WinMagic : Subquery Elimination Using Window Aggregation

Calisto Zuzarte	Hamid Pirahesh	Wenbin Ma	Qi Cheng	Linqi Liu	Kwai Wong
IBM	IBM	IBM	IBM	IBM	IBM
8200 Warden Ave.	650 Harry Road	8200 Warden Ave.	8200 Warden Ave.	8200 Warden Ave.	8200 Warden Ave.
Markham,On,Canada	San Jose, California	Markham,On,Canada	Markham,On,Canada	Markham,On,Canada	Markham,On,Canada
(905) 413 2530	(408) 927 1754	(905) 413 4192	(905) 413 2804	(905) 413 3898	(905) 413 2818
calisto@ca.ibm.com	pirahesh@almaden.ibm.com	wenbinm@ca.ibm.com	qicheng@ca.ibm.com	liulinqi@ca.ibm.com	kwaiwong@ca.ibm.com

ABSTRACT

Database queries often take the form of correlated SQL queries. Correlation refers to the use of values from the outer query block to compute the inner subquery. This is a convenient paradigm for SQL programmers and closely mimics a function invocation paradigm in a typical computer programming language. Queries with correlated subqueries are also often created by SQL generators that translate queries from application domain-specific languages into SQL. Another significant class of queries that use this correlated subquery form is that involving temporal databases using SQL. Performance of these queries is an important consideration particularly in large databases. Several proposals to improve the performance of SQL queries containing correlated subqueries can be found in database literature. One of the main ideas in many of these proposals is to suitably decorrelate the subquery internally to avoid a tuple-at-a-time invocation of the subquery. Magic decorrelation is one method that has been successfully used. Another proposal is to cache the portion of the subquery that is invariant with the changing values of the outer query block. What we propose here is a new technique to handle some typical correlated queries. We go a step further than to simply decorrelate the subquery. By making use of extended window aggregation capabilities, we eliminate redundant access to common tables referenced in the outer query block and the subquery. This technique can be exploited even for non-correlated subqueries. It is possible to get a huge boost in performance for queries that can exploit this technique, which we call WinMagic. This technique was implemented in IBM® DB2® Universal Database™ Version 7 and Version 8. In addition to improving DB2 customer queries that contain aggregation subqueries, it has provided significant improvements in a number of TPCH benchmarks that IBM has published since late in 2001.

1. INTRODUCTION

Large database systems are often associated with complex queries because applications, competing for processing time, try to retrieve much of the information in a single query against the database. A common structure for such queries is one that uses correlated subqueries and is often associated with aggregation. Correlation refers to the use of values from the outer query block to compute the inner subquery. An example of such an SQL statement is the query asking for the list of employees from a specific location, their departments and their salaries, and where the salary is greater than the average salary within a department:

```
Query 1:

SELECT emp_id, emp_name, dept_name
FROM employee E, department D
WHERE E.dept_num = D.dept_num AND
      E.state = 'CALIFORNIA' AND
      E.salary > (SELECT AVG(salary)
                  FROM employee E1
                   WHERE E1.dept_num = D.dept_num);
```

For each department that is needed in the outer query block, we need to go back to the employee table and compute the average salary of employees within that department. Depending on the extent of the restrictions, other than the subquery predicate, on the employee table, we access a good portion of the employee table more than once for both the outer query block and the subquery block. In a partitioned (shared-nothing) environment, this could also mean a significant amount of network traffic to evaluate the subquery remotely with each value from the outer row.

A similar issue is seen in temporal databases, where the tables contain some aspect of time. Consider the following query:

```
Query 2:

SELECT * FROM empl E1
WHERE eff_date = (SELECT MAX(eff_date)
          FROM empl E2
          WHERE E1.emplid = E2.emplid) AND
      seq = (SELECT MAX(seq)
          FROM empl E3
          WHERE E1.emplid = E3.emplid and
               E1.eff_date = E3.eff_date)
```

Here, the employee table contains records of the employees' past lives within the organization. In the query above, we are looking for the most recently updated employee row given that the effective_date and sequence (seq) column values provide us with the latest updates.

One can see that this query, as is, would not have an efficient plan. Later in this paper (query 8) we show how this query can be rewritten automatically using window aggregate functions in a

SIGMOD 2003, June 9-12, 2003, San Diego, CA.

much more efficient way avoiding correlation and the joins to multiple instances of the same table.

2. PRIOR ART

One could comprehend splitting query 1 into two steps. The first step might compute the average salary of the employees in every relevant department by joining the relevant employee data from the employee table and the department. This data could be stored within the application or in a temporary table containing information of the department number, name and average salary (let us call this column avgsal). Next we could go back and access the employee table and join this to the temporary table to apply the salary > avgsal predicate to get the result. This could be a disaster when there is a large amount of data involved. It is not difficult to realize that there are huge benefits to evaluating the result of the query in one single statement.

The traditional approach to evaluating such correlated subqueries is to use a nested iteration approach. Here the subquery is executed for each row of the outer query block in literal compliance with the semantics of the SQL statement. Although this approach may be adequate in some circumstances, it can be expensive, particularly in a massively parallel system using a shared-nothing architecture. In such a system, with the data being distributed in different partitions, this tuple-at-a-time approach can be expensive. The employee table could be split across several partitions and there would be a need to consolidate the average salary for a department at a central coordinator node before applying the subquery.

More recently, particularly beneficial for a massively parallel system (shared-nothing) environment, methods to decorrelate this query have been proposed. In [1], certain fixed forms of complex queries were recognized and rewritten. The work of [2] improved on the technique where the use of the outer join solved the wrong result issue when the result of the subquery was empty. In [3], correlation values are collected in a temporary table and a distinct collection is projected before joining to the subquery.

In [4],[5],[6] a technique called magic decorrelation is developed where the relevant distinct values of the outer references are extracted and, based on these values, all the possible results from the subquery are materialized. The materialized results are joined with the outer query block on the outer referenced values. Although the rewritten query introduces extra views, joins and duplicate removal, we can expect better performance since the subquery is evaluated once with a consolidated temporary relation and avoids a tuple-at-a-time communication overhead.

Decorrelation is not always possible and in some cases, even if possible, may not always be efficient. In [7], a technique is proposed where a portion of the query that is invariant with respect to the changing outer values is cached. The cached result is reused in subsequent executions and combined with the new results in the changing portion of the subquery.

The recognition of redundancy and inefficiency when processing such queries in commercial databases is evident in [8] and [9]. In these papers an extension of the SQL syntax is proposed that allows more efficient processing to be done on a group-by-group basis. This makes the queries simpler and easier to handle in the optimizer. The SQL standard compliant window aggregate functions syntax already implemented in DB2 Universal Database is a more powerful syntax. It also provides a way of expressing the queries that allows a reduction of redundancy and inefficiency. The subject of our paper is to transform queries automatically to exploit this relatively new feature.

In [10], decorrelation techniques adopted in the Microsoft® SQL Server product are described. The concept that is most relevant is one called SegmentApply. Whenever a join connects two instances of an expression and one of the expressions has an extra aggregate and/or a filter, they try to generate a common sub-expression (CSE). The extra aggregation is done on one consumer of the CSE and is joined to all rows in that group from the other consumer of the CSE. This is done one group at a time. They also consider pushing appropriate joins through the CSE. This technique is closest to our proposal, the major difference being that with window aggregation, we go a step further and do not require a CSE and the join.

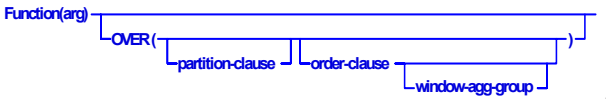
3. WINMAGIC

The target queries that can best exploit the transformation that we propose are often those that are optimized using decorrelation techniques and contain aggregation in the subqueries. What we propose is a new technique where we not only decorrelate the subquery but also go a step further and eliminate the subquery. By making use of extended window aggregation capabilities, we eliminate access to common tables referenced in the outer query block and the subquery. This provides a huge boost in performance. Note that this technique is also applicable to noncorrelated subqueries as we show later. Let us first briefly discuss what window aggregate functions are.

3.1 Window aggregate functions

While most SQL users are familiar with regular aggregation functions like MAX, MIN, SUM and AVG, there has been a relatively recent adoption of another class of aggregation functions. These are window aggregation functions that work on a specified group of rows and report the result on the current row being evaluated. This is both an aggregation function and in some sense a scalar function since it does not collapse the rows involved when computing the aggregation.

The general format of such a function that has been adopted by the SQL standard is:



The OVER clause specifies the three primary attributes of the function. These three attributes are optional. The order-clause is like an ORDER BY clause of a statement except that the order is only relevant in the context of the function. The partition-clause is similar to the commonly used GROUP BY clause but again is relevant only in the context of the function. The window-agg-group clause allows the specification of a window of rows to which the aggregation is applied.

For the purpose of this paper, only the partition-clause will be expanded upon. This will help illustrate the intended WinMagic transformations.

To illustrate the use of the partition-clause, in query 3, for each employee, we get the department, salary and the sum of all salaries within the employee's department. Note that the deptsum column value is repeated for each row that corresponds to that department. This repetitive information may or may not be output directly but could be used to compute other useful information. For example, in the statement below, the final column gives the percentage of the employee's salary in proportion to the total salary of all employees within the department.

```
Query 3:

SELECT empnum, dept, salary,
   SUM(salary) OVER (partition by dept) AS deptsum
   DECIMAL(salary,17,0) * 100 /
           SUM(salary) OVER(partition by
         dept) AS salratio
FROM employee;
```

EMPNUM	DEPT	SALARY	DEPTSUM	SALRATIO
1	1	78000	383000	20.365
2	1	75000	383000	19.582
5	1	75000	383000	19.582
6	1	53000	383000	13.838
7	1	52000	383000	13.577
11	1	50000	383000	13.054
4	2	-	51000	-
9	2	51000	51000	100.000
8	3	79000	209000	37.799
10	3	75000	209000	35.885
12	3	55000	209000	26.315
0	-	-	84000	-
3	-	84000	84000	100.000

3.2 WinMagic Transformation

Consider the following query from the TPCH benchmark.

```
Query 4:

SELECT SUM(l_extendedprice) / 7.0 AS avg_yearly
FROM   tpcd.lineitem, tpcd.part
WHERE p_partkey = l_partkey AND
      p_brand = 'Brand#23' AND
      p_container = 'MED BOX' AND
      l_quantity<(SELECT 0.2*avg(l_quantity)
                  FROM tpcd.lineitem
                  WHERE l_partkey = p_partkey);
```

To transform the query we perform these steps:

(1) Check the outer block to ensure (a) that it contains a subquery with aggregation, and (b) that we have no conditions that prevent breaking up the main query block such as functions with side effects.
(2) Check the subquery block to ensure (a) that there are no odd functions, and (b) that there are no constructs such as ORDER BY and "fetch first n rows", and (c) that the subquery is not used as a common sub-expression.
(3) Check the aggregation function to ensure (a) that it has an equivalent window aggregation function, and (b) that there is no DISTINCT within the aggregation function.
(4) Match the subquery to the outer block. A temporary supplementary query block is constructed with candidate tables from the outer block including common tables, those involved in the correlation. This is done to be able to call the matching routines used for matching materialized views. Once this is done, we can get rid of this supplementary query block.

The main steps used to rewrite the query are:

(1) Replace the aggregation by the window aggregation function as an extra column in the subquery. The partition-clause in the window aggregation function is used to define the grouping.
(2) Pull down the tables involved in the correlation from the outer query block to the subquery. No costing is required if the tables being joined in the outer query block are primary key or unique column joins. If this is the case, then we do not multiply the number of rows that go into the aggregation. If this is not the case, then we need to cost the transformation and compensate within the aggregation by adding a key.
(3) Finally redirect all the columns required in the outer query to those flowing through the subquery and get rid of the useless unreferenced tables in the outer query block.

