Data enrichment:

a) The data enrichment part involved searching Reddit for the same keywords that were searched for Twitter (gun control). This part involved did not involve adding any new constraints on the original tables, it however included adding more tables and adding some restrictions to the new tables. The new tables are "RedditUser", "RedditPost", "RedditEmotion", and "RedditSentiment". These tables are related to a new source of data, i.e., Reddit.

The "RedditUser" table contains information about the user who posted a post on Reddit, and it has only one column, "user_name", which is the primary key since that is all the was obtained from the data source.

The "RedditPost" table contains information about a post on Reddit. It has columns for post ID ("pid"), the author's username ("author_name"), the title of the post ("title"), the post's URL ("url"), the subreddit it was posted in ("subreddit"), and the post's date ("post_date"). The "author_name" column is a foreign key referencing the "RedditUser" table. Here the title is used instead of the posts to allow some of fairness in comparing to the tweets in terms of the number of characters allowed. In twitter the limit 280 and the Reddit titles have a limit of 140 but there does not exist any limit to the Reddit post body. Finally, we thought that it will cause any biases since one the small sample of test data, we found that titles are normally very indicative of the content of the post itself.

The "RedditEmotion" table contains emotion analysis data for a post on Reddit. It has columns for the post ID ("post_id"), the name of the emotion model used ("emotion_model_name"), and the probabilities of joy, optimism, sadness, and anger. The "post_id" column is a foreign key referencing the "RedditPost" table.

The "RedditSentiment" table contains sentiment analysis data for a post on Reddit. It has columns for the post ID ("post_id"), the name of the sentiment model used ("sentiment_model_name"), and the probabilities of positive, negative, and neutral sentiment. The "post_id" column is a foreign key referencing the "RedditPost" table.

The main constraints on that are present in the newly added tables are:

- Primary key constraint: this is a built-in constraint stating that each primary key must uniquely identify each row and that the primary key cannot be null.
- Referential constraint: this is a built-in constraint stating that each foreign key is valid only if they appear as values of a primary key in the referred table or it can be null.

Finally, it can be observed that all the foreign keys are set to CASCADE on DELETE and UPDATE, which means that if a user or post is deleted or updated, the related data in the other tables will also be deleted or updated.

b) The new data allowed us to mainly compare the level of negativity regarding the topic of guns on different platforms, namely Reddit and Twitter. This allowed us to have a greater insight into the patterns that can be observed in different social media and more especially Twitter. First, in order to achieve that we compared the highly negative posts on different platforms. We defined highly negative as having a score of 80% or more in the negative probability in the sentiment analysis.
This query counted the number of each highly negative Reddit post where the negative probability was rounded to two decimal places:

```
SELECT ROUND(CAST(RS.negative_prob AS numeric), 2), COUNT(*)
FROM "RedditPost" RP, "RedditSentiment" RS
WHERE RP.pid = RS.post_id AND RS.negative_prob >= '0.80'
GROUP BY ROUND(CAST(RS.negative_prob AS numeric), 2)
ORDER BY COUNT(*) DESC;
```

Then the following query was run on Twitter posts to get the same result:

```
SELECT ROUND(CAST(S.negative_prob AS numeric), 2), COUNT(*)
FROM "Tweet" T, "Sentiment" S
WHERE T.tid = S.tweeter_id AND S.negative_prob >= '0.80'
GROUP BY ROUND(CAST(S.negative_prob AS numeric), 2)
ORDER BY COUNT(*) DESC;
```

Secondly, in order to see some of the highly negative Reddit Post, the following SQL query was run:

```
SELECT RP.title, RS.negative_prob
FROM "RedditPost" RP, "RedditSentiment" RS
WHERE RP.pid = RS.post_id AND RS.negative_prob >= '0.80'
ORDER BY RS.negative_prob;
```

Finally, since the Reddit data included more recent records, it was suggested to find the number of Reddit posts that were created after Saturday, 3 December 2022 05:00:00 grouped by the post date to know how many posts were created at each timestamp. This was done by the following query:

```
SELECT RP.post_date, COUNT(*) AS number_of_posts
FROM "RedditPost" RP
WHERE RP.post_date >= to_timestamp(1670043600)
GROUP BY RP.post_date;
```

And to see some of the posts that occurred after the previously mentioned date, the following query was run:

```
SELECT RP.author_name, RP.title
FROM "RedditPost" RP
WHERE RP.post_date >= to_timestamp(1670043600)
ORDER BY RP.post_date;
```

c) This part shows for every query the execution time before and after adding an index. As it can be observed from the queries below, after adding the index on a specific column that was mainly accessed, the execution time on average was reduced compared to the execution time before adding the index.

