.py file for creating tables and databases.
.sql file for querying

Users table:

```
select * from users
limit 50;
```

user_id	username	email	country	created_at	status
1	clevine	masseydavid@example.com	Brazil	2023-06-23	active
2	wschultz	steven49@example.com	Norfolk Island	2023-06-25	inactive
3	joshua34	awarner@example.org	Myanmar	2023-11-28	inactive
4	kimberlyrichardson	danielleforbes@example.com	Macao	2025-09-18	inactive
5	darryl11	richard59@example.com	Faroe Islands	2025-10-19	inactive
6	jason55	rmartinez@example.org	Latvia	2023-12-12	suspended
7	ashleyallen	drivera@example.net	Nicaragua	2023-08-24	active
8	zfrench	smiththomas@example.com	Nicaragua	2024-06-01	active

Comments table:

```
select * from comments
limit 50;
```

comment_id	post_id	user_id	comment_text	created_at	likes
1	1070707	171056	Along rule example environmental vote far save...	2025-10-08	77
2	1136274	478289	Require though student still international light w...	2024-12-22	49
3	388616	711556	Above even opportunity color. Always notice so...	2024-09-02	17
4	800700	912783	Effect serve bed according within report need c...	2024-07-29	21
5	821864	329112	Me probably chance especially. Agreement stop...	2024-10-12	96
6	52592	542221	Agent machine establish yeah protect live have ...	2024-12-04	56
7	562636	370771	Election others how natural future head. It tota...	2024-02-02	81
8	138723	545201	Far doctor green ok hotel generation sport. Agr...	2025-10-11	70

Posts table:

```
select * from posts
limit 50;
```

post_id	user_id	title	category	created_at	views	likes
1	573272	See visit charge ago few week price stand.	Technology	2025-05-23	412	959
2	744867	Stop world throughout tonight can happy accept.	Sports	2024-06-06	6257	186
3	848434	Keep yet before read meeting.	Travel	2025-10-10	2692	379
4	394678	Them near should structure security partner wo...	Travel	2024-10-03	7894	852
5	548659	Capital act find pay consider early.	Food	2023-12-08	9757	884
6	838699	Start high available establish for.	Food	2023-11-12	2144	642
7	811830	Evidence himself page another bank such.	Fashion	2024-02-23	4021	858
8	949036	Through seek smile system price guy.	Fashion	2025-01-18	2610	195

Unoptimized query by AI:

```
SELECT
    u.user_id,
    u.username,
    u.email,
    u.country,
    (SELECT COUNT(*)
     FROM posts p
     WHERE p.user_id = u.user_id
       AND YEAR(p.created_at) = 2024
       AND p.category IN ('Technology', 'Travel')) as posts_count,
    (SELECT SUM(p2.views)
     FROM posts p2
     WHERE p2.user_id = u.user_id
       AND YEAR(p2.created_at) = 2024
       AND p2.category IN ('Technology', 'Travel')) as total_views,
    (SELECT COUNT(*)
     FROM comments c
     WHERE c.user_id = u.user_id
       AND YEAR(c.created_at) = 2024) as comments_count,
    (SELECT AVG(c2.likes)
     FROM comments c2
     WHERE c2.user_id = u.user_id
       AND YEAR(c2.created_at) = 2024) as avg_comment_likes
FROM users u
WHERE u.user_id IN (
    SELECT p3.user_id
    FROM posts p3
    WHERE YEAR(p3.created_at) = 2024
      AND p3.category IN ('Technology', 'Travel')
)
AND u.status = 'active'
ORDER BY total_views DESC
```

LIMIT 100;

How it works:

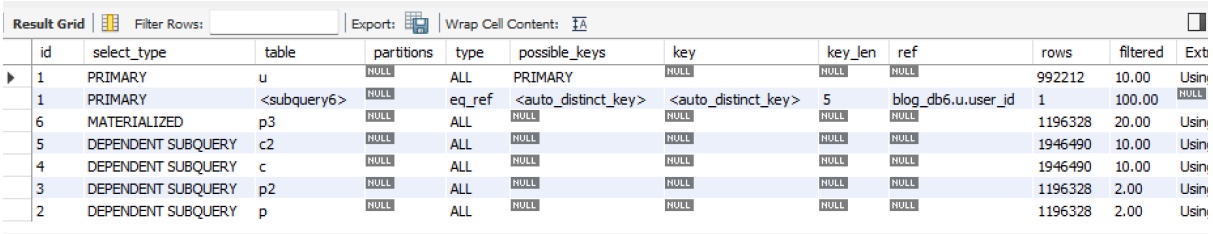
Iterates over all users, who have posted in 2024 in Technology or Travel category and is active. For each user, it iterates over posts and counts every 2024 posts in Technology and Travel for this user, then it iterates again, getting total views with the same where. Then it iterates over comments table, gets count of the comments the person has left in 2024 and then it iterates again, getting average of likes on these comments for person. Then it selects these metrics and orders the table by total_views descending and limits to 100 showing.

