OPUS

Query Optimization Documentation

Things I can utilize:

* Selection on specific fields
* Selection on specific rows
* Avoiding select distinct if not needed
* Creating joins with inner joins instead of where
* Using where instead of HAVING
* Using limit to sample query results

Query 1

Company/News

The query is being used to fetch the recent news of the specified company and its peers from the database. We highlight how the optimizations result in a speed up.

Unoptimized:

WITH Temp AS (  
 WITH T1 AS (SELECT DISTINCT peerID, "PEER" as cmpRel  
 FROM Peers  
 WHERE symbol LIKE '%A%'  
 UNION ALL  
 SELECT DISTINCT peerID, "SELF" as cmpRel  
 FROM Peers  
 WHERE peerID LIKE '%A%'  
 )  
 SELECT \*  
 FROM CompanyNews CN, T1  
 WHERE CN.symbol = T1.peerID)  
 SELECT \*  
 FROM Temp  
 ORDER BY publishedDate DESC

LIMIT 0, 1000;

* Execution Time: 9s 106ms

Optimized:

WITH Temp AS (  
 WITH T1 AS (SELECT DISTINCT peerID, "PEER" as cmpRel  
 FROM Peers  
 WHERE symbol LIKE 'A'  
 UNION ALL  
 SELECT DISTINCT peerID, "SELF" as cmpRel  
 FROM Peers  
 WHERE peerID LIKE 'A'  
 )  
 SELECT symbol,  
 publishedDate,  
 title,  
 image,  
 site,  
 text,  
 cmpRel,  
 url  
 FROM CompanyNews CN JOIN T1 ON CN.symbol = T1.peerID)  
 SELECT symbol,  
 publishedDate,  
 title,  
 image,  
 site,  
 text,  
 cmpRel,  
 url  
 FROM Temp  
 WHERE title IS NOT NULL  
 ORDER BY publishedDate DESC  
 LIMIT ${offset}, ${pagesize};

* Execution Time: 859 ms

Optimizations Made:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SNo | Type of Optimization | Unoptimized Query | Optimized Query | Implication |
| 1 | **SELECT** | * All fields (SELECT \*) are being retrieved in both the CTE expression and the final SELECT call. * The fields are being filtered on frontend via JavaScript NodeJS. | * Only fields needed by subsequent queries are retrieved in CTEs and fields to be displayed on the Front End are projected in the main SELECT call. | Project on Specific Fields |
| 2 | **WHERE** | * No selection condition to filter the number of rows in the final result. | * Selection condition in WHERE clause to filter out rows with erroneous News Title. | Selection on specific rows |
| 3 | **INNER JOIN vs**  **CARTESIAN CROSS PRODUCT** | * Cross product with Where clause to filter out relevant rows after the execution of Join. * Results in more comparisons to join tables. | * Inner Join with the Join condition specified on Primary Key – Foreign Key. * Results in filtering of tables on join condition before the execution of join and thereby significantly lesser number of comparisons to be made. | Creating inner joins instead of cross products with selection condition. |
| 4 | **LIMIT** | * All the results in the final table are retrieved by the server and pagination done on FrontEnd. | * LIMIT implemented to enforce pagination at database level. * As a result only the data required to be displayed at the front end is queried from the database. * Consequently a smaller size of the data is sent to the server. | Using limit to sample query results with dynamic queries. |
| 5 | **INDEX** |  |  | As joins are made on Primary Key – Foreign Keys the attributes are already indexed and no new indexes are created. |
| 6 | **Wildcard Optimization ‘%’** | * Use of wildcard while matching symbol resulted in many more results than required. | * Wildcards removed to promote exact match and thereby reduce the number of results returned by the database. | Use of ‘%’ wildcard character with *LIKE* matches all strings when sometimes only specific matches are needed. |
|  | **Execution Time** | 9s 106ms | 859 ms |  |
|  | Speed Up– 10.6x | | | |

Note: The queries for the pages Company/Info, Company/Sentiment, Company/Jobs are optimized in a similar way.

Query Optimization 2

Home Page

The query is used to retrieve all companies matching the filters specified by the user on home page such as any matching characters in the company name, min/max sentiment score, market cap min/max etc.

Unoptimized Query:

WITH tmp1 AS

        (SELECT symbol, companyName, fullTimeEmployees, mktCap

        FROM CompanyInformation

        WHERE companyName LIKE '%${cmpName}%' and

        fullTimeEmployees BETWEEN ${numEmployeesLow} AND ${numEmployeesHigh}

        AND mktCap BETWEEN ${mktcapLow} AND ${mktcapHigh}),

        tmp2 AS (SELECT s.symbol, tmp1.companyName, tmp1.fullTimeEmployees, tmp1.mktCap, s.sentiment

        FROM CompanySentiments s

        LEFT JOIN tmp1

        ON tmp1.symbol= s.symbol

        WHERE sentiment BETWEEN ${sentiLow} AND ${sentiHigh})

        SELECT \*, COUNT(jobLink) as JobCount

        FROM IndeedJobs i

        JOIN tmp2

        ON tmp2.symbol=i.companySymbol

        GROUP BY i.companySymbol

        HAVING COUNT(jobLink)> ${jobNum}

        LIMIT 1000 OFFSET 0;

Optimized Query: