Original Article
International Journal of Fuzzy Logic and Intelligent Systems
Vol. 16, No. 1, March 2016, pp. 36-43
http://dx.doi.org/10.5391/IJFIS.2016.16.1.36

ISSN(Print) 1598-2645
ISSN(Online) 2093-744X

Development of Query Transformation Method by Cost Optimization

Aigerim Bakatkaliyevna Altayeva, Youngmi Yoon, and Young Im Cho
Department of Computer Engineering, Gachon University, Seongnam, Korea

Abstract

The transformation time among queries in the database management system (DBMS) is responsible for the execution time of users' queries, because a conventional DBMS does not consider the transformation cost when queries are transformed for execution. To reduce the transformation time (cost reduction) during execution, we propose an optimal query transformation method by exploring queries from a cost-based point of view. This cost-based point of view means considering the cost whenever queries are transformed for execution. Toward that end, we explore and compare set off heuristic, linear, and exhaustive cost-based transformations. Further, we describe practical methods of cost-based transformation integration and some query transformation problems. Our results show that, some cost-based transformations significantly improve query execution time. For instance, linear and heuristic transformed queries work 43% and 74% better than exhaustive queries.

Keywords: Cost-based transformation, Join, Optimization, Oracle DBMS, Subquery, Window function

1. Introduction

Database performance is one of the main challenging aspects of database operations organization. Conventional database query optimization normally consists of two stages of processing: logical and physical optimization. At the stage of a given query, optimization logic is rewritten (usually on the basis of heuristics or rules) to an equivalent but potentially more declarative or optimal form. The traditional physical optimizer works within one block of the request, which operates a plurality of base tables with limits of projection and connection.

In general, the transformation time among queries in the DBMS is mainly responsible for the execution time of users' queries, because a conventional DBMS does not consider the transformation cost when queries are transformed for execution. To reduce the transformation time (cost reduction) during execution, we propose an optimal query transformation method by exploring queries from a cost-based point of view. This cost-based point of view means considering the cost whenever queries are transformed for execution.

The motivation of our paper is exploring database query transformation methods and to determine which method is best from the cost-based point of view. After determining the best method, we replace the queries with the low-cost equivalent and reduce the database query cost. To achieve the goal, we explore exhaustive, linear, and heuristic query transformation methods. In the experimental part, we execute all queries separately and set of queries by transformation type. After determining the execution types, we can determine which type of

Received: Dec. 8, 2015
Revised : Mar. 20, 2016
Accepted: Mar. 22, 2016

Correspondence to: Young Im Cho
(yicho@gachon.ac.kr)

method is better.

The rest of this paper consists of four sections. We will explain the related work in Section 2, our proposed method in Section 3, and the simulation results in Section 4. Finally, we will conclude our paper in Section 5.

2. Related Work

The system conversion requests based on rules can be exhaustively listed by the thick wood conversion [1] and to assess the final plans; this leads to an exponential increase in the number of plans, and may be unsuitable for commercial optimizers. Garcia-Molina et al. [1] showed that for some queries, an external connection can switch to the internal connections by introducing the concept of a generalized external connection. A study was conducted to optimize the unification of the two-phase reform. Chaudhuri and Shim [2] explored generation of a plan optimizer to offer a conversion and attempt to apply it. In addition, Chaudhuri and Shim [2] defined a similar transformation based on pulling and pushing units, but also demonstrated how this conversion can be performed on the basis of valuation. In addition, they proposed the concept of rewriting that is sensitive to the cost. In Galindo-Legaria and Rosenthal [3] and Ganske [4], work query transformation cannot eliminate nesting in queries with subqueries. In our experience, with the exception of nested sub-queries, which lead to the generation of ideas, it does not always provide a more effective plan. Garcia-Molina et al. [1] and Grief et al. explored similar methods to eliminate nesting sub-queries SPJ-actively studied for solving sub-queries with aggregates. Chaudhuri [5] and Gupta [7] demonstrated conversion assemblies that are pushed into the position preceding the connection. Chaudhuri [1] examined whether a greedy and conservative heuristics leads to cheaper scanning or connection. Gupta [7] demonstrated a canonical abstraction external connection, which allows the optimizer to use different formations among tables joined externally and internally.

Heller stein [8] and Levy [9] proposed a method that is a synthesis of these approaches by drawing predicates up the tree query, and then pushing them down, so that when scanning the base table, a greater number of single-table predicates can be used. Kim [10] and Morzy [11] described the transformation of logical sets (magic set), which leads to new insights into the costly units' request, thus allowing the volume of data being processed to be reduced.

One problem with most of these translation schemes is that they require knowledge of the semantics of the source schema, and thus they are intended to be performed manually by a database designer. Another problem is that there were no comparisons between sets of queries, and just single query transformation results were performed. In this study, we compare each query transformation result separately, and set of queries for each heuristic, linear, and exhaustive search. Further, we give the common query execution time, by comparing the results obtained for each type of query transformation.

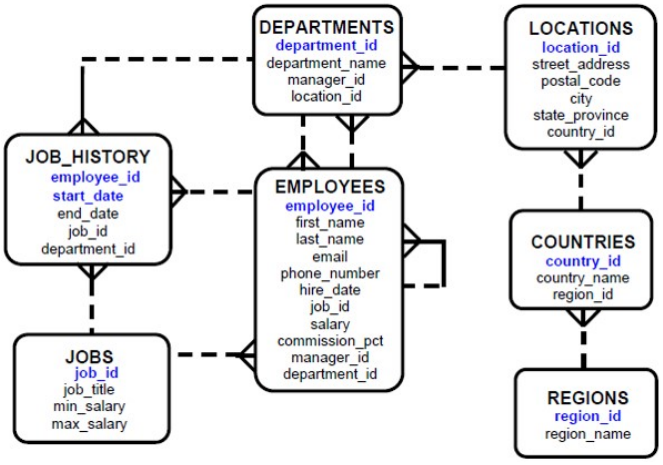

Figure 1. Human resources (HR) schema.

3. Proposed Optimized Query Transformation Method

3.1 Multiple Query Transformations

For multiple query transformation, we use Human Resources (HR) and Order Entry (OE) schemes from Oracle Academy that consist of the following seven tables, as shown in Figure 1 (employees, departments, locations, countries, regions, jobs, and job history), and order items, orders, customers, product_information, product_description, inventories, and warehouses.

We know that the requests may be expressed in a number of different forms. Furthermore, even within a given language, each request can be a series of semantically equivalent expressions [1, 4]. The subject of this section is the transformation of a query expression into an equivalent expression on the basis of properly defined rules. Here is an example in Q1 that shows how to compare the wages of each employee to the average wage in the division where he or she works:

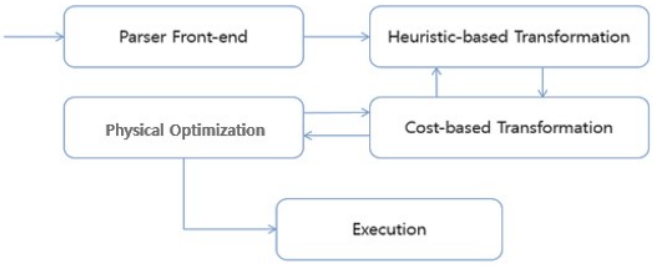

Figure 2. Query processing in Oracle database management system.

```
Q1
SELECT depname, empno, salary, avg (salary)
      OVER (PARTITION BY depname)
FROM empsalary;
```

Moreover, we use window functions to improve the query execution time. In Q2 we use the window function, which provides the aggregate sum of all the rows Figure 3 and Table 2, and its resulting data would look like this:

```
Q2
SELECT first_name, department_id, job_id,
SUM(salary) OVER (PARTITION BY
department_id, job_id) sum_sal
FROM employees;
```

We consider some selection approaches as heuristic, exhaustive, and linear selection. In exhaustive search, the prices of all possible physical query plans derivable from the logical query plan are considered, and the least expensive plan is chosen [6]. In heuristic selection, a sequence of choices for annotations is made on the basis of heuristics [8]. Linear search is based on the search of the dynamic programming approach, which assumes that for a query consisting of several objects, it is sufficient to analyze the conversion for only a subset of these objects, and then expand it with additional transformations of other objects [11].

Oracle performs multiple query transformations, some of which are cost based and others are heuristic based. Cost-based transformation is combined with logic and physical optimization to generate an optimal execution plan. This is illustrated in Figure 2.

It can be assumed that the logical transformation has two separate components: heuristic-based conversion and cost-based conversion. Different elements of wood used different conversion requests.

3.2 State Space Search Methods

The fundamental question related to transformations based on the evaluation value is whether these changes will lead to the rapid growth of combinatorial alternatives that must be assessed, and if they provide a balance between cost optimization and cost performance.

Exhaustive Search. If we consider an exhaustive search of all possible states, n 2N status of space objects. For example, for the query Q11, we consider the four states (0,0), (0,1), (1,0), and (1,1). This search is guaranteed to find the best solution.

Iterative Improvement. The Iterative Improvement method is used to reduce the search space. The main idea of this method is that we start with some initial condition and move to the next adjacent of using some of the techniques aimed at finding a local minimum by continuously selecting the downward direction of movement.

Linear Search. This method is based on the search of the dynamic programming approach, which assumes that in a query consisting of several objects, it is sufficient to analyze the conversion for only a subset of these objects, and then expand it with additional transformations of other objects.

3.3 Query Transformation Methods

Traditional transformation was performed based on heuristics for early selection operations and projection removing redundant operations, minimizing the number of request blocks. Minimizing the number of request blocks by merging them with other blocks a request to remove the restrictions set permutation operations compounds that can be generated, allowing thus reordered more tables.

Subquery Un-nesting. An important transformation, which is usually implemented by a commercial DBMS. Subquery nesting is calculated with unresolved several times using a tuple iteration semantics (TIS), which is similar to the connection execution nested loops, and therefore in this case cannot be taken into account many effective ways to access and serial connection. Consider the following query Q3 that returns information about department staff who receive high salaries:

```
Q3
SELECT d.deptartment_name
FROM departaments d     WHERE EXISTS (SELECT 1
FROM employees e     WHERE d.department_id =
e.department_id     AND e.salary > 10000);
```

This query is transformed into the following equivalent query Q4:

```
Q4
SELECT d.deptartment_name
FROM departaments d, employees e
WHERE d.department_id S= e.department_id
AND e.salary > 10000;
```

Transformation merges with the outer query sub query generally allows additional methods and compounds of the sequence of operations.

Join Elimination removes the table from the query, if there are restrictions on the join column of the table, such that the elimination of the compound does not affect the results of the query. Consider the following two queries Q5 and Q6, which selects some information about employees:

```
Q5
SELECT e.name, e.salary
FROM employees e, departaments d
WHERE e.department_id = d.department_id;
```

Because department_id in the employees table is a foreign key, referencing the primary key of the table departments, the connection to the departments of Q5 can be eliminated. The request Q6 Connection column in the right table of external connections is unique. Because the outer join contains all tuples left the table, and equijoin with a unique column does not create a duplicate table departments be deleted.

```
Q6
SELECT e.name, e.salary
FROM employees e left outer join
departments d on e.department_id = d.department_id;
```

Use of the compounds to eliminate the transformation of Q5 and Q6 leads to the generation of the query Q7. If Q5 e.department_id may contain null values, the section of WHERE Q7 must be added to the predicate e.department_id is not null.

```
Q7
SELECT e.name, e.salary FROM employees e;
```

Obviously, the elimination of redundant connections to improve query performance, and therefore the elimination of connections, is always executed in Oracle, if appropriate.

In Filter Predicate Move Around predicates, cheap shifted to block the submission of requests for the performance of the earlier filter. Consider a query Q8 with two disjunctive subqueries with EXISTS; sub-queries contain the same predicate correlated, but different conjunctive predicate filtering:

```
Q8
SELECT employee_id, last_name, job_id, department_id
from employees outer where
EXISTS (SELECT * From employees
WHERE manager_id = outer.employee_id)
OR exists (SELECT * from employees
WHERE (commission_pct = outer.commission_pct));
```

Splicing together two EXISTS sub-queries with a single EXIST sub query with a disjunction predicate filtering results in the query Q9:

```
Q9
SELECT employee_id, last_name, job_id, department_id
FROM employees outer
WHERE EXISTS (select * FROM employees WHERE
manager_id = outer.employee_id
OR (commission_pct = outer.commission_pct;
```

Group Pruning, another imperative transformation that removes the representations of the group, is not required in the outer query block. Expression SQL CUBE is an extension of the operator-SQL, which allows multiple clusters to be retrieved in a single SQL statement for queries that contain representations with expressions cube, sometimes the amount of data needed to assess the request can be reduced. For example, consider the following query Q10:

```
Q10
SELECT hire_date, last_name, department_id
FROM ( SELECT hire_date, last_name, department_id,
sum(salary) as revenue
FROM employees group by cube (hire_date, last_name,
department_id))
WHERE hire_date like '%2015';
```

Query Q10 can be transformed to the Q11 query:

```
Q11
SELECT hire_date, last_name, department_id
FROM ( SELECT hire_date, last_name, department_id,
sum(salary) as revenue
FROM employees
WHERE hire_date like '%2015' group by
hire_date, cube (last_name, department_id)) WHERE
hire_date like '%2015';
```

3.4 Cost-Based Transformation Methods

Here we will briefly discuss some of the changes that are performed in Oracle-based valuation.

In the case of multiple sub-queries with EXISTS or ANY, as a rule, it is not possible simply to merge sub-queries with a query containing the same unit, owing to the possible undesirable effect of the occurrence of duplicate rows that were not in the results of the original query. In these cases, we should create embedded views that contain table sub-queries to eliminate nesting correlated sub-queries containing aggregates also

requires the creation of embedded representations with GROUP BY. Once again consider the request Q12:

```
Q12
SELECT e1.last_name, j.job_title
FROM employees e1, job_history j
WHERE e1.employee_id = j.employee_id and    j.start_date >
'01-01-1998'  and
e1.salary > (SELECT AVG (e2.salary)
FROM employees e2   WHERE e2.department_id =
e1.department_id) and  e1_department_id IN  (SELECT
department_id    FROM departaments d, locations l    WHERE
d.location_id = l.location_id
and  l.country_id = 'US');
```

Consider the transformed query Q13, which first nested the sub-query resolved by the formation of the built-in views:

```
Q13
 SELECT e1.last_name, j.job_title
FROM employees e1, job_history j,
(SELECT AVG (e2.salary) avg_sal, department_id    FROM
employees e2
GROUP BY department_id) V
WHERE e1.employee_id  = j.employee_id  and
j.start_date > '01-01-1998'    and
e1.department_id = V.department_id  and
e1.salary > V.avg_sal and  e1_department_id  IN
(SELECT department_id    FROM departments d, locations l
WHERE d.location_id = l.location_id
and   l.country_id = 'US');
```

The untransformed query Q12 may be better performed using a strategy TIS, if the outer query block significantly reduces the number of tuples of the table employee, for which the excess of the average wage is to be calculated. In addition, TIS can be very effective if the local column predicate has a correlation index.

Group-by View Merging allows one to merge the presentation, comprising the steps of GROUP BY (or DISTINCT), with its external power request. This allows the optimizer to consider additional join orders and access paths and to postpone the calculation of aggregates until the connection is completed. Consider query Q14, which is obtained by the merger of views with a GROUP BY clause in the query Q13. The transformed query Q13 requires that aggregation be performed on the entire table of employees. The conversion request Q14 connects two employee's tables and the table job history, and applies a second filtering sub query before aggregation is performed:

These considerations are the basis for why the decision should be based on an assessment of the cost. For queries that aggregate the latter, we also consider the possibility of converting the query to perform early aggregation.

The Join Predicate Pushdown transformation encounters join predicates within the view. Pushes join predicates, and being within the view, they act like the correlation, thus making a new path available.

```
Q14
SELECT e1.last_name, j.job_title
FROM employees e1, job_history j,
employees e2  WHERE   e1.employee_id  = j.employee_id
and  j.start_date > 01-01-1998  and
e2.department_id = e1.department_id and
e1_department_id IN
(SELECT department_id
FROM departaments d, locations l
 WHERE d.location_id = l.location_id and
 l.country_id = 'US')    GROUP BY e2.department_id,
e1.employee_id, j.rowid,    e1.last_name,
j.job_title, e1.salary    HAVING e1.salary > AVG (e2.salary);
```

A similar optimization is possible for submissions to the operation DISTINCT, as shown in the query Q15. Consider the following query that returns information about the employees and work history (job history) of staff (employees) who are working in offices (departments) located in the United Kingdom (UK) and the United States (US):

```
Q15
SELECT e1.last_name, j.job_title,
e2.last_name as mgr_name
FROM employees e1, job_history j,
employees e2,
(SELECT DISTINCT department_id
FROM departments d, locations l
WHERE d.location_id = l.location_id and
l.country_id IN ('U.K.','U.S.')) V
 WHERE e1.employee_id  = j.employee_id  and
j.start_date > '01-01-1998'   and
 e1.manager_id = e2.employee_id  and
e1.department_id = V.department_id;
```

Query Q15 is converted on the basis of the push join predicate to Query Q16. This transformation allows us to delete from the expensive operation DISTINCT. The internal (inner) compound is converted into a semi-join that imposes a partial order connection, where e1 must precede V:

```
Q16
SELECT e1.last_name, j.job_title,
e2.last_name as mgr_name
FROM employees e1, job_history j,
employees e2,  (SELECT department_id
FROM departments d, locations l
WHERE d.location_id = l.location_id and
l.country_id IN ('U.K.','U.S.') and
e1.department_id = d.department_id) V
WHERE e1.employee_id  = j.employee_id  and
j.start_date > '01-01-1998'  and
 e1.manager_id = e2.employee_id;
```

In Q16 for connecting the V method can be used nested loops; it can be quite effective if d.department_id is an index, and the number of tuples in the outer query is relatively small. Nevertheless, determining which of the requests Q15 and Q16

will provide a more optimal execution plan requires a decision based on the evaluation value. In Oracle, pushing join predicates can be applied to stick together, and to non-drainable representations.

Join Factorization is used for queries with UNION/UNION ALL, UNION ALL branches which contain general joined the table. This factorization prevents multiple accesses to shared tables. Using factorization connections, query Q17 can be converted to a query Q18:

```
Q17
SELECT e.first_name, e.last_name, job_id,
d.departament_name, l.city FROM employees e,
departaments d, locations l WHERE e.department_id =
d.department_id and d.location_id = l.location_id
UNION ALL SELECT e.first_name, e.last_name, j.job_id,
d.departament_name, l.city FROM employees e,
job_history j, departaments d, locations l
WHERE e.employee_id = j.employee_id and
j.department_id = d.department_id and
d.location_id = l.location_id;
```

```
Q18
SELECT V.first_name, V.last_name, V.job_id,
d.departament_name, l.city
FROM departaments d, locations l,
(SELECT e.first_name, e.last_name
e.job_id, e.department_id FROM employees e
UNION ALL SELECT e.first_name, e.last_name,
j.job_id, j.department_id
FROM employees e, job_history j
WHERE e.employee_id = j.employee_id) V
WHERE d.department_id = V.department_id and
d.location_id = l.location_ id;
```

Predicate Pull-up filtration pulls expensive filtering predicates in the query from the view containing the submission. Currently, a predicate is considered expensive if it contains SQL procedural functions, user-defined operations, or sub-queries. Convert "Predicate Pull-up" is being taken into consideration only if the query containing view set predicate grownup and representation has a blocking operation. Consider the following query, Q19, which contains the presentation of two expensive predicates:

```
Q19
SELECT * FROM (SELECT document_id
FROM product_docs
WHERE containts(summary,'optimizer',1) > 0 AND
containts(full_text,'execution',2) > 0
ORDER BY create_date) V WHERE rownum <= 20;
```

Because the introduction of an expensive predicate has two, there are three ways in which the conversion can be accomplished predicate drawing, one of which is shown below:

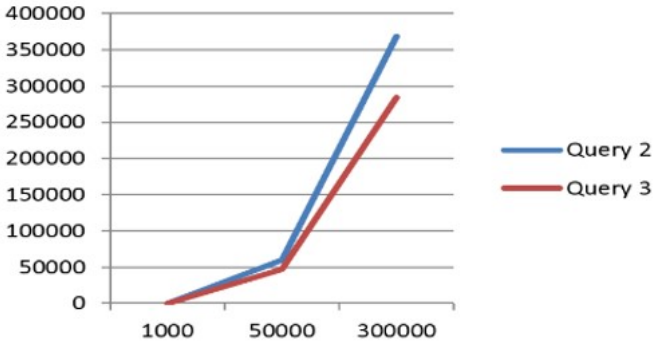

Figure 3. Comparison of query execution time between Q2 and Q3 queries.

Table 1. Query execution times of group pruning queries

Row count	100	1,000	150,000	200,000	250,000	300,000
Query execution time of Q9 (s)	132	152	201	239	281	392
Query execution time of Q10 (s)	91	114	178	181	196	347

```
Q20
SELECT * FROM (SELECT document_id, value(r) as vr
FROM product_docs
WHERE containts(full_text,'execution',2) > 0
ORDER BY create_date) V
WHERE containts(summary,'optimizer',1) > 0
AND rownum <= 20;
```

Running late check expensive predicate to a significant reduction in the data set can, in some cases, improve query performance. If filtered predicates have very few lines, we can avoid implementing expensive predicates on the full data set. This transformation acts in the opposite way with respect to the common method of optimizing push into view predicate filtering.

4. Experiment Results

We conducted experiments to study the efficiency of conversion requests, cost-based values. To run queries, we use Oracle 11g PL SQL Developer 3.2 Platform that includes HR, OE, and Financial schemes (Figures 1 and 2). We compared the query execution times of each query separately and each type of set of queries, such as heuristic-based and cost-based queries. In addition, query performance and query execution times are compared.

In Figure 3, we compare the query execution times of two queries for 1,000, 50,000, and 300,000 rows of data.

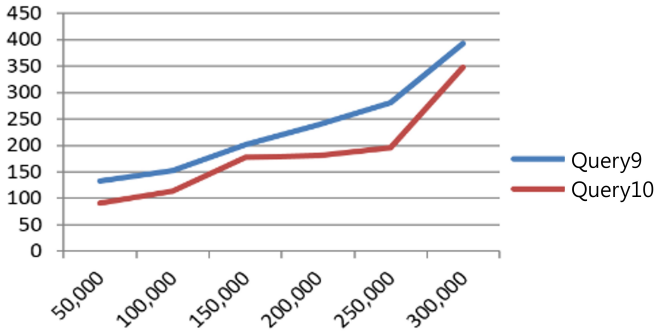

Figure 4. Comparison execution times between Q9 and Q10 queries for tables with from 50,000 to 300,000 rows.

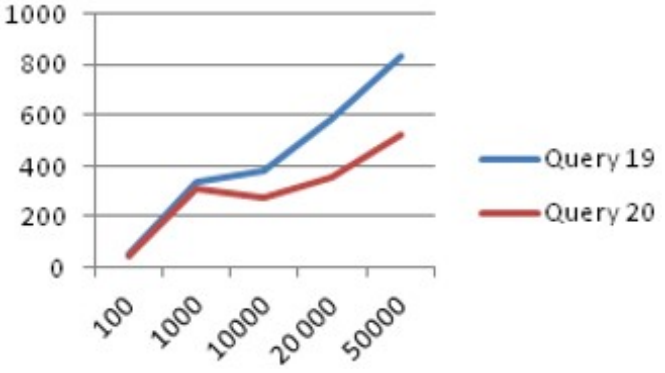

Figure 5. Comparison of execution times between Q19 and Q20 queries for tables from 100 to 50,000 rows.

In Figure 4, the execution times of two queries using sub query (Q9) and join (Q10) for 100, 1000, 10,000, 20,000, and 50,000 rows of records can be seen. It is clear that using join results in much better performance than sub-queries. In the case of group pruning for CUBE queries, we run queries for the same table with rows from 50,000 to 300,000 rows, and Q10 has 15% less execution time that Q9 (Figure 4).

This includes SQL becomes much less aggregation, because the amount of data that must be broken is vastly reduced and the number of units is further reduced. This is an important transformation for SQL generators in analytical applications, because these tools may need to request logical "cubes" that were previously defined with a view containing the operator CUBE. Figure 4 shows comparison of the execution times for each experiment from 50,000 to 300,000 rows. In Figure 5, it is observed that the execution time of Q19 is less than the execution time of Q20. Figure 5 shows the result of queries with a sub query and the MINUS set operation.

One of the main problems of transformation based on an assessment of cost optimization is to increase the time. We have noted that in the experiments described in the previous sections, the optimization time increased.

In Table 3, we present the optimization time and the number of states for a variety of search strategies in the state space for a single query, with a transformation to eliminate nesting. The increased optimization time for different search strategies compared to the heuristic mode (that is, without change, based on an assessment of the value) is not significant owing to the reuse of the annotations cost sub trees request.

Table 2. Comparison with query optimization time

	Optimization time (ms)
Heuristic	452
Linear	987
Exhaustive	1756

5. Conclusion

The paper explored query transformation methods and their performance based on two types of experiments: The first one compared the results of single queries, and the second one compared query set results of each optimization type. During the research, heuristic, linear, and exhaustive cost-based transformations were explored, and their execution times were compared with experiments to determine the best method depending on the query types.

Conflict of Interest

No potential conflict of interest relevant to this article was reported.

Acknowledgements

This work was funded by the National Reseach Foundation (NRF) in Korea Government (Ministry of Science, ICT & Future Planning) (No. 2015R1A2A2A03004088) and by Small & Medium Business Administration's University–Industry Cooperation project (No. C0298842).

References

[1] H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database System Implementation*. Upper Saddle River, NJ: Prentice Hall, 2000.

[2] S. Chaudhuri and K. Shim, "Optimizing queries with aggregate views," in *Proceedings of 5th International Conference on Extending Database Technology*, Avignon, France, 1996, pp. 167-182. http://dx.doi.org/10.1007/BFb0014151

[3] C. Galindo-Legaria and A. Rosenthal, "Outerjoin simplification and reordering for query optimization," *ACM Transactions on Database Systems*, vol. 22, no. 1, pp. 43-74, 1997. http://dx.doi.org/10.1145/244810.244812

[4] C. Fraser, L. Giakoumakis, V. Hamine, and K. F. Moore-Smith, "Testing cardinality estimation models in SQL server," in *Proceedings of 5th International Workshop on Testing Database Systems (DBTest'12)*, Scottsdale, AZ, 2012. http://dx.doi.org/10.1145/2304510.2304526

[5] T. Neumann, "Query simplification: graceful degradation for join-order optimization," in *Proceedings of ACM SIGMOD International Conference on Management of data (SIGMOD'09)*, Providence, RI, 2009, pp. 403-414. http://dx.doi.org/10.1145/1559845.1559889

[6] T. Neumann and C. Galindo-Legaria, "Taking the edge off cardinality estimation errors using incremental execution," in *Proceedings of Datenbanksysteme fur Business, Technologie und Web (BTW2013)*, Magdeburg, Germany, 2013, pp. 73-92.

[7] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacigumus, and J. F. Naughton, "Predicting query execution time: are optimizer cost models really unusable?," in *Proceedings of IEEE 29th International Conference on Data Engineering (ICDE'13)*, Brisbane, Australia, 2013, pp. 1081-1092. http://dx.doi.org/10.1109/ICDE.2013.6544899

[8] G. Lohman, "Is query optimization a "solved" problem?," Available http://wp.sigmod.org/?p=1075

[9] S. Chaudhuri, "Query optimizers: time to rethink the contract?," in *Proceedings of ACM SIGMOD International Conference on Management of data (SIGMOD'09)*, Providence, RI, 2009, pp. 961-968. http://dx.doi.org/10.1145/1559845.1559955

[10] S. Bellamkonda, H. G. Li, U. Jagtap, Y. Zhu, V. Liang, and T. Cruanes, "Adaptive and big data scale parallel execution in oracle," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1102-1113, 2013. http://dx.doi.org/10.14778/2536222.2536235

[11] F. Liu and S. Blanas, "Forecasting the cost of processing multi-join queries via hashing for main-memory databases," in *Proceedings of the 6th ACM Symposium on Cloud Computing (SoCC'15)*, Kohala Coast, HI, 2015, pp. 153-166. http://dx.doi.org/10.1145/2806777.2806944

[12] F. M. Waas, L. Giakoumakis, and S. Zhang, "Plan space analysis: an early warning system to detect plan regressions in cost-based optimizers," in *Proceedings of the 4th International Workshop on Testing Database Systems (DBTest'11)*, Athens, Greece, 2011. http://dx.doi.org/10.1145/1988842.1988844

Aigerim Bakatkaliyevna Altayeva received her B.S., M.Sc., from the Department of Computer Science, IITU, Kazakhstan, in 2012, 2014. She is a Ph.D. student at Gachon University. Her interesting part is AI, smart city, big data etc.
Tel: +82-31-750-5800
E-mail: aikosha1703@gmail.com

Youngmi Yoon received her B.S from Seoul National University in 1981; the M.S. from Stanford University in 1984, and the Ph.D. degree in computer science from Yonsei University in 2008. She is a professor at Gachon University. Her research interest includes database, data mining, and bioinformatics.
Tel: +82-31-750-4755
Fax: +82-31-750-5662
E-mail: ymyoon@gachon.ac.kr

Young Im Cho received her B.S., M.Sc., and Ph.D. from the Department of Computer Science, Korea University, Korea, in 1988, 1990 and 1994, respectively. She is a professor at Gachon University. Her research interest includes AI, big data, information retrieval, smart city etc.
Tel: +82-31-750-5800
Fax: +82-31-750-5662
E-mail: yicho@gachon.ac.kr