The resulting query can be written as follows:

```
Query 5:

 WITH WinMagic AS
  (SELECT l_extendedprice, l_quantity,
    avg(l_quantity)over(partition by p_partkey)
            AS avg_l_quantity
   FROM  tpcd.lineitem, tpcd.part
   WHERE p_partkey = l_partkey and
         p_brand = 'Brand#23' and
         p_container =  'MED BOX' )
 SELECT SUM(l_extendedprice) / 7.0 as avg_yearly
 FROM WinMagic
 WHERE l_quantity < 0.2 * avg_l_quantity;
```

The original query accessed the LINEITEM table twice, once in the main query block and once in the subquery block. The major benefit of our transformation is the elimination of an access of the LINEITEM table without the need for a CSE and a join. In addition the PART table is joined to the LINEITEM table before the aggregation. This allows us to compute the aggregation for the partitions that are relevant to the query, which is what effectively what sideways-information-passing (SIP) does in magic decorrelation [5],[6].

3.3 General considerations for WinMagic

We have seen the simple case based on the following structure where the dotted line shows correlation:

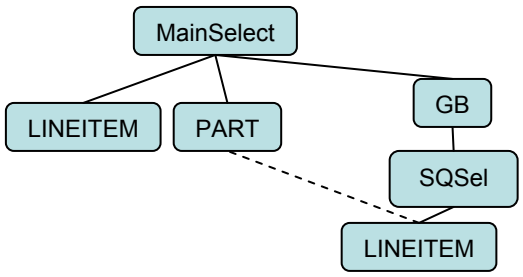


Figure 1 : Representation of Query 4

More complex scenarios can be handled within DB2. A generalized representation is shown below:

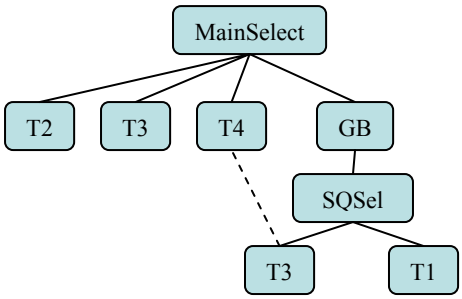


Figure 2 : Query with correlated subquery

where

- T1, T2, T3, T4 could be a set of one or more tables or views.
- T1 in the subquery is a lossless join.
- T2 appears in the main query but is not joined to T4.

Because of the power of the materialized view matching in DB2 Universal Database [11], it was easy to cover a very general query. The resulting transformation of the general query is shown in figure 3:

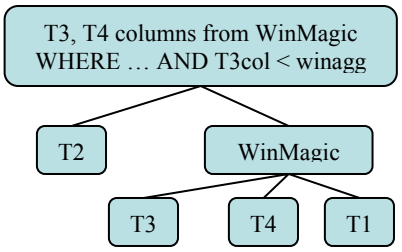


Figure 3 : Internal query after WinMagic

The portion of the subquery referencing the overlapping tables must typically subsume the corresponding portion of the outer query block in order to compute the aggregation correctly and eliminate one invocation of the common tables. However, as an extension to this technique, if there are more restrictive predicates in the subquery, the window aggregate function could be suitably modified to account for the additional restriction as part of the computation of the aggregate. For example:

```
Query 6:

SELECT cntrycode, COUNT(*) AS numcust,
        SUM(c_acctbal) AS totacctbal
FROM (SELECT SUBSTR(c_phone, 1, 2) AS cntrycode,
             c_acctbal
      FROM  tpcd.customer
      WHERE SUBSTR(c_phone, 1, 2) IN ('13','31','23')
        AND c_acctbal >
         (SELECT avg(c_acctbal)
          FROM  tpcd.customer
          WHERE c_acctbal > 0.00 AND
          SUBSTR(c_phone, 1, 2) IN ('13','31','23'))
     ) as cstsale
GROUP BY cntrycode ORDER BY cntrycode;
```