Returns:

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

user_id	username	email	country	posts_count	total_views	comments_count	avg_comment_likes
1321	joseph24	matthewwest@example.org	Albania	2	16771	3	36.6667
1649	pereztroy	lspencer@example.net	Montenegro	2	15367	0	NULL
103	lorileonard	garciacorey@example.com	Bosnia and Herzegovina	3	14466	2	77.5000
1392	jerry32	tward@example.com	Trinidad and Tobago	2	14369	1	33.0000
1094	hermanamy	heidiwhite@example.org	Saint Pierre and Miquelon	2	12900	0	NULL
776	sarahhicks	rebecca25@example.org	Lao People's Democratic Republic	2	12775	1	59.0000
1319	cwest	dawnalvarez@example.com	Barbados	2	12125	0	NULL
821	cathy91	kruegernicholas@example.com	Tanzania	2	11822	2	47.0000

Explain statement:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Ext
1	PRIMARY	u	NULL	ALL	PRIMARY	NULL	NULL	NULL	992212	10.00	Usin
1	PRIMARY	<subquery6>	NULL	eq_ref	<auto_distinct_key>	<auto_distinct_key>	5	blog_db6.u.user_id	1	100.00	NULL
6	MATERIALIZED	p3	NULL	ALL	NULL	NULL	NULL	NULL	1196328	20.00	Usin
5	DEPENDENT SUBQUERY	c2	NULL	ALL	NULL	NULL	NULL	NULL	1946490	10.00	Usin
4	DEPENDENT SUBQUERY	c	NULL	ALL	NULL	NULL	NULL	NULL	1946490	10.00	Usin
3	DEPENDENT SUBQUERY	p2	NULL	ALL	NULL	NULL	NULL	NULL	1196328	2.00	Usin
2	DEPENDENT SUBQUERY	p	NULL	ALL	NULL	NULL	NULL	NULL	1196328	2.00	Usin

We can see that every table is a full scan, and there are full scans of multiple same tables, making this query impossible to run in a table with more than 10 000 rows (i've tried). There are no indexes as well.

Indexes:

create index users_indx on users(status, user_id);
create index posts_indx on posts(created_at, category, user_id, views);
create index comments_indx on comments(created_at, user_id, likes);

Optimized query:

```
with posts_year_select as (
    select p.user_id, count(*) as count_posts, sum(p.views) as sum_views
    from posts p use index (posts_indx)
    where p.created_at >= '2024-01-01'
          and p.created_at < '2025-01-01'
          and p.category in ('Technology', 'Travel')
    group by p.user_id
    ),
comments_year_select as (
          select c.user_id, count(*) as count_comments, avg(c.likes) as avg_likes
    from comments c use index(comments_indx)
    where c.created_at >= '2024-01-01'
    and c.created_at < '2025-01-01'
    group by c.user_id
)
SELECT
    u.user_id,
    u.username,
    u.email,
    u.country,
    p.count_posts,
    p.sum_views,
    coalesce(c.count_comments, 0) as count_comments,
    c.avg_likes as avg_likes
    from users u use index (users_indx)
    inner join posts_year_select p on u.user_id = p.user_id
    left join comments_year_select c on u.user_id = c.user_id
    where u.status = "active"
    order by p.sum_views desc
    limit 100;
```

How it works:

First we use a cte in the posts table, selecting user_id, count and sum of views for posts in 2024 in Technology and Travel, grouped by user_id. Then we do a similar cte in the comments table, selecting user_id, count and avg of likes for 2024 comments, grouped by user_id. Then we select some columns from active users inner joined with posts_year_selected by user_id, then left join with comments. After we order by sum_views descending and limit to 100 showing.

