PART 3

--CREATING TABLES (with postgresql)

```
CREATE TABLE users (
   Id INT NOT NULL,
   Reputation INT DEFAULT NULL,
   CreationDate TIMESTAMP DEFAULT NULL,
   DisplayName VARCHAR(255) DEFAULT NULL,
   LastAccessDate TIMESTAMP DEFAULT NULL,
   WebsiteUrl VARCHAR(255) DEFAULT NULL,
   Location VARCHAR(255) DEFAULT NULL,
   AboutMe TEXT DEFAULT NULL,
   Views INT DEFAULT NULL,
   UpVotes INT DEFAULT NULL,
   DownVotes INT DEFAULT NULL,
   AccountId INT DEFAULT NULL,
   Age INT DEFAULT NULL,
   ProfileImageUrl VARCHAR(255) DEFAULT NULL,
   PRIMARY KEY (Id)
);

CREATE TABLE posts (
   Id INT NOT NULL,
   PostTypeId INT DEFAULT NULL,
   AcceptedAnswerId INT DEFAULT NULL,
   CreationDate TIMESTAMP DEFAULT NULL,
   Score INT DEFAULT NULL,
   ViewCount INT DEFAULT NULL,
   Body TEXT DEFAULT NULL,
   OwnerUserId INT DEFAULT NULL,
   LasActivityDate TIMESTAMP DEFAULT NULL,
   Title VARCHAR(255) DEFAULT NULL,
   Tags VARCHAR(255) DEFAULT NULL,
   AnswerCount INT DEFAULT NULL,
   CommentCount INT DEFAULT NULL,
   FavoriteCount INT DEFAULT NULL,
   LastEditorUserId INT DEFAULT NULL,
   LastEditDate TIMESTAMP DEFAULT NULL,
   CommunityOwnedDate TIMESTAMP DEFAULT NULL,
   ParentId INT DEFAULT NULL,
   ClosedDate TIMESTAMP DEFAULT NULL,
   OwnerDisplayName VARCHAR(255) DEFAULT NULL,
   LastEditorDisplayName VARCHAR(255) DEFAULT NULL,
   PRIMARY KEY (Id),
```

```
  CONSTRAINT posts_LastEditorUserId_fkey FOREIGN KEY (LastEditorUserId)
    REFERENCES users (Id),
  CONSTRAINT posts_OwnerUserId_fkey FOREIGN KEY (OwnerUserId)
    REFERENCES users (Id),
  CONSTRAINT posts_ParentId_fkey FOREIGN KEY (ParentId)
    REFERENCES posts (Id)
);

CREATE TABLE postLinks (
  Id INT NOT NULL,
  CreationDate TIMESTAMP DEFAULT NULL,
  PostId INT DEFAULT NULL,
  RelatedPostId INT DEFAULT NULL,
  LinkTypeId INT DEFAULT NULL,
  PRIMARY KEY (Id),
  CONSTRAINT postlinks_stripped_PostId_fkey FOREIGN KEY (PostId)
    REFERENCES posts (Id),
  CONSTRAINT postlinks_stripped_RelatedPostId_fkey FOREIGN KEY (RelatedPostId)
    REFERENCES posts (Id)
);

CREATE TABLE tags (
  Id INT NOT NULL,
  TagName VARCHAR(255) DEFAULT NULL,
  Count INT DEFAULT NULL,
  ExcerptPostId INT DEFAULT NULL,
  WikiPostId INT DEFAULT NULL,
  PRIMARY KEY (Id),
  FOREIGN KEY (ExcerptPostId)
    REFERENCES posts (Id)
);

CREATE TABLE badges (
  Id INT NOT NULL,
  UserId INT DEFAULT NULL,
  Name VARCHAR(255) DEFAULT NULL,
  Date timestamp DEFAULT NULL,
  PRIMARY KEY (Id),
  FOREIGN KEY (UserId)
    REFERENCES users (Id)
);

CREATE TABLE comments (
  Id INT NOT NULL,
```

```
    PostId INT DEFAULT NULL,
    Score INT DEFAULT NULL,
    Text TEXT DEFAULT NULL,
    CreationDate timestamp DEFAULT NULL,
    UserId INT DEFAULT NULL,
    UserDisplayName VARCHAR(255) DEFAULT NULL,
    PRIMARY KEY (Id),
    FOREIGN KEY (PostId)
        REFERENCES posts (Id),
    FOREIGN KEY (UserId)
        REFERENCES users (Id)
);

CREATE TABLE postHistory (
    Id INT NOT NULL,
    PostHistoryTypeId INT DEFAULT NULL,
    PostId INT DEFAULT NULL,
    RevisionGUID VARCHAR(255) DEFAULT NULL,
    CreationDate timestamp DEFAULT NULL,
    UserId INT DEFAULT NULL,
    Text TEXT DEFAULT NULL,
    Comment TEXT DEFAULT NULL,
    UserDisplayName VARCHAR(255) DEFAULT NULL,
    PRIMARY KEY (Id),
    FOREIGN KEY (PostId)
        REFERENCES posts (Id),
    FOREIGN KEY (UserId)
        REFERENCES users (Id)
);

CREATE TABLE votes (
    Id INT NOT NULL,
    PostId INT DEFAULT NULL,
    VoteTypeId INT DEFAULT NULL,
    CreationDate timestamp DEFAULT NULL,
    UserId INT DEFAULT NULL,
    BountyAmount INT DEFAULT NULL,
    PRIMARY KEY (Id),
    FOREIGN KEY (PostId)
        REFERENCES posts (Id),
    FOREIGN KEY (UserId)
        REFERENCES users (Id)
);
```