The extra subquery predicate must not be applied to the rows of the outer query block (although in this specific case it would be OK). However, the window aggregation function should account for the predicate through judicious use of the CASE expression to ensure that the aggregation is unaffected by rows that do not satisfy the extra predicate.

```
Query 7:

SELECT cntrycode, COUNT(*) AS numcust,
          SUM(c_acctbal) AS totacctbal
FROM (SELECT SUBSTR(c_phone, 1, 2)AS cntrycode,
        c_acctbal,
        AVG(CASE WHEN (c_acctbal > 0) THEN c_acctbal
              ELSE NULL END) OVER( ) AS AVGACCTBAL,
      FROM  tpcd.customer
      WHERE SUBSTR(c_phone,1,2) IN ('13', '31', '23')
      ) AS cstsale
WHERE c_acctbal > AVGACCTBAL
GROUP BY cntrycode ORDER BY cntrycode;
```

WinMagic is not only applicable to correlated subqueries. Evaluate-at-open subqueries could also be handled. WinMagic can also be extended to cover cases that do not contain aggregation in the subquery. As in query 2 provided earlier in the paper, we can show that the query can be simplified by eliminating the joins as follows:

```
Query 8 (WinMagic for temporal database query)

SELECT … FROM
 (SELECT E1.*,
   max(eff_date) over (partition by emplid) as ME
   max(seq) over (partition by emplid,eff_date) as MS
  FROM empl E1) as E
WHERE eff_date = ME AND seq = MS;
```

4. PERFORMANCE RESULTS

WinMagic was used with significant impact in some of the recent TPCH benchmarks. In a 100GB scale TPCH database, the test environment included IBM p660s running the AIX® operating system using a 5-node partitioned database. The results are shown below with the second run using a 5-node partitioned database with intra-partition parallelism set to degree 4.

The result of query 4 (TPCH Q17) showed a significant improvement with a performance gain of about 50%. Much of this is attributed to eliminating the access to the LINEITEM table.

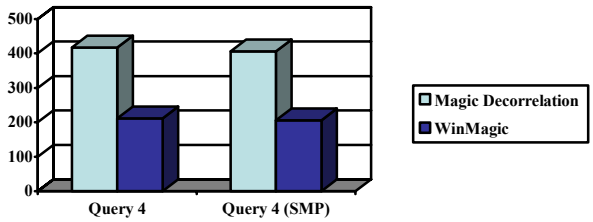


Figure 4: About 50% improvement with WinMagic for query 4 (100 GB TPCH Q17)

The other query where WinMagic was used was TPCH Q2:

```
Query 9:

SELECT  s_acctbal, s_name, n_name, p_partkey, p_mfgr,
        s_address, s_phone,  s_comment
FROM tpcd.part, tpcd.supplier, tpcd.partsupp,
     tpcd.nation, tpcd.region
WHERE p_partkey = ps_partkey AND
      s_suppkey = ps_suppkey AND
      p_size = 15 AND p_type like '%BRASS' AND
      s_nationkey = n_nationkey AND
      n_regionkey =  r_regionkey AND
      r_name = 'EUROPE' AND
```

```
      ps_supplycost =
       (SELECT MIN(ps_supplycost)
        FROM tpcd.partsupp, tpcd.supplier, tpcd.nation,
             tpcd.region
        WHERE p_partkey = ps_partkey AND
              s_suppkey = ps_suppkey AND
              s_nationkey = n_nationkey AND
              n_regionkey = r_regionkey AND
              r_name = 'EUROPE' )
ORDER BY s_acctbal desc, n_name, s_name, p_partkey
FETCH FIRST 100 ROWS ONLY;
```

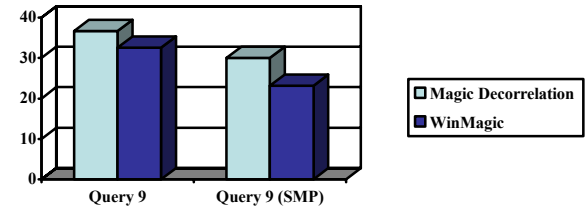


Figure 5 : !0%-15% impprovement with WinMagic for query 9 (100Gb TPCH Q2)

Even though multiple table accesses and joins were eliminated using WinMagic in Query 9, the performance gain was not as significant since the tables involved were small and the pages for the redundant table access was most likely in the bufferpool for the magic decorrelation version.

