Automated Selection of Materialized Views and Indexes for SQL Databases

Sanjay Agrawal *Surajit Chaudhuri* *Vivek Narasayya*

Presentation By:
Tarun Jain

Department of Computer Science & Engineering
Indian Institute of Technology Bombay

Outline

Review : Materialized Views

- A **materialized view** is a view whose contents are computed and stored.

Example : Consider the view
c**reate view** *branch_total_loan*(*branch_name, total_loan*) **as**
select *branch_name*, **sum**(*amount*)
from *loan*
group by *branch_name*

- Materializing the above view would be very useful if the total loan amount is required frequently

- Materialized views may also be indexed.

- The join results are computed once (or as often as you refresh your materialized view), rather than each time you select from the materialized view.

Motivation

- Selecting appropriate set of Indexes and Materialized views influenced by workload of the system.
- A workload consists of a set of SQL data manipulation statements.
- DBA have to administer manually – create indexes, materialized views, indexes on materialized views for performance tuning

  Such an approach is
    1) Time Consuming
    2) Error Prone
    3) Might not be able to handle continuosly changing or growing workloads

Motivation Example

Consider an Online game which store the records of players in a table **LeaderBoard(player_id,region,score,timestamp)**

Mv_1 : Select max(score) from LeaderBoard;

Mv_2 : Select region , max(score) from LeaderBoard Group By region;

Mv_3 : Select region, sum(score) , max(score) from LeaderBoard Group BY region;

Mv_4 : Select Top 10 players on the leaderboard

To answer the following queries :

1) Current maximum score	- Mv_1 , Mv_2 ,Mv_3
2) maximum score of region 'A'	- Mv_2 , Mv_3
3) region with overall best total score	- Mv_3

Introduction

- Both indexes and materialized views are fundamentally similar – both are redundant structures that speed up query execution.
- Index can logically be considered as single-table, projection only materialized view.
- Though they are similar but a materialized view may be defined over multiple tables, and can have selections and GROUP BY over multiple columns.
- Need for efficient ways for dealing with the large space of potentially interesting materialized views for a given set of SQL queries

Key Contributions

- The paper present an architecture and novel algorithms for addressing automated materialized view selection.
- Takes into account the significant enhancement that can be achieved by interaction between indexes and materialized views
- Introduce a principled way to identify a much smaller set of candidate materialized views .
- Database design tool that can determine an appropriate set of indexes, materialized views for a given database and workload consisting of SQL queries and updates.
- This tool became part of Microsoft SQL Server 2000 and onward releases.

Architecture for
Index and Materialized View Selection

<u>Key Components of this Architecture:</u>

- Syntactic structure selection
- Candidate selection
  - -> Index Selection
  - -> Materialized View Selection
- Configuration enumeration
- Configuration simulation and Cost estimation

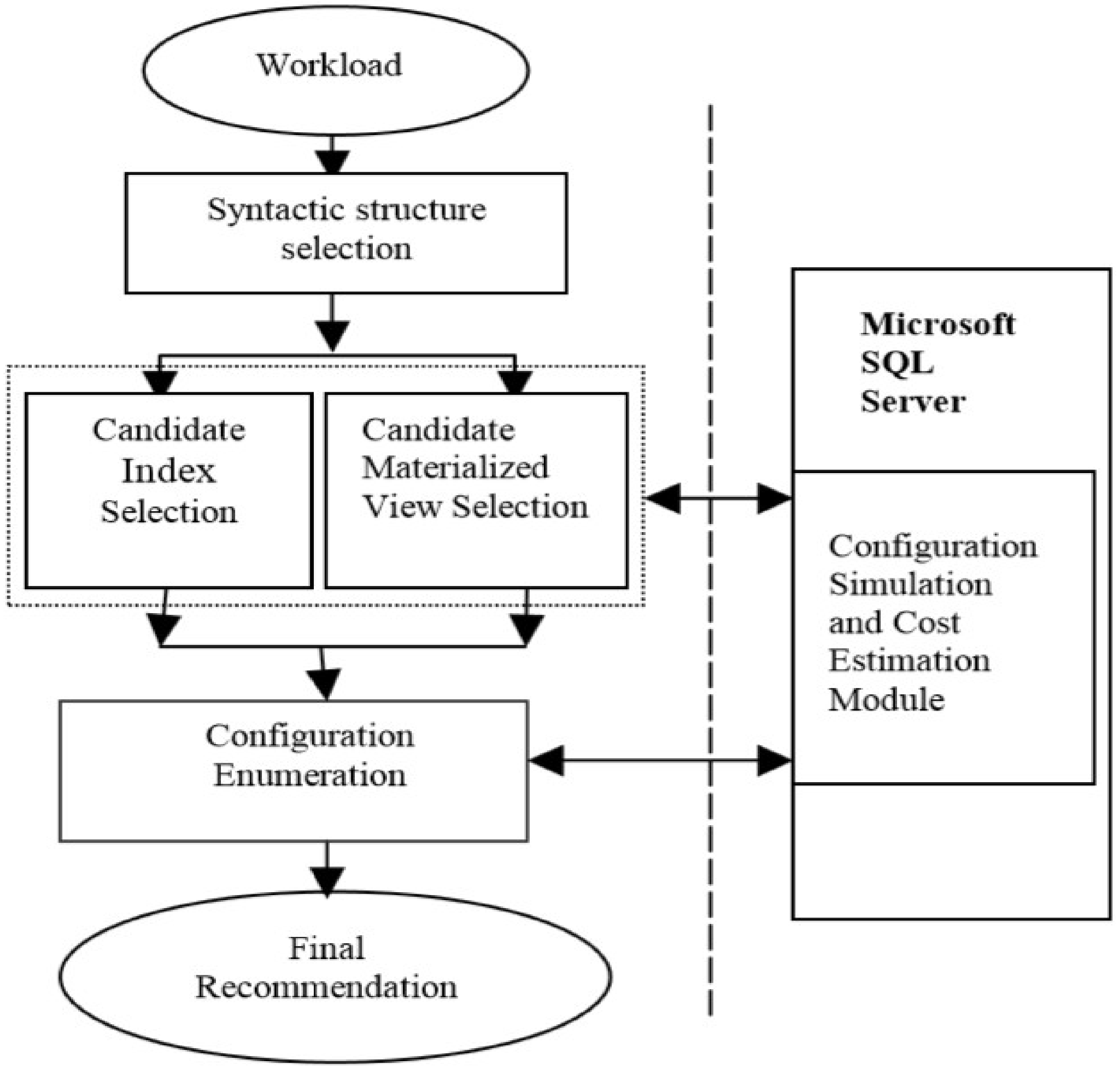

Workload
Syntactic structure selection
Candidate Index Selection
Candidate Materialized View Selection
Configuration Enumeration
Final Recommendation
Microsoft SQL Server
Configuration Simulation and Cost Estimation Module

Figure . Architecture of Index and Materialized View Selection Tool

Architecture for Index and Materialized View Selection

Step – 1 : Syntactic Structure selection

To identify syntactically relevant indexes , materialized views and indexes on materialized views that can potentially be used to answer the query.

For example:

Query **Q**: *SELECT Sum(Sales) FROM Sales_Data*
WHERE City = 'Delhi'

The ***syntactically relevant materialized views****:*

v1: *SELECT Sum(Sales) FROM Sales_Data*
WHERE City =‘Delhi’

v2: *SELECT City, Sum(Sales) FROM Sales_Data*
GROUP BY City

v3: *SELECT City, Product, Sum(Sales)*
FROM Sales_Data GROUP BY City, Product

Architecture for Index and Materialized View Selection

Step – 2 : Candidate Selection

- Eliminate spurious candidates and focus on smaller search space.

- Candidate selection is responsible for Identifying a sets of structure for the given workload which are worthy of further exploration

Note:

- This paper focuses only on efficient selection of candidate materialized views.
- The candidate index selection is assumed to be already done.
- The issues related to selection of indexes on materialized views is not discussed in this paper.

Architecture for Index and Materialized View Selection

Step – 3 : Configuration Enumeration

- Search among structures selected in Step-2 , inorder to determine ideal physical design- called **configuration**.
- *Configuration consists of set of traditional indexes , materialized views and indexes on materialized views.*
- Search using the naive approach is infeasible
- Thus we adopt the GREEDY algorithm for configuration enumeration.

Architecture for Index and Materialized View Selection

Step – 3 : Configuration Enumeration

The Naive Approach(Index Selection):

- There are n candidate indexes, and we are asked to find optimal configuration of size at most k structures
- Enumerate all subsets of the candidate structures of size k or less
- Pick the one with lowest total cost.
- This gurantees an optimal solution , but complexity of search is exponentially large.
- For Example :
  n = 40 , K = 10

Architecture for Index and Materialized View Selection

Step – 3 : Configuration Enumeration(continued)

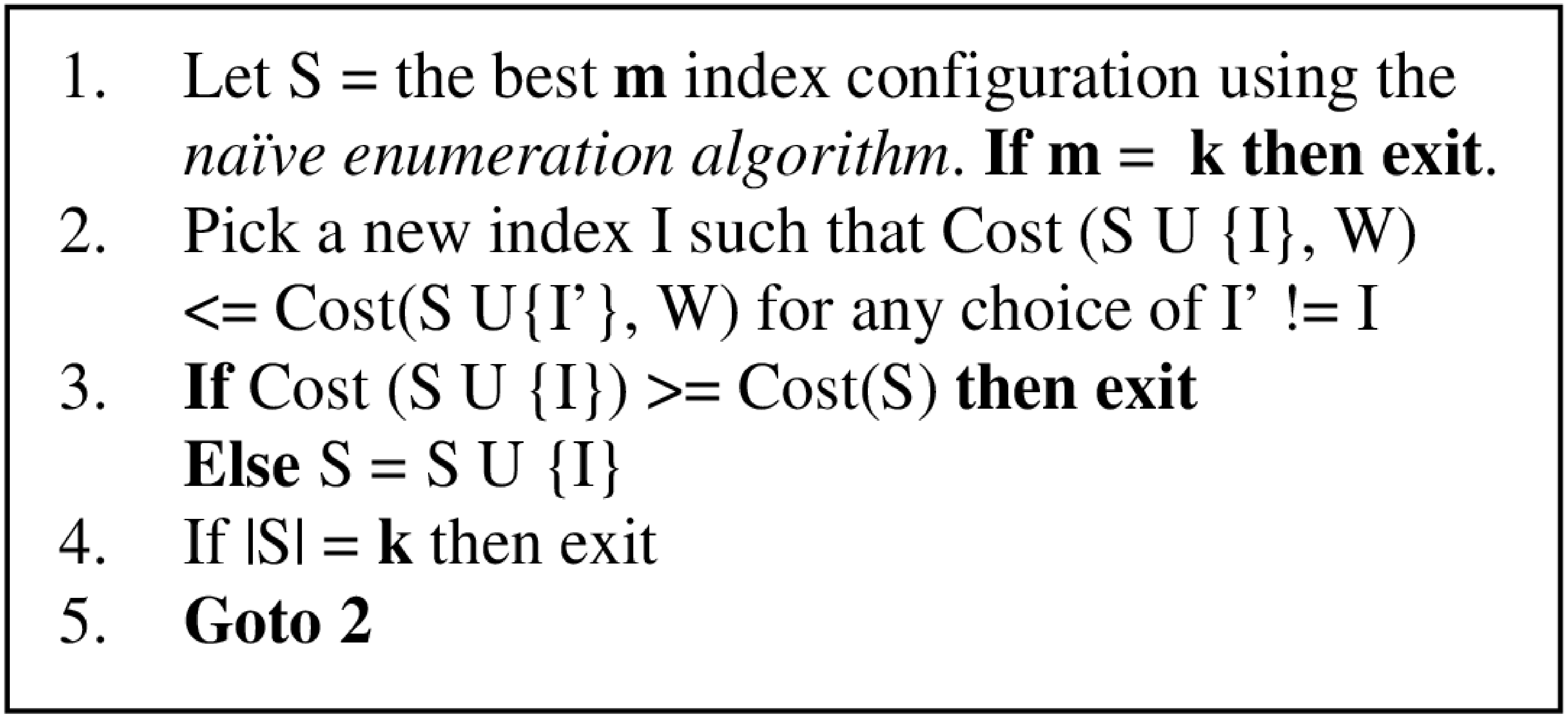
The Greedy(m,k) Algorithm :

- Returns a configuration consisting of a total of k indexes and materialized views
- It first picks an optimal configuration of size up to m (≤ k) by exhaustively enumerating all configurations of size up to m. (seed)
- Each greedy step considers all possible choices for adding one more index and adds the one resulting in the highest cost reduction
- The alogorithm continues until all k indexes and materialized views have been chosen, or no further reduction in cost is possible by adding a structure.

Architecture for Index and Materialized View Selection

Step – 3 : Configuration Enumeration(continued)

The Greedy(m,k) Algorithm for Index Selection :



```
1. Let S = the best m index configuration using the
   naïve enumeration algorithm. If m = k then exit.
2. Pick a new index I such that Cost (S U {I}, W)
   <= Cost(S U{I'}, W) for any choice of I' != I
3. If Cost (S U {I}) >= Cost(S) then exit
   Else S = S U {I}
4. If |S| = k then exit
5. Goto 2
```

Figure . The Greedy(m, k) enumeration algorithm.

Architecture for Index and Materialized View Selection

Step – 3 : Configuration Enumeration(continued)

The Greedy(m,k) Algorithm :

Note that :

- If the parameter m = 0 , then the algorithm takes a pure greedy approach.
- On the other hand, if m = k , the algorithm is identical to the naive enumeration algorithm.
- Therefore, the use of the algorithm is computationally efficient only if m is small relative to k. In such a case, the enumeration exhibits near greedy behavior.

Architecture for Index and Materialized View Selection

Step – 4 : Configuration simulation and Cost estimation

- Responsible for evaluating the cost of configurations.

Naive Approach (Index Selection):

- The cost-evaluator asks the optimizer for a cost estimate for each query in the workload.
- For M configurations and Q queries in the workload, such estimation requires asking the optimizer to optimize M*Q queries.
- Invoking the optimizer many times can be expensive

Architecture for Index and Materialized View Selection

Step – 4 : Configuration simulation and Cost estimation

Notion of Atomic Configurations:

- Configuration C is atomic for a workload if for some query in the workload there is a possible execution of the query that uses all indexes in C.

To find Cost(Q,C):

- C is a configuration that is not atomic and Q is a Select/Update query in the workload.
- Consider all atomic configurations Ci of Q that are subsets of C.
- Optimizer will choose the atomic configuration from the above set of Ci that has the minimal cost

Architecture for Index and Materialized View Selection

Step – 4 : Configuration simulation and Cost estimation

- **Therefore, we can derive**
  Cost (Q, C) = Min {(Cost(Q, Ci)}

- Intuitively in a Select query, it will suffice to take the minimum cost over the largest atomic configurations of Q that are subsets of C.

Example :

Consider following Indexes in C :
I1 reduces cost by 50 units , I2 reduces cost by 20 units
I3 reduces cost by 30 units , I4 reduces cost by 40 units

Possible Ci's = {I1,I2} , {I1,I4} , {I3,I4}
{I1,I4,I2} , {I1,I3,I4}

Architecture for Index and Materialized View Selection

Step – 4 : Configuration simulation and Cost estimation

The cost of an Insert/Delete query for a non-atomic configuration C

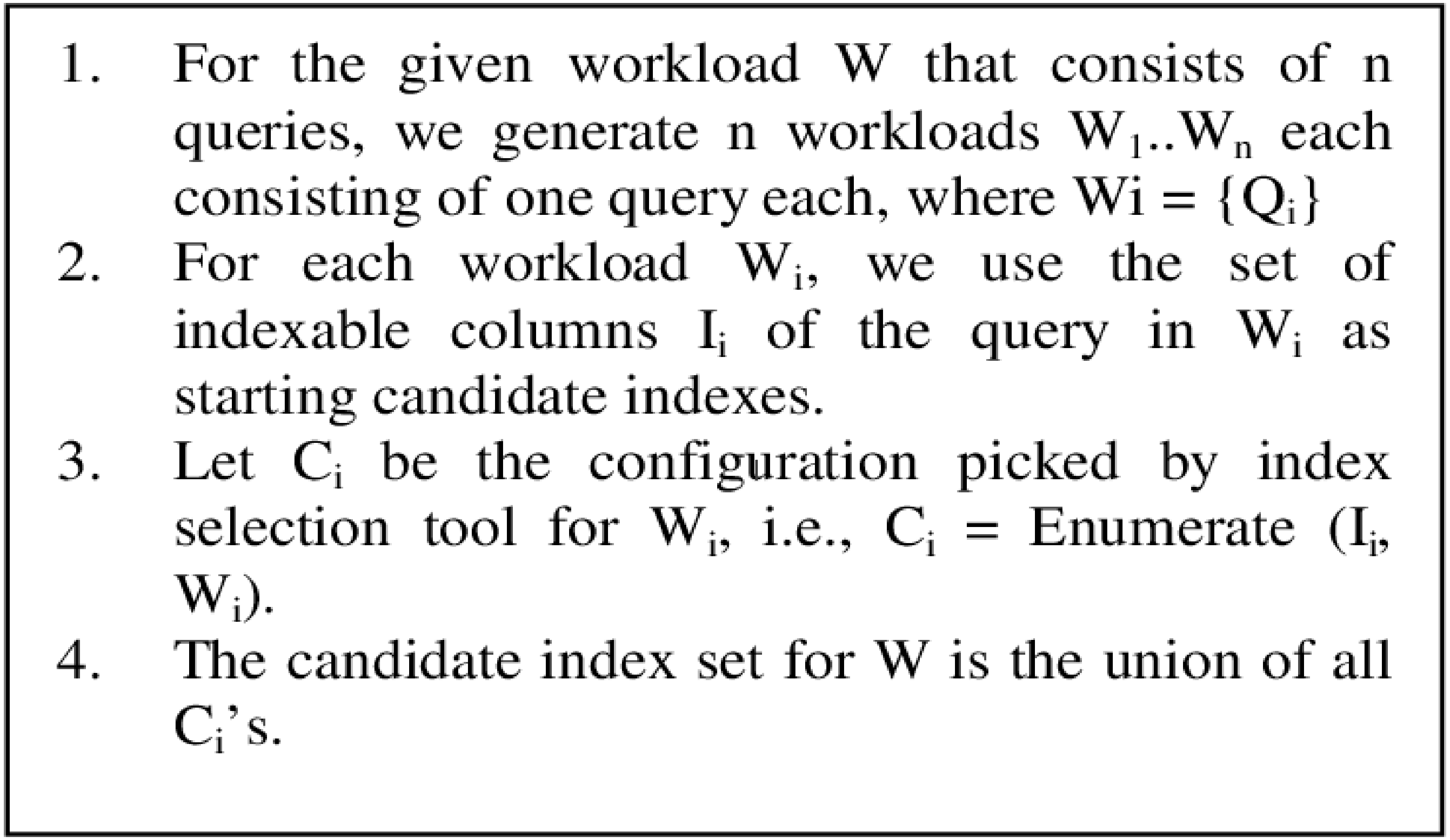
- Divided in three components:
  (a) Cost of selection
  (b) Cost of updating the table and the indexes used for selection
  (c) Cost for updating indexes that do not affect the selection cost.

- Note : cost for updating each index in (c) is independent of each other and can be assumed to be independent of the plan chosen for (a) and (b).

$$\text{Total cost} = T + \sum j\ (\text{Cost}(Q, \{Ij\}) - \text{Cost}(Q, \{\}))$$

(As in a Select/Update query, we can derive T)

Candidate Index Selection



1. For the given workload W that consists of n queries, we generate n workloads $W_1..W_n$ each consisting of one query each, where $Wi = \{Q_i\}$
2. For each workload W_i, we use the set of indexable columns I_i of the query in W_i as starting candidate indexes.
3. Let C_i be the configuration picked by index selection tool for W_i, i.e., C_i = Enumerate (I_i, W_i).
4. The candidate index set for W is the union of all C_i's.

Figure . Candidate index selection algorithm.

Candidate Materialized View Selection

Goal :

To eliminate materialized views that are syntactically relevant to one or two queries but are never used in answering any query.

Naive Approach :

Selecting one candidate materialized view per query that exactly matches each query in the workload.

Candidate Materialized View Selection

Scenario 1: Storage Constrained Environments

Consider a workload consisting of 1000 queries of the form:

```
        SELECT   attr_A, SUM(attr_B)
          FROM   table_T
         WHERE   attr_C  BETWEEN <val1> and <val2>
      GROUP BY  attr_A
```

Assume different constants for <val1> and <val2>

Naive Approach :
1000 Materialized Views for each query.

Better Alternative :

```
        SELECT  attr_C,attr_A, SUM(attr_B)
          FROM  table_T
      GROUP BY attr_C, attr_A
```

Observation :
Ignoring the commonality across queries in the workload can result in sub-optimal quality. The problem is more severe in case of larger workloads.

Candidate Materialized View Selection

Scenario 2:

Consider a workload consisting of 100 queries:

Total Cost of all queries : 10,000 units

Let 'T' be a table-subset that occurs in 25 queries:

Total Cost these 25 queries : 50 units

Then even if we considered all syntactically relevant materialized views on T, the maximum possible benefit of those materialized views for the workload is 0.5%.

Observation **:**

- There are certain table-subsets such that, even if we were to propose materialized views on those subsets it would only lead to a small reduction in cost for the entire workload.

Candidate Materialized View Selection

Scenario 3:

Consider a large set of queries in which some tables table_P, table_Q, table_R, table_S co-occur.

Assume that : table_P has 5 million tuples & table_Q has 4 million tuples
table_R has 100 tuples & table_S has 50 tuples

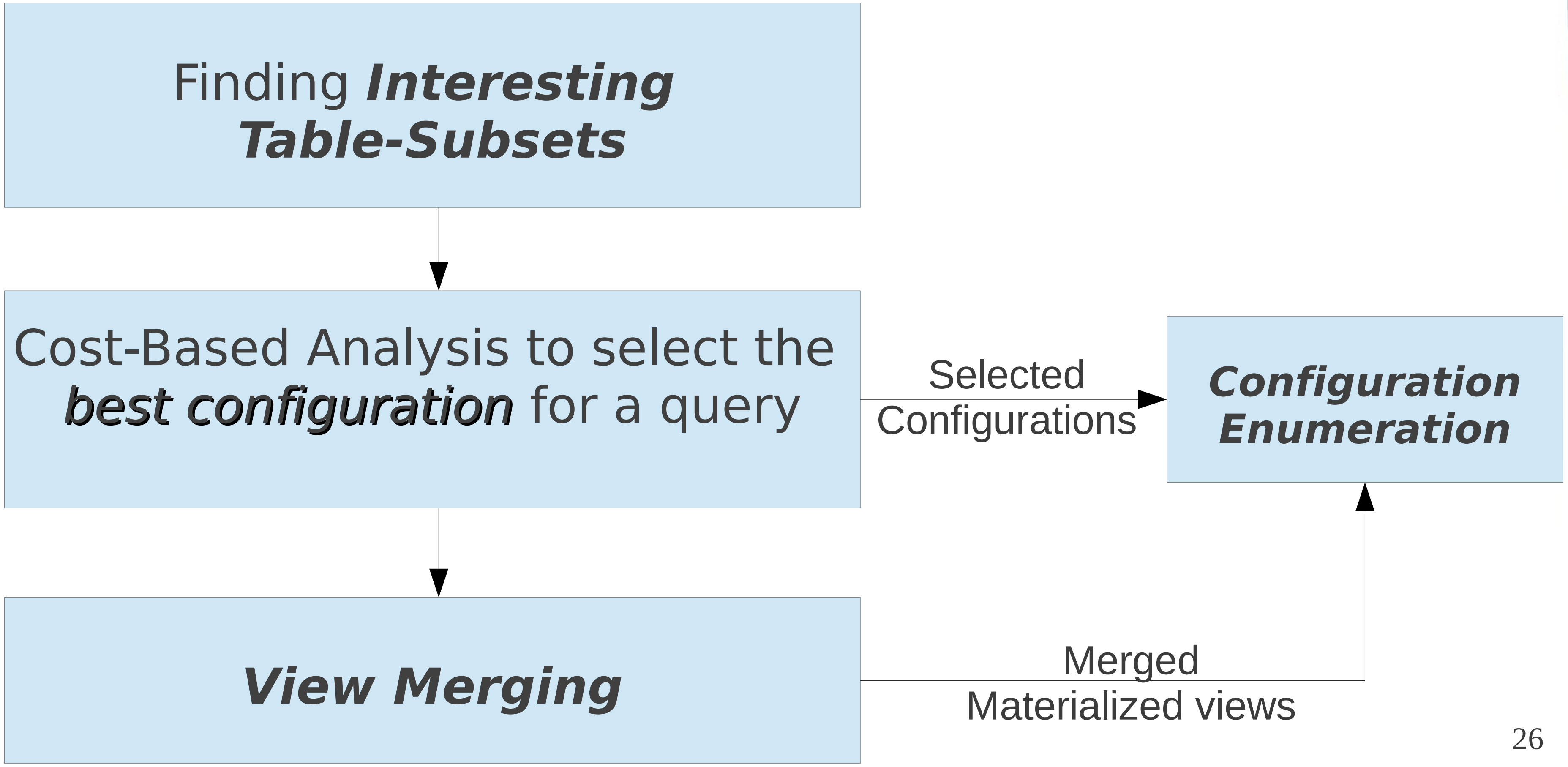
Hence from above statistics , it is likely that the materialized view on table-subset {table_P,table_Q} is **more useful** than that on {table_R,table_S}.

Observation :

The benefit of pre-computing the portion of the queries involving {table_R,table_S} **is insignificant** compared to the benefit of pre-computing the portion of the query involving {table_P,table_Q}.

Candidate Materialized View Selection

Based on the previous observations Candidate Materialized View selection can be done in three steps:



Finding ***Interesting Table-Subsets***
Cost-Based Analysis to select the ***best configuration*** for a query
Selected Configurations
Configuration Enumeration
View Merging
Merged Materialized views

Candidate Materialized View Selection

1) Finding Interesting Table-Subsets

- The a table-subset T is ***interesting*** if materializing one or more views on T has the potential to reduce the cost of the workload significantly, i.e., above a given threshold.
- Define a metric that captures the relative importance of a table-subset.
- Two table-subset metric we define here:

  a) TS-Cost(T)

  b) TS-Weight(T)

Candidate Materialized View Selection

TS-Cost(T) **=** total cost of all queries in the workload (for the current database) where table-subset T occurs.

- Monotonicity of TS-Cost(T)

  For table subsets T1, T2:

  If T1 is subset of T2 , then TS-Cost(T1) >= TS-Cost(T2)

TS-Weight(T) **=** $$\frac{\sum_i Cost(Q_i) * (sum\ of\ sizes\ of\ tables\ in\ T)}{(sum\ of\ sizes\ of\ all\ tables\ referenced\ in\ Q_i)}$$

- Also for any threshold C , the following holds good :

$$TS\text{-}Weight(T) \geq C \Rightarrow TS\text{-}Cost(T) \geq C$$

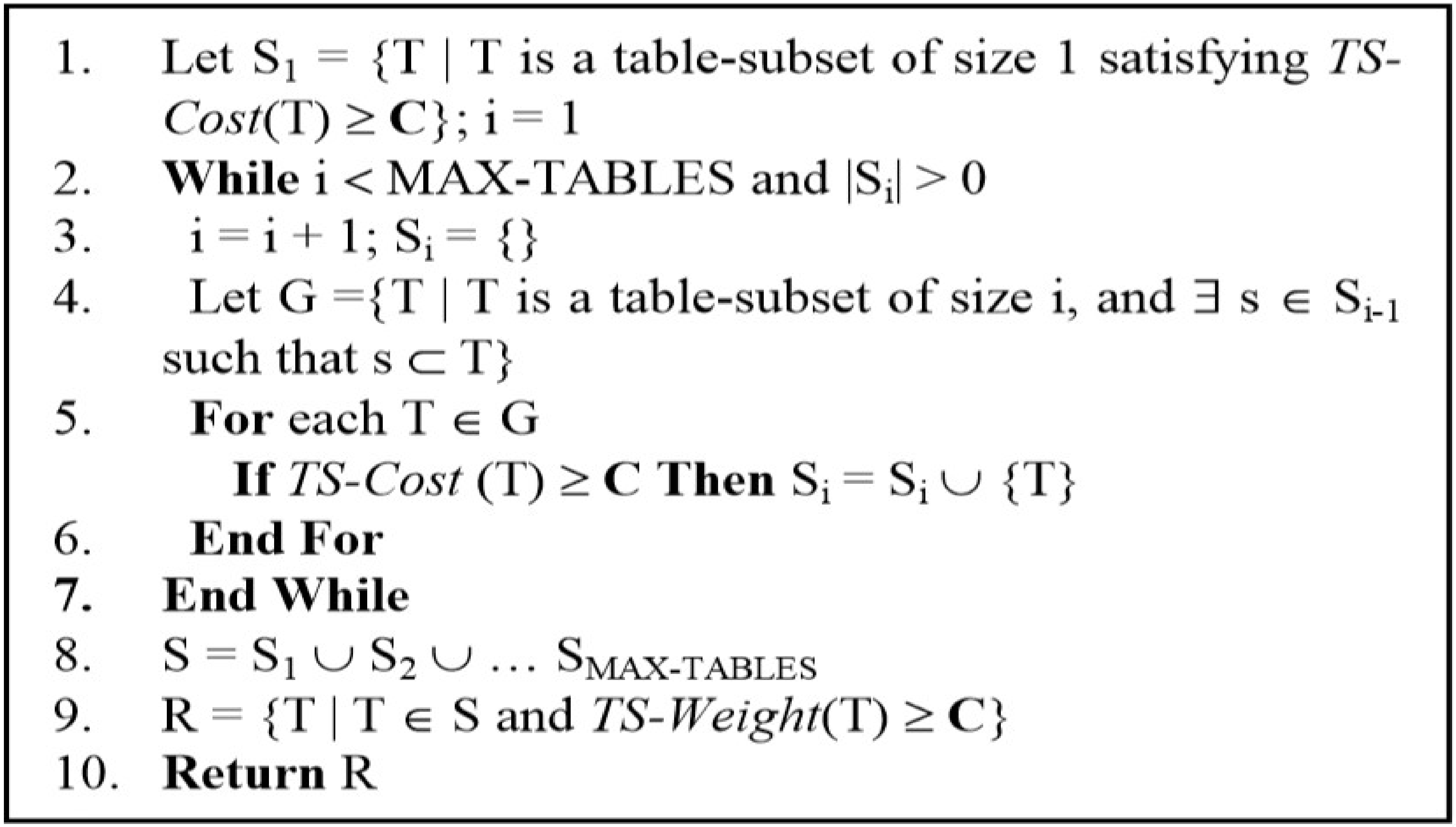
Candidate Materialized View Selection

<u>TS-Cost(T)</u> ***v/s*** ***<u>TS-Weight(T)</u>*** ***:***

1) TS-Cost(T) is simple , but not good measure of relative importance of a table-subset. While TS-Weight(T) can discriminate between table-subsets even if ther occur in exactly the same queries in the workload.

2) No efficient algorithm for finding all table subsets whose TS-Weight(T) exceeds a given threshold , while it is possible for TS-Cost(T) as it is monotonic.

Candidate Materialized View Selection



```
1.  Let $S_1$ = {T | T is a table-subset of size 1 satisfying $TS\text{-}Cost$(T) ≥ C}; i = 1
2.  While i < MAX-TABLES and $|S_i| > 0$
3.    i = i + 1; $S_i$ = {}
4.    Let G ={T | T is a table-subset of size i, and ∃ s ∈ $S_{i-1}$ such that s ⊂ T}
5.    For each T ∈ G
        If $TS\text{-}Cost$ (T) ≥ C Then $S_i = S_i \cup \{T\}$
6.    End For
7.  End While
8.  $S = S_1 \cup S_2 \cup \ldots S_{MAX\text{-}TABLES}$
9.  R = {T | T ∈ S and $TS\text{-}Weight$(T) ≥ C}
10. Return R
```

Figure . Algorithm for finding interesting table-subsets in the workload.

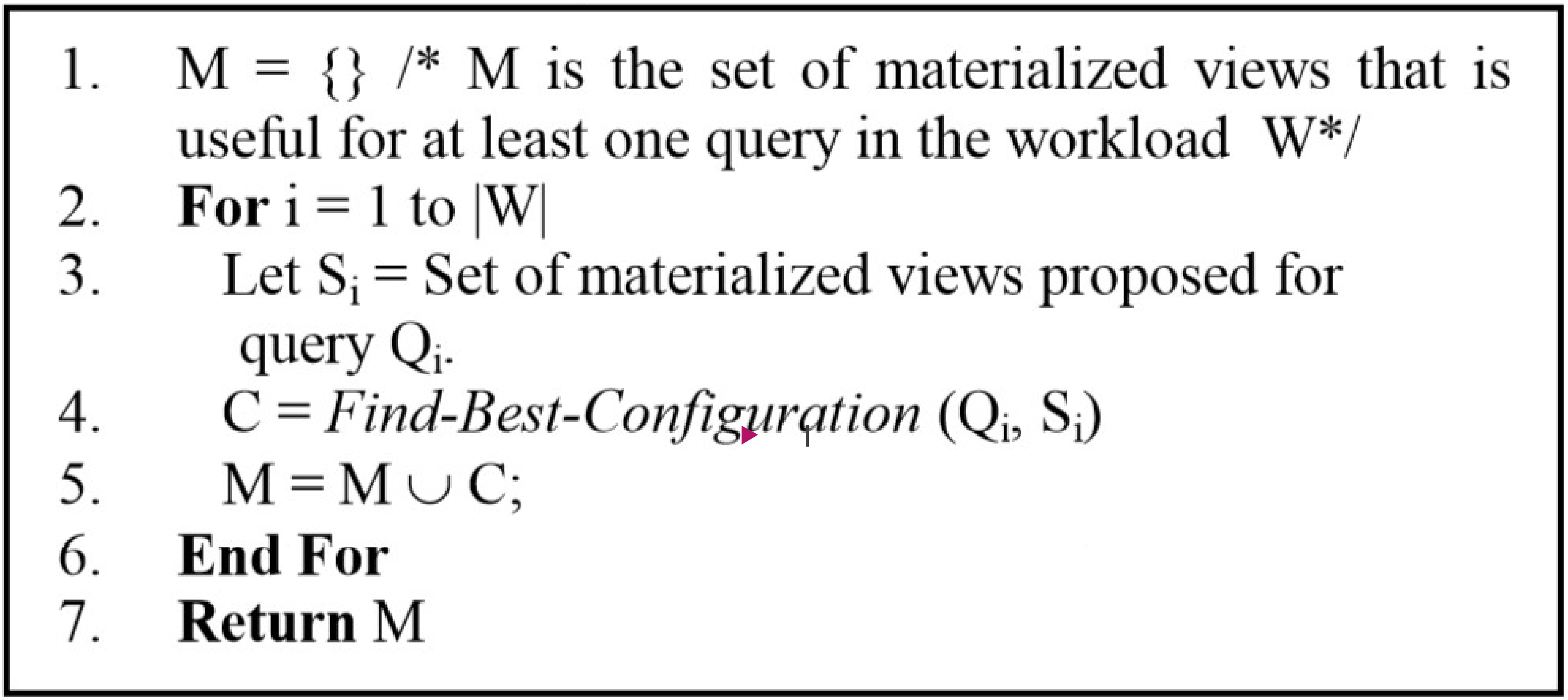
Candidate Materialized View Selection

2) Pruning Syntactically Relevant Materialized Views :

- The goal is to prevent syntactically relevant materialized views that are not used in answering any query.

- Intuitively if a materialized view is not part of the best solution for even a single query in the workload, then it is unlikely to be part of the best solution for the entire workload.

Candidate Materialized View Selection



```
1.  M = {} /* M is the set of materialized views that is
    useful for at least one query in the workload W*/
2.  For i = 1 to |W|
3.      Let $S_i$ = Set of materialized views proposed for
        query $Q_i$.
4.      C = Find-Best-Configuration ($Q_i$, $S_i$)
5.      M = M $\cup$ C;
6.  End For
7.  Return M
```

Figure . Cost-based pruning of syntactically relevant materialized views.

Candidate Materialized View Selection

Step (3) of the Algorithm :

Which syntactically relevant materialized views should be proposed for a query Qi?

- It is not sufficient to propose materialized views only on the table-subset that exactly matches the tables referenced in Qi
- Due to the pruning of table-subsets in previous step the table-subset that exactly matches the tables referenced in the query may not even be deemed interesting. In such cases, it again becomes important to consider smaller interesting table-subsets that occur in Qi.

Candidate Materialized View Selection

Step (3) of the Algorithm :

For each such interesting table-subset T, we propose :

(1) A “pure-join” materialized view on T containing join and selection conditions in Qi on tables in T.

(2) If Qi has grouping columns, then a materialized view similar to (1) but also containing GROUP BY columns and aggregate expression from Qi on tables in T.

Note:
For each materialized view proposed, also propose a set of clustered and non-clustered indexes on the materialized view.

Candidate Materialized View Selection

Step (4) of the Algorithm :

- **Find-Best-Configuration(Q, S)** for query Q and a set S of materialized views(with index on them) proposed for Q , returns the best configuration for Q from S.
- The best configuration for a query is that which the optimizer estimates as having the lowest cost for Q.
- Any suitable search method can be used in this function, e.g., the Greedy(m,k) algorithm described earlier.

Candidate Materialized View Selection

3) View Merging

Example :

Emp(ssn,name,sex,dno)   &   Works(ssn,pno,hours)

M_v1 : Select dno,count(ssn) from Emp,Works Where emp.ssn = Works.ssn And pno = 'p1' Group By dno

M_v1 : Select pno,count(ssn) from Emp,Works Where emp.ssn = Works.ssn And dno = 302 Group By pno

Merged View :
Select dno , pno , count(ssn) from Emp,Works Where emp.ssn = Works.ssn Group By dno,pno

Candidate Materialized View Selection

Emp table

SSN	NAME	SEX	Dno
E101	James	M	301
E102	Nick	M	301
E103	Ted	M	302
E104	Laura	F	302

Works table

SSN	Pno	Hrs
E101	P1	30
E101	P2	40
E102	P2	56
E103	P1	76
E103	P2	78
E104	P1	34

M_v1 :

Dno	Count
301	1
302	2

M_v2 :

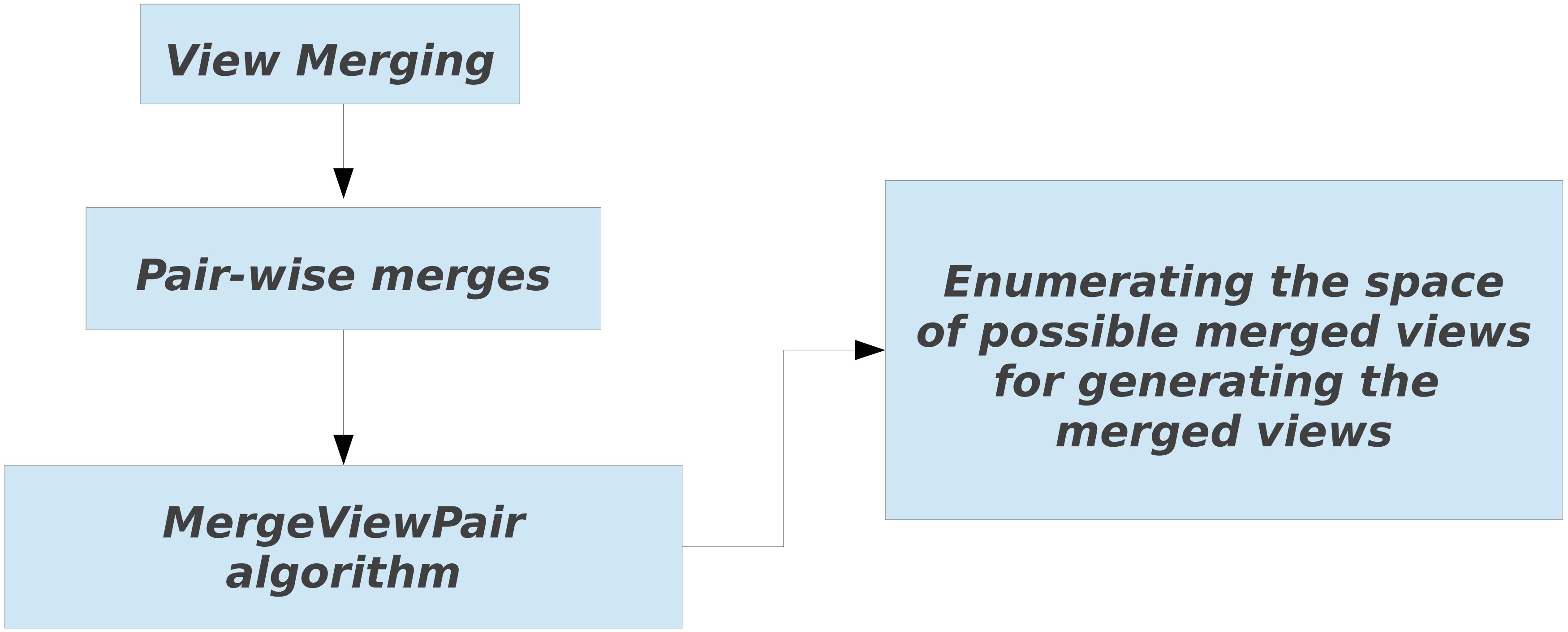
Pno	Count
P1	2
P2	1

Merged View :

Dno	Pno	Count
301	P1	1
301	P2	2
302	P1	2
302	P2	1

Candidate Materialized View Selection

Important steps in View Merging:


View Merging
Pair-wise merges
MergeViewPair algorithm
Enumerating the space of possible merged views for generating the merged views

Candidate Materialized View Selection

<u>Properties of View Merging :</u>

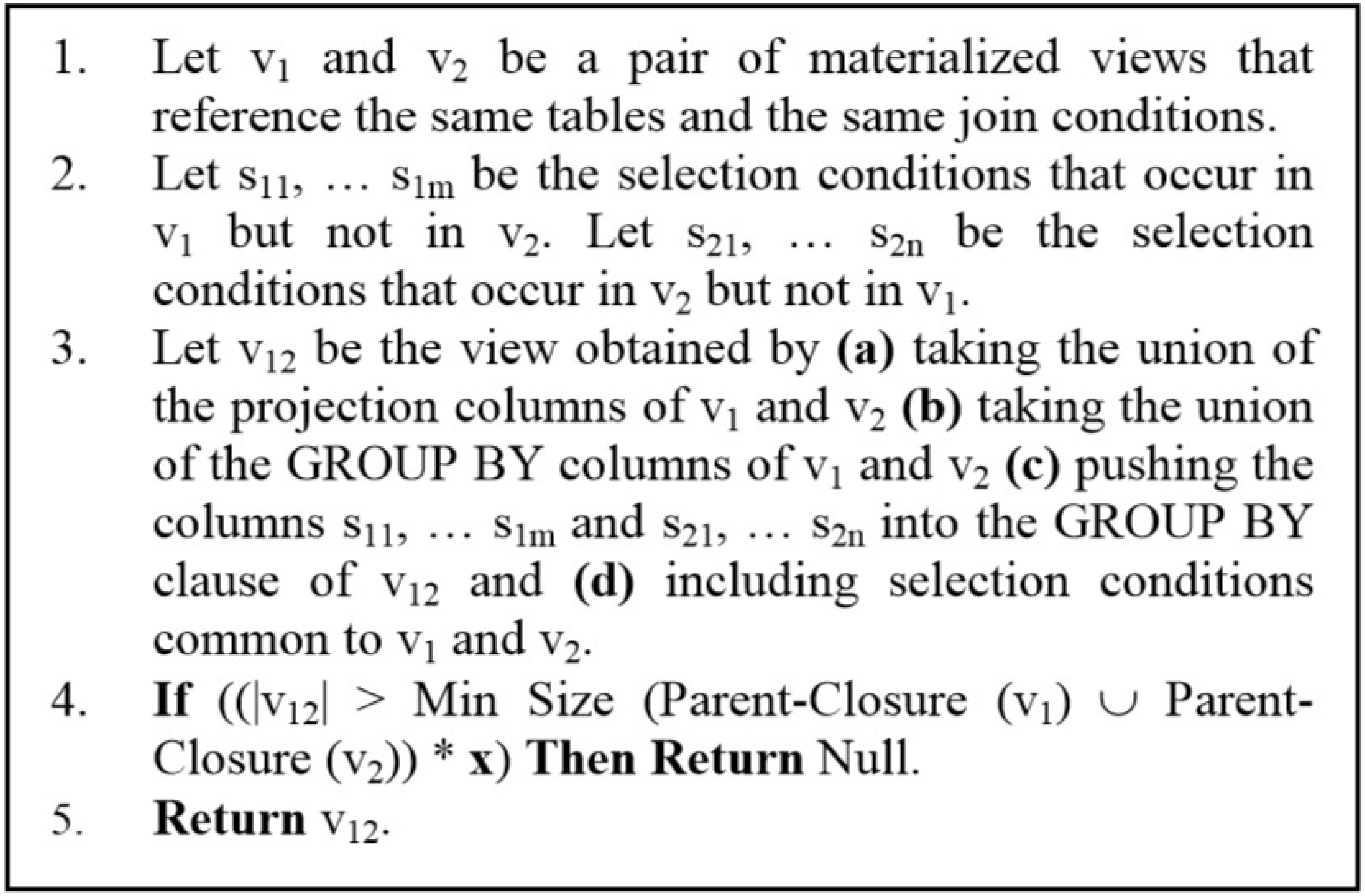
While merging two parent views to generate the merged view , following properties must hold true :

a) All queries that can be answered using either of the parent views should be answerable using the merged view.

b) The cost of answering these queries using the merged view should not be significantly higher than the cost of answering the queries using views in M.

Parent-Closure(v) as the set of views in M from which v is derived.
x = Size increase threshold (between 1-2)

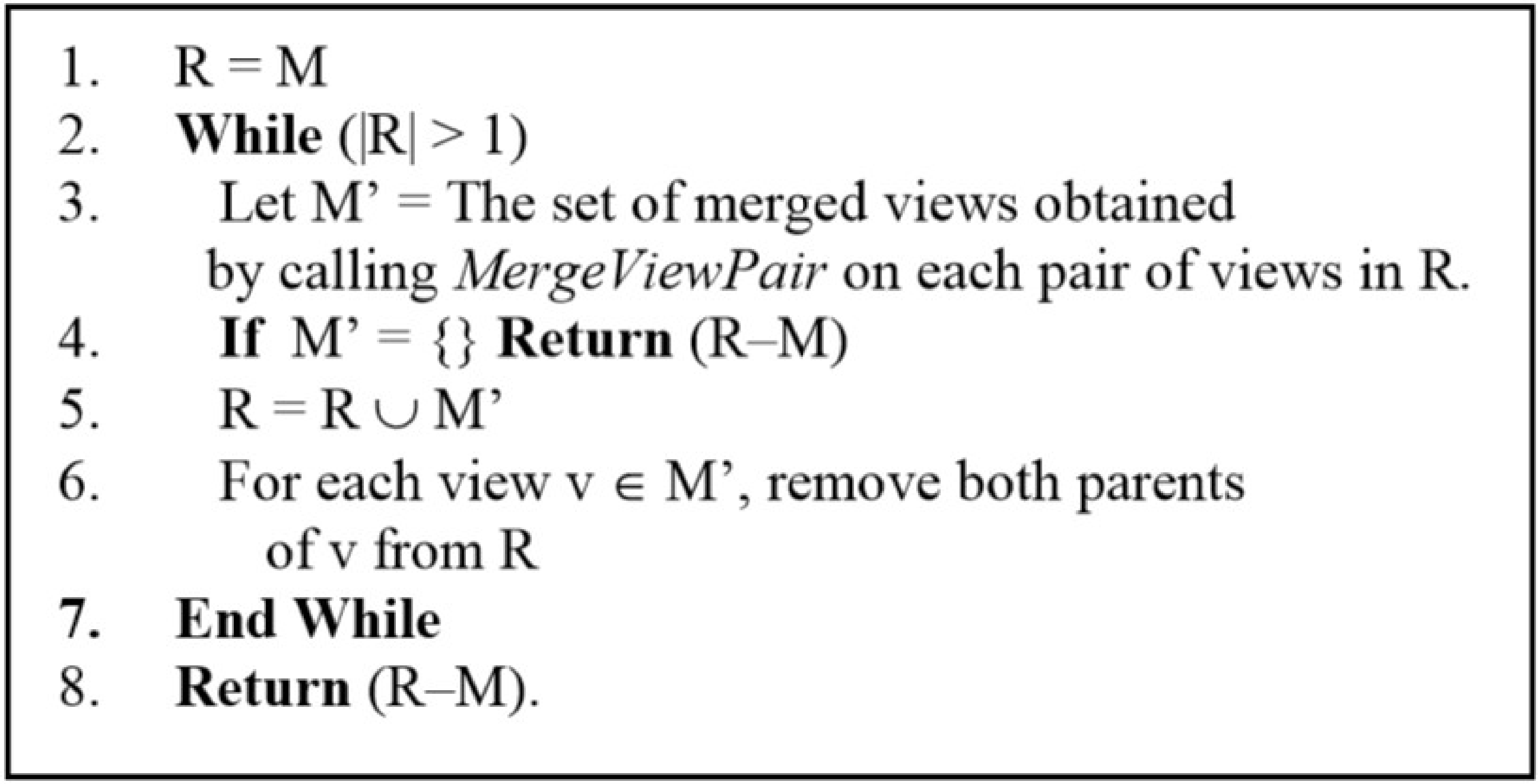
Candidate Materialized View Selection



1. Let v_1 and v_2 be a pair of materialized views that reference the same tables and the same join conditions.
2. Let s_{11}, … s_{1m} be the selection conditions that occur in v_1 but not in v_2. Let s_{21}, … s_{2n} be the selection conditions that occur in v_2 but not in v_1.
3. Let v_{12} be the view obtained by **(a)** taking the union of the projection columns of v_1 and v_2 **(b)** taking the union of the GROUP BY columns of v_1 and v_2 **(c)** pushing the columns s_{11}, … s_{1m} and s_{21}, … s_{2n} into the GROUP BY clause of v_{12} and **(d)** including selection conditions common to v_1 and v_2.
4. **If** $((|v_{12}| >$ Min Size (Parent-Closure $(v_1) \cup$ Parent-Closure $(v_2)) * \mathbf{x})$ **Then Return** Null.
5. **Return** v_{12}.

Figure MergeViewPair algorithm

Candidate Materialized View Selection



```
1.  $R = M$
2.  **While** ($|R| > 1$)
3.     Let $M'$ = The set of merged views obtained
      by calling *MergeViewPair* on each pair of views in R.
4.     **If** $M' = \{\}$ **Return** ($R – M$)
5.     $R = R \cup M'$
6.     For each view $v \in M'$, remove both parents
         of v from R
7.  **End While**
8.  **Return** ($R – M$).
```

Figure . Algorithm for generating a set of merged views from a given set of views M

Candidate Materialized View Selection

Properties of previous algorithm:

- The number of new merged views can exponential in the size of M in the worst case.
- New merged views can be merged further(implies more than 2 views can also get merged)
- The set of merged views returned by the algorithm does not depend on the exact sequence in which views are merged.

Alternate approaches for Index & Materialized View Selection

The approach we have used so far considers the *joint enumeration* of the space of candidate indexes and materialized views.The following are alternatives to this :

Approach 1 : MVFIRST

To pick *materialized views* first, and then select indexes for the workload given the materialized views picked earlier.

Approach 2 : INDFIRST

To pick *indexes* views first, and then select materialized views for the workload given the indexes picked earlier.

Alternate approaches for Index & Materialized View Selection

<u>Drawback of alternate approaches against Join Enumeration :</u>

1) Some **interactions between candidate indexes and candidate materialized views** that are eliminated in these approaches.

<u>For example:</u>
Consider a query Q for which indexes Ind_1, Ind_2 and materialized view M_v are candidates.
Assume that Ind_1 alone reduces the cost of Q by 25 units
Ind_2 reduces the cost by 30 units
Ind_1 and M_v together reduce the cost by 100 units.

Then, using INDFIRST, Ind_2 would eliminate Ind_1 when indexes are picked, and we would not be able to get the optimal recommendation {Ind_1,M_v}.

Alternate approaches for Index & Materialized View Selection

Drawback of alternate approaches against Join Enumeration :

2) A further drawback of MVFIRST is that selecting materialized views first are likely to **preclude selection of potentially useful candidate indexes** for the workload.

3) Another problem relevant to both INDFIRST and MVFIRST is **redundant recommendations** if the feature selected second is better for a query than the feature selected first.

EXPERIMENTS

- Algorithms presented in this paper are implemented on **Microsoft SQL Server 2000 release** and tested on **TPC-H 1 GB database.**

Hypotheses set:

- Selecting Candidate materialized views
  - Identifying interesting table-subsets substantially reduces materialized views without eliminating useful ones
  - View-merging algorithms significantly improves performance especially under storage constraints

EXPERIMENTS

Architectural issues :

1)Candidate selection module reduces runtime maintaining quality recommendations

2)Configuration enumeration module Greedy(m,k) gives results significantly faster than exhaustive one but still comparable.

3)JOINTSEL better than MVFIRST or INDFIRST

Identifying interesting table-subsets

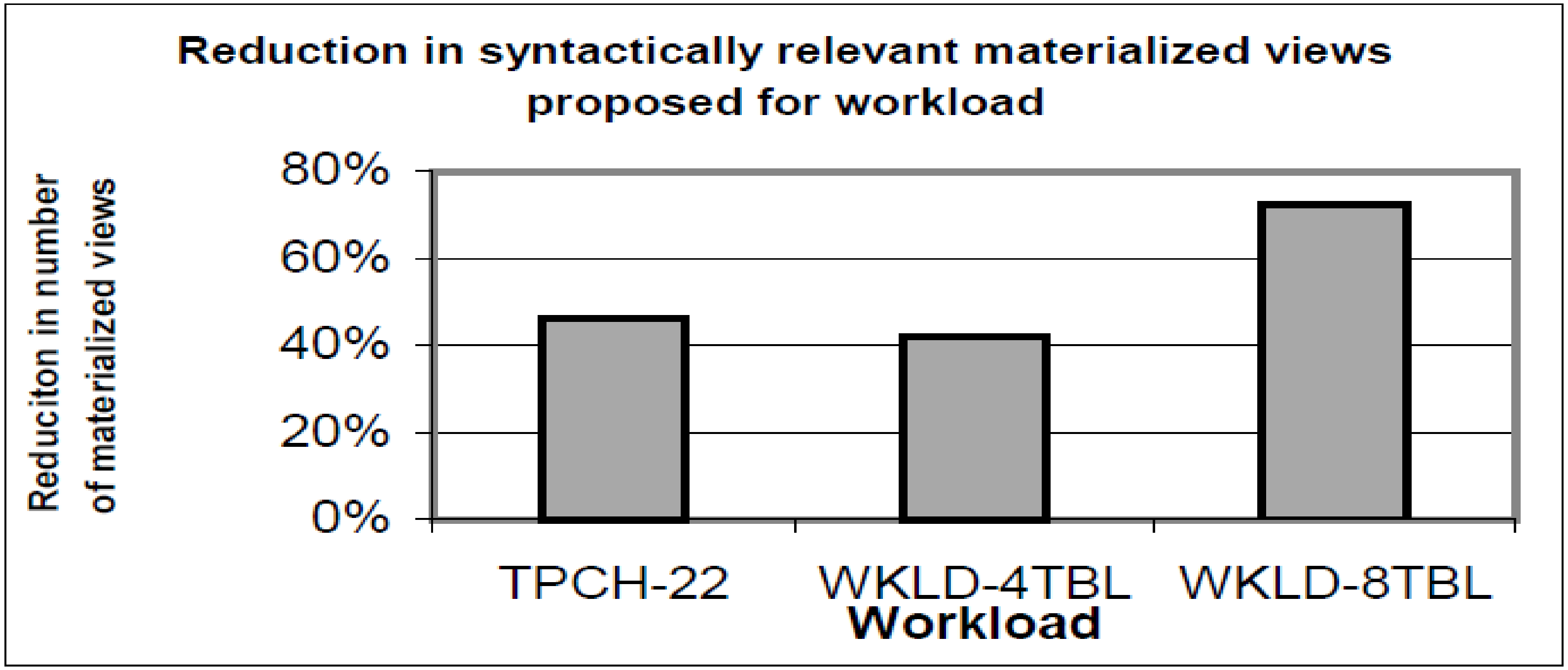
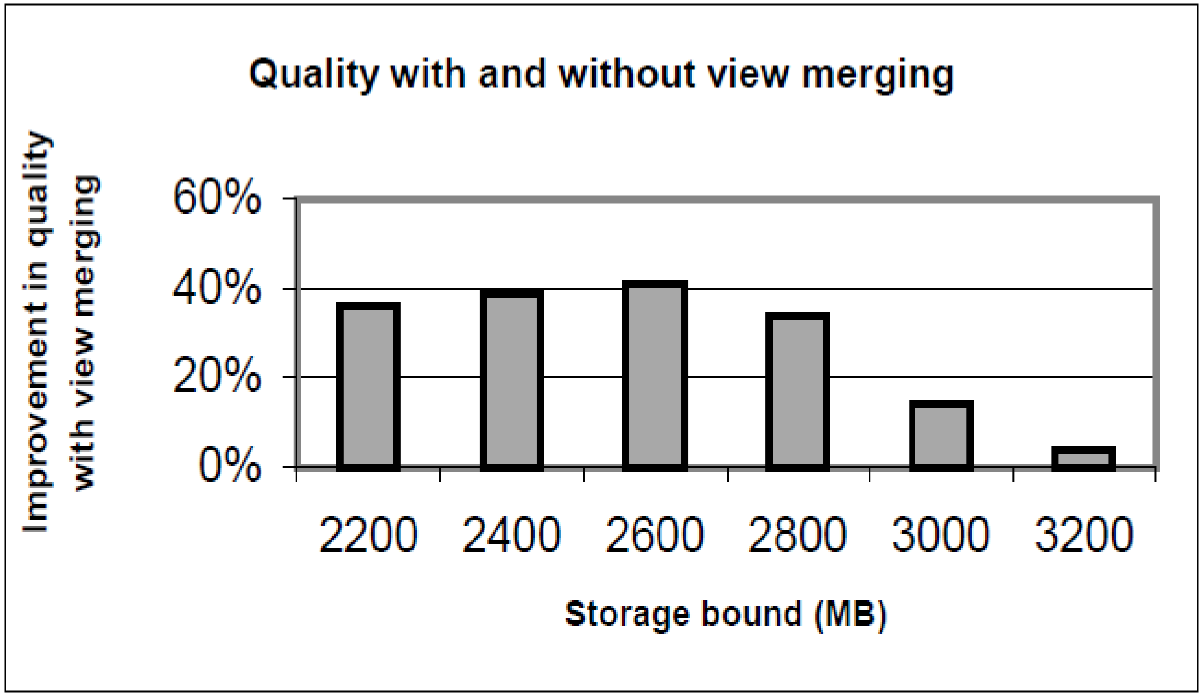


Figure . Reduction in syntactically relevant materialized views proposed compared to Exhaustive

Threshold **C** = 10%

Significant Pruning of space

View Merging



Add. Merged views:19%

Increase in runtime: 9%

Figure . Quality vs. storage bound with and without view merging.

- In low storage scenario the version with view merging significantly outperforms the version without view merging.

Candidate Selection

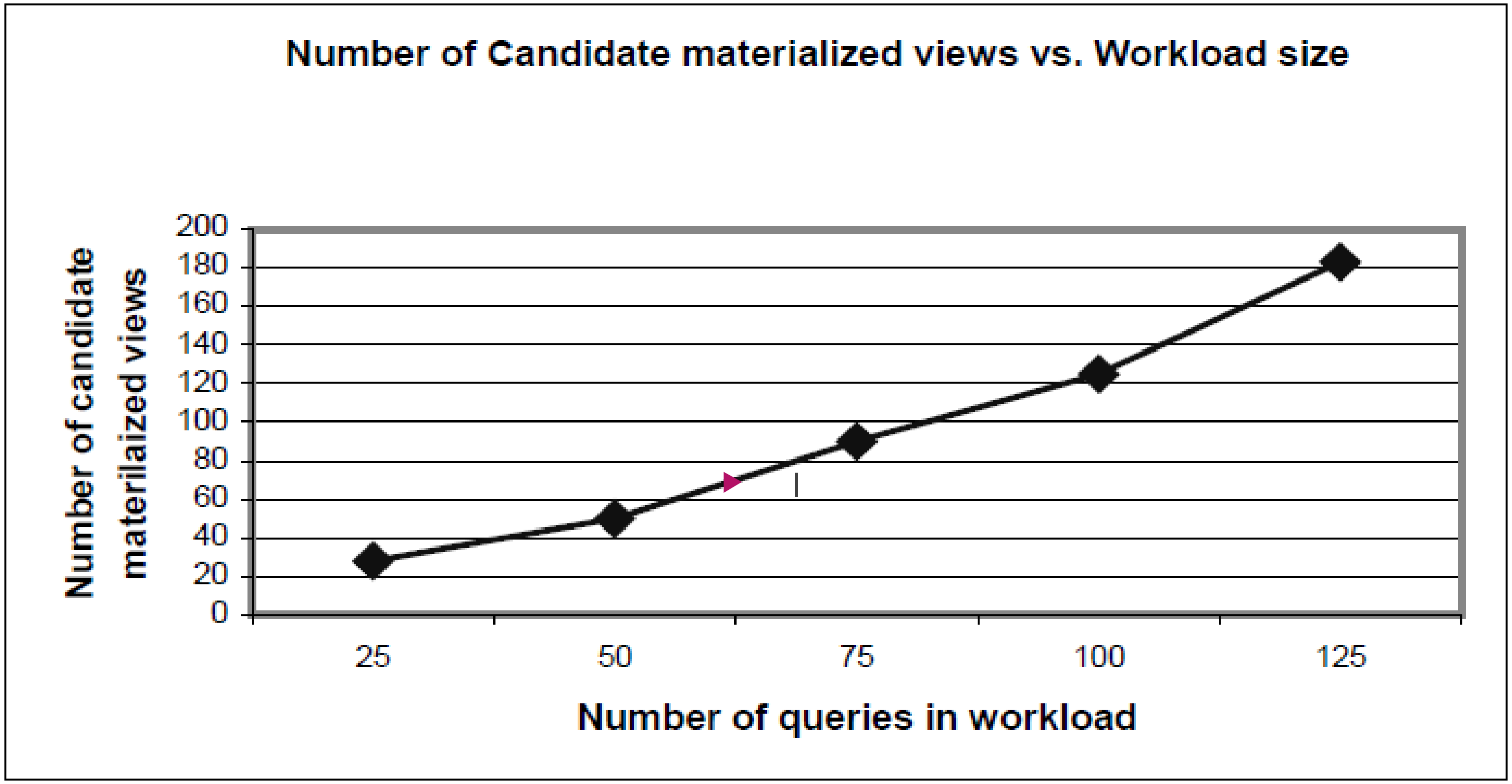
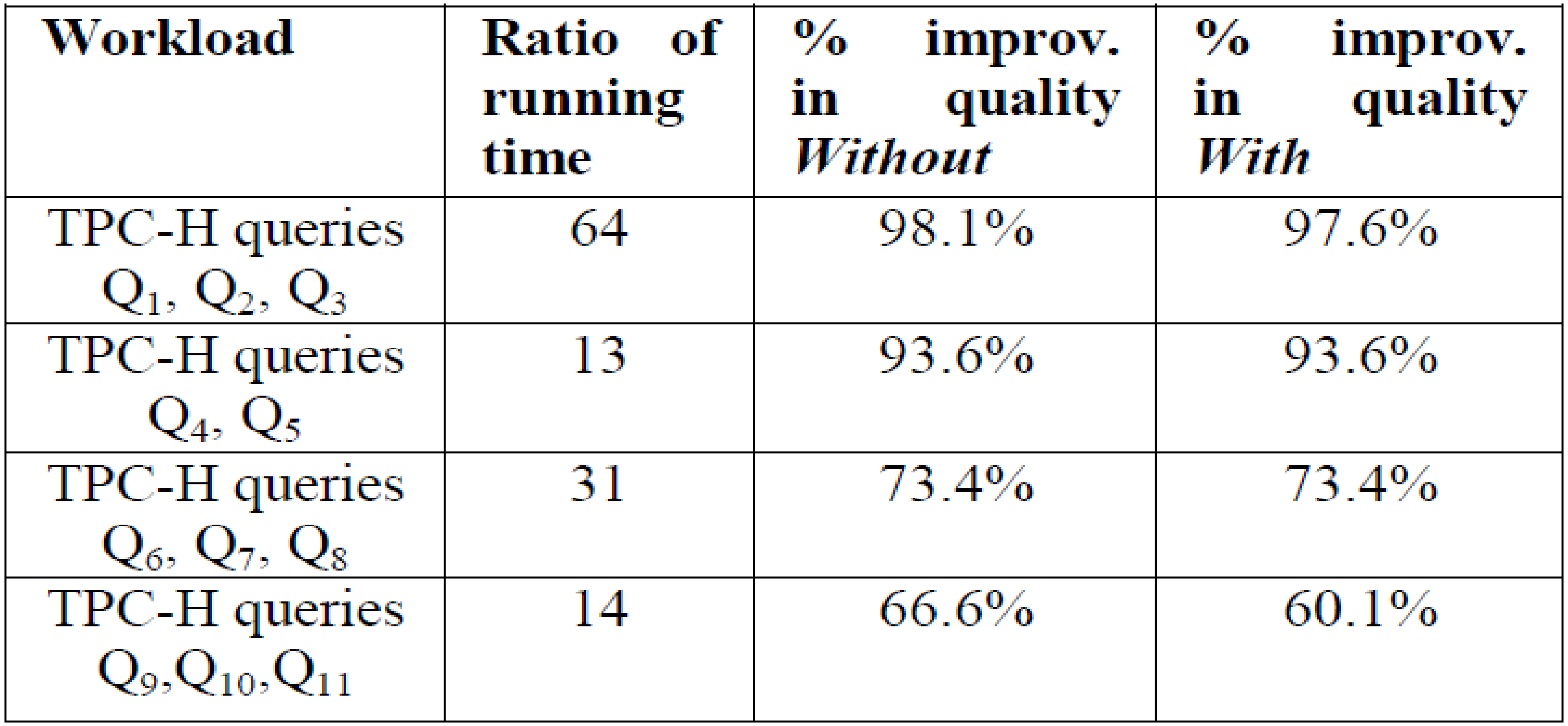