Returns:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	username	email	country	count_posts	sum_views	count_comments	avg_likes
1321	joseph24	matthewwest@example.org	Albania	2	16771	3	36.6667
1649	pereztroy	lspencer@example.net	Montenegro	2	15367	0	NULL
103	lorileonard	garciacorey@example.com	Bosnia and Herzegovina	3	14466	2	77.5000
1392	jerry32	tward@example.com	Trinidad and Tobago	2	14369	1	33.0000
1094	hermanamy	heidiwhite@example.org	Saint Pierre and Miquelon	2	12900	0	NULL
776	sarahhicks	rebecca25@example.org	Lao People's Democratic Republic	2	12775	1	59.0000
1319	cwest	dawnalvarez@example.com	Barbados	2	12125	0	NULL
821	cathy91	kruegernicholas@example.com	Tanzania	2	11822	2	47.0000
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	NULL

Explain:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	<derived2>	NULL	ALL	NULL	NULL	NULL	NULL	119632	100.00	Using where; Using filesort
1	PRIMARY	u	NULL	eq_ref	users_indx	users_indx	1027	const,p.user_id	1	100.00	NULL
1	PRIMARY	<derived3>	NULL	ref	<auto_key0>	<auto_key0>	5	p.user_id	10	100.00	NULL
3	DERIVED	c	NULL	range	comments_indx	comments_indx	4	NULL	973245	100.00	Using where; Using index; Using t
2	DERIVED	p	NULL	index	posts_indx	posts_indx	1037	NULL	1196328	10.00	Using where; Using index; Using t

We can a much better type of selection, only having 1 full-table scan in a derived table and it doesnt have that many rows. Others are index or 1/multiple row scans, greatly benefitting the performance. We can also see all 3 indexes used in their related tables. Takes about 20-25 seconds to perform. Original stopped querying at 10 minutes mark, disconnecting from sql server.

Explain analyze:

-> Limit: 100 row(s) (cost=418698 rows=0) (actual time=21054..21126 rows=100 loops=1)
    -> Nested loop left join (cost=418698 rows=0) (actual time=21054..21126 rows=100 loops=1)
        -> Nested loop inner join (cost=119617 rows=0) (actual time=2431..2501 rows=100 loops=1)
            -> Sort: p.sum_views DESC (cost=2.6..2.6 rows=0) (actual time=2429..2429 rows=338 loops=1)
                -> Filter: (p.user_id is not null) (cost=2.5..2.5 rows=0) (actual time=2144..2205 rows=181471 loops=1)
                    -> Table scan on p (cost=2.5..2.5 rows=0) (actual time=2144..2180 rows=181471 loops=1)
                        -> Materialize CTE posts_year_select (cost=0..0 rows=0) (actual time=2144..2144 rows=181471 loops=1)
                            -> Table scan on <temporary> (actual time=2057..2093 rows=181471 loops=1)
                                -> Aggregate using temporary table (actual time=2057..2057 rows=181471 loops=1)
                                    -> Filter: ((p.created_at >= DATE'2024-01-01') and (p.created_at < DATE'2025-01-01') and (p.category in ('Technology','Travel'))) (cost=126136 rows=119633) (actual time=51..1642 rows=200488 loops=1)
                                        -> Covering index scan on p using posts_indx (cost=126136 rows=1.2e+6) (actual time=0.132..1135 rows=1.2e+6 loops=1)

-> Single-row index lookup on u using users_indx (status='active', user_id=p.user_id) (cost=1 rows=1) (actual time=0.214..0.214 rows=0.296 loops=338)
-> Index lookup on c using <auto_key0> (user_id=p.user_id) (cost=0.25..2.5 rows=10) (actual time=186..186 rows=0.61 loops=100)
-> Materialize CTE comments_year_select (cost=0..0 rows=0) (actual time=18622..18622 rows=632176 loops=1)
-> Table scan on <temporary> (actual time=12972..13799 rows=632176 loops=1)
-> Aggregate using temporary table (actual time=12972..12972 rows=632175 loops=1)
-> Filter: ((c.created_at >= DATE'2024-01-01') and (c.created_at < DATE'2025-01-01')) (cost=196784 rows=973245) (actual time=4.05..2125 rows=1e+6 loops=1)
-> Covering index range scan on c using comments_indx over ('2024-01-01' <= created_at < '2025-01-01') (cost=196784 rows=973245) (actual time=4.04..1373 rows=1e+6 loops=1)