On a larger-scale 10TB TPCH database, the performance gains for both queries were 50% to 60% compared to the magic decorrelation technique.

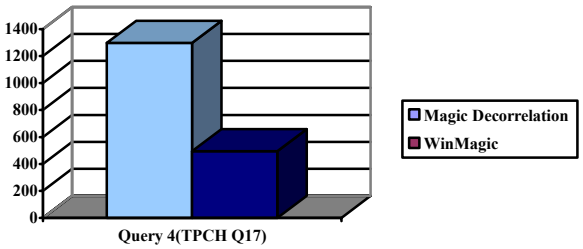


Figure 6: 60%- impprovement with WinMagic for query 4 (10TB TPCH Q17)

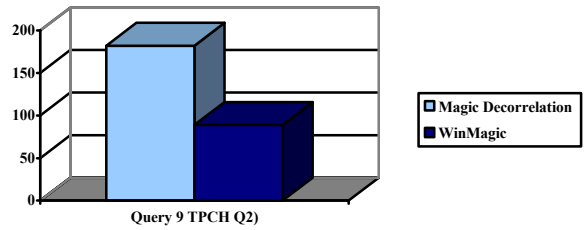


Figure 7: 50%- impprovement with WinMagic for query 9 (10TB TPCH Q2)

5. CONCLUSION

Using the power of the existing materialized view matching algorithms [11] and the new window aggregation functions, we could easily implement this powerful transformation with minimal effort. Many customers have such common queries written in the traditional form. Given that most customers do not have the luxury of rewriting their applications to exploit the new window aggregation syntax, this internal rewrite can be expected to have a huge impact on improving the performance of such queries.

6. REFERENCES

[1] W. Kim. “On Optimizing an SQL-Like Nested Query”, ACM Transactions on Database Systems, 7 Sep 1982.

[2] U. Dayal: “Of Nests and Trees: A Unified Approach to Processing Queries that Contain Nested Subqueries, Aggregates and Quantifiers”. Proceedings on the Eighteenth International Conference on Very Large Databases (VLDB) pp. 197-208, 1987

[3] R. Ganski and H. Wong “Optimization of Nested SQL Queries Revisited”, Proceedings of ACM SIGMOD, San Francisco, California, U.S.A., 1987 pp 22-33

[4] I. S. Mumick, H. Pirahesh, and R. Ramakrishnan. The Magic of Duplicates and Aggregates. In Proceedings. 16th International Conference on Very Large Data Bases, Brisbane, August 1990.

[5] C. Leung, H. Pirahesh, P. Seshadri and J. Hellerstein. "Query Rewrite Optimization Rules in IBM DB2 Universal Database". In Readings in Database Systems, Third Edition, M.Stonebraker and J.Hellerstein (eds.), Morgan Kaufmann, pp. 153-168, 1998.

[6] P. Seshadri, H. Pirahsh and T.Y.C. Leung. "Complex Query Decorrelation". Proceedings of the International Conference on Data Engineering (ICDE), Louisiana, USA, February 1996.

[7] Jun Rao and Kenneth A. Ross. "A New Strategy for Correlated Queries". Proceedings of the ACM SIGMOD Conference, pages 37-48, ACM Press, New York, 1998.

[8] D. Chatziantoniou and K. A. Ross. Querying multiple features of groups in relational databases. In Proceedings of the 22rd International Conference on Very Large Databases, pages 295-306, 1996.

[9] D. Chatziantoniou and K. A. Ross. Groupwise processing of relational queries. In Proceedings of the 23rd International Conference on Very Large Databases, Athens, pp 476–485, 1997.

[10] C.A. GalindoLegaria and M. Joshi. Orthogonal Optimization of Subqueries and Aggregation. In Proceedings of ACM SIGMOD, International Conference on Management of Data, Santa Barbara, California, U.S.A 2001

[11] M.Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh and M. Urata. Answering complex SQL queries using automatic summary tables. In SIGMOD 2000, pages 105-116