--IMPORTING DATA (substitute for personalized csv directories)

-Using ‘Import/Export data’ in PGAdmin4 GUI, or alternatively by running the following queries:

```
copy public.users (id, reputation, creationdate, displayname, lastaccessdate, websiteurl, location, aboutme, views, upvotes, downvotes, accountid, age, profileimageurl) FROM 'C:/Users/Desktop/Tables/users.csv' DELIMITER ',' CSV HEADER ENCODING 'UTF8' QUOTE '\"' NULL 'null' ESCAPE '\"';"

copy public.posts (id, posttypeid, acceptedanswerid, creationdate, score, viewcount, body, owneruserid, lasactivitydate, title, tags, answercount, commentcount, favoritecount, lasteditoruserid, lasteditdate, communityowneddate, parentid, closeddate, ownerdisplayname, lasteditordisplayname) FROM 'C:/Users/Desktop/Tables/posts.csv' DELIMITER ',' CSV HEADER ENCODING 'UTF8' QUOTE '\"' NULL 'null' ESCAPE '\"'

copy public.votes (id, postid, votetypeid, creationdate, userid, bountyamount) FROM 'C:/Users/Desktop/Tables/votes.csv' DELIMITER ',' CSV HEADER ENCODING 'UTF8' QUOTE '\"' NULL 'null' ESCAPE '\"'

copy public.tags (id, tagname, count, excerptpostid, wikipostid) FROM 'C:/Users/Desktop/Tables/tags.csv' DELIMITER ',' CSV HEADER ENCODING 'UTF8' QUOTE '\"' NULL 'null' ESCAPE '\"'

copy public.postlinks (id, creationdate, postid, relatedpostid, linktypeid) FROM 'C:/Users/Desktop/Tables/postLinks.csv' DELIMITER ',' CSV HEADER ENCODING 'UTF8' QUOTE '\"' NULL 'null' ESCAPE '\"';"

copy public.posthistory (id, posthistorytypeid, postid, revisionguid, creationdate, userid, text, comment, userdisplayname) FROM 'C:/Users/Desktop/Tables/postHistory.csv' DELIMITER ',' CSV HEADER ENCODING 'UTF8' QUOTE '\"' NULL 'null' ESCAPE '\"';"

copy public.comments (id, postid, score, text, creationdate, userid, userdisplayname) FROM 'C:/Users/Tables/comments.csv' DELIMITER ',' CSV HEADER ENCODING 'UTF8' QUOTE '\"' NULL 'null' ESCAPE '\"';"

copy public.badges (id, userid, name, date) FROM 'C:/Users/Desktop/Tables/badges.csv' DELIMITER ',' CSV HEADER ENCODING 'UTF8' QUOTE '\"' NULL 'null' ESCAPE '\"';"
```

