

Description automatically generated**Run Time for *sales\_customerandcategories* – without Indices (left) and with indices (right)

**A screenshot of a cell phone

Description automatically generated**

***A screenshot of a cell phone

Description automatically generated*Sales Cube – Queries for #6**

|  |  |  |
| --- | --- | --- |
| **Query Type** | **Run Time (with indices)** | **Run Time (without indices)** |
| Original | 16 secs 545 msec | 14 secs 986 msec |
| With Pre-Computed Tables | 813 msec | 802 msec |

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

Original

**A screenshot of a cell phone

Description automatically generated**

With Pre-Computed Tables

**Sales Cube - Indices Created**

|  |  |  |  |
| --- | --- | --- | --- |
| **Table** | **Column** | **Used In Clause** | **SQL Statement** |
| sales | customer\_id  (Foreign Key) | JOIN | CREATE INDEX sales\_customerid\_idx ON sales(customer\_id); |
| sales | product\_id  (Foreign Key) | JOIN | CREATE INDEX sales\_productid\_idx ON sales(product\_id); |
| products | category\_id  (Foreign Key) | JOIN | CREATE INDEX products\_categoryid\_idx ON products(category\_id); |
| categories | name | GROUP BY | CREATE INDEX categories\_name\_idx ON categories(name); |

**Sales Cube – Summary of Observations**

* Pre-Computed Tables
  + “top20\_customers” table: Finds the top 20 ranked customers based on the price paid. It computes the join between sales and customers, in addition to the rank. This is a nice summary for anyone that might be interested in who are the customers spending the most money in the organization's products or services.
  + “top20\_categories” table: Finds the top 20 ranked categories based on the price paid. It computes the three joins among tables sales, products, and categories; in addition to the rank. This is also a nice summary for anyone that might be interested in maybe understanding what categories are bringing the most revenue or perhaps are the most popular among customers.
  + “sales\_customerandcategories” table: Aggregates all the "sales" with the "product" and "categoties" table information. This pre-computed table seems very useful for users that query the "sales" table frequently but also want information coming from the "products" or "categories" tables, therefore, instead of joining all three tables in every single query (for each user) we can have an aggregation that becomes computationally cheaper in the long run.
* Indices
  + On Pre-Computed Tables: Indices seem to only be beneficial for table "sales\_customerandcategories" by lowering the run time from 8secs 803msec to 6secs 875msec, while "top20\_customers" decreased in performance by almost 1 second and "top20\_categories" observed no change. Is likely that "sales\_customerandcategories" table is benefiting the most from indices because the number of tuples (1,500,000) visited in memory is many orders of magnitude higher as compared to the other top 20 tables (with ~20 tuples).
  + On queries: There is no run time improvement when testing the queries with indices. The original query (no pre-computed tables) increased from 14secs 986msecs to 16secs 545msec when indices were applied, and the query with pre-computed tables also increased from 802msec to 813msec; the latter one is probably within variation (~10msec change).
* Conclusion
  + The cost to generate the query with pre-computed tables (including indices) is about 20 seconds, this is almost 4 seconds slower than the original query with a run time of 16secs 545msec. Given that pre-computed tables are giving worst performance, this is one of those cases where possible applications have to be considered for the tables generated, and how they bring utility to the end users. As mentioned above, someone may be particularly interested in just getting the top 20 customers or categories, or maybe query a single table to get all the sales information. All and all, if the needs of the data ecosystem consider more flexibility for data availability and cheaper performance over the long run it would make sense to implement pre-computed tables, but if optimizing the whole query is the main goal a better option would be to simply implement indices or perhaps to think about other possible optimization alternatives (i.e. re-writing the query or re-designing the schema).