Figure . Scalability of candidate materialized view selection with workload size

No. of mat views grows linearly with workload size – hence scalable

Candidate Selection

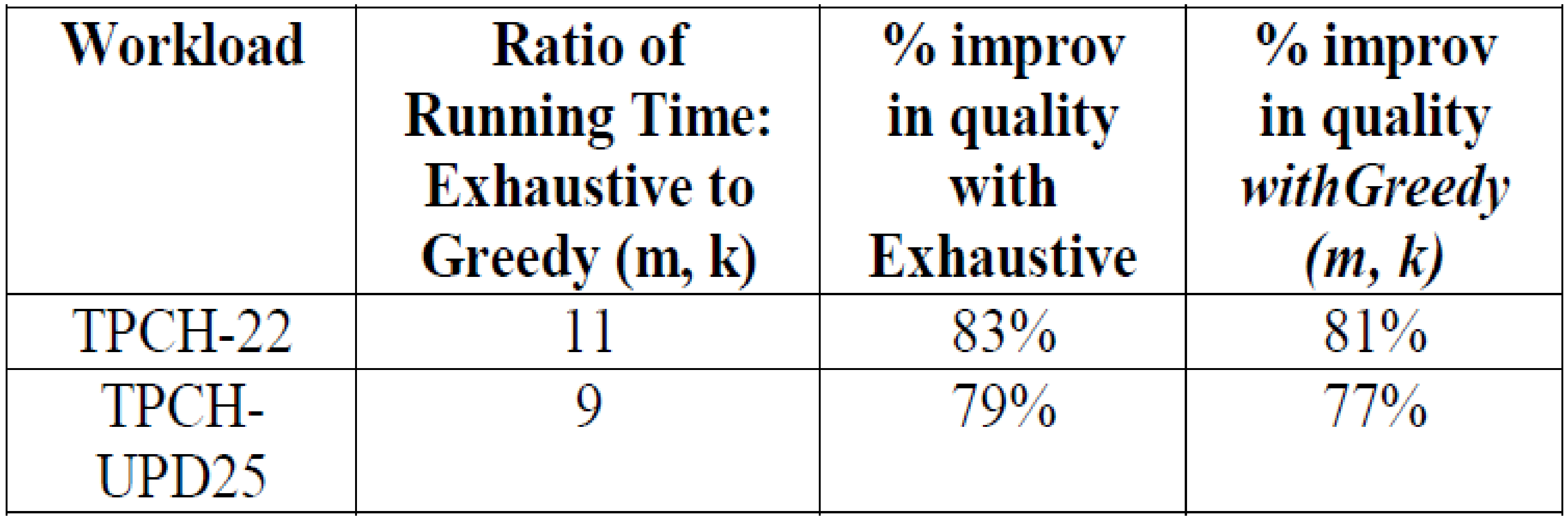


Workload	Ratio of running time	% improv. in quality *Without*	% improv. in quality *With*
TPC-H queries Q_1, Q_2, Q_3	64	98.1%	97.6%
TPC-H queries Q_4, Q_5	13	93.6%	93.6%
TPC-H queries Q_6, Q_7, Q_8	31	73.4%	73.4%
TPC-H queries Q_9,Q_{10},Q_{11}	14	66.6%	60.1%

Table . Comparison of schemes *with* and *without* the candidate selection module.

candidate selection not only reduces the running time by several orders of magnitude, but the drop in quality resulting from this pruning is very small

Configuration Enumeration

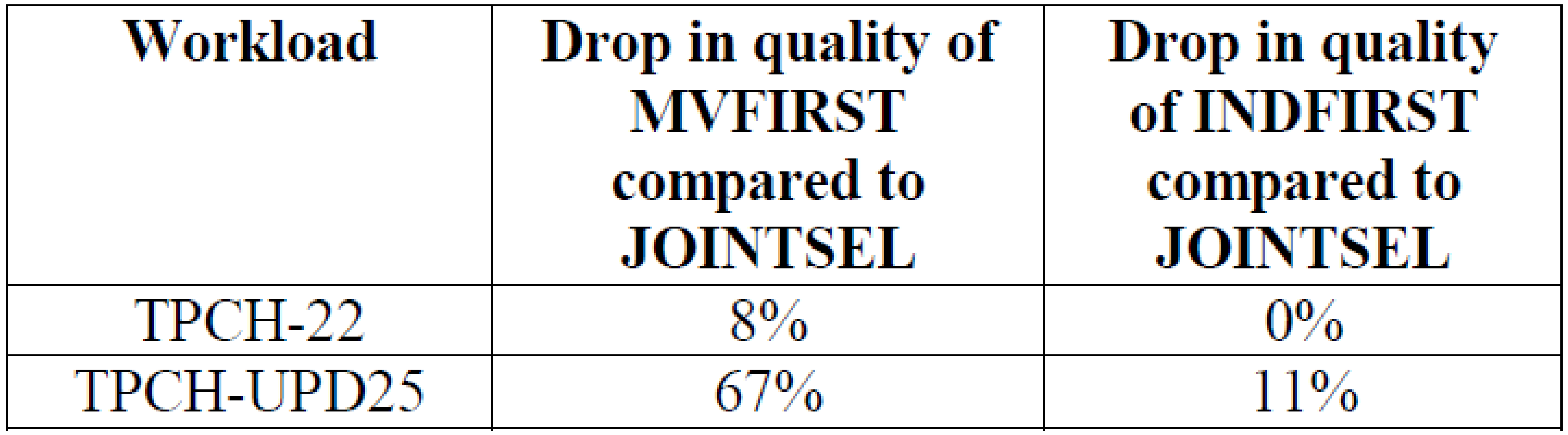


Workload	Ratio of Running Time: Exhaustive to Greedy (m, k)	% improv in quality with Exhaustive	% improv in quality *with Greedy (m, k)*
TPCH-22	11	83%	81%
TPCH-UPD25	9	79%	77%

Table . Comparison of Greedy(m,k) and exhaustive enumeration algorithms. $m=2$

Greedy(m,k) gives a solution comparable in quality to exhaustive enumeration. Yet, in time magnitudes faster

JoinSel vs MVFirst vs INDFirst



Workload	Drop in quality of MVFIRST compared to JOINTSEL	Drop in quality of INDFIRST compared to JOINTSEL
TPCH-22	8%	0%
TPCH-UPD25	67%	11%

Table . Comparison of alternative schemes without storage bound (i.e., storage = ∞)

- MVFIRST is significantly worse than the quality of JOINTSEL, particularly in the presence of updates in the workload.

- This confirms our intuition that picking materialized views first adversely affects the subsequent selection of indexes

REFERENCES

[1] Automated Selection of Materialized Views and Indexes for SQL Databases. Surajit Chaudhuri, Vivek Narasayya, and Sanjay Agrawal., VLDB 2000

[2] AutoAdmin "What-If" Index Analysis Utility. Chaudhuri S., Narasayya V., ACM SIGMOD 1998.

[3] An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. Chaudhuri S., Narasayya V., VLDB 1997.

Thank You!