Group Number: INFO3404 Assignment Group R12A1

Original Query:

```
SELECT P.id, P.score, P.answerCount, P.viewCount, P.title, P.body, P.creationDate,
P.lastActivityDate, P.ownerUserId, U.displayName, U.profileImageUrl, U.reputation, COUNT(B.id) AS
numBadges
FROM Posts P
JOIN Users U ON (ownerUserID=U.Id)
LEFT OUTER JOIN Badges B ON (ownerUserID=userID)
GROUP BY P.id, U.id
ORDER BY P.lastActivityDate DESC
LIMIT 30 OFFSET 30;
```

```
Limit  (cost=7779.13..7779.21 rows=30 width=988)
```

Modified Query:

```
SELECT P.id, P.score, P.answerCount, P.viewCount, P.title, P.body, P.creationDate, P.lastActivityDate,
P.ownerUserId, U.displayName, U.profileImageUrl, U.reputation, COUNT(B.id) AS numBadges
FROM Users U
JOIN Posts P ON (U.Id=P.ownerUserID)
Left Outer JOIN Badges B ON (U.Id=userID)
GROUP BY P.id,U.id
ORDER BY P.lastActivityDate DESC
LIMIT 30 OFFSET 30
```

```
Limit  (cost=784.59..784.67 rows=30 width=988)
```

Q2:

```
SELECT T.id, T.count, T.tagName, P.body, COUNT(Q.id) AS posted_today
FROM Tags T
JOIN Posts P ON (T.excerptPostId=P.Id)
LEFT OUTER JOIN Posts Q ON (Q.creationDate=current_date AND Q.tags LIKE '%<' ||
T.tagName || '>%')
WHERE P.postTypeId=4
GROUP BY T.id, P.id
ORDER BY P.creationDate DESC
LIMIT 30 OFFSET 0
```

Limit (cost=538.19..538.20 rows=1 width=836)

Q3.

```
explain SELECT P.id, P.score, P.answerCount, P.viewCount, P.title, P.body, P.creationDate,
P.ownerUserId, U.displayName, U.profileImageUrl, U.reputation, COUNT(B.id) AS numBadges
        FROM Posts P
        JOIN Users U ON (ownerUserID=U.Id)
        LEFT OUTER JOIN Badges B ON (ownerUserID=userID)
        WHERE P.tags LIKE ('%<' || 'machine-learning' || '>%')
        GROUP BY P.id, U.id
        ORDER BY P.score DESC
        LIMIT 30 OFFSET 0;
```

Optimisation:

```
CREATE INDEX covering_index on votes (voteTypeID, postId, userID);
CREATE INDEX postidtype on posts (parentId,postTypeId);

CREATE INDEX posthash on posts USING hash (id);
CREATE INDEX userhash on users USING hash (id);
CREATE INDEX  comhash on comments USING hash (postId);
CREATE INDEX  linkhash on PostLinks USING hash (relatedPostId);
```

Analyse of the workload

Which queries/updates have you optimised, and why?

The first optimised query is the one to see the question page; Before the optimisation, the estimated total cost is 7779.21, with the limit of 30 and offset of 30. Obviously, the query is not cost-effective and needs to be improved

Before optimization, the query about voting (explain SELECT * FROM Votes V WHERE V.voteTypeId IN (2,3,5) AND V.userId=null AND V.postId=null) has a total cost of 125.71 (cost=0.00..125.71 rows=1 width=24). The sequence scannig used in this query may be improve by covering composite index.

Choice of optimisations

What changes have you made, such as additional indexes, and why? You should be specific about the type of index you create (B+-Tree, Hash, clustered, etc.)

Corresponding to the first query being optimised, we exchange the places between relation Post and relation users. Originally, the Post table as outer relation is joint to Users table with hash join. The cumbersome size of Post table is undoubtedly the bane of heavier cost. Compared to Post table, Users table is lighter and can be used as outer relation. It turns out that the cost of this query drops significantly after change(From 7779.21 to 784.67). And the outputs hold the same (rows=30 width=988).

In terms of query about voting, the sequence scanning used in this query may be improve by covering composite index. Therefore, we constructed of covering index on voteTypeID, postId and userID.

Analysis of tuning decisions

What evidence do you have that you tuning has improved performance?

After the modification in first query (question page), it turns out that the cost of this query drops significantly after change(From 7779.21 to 784.67). And the outputs hold the same (rows=30 width=988).

The cost on query about voting has dropped slightly, after the construction of covering index, from 125.71 to 116.74 (cost=0.00..116.74 rows=1 width=24). It is also notable that the output is identical to the one before adding index.

Broader impact of tuning

What are the potential down-sides to your optimisations? For instance, how will updates be affected?