--10 Reports (Queries)

```
-- 1) Showing top 10 users with the most badges (based on the ‘UserId’ column in the ‘badges’ table)
SELECT u.DisplayName, COUNT(*) as TotalBadges
FROM users u, badges b
```

```
WHERE b.UserId = u.Id
GROUP BY u.DisplayName
ORDER BY TotalBadges DESC
LIMIT 10;
```

	displayname character varying (255)	totalbadges bigint
1	whuber	456
2	Glen_b	318
3	chl	282
4	gung	256
5	Jeromy Anglim	222
6	John	182
7	Macro	176
8	Tal Galili	159
9	Rob Hyndman	156
10	Peter Flom	152

```
-- 2) Showing top 10 posts with the most comments (based on the 'PostId' column in the
'comments' table)
SELECT p.Title, COUNT(*) as TotalComments
FROM posts p, comments c
WHERE c.PostId = p.id
GROUP BY p.title
ORDER BY TotalComments DESC
LIMIT 10;
```

	title character varying (255)	totalcomments bigint
1	[null]	83628
2	Does the presence of an outlier increase the probability that another outlier will also be present on the same observa...	37
3	Amazon interview question—probability of 2nd interview	35
4	Which test is better suited to compare averaged versus single data sets	33
5	How to fit a mixture of Gamma distributions to the PMF of a discrete distribution?	33
6	How do I incorporate an innovative outlier at observation 48 in my ARIMA model?	33
7	Binomial GLM - Predicting the time of buying of a product	32
8	Rejection sampling from a normal distribution	31
9	Composition of probability density	29
10	What are the chances my wife has lupus?	29

-- 3) Showing top 10 users who have made the most edits to posts (based on the 'UserId' column in the 'postHistory' table)

```
SELECT u.DisplayName, COUNT(*) as TotalEdits
FROM users u, postHistory p
WHERE p.userId = u.Id
GROUP BY u.displayname
ORDER BY TotalEdits DESC
LIMIT 10;
```

	displayname character varying (255)	totaledits bigint
1	gung	7321
2	whuber	7097
3	Glen_b	6900
4	mbq	6570
5	chl	6005
6	Community	4511
7	Jeromy Anglim	3107
8	Nick Cox	3034
9	Peter Flom	2739
10	Nick Stauner	2175

-- 4) Showing the total number of new posts per year, followed by the number of new users per year

```
SELECT EXTRACT(YEAR FROM p.CreationDate) AS Year,
    COUNT(*) AS NumberOfNewPosts
FROM posts p
GROUP BY Year
ORDER BY Year ASC;
```

	year numeric	numberofnewposts bigint
1	2009	18
2	2010	5450
3	2011	13163
4	2012	20113
5	2013	26771
6	2014	26461

```
SELECT EXTRACT(YEAR FROM u.CreationDate) AS Year,
      COUNT(*) AS NumberOfNewUsers
FROM users u
GROUP BY Year
ORDER BY Year ASC;
```

	year numeric	numberofnewusers bigint
1	2010	1678
2	2011	4430
3	2012	7544
4	2013	12231
5	2014	14442

We may note that, oddly enough, this dataset had no new users in the year 2009 yet had 18 posts that year. This may be due to the fact that StackOverFlow started as a company in 2008 and was still in its early stages. We may also notice that as the number of users increases, so does the number of posts, with the slight exception of the number of posts remaining more or less the same between 2013 and 2014.

```
-- 5) Showing the 10 lowest reputable users followed by the 10 highest reputable users along
with their account creation dates
(SELECT u1.DisplayName, u1.Reputation, u1.UpVotes, u1.DownVotes
FROM users u1
WHERE u1.UpVotes > 10
 AND u1.DownVotes > 10
ORDER BY u1.Reputation ASC
LIMIT 10)
UNION ALL
(SELECT u2.DisplayName, u2.Reputation, u2.UpVotes, u2.DownVotes
FROM users u2
ORDER BY u2.Reputation DESC
LIMIT 10);
```

	displayname character varying (255)	reputation integer	upvotes integer	downvotes integer
1	Community	1	5007	1920
2	hadley	111	59	15
3	Deer Hunter	224	96	12
4	Joel Reyes Noche	245	41	11
5	Mike John	292	12	16
6	subhash c. davar	298	64	12
7	Bill the Lizard	379	163	49
8	garciaj	393	150	19
9	Gschneider	406	15	13
10	RioRaider	551	418	18
11	whuber	87393	11273	779
12	Glen_b	65272	7035	143
13	Peter Flom	44152	2156	82
14	gung	37083	8641	125
15	chl	31170	10523	214
16	Greg Snow	25123	582	4
17	Jeromy Anglim	22625	2496	45
18	Michael Chernick	22275	2619	42
19	Frank Harrell	19585	155	56
20	Rob Hyndman	18283	1014	59

In this case, the lowest reputable users with low upvotes/downvotes were omitted as they all had a reputation of 1. Having a count of upvotes/downvotes greater than or equal to 10 immediately filtered out these users, as it represents users who interact more with StackOverFlow. Therefore, with this filter criteria, it may be noted that user reputation may range from about 100 to 87000. There also seems to be no correlation between the number of upvotes/downvotes and the reputation among the 10 least and most reputable users individually.

```
-- 6) Selecting top 10 users with the most badges
SELECT u.DisplayName, COUNT(b.Id) AS num_badges
FROM users u, badges b
WHERE u.id = b.userid
GROUP BY u.DisplayName
ORDER BY num_badges DESC
```

	displayname character varying (255)	num_badges bigint
1	whuber	456
2	Glen_b	318
3	chl	282
4	gung	256
5	Jeromy Anglim	222
6	John	182
7	Macro	176
8	Tal Galili	159
9	Rob Hyndman	156
10	Peter Flom	152

Interestingly enough, among the top 10 most reputable users (as seen in query #5), 7 of them are among the top 10 badge earners. This indicates a sound correlation between reputation and badge number.

```
-- 7) Selecting Title, ViewCount and Score of top 10 lowest and highest scored posts with a significant viewcount
(SELECT p.Title, p.ViewCount, p.Score
 FROM posts p
 WHERE p.ViewCount IS NOT NULL
 AND p.ViewCount > 10000
 ORDER BY p.Score ASC
 LIMIT 10)
UNION ALL
(SELECT p.Title, p.ViewCount, p.Score
 FROM posts p
 WHERE p.ViewCount IS NOT NULL
 ORDER BY p.Score DESC
 LIMIT 10);
```

	title character varying (255)	viewcount integer	score integer
1	What is the difference between normal distribution and standard normal distributi...	16957	-4
2	How do I interpret the results from the F-test in excel	14046	0
3	What is the "root mse" in stata?	12116	1
4	How do I interpret a probit model in Stata?	16604	1
5	How do you interpret results from unit root tests?	12859	1
6	Working with Likert scales in SPSS	17323	1
7	How to stop excel from changing a range when you drag a formula down?	25583	1
8	How to determine which variables are statistically significant in multiple regressi...	16587	1
9	How to make a forest plot with Excel?	23434	2
10	How to compute standard deviation of difference between two data sets?	18883	2
11	Python as a statistics workbench	60964	192
12	Making sense of principal component analysis, eigenvectors & eigenvalues	66071	184
13	What is your favorite "data analysis" cartoon?	64481	156
14	The Two Cultures: statistics vs. machine learning?	29229	152
15	Famous statistician quotes	34780	124
16	Why square the difference instead of taking the absolute value in standard deviati...	39118	122
17	What are common statistical sins?	10402	121
18	Is Facebook coming to an end?	25744	110
19	Is R^2 useful or dangerous?	8688	106
20	Bayesian and frequentist reasoning in plain English	21916	102