Query 1:

```
SELECT ROUND(CAST(RS.negative_prob AS numeric), 2), COUNT(*)
FROM "RedditPost" RP, "RedditSentiment" RS
WHERE RP.pid = RS.post_id AND RS.negative_prob >= '0.80'
GROUP BY ROUND(CAST(RS.negative_prob AS numeric), 2)
ORDER BY COUNT(*) DESC;
```

Before index:

	QUERY PLAN
1	Sort (cost=561.14..563.66 rows=1010 width=40) (actual time=6.070..6.073 rows=17 loops=1)
2	Sort Key: (count(*)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=495.59..510.74 rows=1010 width=40) (actual time=6.049..6.060 rows=17 loops=1)
5	Group Key: round((rs.negative_prob)::numeric, 2)
6	Batches: 1 Memory Usage: 73kB
7	-> Hash Join (cost=213.62..490.46 rows=1026 width=32) (actual time=1.785..5.675 rows=1026 loops=1)
8	Hash Cond: (rp.pid = rs.post_id)
9	-> Index Only Scan using "RedditPost_pkey" on "RedditPost" rp (cost=0.29..227.38 rows=9161 width=10) (actual time=0.010..1.442 rc
10	Heap Fetches: 518
11	-> Hash (cost=200.51..200.51 rows=1026 width=18) (actual time=1.739..1.740 rows=1026 loops=1)
12	Buckets: 2048 Batches: 1 Memory Usage: 67kB
13	-> Seq Scan on "RedditSentiment" rs (cost=0.00..200.51 rows=1026 width=18) (actual time=0.010..1.475 rows=1026 loops=1)
14	Filter: (negative_prob >= '0.8'::double precision)
15	Rows Removed by Filter: 8135
16	Planning Time: 0.307 ms
17	Execution Time: 6.178 ms

After index (on RedditSentiment.negative_prob):

	QUERY PLAN
1	Sort (cost=472.09..474.61 rows=1010 width=40) (actual time=3.052..3.055 rows=17 loops=1)
2	Sort Key: (count(*)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=406.54..421.69 rows=1010 width=40) (actual time=3.039..3.047 rows=17 loops=1)
5	Group Key: round((rs.negative_prob)::numeric, 2)
6	Batches: 1 Memory Usage: 73kB
7	-> Hash Join (cost=124.57..401.41 rows=1026 width=32) (actual time=0.472..2.848 rows=1026 loops=1)
8	Hash Cond: (rp.pid = rs.post_id)
9	-> Index Only Scan using "RedditPost_pkey" on "RedditPost" rp (cost=0.29..227.38 rows=9161 width=10) (actual time=0.006..0.893 rc
10	Heap Fetches: 518
11	-> Hash (cost=111.46..111.46 rows=1026 width=18) (actual time=0.445..0.446 rows=1026 loops=1)
12	Buckets: 2048 Batches: 1 Memory Usage: 67kB
13	-> Bitmap Heap Scan on "RedditSentiment" rs (cost=12.64..111.46 rows=1026 width=18) (actual time=0.057..0.324 rows=1026 lo
14	Recheck Cond: (negative_prob >= '0.8'::double precision)
15	Heap Blocks: exact=86
16	-> Bitmap Index Scan on reddit_neg_prob (cost=0.00..12.38 rows=1026 width=0) (actual time=0.042..0.042 rows=1026 loops=
17	Index Cond: (negative_prob >= '0.8'::double precision)
18	Planning Time: 0.224 ms
19	Execution Time: 3.132 ms

Query 2:

```
SELECT ROUND(CAST(RS.negative_prob AS numeric), 2), COUNT(*)
FROM "RedditPost" RP, "RedditSentiment" RS
WHERE RP.pid = RS.post_id AND RS.negative_prob >= '0.80'
GROUP BY ROUND(CAST(RS.negative_prob AS numeric), 2)
ORDER BY COUNT(*) DESC;
```

Before index:

	QUERY PLAN
1	Sort (cost=8828.50..8891.35 rows=25143 width=40) (actual time=296.715..296.719 rows=18 loops=1)
2	Sort Key: (count(*)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=6613.67..6990.81 rows=25143 width=40) (actual time=296.623..296.706 rows=18 loops=1)
5	Group Key: round((s.negative_prob)::numeric, 2)
6	Batches: 1 Memory Usage: 793kB
7	-> Hash Join (cost=2775.36..6474.04 rows=27925 width=32) (actual time=18.686..210.194 rows=27802 loops=1)
8	Hash Cond: (t.tid = s.tweeter_id)
9	-> Index Only Scan using "Tweet_pkey" on "Tweet" t (cost=0.42..2863.92 rows=111021 width=8) (actual time=0.012..14.584 rows=11
10	Heap Fetches: 21216
11	-> Hash (cost=2425.88..2425.88 rows=27925 width=16) (actual time=18.553..18.555 rows=27802 loops=1)
12	Buckets: 32768 Batches: 1 Memory Usage: 1560kB
13	-> Seq Scan on "Sentiment" s (cost=0.00..2425.88 rows=27925 width=16) (actual time=0.013..12.012 rows=27802 loops=1)
14	Filter: (negative_prob >= '0.8'::double precision)
15	Rows Removed by Filter: 83201
16	Planning Time: 0.381 ms
17	Execution Time: 296.869 ms

After index (on Sentiment.negative_prob):

	QUERY PLAN
1	Sort (cost=8091.30..8154.14 rows=25136 width=40) (actual time=75.457..75.462 rows=18 loops=1)
2	Sort Key: (count(*)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=5877.14..6254.18 rows=25136 width=40) (actual time=75.376..75.448 rows=18 loops=1)
5	Group Key: round((s.negative_prob)::numeric, 2)
6	Batches: 1 Memory Usage: 793kB
7	-> Hash Join (cost=2038.95..5737.55 rows=27918 width=32) (actual time=11.387..58.849 rows=27802 loops=1)
8	Hash Cond: (t.tid = s.tweeter_id)
9	-> Index Only Scan using "Tweet_pkey" on "Tweet" t (cost=0.42..2863.92 rows=111021 width=8) (actual time=0.011..12.981 rows=11
10	Heap Fetches: 21216
11	-> Hash (cost=1689.56..1689.56 rows=27918 width=16) (actual time=11.335..11.337 rows=27802 loops=1)
12	Buckets: 32768 Batches: 1 Memory Usage: 1560kB
13	-> Bitmap Heap Scan on "Sentiment" s (cost=302.58..1689.56 rows=27918 width=16) (actual time=1.802..7.570 rows=27802 loop
14	Recheck Cond: (negative_prob >= '0.8'::double precision)
15	Heap Blocks: exact=1038
16	-> Bitmap Index Scan on twitter_neg_prob (cost=0.00..295.60 rows=27918 width=0) (actual time=1.669..1.669 rows=27802 lo
17	Index Cond: (negative_prob >= '0.8'::double precision)
18	Planning Time: 0.217 ms
19	Execution Time: 75.666 ms

Query 3:

```
SELECT RP.title, RS.negative_prob
FROM "RedditPost" RP, "RedditSentiment" RS
WHERE RP.pid = RS.post_id AND RS.negative_prob >= '0.80'
ORDER BY RS.negative_prob;
```

Before index:

	QUERY PLAN
1	Sort (cost=673.88..676.44 rows=1026 width=77) (actual time=4.773..4.892 rows=1026 loops=1)
2	Sort Key: rs.negative_prob
3	Sort Method: quicksort Memory: 242kB
4	-> Hash Join (cost=213.34..622.56 rows=1026 width=77) (actual time=1.736..4.450 rows=1026 loops=1)
5	Hash Cond: (rp.pid = rs.post_id)
6	-> Seq Scan on "RedditPost" rp (cost=0.00..364.61 rows=9161 width=79) (actual time=0.008..0.793 rows=9161 loops=1)
7	-> Hash (cost=200.51..200.51 rows=1026 width=18) (actual time=1.718..1.719 rows=1026 loops=1)
8	Buckets: 2048 Batches: 1 Memory Usage: 67kB
9	-> Seq Scan on "RedditSentiment" rs (cost=0.00..200.51 rows=1026 width=18) (actual time=0.008..1.417 rows=1026 loops=1)
10	Filter: (negative_prob >= '0.8'::double precision)
11	Rows Removed by Filter: 8135
12	Planning Time: 0.221 ms
13	Execution Time: 5.017 ms

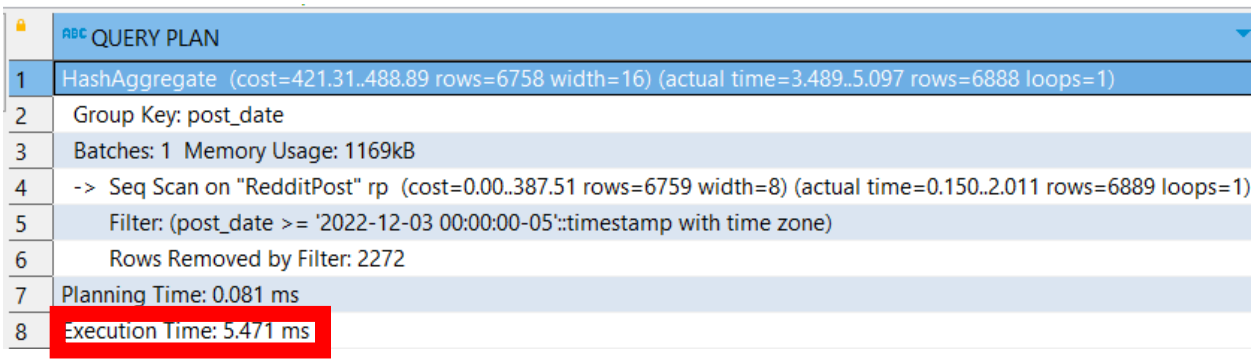
After index (on RedditSentiment.negative_prob):

	QUERY PLAN
1	Sort (cost=584.82..587.39 rows=1026 width=77) (actual time=2.723..2.814 rows=1026 loops=1)
2	Sort Key: rs.negative_prob
3	Sort Method: quicksort Memory: 242kB
4	-> Hash Join (cost=124.29..533.51 rows=1026 width=77) (actual time=0.467..2.482 rows=1026 loops=1)
5	Hash Cond: (rp.pid = rs.post_id)
6	-> Seq Scan on "RedditPost" rp (cost=0.00..364.61 rows=9161 width=79) (actual time=0.005..0.667 rows=9161 loops=1)
7	-> Hash (cost=111.46..111.46 rows=1026 width=18) (actual time=0.453..0.454 rows=1026 loops=1)
8	Buckets: 2048 Batches: 1 Memory Usage: 67kB
9	-> Bitmap Heap Scan on "RedditSentiment" rs (cost=12.64..111.46 rows=1026 width=18) (actual time=0.071..0.324 rows=1026 loops=1)
10	Recheck Cond: (negative_prob >= '0.8'::double precision)
11	Heap Blocks: exact=86
12	-> Bitmap Index Scan on reddit_neg_prob (cost=0.00..12.38 rows=1026 width=0) (actual time=0.056..0.056 rows=1026 loops=1)
13	Index Cond: (negative_prob >= '0.8'::double precision)
14	Planning Time: 0.212 ms
15	Execution Time: 2.914 ms

Query 4:

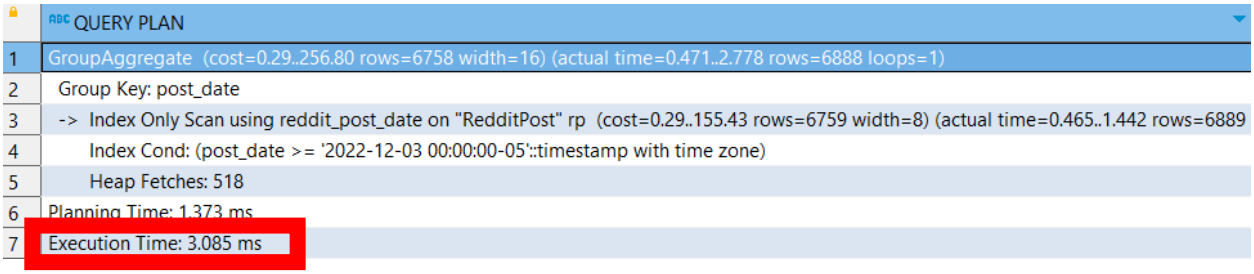
```
SELECT RP.post_date, COUNT(*) AS number_of_posts
FROM "RedditPost" RP
WHERE RP.post_date >= to_timestamp(1670043600)
GROUP BY RP.post_date;
```

Before index:



	QUERY PLAN
1	HashAggregate (cost=421.31..488.89 rows=6758 width=16) (actual time=3.489..5.097 rows=6888 loops=1)
2	Group Key: post_date
3	Batches: 1 Memory Usage: 1169kB
4	-> Seq Scan on "RedditPost" rp (cost=0.00..387.51 rows=6759 width=8) (actual time=0.150..2.011 rows=6889 loops=1)
5	Filter: (post_date >= '2022-12-03 00:00:00-05'::timestamp with time zone)
6	Rows Removed by Filter: 2272
7	Planning Time: 0.081 ms
8	Execution Time: 5.471 ms

After index (on RedditPost.post_date):

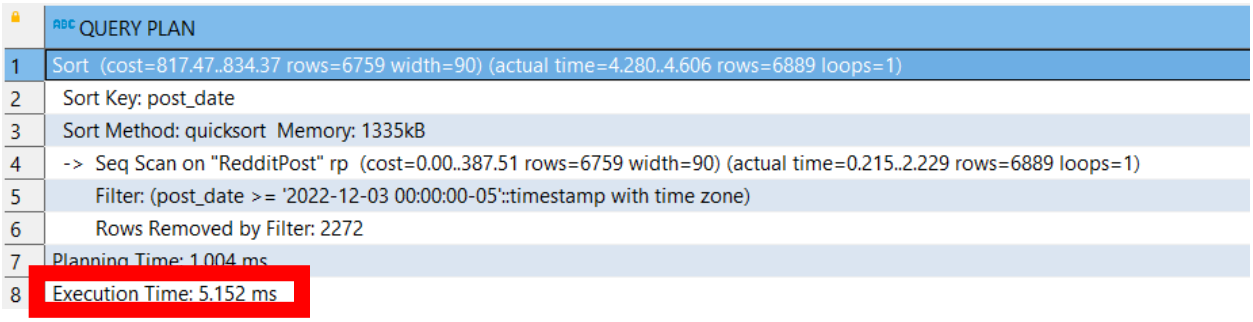


	QUERY PLAN
1	GroupAggregate (cost=0.29..256.80 rows=6758 width=16) (actual time=0.471..2.778 rows=6888 loops=1)
2	Group Key: post_date
3	-> Index Only Scan using reddit_post_date on "RedditPost" rp (cost=0.29..155.43 rows=6759 width=8) (actual time=0.465..1.442 rows=6889
4	Index Cond: (post_date >= '2022-12-03 00:00:00-05'::timestamp with time zone)
5	Heap Fetches: 518
6	Planning Time: 1.373 ms
7	Execution Time: 3.085 ms

Query 5:

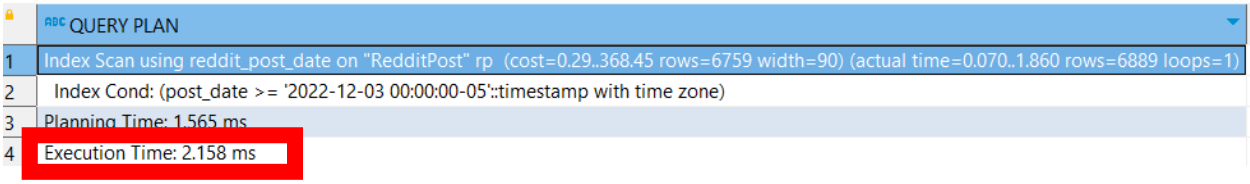
```
SELECT RP.author_name, RP.title
FROM "RedditPost" RP
WHERE RP.post_date >= to_timestamp(1670043600)
ORDER BY RP.post_date;
```

Before index:



	QUERY PLAN
1	Sort (cost=817.47..834.37 rows=6759 width=90) (actual time=4.280..4.606 rows=6889 loops=1)
2	Sort Key: post_date
3	Sort Method: quicksort Memory: 1335kB
4	-> Seq Scan on "RedditPost" rp (cost=0.00..387.51 rows=6759 width=90) (actual time=0.215..2.229 rows=6889 loops=1)
5	Filter: (post_date >= '2022-12-03 00:00:00-05'::timestamp with time zone)
6	Rows Removed by Filter: 2272
7	Planning Time: 1.004 ms
8	Execution Time: 5.152 ms

After index (on RedditPost.post_date):



	QUERY PLAN
1	Index Scan using reddit_post_date on "RedditPost" rp (cost=0.29..368.45 rows=6759 width=90) (actual time=0.070..1.860 rows=6889 loops=1)
2	Index Cond: (post_date >= '2022-12-03 00:00:00-05'::timestamp with time zone)
3	Planning Time: 1.565 ms
4	Execution Time: 2.158 ms