To be considered “significant” an arbitrary viewcount of 10000 views was selected to filter out posts with low scores and viewcounts. To answer the question of whether a post’s increased viewcount corresponded to a higher score, this query shows that there are posts with up to 25000 views (index 7 in this table) that receive scores of 1, while there are posts with 25000 views that receive scores of 110 (index 18 in this table).

```
-- 8) Selecting display name, number of comments, average post score and average view count
of the top 10 users with the most comments on their posts
SELECT u.DisplayName,
    COUNT(c.Id) AS num_comments,
    AVG(p.Score) AS avg_score,
    AVG(p.ViewCount) AS avg_views
FROM users u, posts p, comments c
WHERE u.id = p.OwnerUserId
AND p.Id = c.PostId
AND p.viewcount IS NOT NULL
GROUP BY u.DisplayName
```

```
ORDER BY num_comments DESC
LIMIT 10;
```

	displayname character varying (255)	num_comments bigint	avg_score numeric	avg_views numeric
1	Tim	551	2.372050816	508.1197822
2	JohnK	287	3.125435540	156.2543554
3	Stéphane Laurent	269	4.200743494	426.7806691
4	Chris	242	2.161157024	429.4834710
5	Tal Galili	238	6.962184873	1480.840336
6	David	217	5.562211981	763.9677419
7	user34790	214	1.775700934	263.1822429
8	luciano	204	2.416666666	366.9068627
9	shabbychef	200	8.575000000	1236.910000
10	Luca	196	1.612244897	1046.673469

Note that these top 10 users with the most number of comments on their posts do not have a high average post view compared to the viewcounts of the posts in query 7. These users are also not among the top 10 most reputable users or top 10 users with the most badges found in queries 5 and 5, respectively. The randomness of these users with low average viewcounts and scores on their posts indicates that posts made on StackOverFlow all have equal reach, opposed to posts on other platforms like Twitter where users with more followers have more comments.

9)

Selecting the top 5 most upvoted posts and least upvoted posts by a user based on the user badge status,

```
(SELECT b."name",  COUNT(u.upvotes) AS upv
FROM badges b, users u
WHERE b.id=u.id
GROUP BY b."name",u.upvotes
ORDER BY upv DESC
LIMIT 5)
Union ALL
(SELECT b."name",  COUNT(u.upvotes) AS upv
FROM badges b, users u
WHERE b.id=u.id
GROUP BY b."name",u.upvotes
ORDER BY upv ASC
```

LIMIT 5)

	name character varying (255)	upv bigint
1	Student	5391
2	Supporter	3090
3	Editor	2846
4	Scholar	2376
5	Teacher	1722
6	Notable Question	1
7	Talkative	1
8	Critic	1
9	Editor	1
10	Scholar	1

10)

Selecting the top 10 most viewed tagnames containing the word “data” in their tag name, followed by the top 10 most viewed tagnames.

```
(SELECT t.tagname, p.viewcount
FROM tags t, posts p
WHERE t.id=p.id and t.tagname LIKE '%data%'
AND p.viewcount IS NOT NULL
GROUP BY t.tagname,p.viewcount
ORDER BY p.viewcount DESC
LIMIT 10)
UNION ALL
(SELECT t.tagname, p.viewcount
FROM tags t, posts p
WHERE t.id=p.id
AND p.viewcount IS NOT NULL
GROUP BY t.tagname,p.viewcount
ORDER BY p.viewcount DESC
LIMIT 10)
```

	tagname character varying (255)	viewcount integer
1	dataset	21925
2	collecting-data	11533
3	binary-data	6700
4	data-acquisition	5593
5	compositional-data	3221
6	data-imputation	811
7	data-visualization	497
8	bootstrap	70255
9	kalman-filter	64481
10	kde	39118
11	optimal-scaling	36801
12	delta-method	34780
13	contingency-tables	29261
14	distributions	29229
15	value-of-information	28878
16	computational-statistics	25597
17	clinical-trials	23985

Note that in this case, there have only been 7 returned tagnames containing the word “data”, meaning that there are only 7 tagnames with that word with a viewcount that is not null. The top tagname turned out to be “bootstrap”, with a viewcount of 70255, while the tag name containing the word “data” with the most views is “dataset”, with a viewcount of 21925.

The indexing:

```
CREATE INDEX idx_badges
On badges (Id,Name);

CREATE INDEX idx_users
On users (Id,DisplayNameD,Creationdate,Reputation,UpVotes,DownVotes);

CREATE INDEX idx_posts
On posts (Id,Title,Viewcount,Score,Creationdate);

CREATE INDEX idx_comments
On comments (Id);

DROP INDEX idx_badges On badges
```

DROP INDEXidx_users On users

DROP INDEX idx_posts On posts

DROP INDEX idx_comments On comments

	QUERY PLAN text
1	Append (cost=274.80..610.25 rows=20 width=16) (actual time=0.363..2.178 rows=17 loops=1)
2	-> Limit (cost=274.80..274.88 rows=10 width=16) (actual time=0.363..0.369 rows=7 loops=1)
3	-> Group (cost=274.80..274.92 rows=15 width=16) (actual time=0.362..0.367 rows=7 loops=1)
4	Group Key: p.viewcount, t.tagname
5	-> Sort (cost=274.80..274.84 rows=15 width=16) (actual time=0.361..0.363 rows=7 loops=1)
6	Sort Key: p.viewcount DESC, t.tagname
7	Sort Method: quicksort Memory: 25kB
8	-> Nested Loop (cost=0.29..274.51 rows=15 width=16) (actual time=0.040..0.351 rows=7 loops=1)
9	-> Seq Scan on tags t (cost=0.00..20.90 rows=31 width=16) (actual time=0.021..0.201 rows=29 loops=1)
10	Filter: ((tagname)::text ~~ '%data%'::text)
11	Rows Removed by Filter: 1003
12	-> Index Scan using posts_pkey on posts p (cost=0.29..8.18 rows=1 width=8) (actual time=0.004..0.004 rows=0 loops=29)
13	Index Cond: (id = t.id)
14	Filter: (viewcount IS NOT NULL)
15	Rows Removed by Filter: 1
16	-> Limit (cost=335.04..335.07 rows=10 width=16) (actual time=1.803..1.806 rows=10 loops=1)
17	-> Sort (cost=335.04..336.26 rows=487 width=16) (actual time=1.801..1.803 rows=10 loops=1)
18	Sort Key: p_1.viewcount DESC
19	Sort Method: top-N heapsort Memory: 26kB
20	-> HashAggregate (cost=319.65..324.52 rows=487 width=16) (actual time=1.676..1.742 rows=242 loops=1)
21	Group Key: p_1.viewcount, t_1.tagname
22	Batches: 1 Memory Usage: 57kB
23	-> Merge Join (cost=0.67..317.22 rows=487 width=16) (actual time=0.025..1.525 rows=242 loops=1)
24	Merge Cond: (t_1.id = p_1.id)
25	-> Index Scan using tags_pkey on tags t_1 (cost=0.28..46.76 rows=1032 width=16) (actual time=0.013..0.274 rows=1032 loops=1)
26	-> Index Scan using posts_pkey on posts p_1 (cost=0.29..13476.00 rows=43434 width=8) (actual time=0.010..0.983 rows=425 loops=1)
27	Filter: (viewcount IS NOT NULL)
28	Rows Removed by Filter: 1327
29	Planning Time: 0.860 ms
30	Execution Time: 2.361 ms

	QUERY PLAN text
4	Group Key: p.viewcount, t.tagname
5	-> Sort (cost=108.49..108.53 rows=15 width=16) (actual time=1.058..1.060 rows=7 loops=1)
6	Sort Key: p.viewcount DESC, t.tagname
7	Sort Method: quicksort Memory: 25kB
8	-> Merge Join (cost=22.09..108.20 rows=15 width=16) (actual time=0.536..1.042 rows=7 loops=1)
9	Merge Cond: (t.id = p.id)
10	-> Sort (cost=21.67..21.75 rows=31 width=16) (actual time=0.488..0.492 rows=29 loops=1)
11	Sort Key: t.id
12	Sort Method: quicksort Memory: 26kB
13	-> Seq Scan on tags t (cost=0.00..20.90 rows=31 width=16) (actual time=0.059..0.468 rows=29 loops=1)
14	Filter: ((tagname)::text ~~ '%data%'::text)
15	Rows Removed by Filter: 1003
16	-> Index Only Scan using idx_posts on posts p (cost=0.42..4328.58 rows=43434 width=8) (actual time=0.040..0.473 rows=409 loops=1)
17	Index Cond: (viewcount IS NOT NULL)
18	Heap Fetches: 0
19	-> Limit (cost=158.26..158.28 rows=10 width=16) (actual time=1.437..1.441 rows=10 loops=1)
20	-> Sort (cost=158.26..159.48 rows=487 width=16) (actual time=1.436..1.439 rows=10 loops=1)
21	Sort Key: p_1.viewcount DESC
22	Sort Method: top-N heapsort Memory: 26kB
23	-> HashAggregate (cost=142.87..147.74 rows=487 width=16) (actual time=1.276..1.357 rows=242 loops=1)
24	Group Key: p_1.viewcount, t_1.tagname
25	Batches: 1 Memory Usage: 57kB
26	-> Merge Join (cost=0.79..140.43 rows=487 width=16) (actual time=0.057..1.063 rows=242 loops=1)
27	Merge Cond: (t_1.id = p_1.id)
28	-> Index Scan using tags_pkey on tags t_1 (cost=0.28..46.76 rows=1032 width=16) (actual time=0.024..0.368 rows=1032 loops=1)
29	-> Index Only Scan using idx_posts on posts p_1 (cost=0.42..4328.58 rows=43434 width=8) (actual time=0.027..0.302 rows=425 loops=1)
30	Index Cond: (viewcount IS NOT NULL)
31	Heap Fetches: 0
32	Planning Time: 0.626 ms
33	Execution Time: 5.171 ms

4 NoSQL Databases:

```
//users
CREATE INDEX idx_id ON users(Id);

//posts
CREATE INDEX idx_id ON posts(Id);
CREATE INDEX idx_lastEditorUserId ON posts(LastEditorUserId);
CREATE INDEX idx_ownerUserId ON posts(OwnerUserId);
CREATE INDEX idx_parentId ON posts(ParentId);

//postLinks
CREATE INDEX idx_id ON postLinks(Id);
CREATE INDEX idx_postId ON postLinks(PostId);
```

```
CREATE INDEX idx_relatedPostId ON postLinks(RelatedPostId);

//tags
CREATE INDEX idx_id ON tags(Id)
CREATE INDEX idx_excerptPostId ON tags(ExcerptPostId)

//badges
CREATE INDEX idx_id ON badges(Id)
CREATE INDEX idx_userId ON badges(UserId)

//comments
CREATE INDEX idx_id ON comments(Id)
CREATE INDEX idx_postId ON comments(PostId)
CREATE INDEX idx_userId ON comments(UserId)

//postHistory
CREATE INDEX idx_id ON postHistory(Id)
CREATE INDEX idx_postId ON postHistory(PostId)
CREATE INDEX idx_userId ON postHistory(UserId)

//votes
CREATE INDEX idx_id ON votes(Id)
CREATE INDEX idx_postId ON votes(PostId)
CREATE INDEX idx_userId ON votes(UserId)
```