Optimizing Queries Using Materialized Views

Paul Larson & Jonathan Goldstein
Microsoft Research

Materialized views

- Precomputed, stored result defined by a view expression
- Faster queries but slower updates
- Issues
  - View design
  - View exploitation
  - View maintenance
- View exploitation: determine whether and how a query (sub)expression can be computed from existing views

Query optimization

- Generate rewrites, estimate cost, choose lowest-cost alternative
- Generating rewrites in SQL Server
  - Apply local algebraic transformation rules to generate substitute expressions
  - Logical exploration followed by physical optimization
  - View matching is a logical rule that fires a view matching algorithm

Example view

```
create view v1 with schemabinding as
  select s_suppkey, s_name, s_nationkey,
         count_big(*) as cnt,
         sum(l_extendedprice*l_quantity) as grv
  from dbo.lineitem, dbo.supplier
  where p_partkey < 1000
    and l_suppkey = s_suppkey
  group by s_suppkey, s_name, s_nationkey


create unique clustered index v1_cidx on v1(s_suppkey)

create index v1_sidx on v1( grv, s_name)
```

Example query

```
Select n_nationkey, n_name,
       sum(l_extendedprice*l_quantity)
from lineitem, supplier, nation
where l_partkey between 100 and 500
  and l_suppkey = s_suppkey
  and s_nationkey = n_nationkey
group by n_nationkey, n_name
```

Execution time on 1GB TPC-R database: 99 sec (cold), 27 sec (hot)

Rewrite using v1

```
Select n_nationkey, n_name, smp
from (select s_nationkey,
             sum(l_extendedprice*l_quantity) as smp
      from lineitem, supplier
      where l_suppkey = s_suppkey
        and l_suppkey between 100 and 500
      group by s_nationkey) as sq1,
      nation
where s_nationkey = n_nationkey
```

```
Select n_nationkey, n_name, smp
from (select s_nationkey, sum(grv)as smp
      from v1
      where s_suppkey between 100 and 500
      group by s_nationkey ) as sq1,
      nation
where s_nationkey = n_nationkey
```

Execution time on 1GB TPCD-R database: less than 1 sec

Outline of the talk

- View matching algorithm
  - Algorithm overview
  - SPJ expressions, same tables referenced
  - Extra tables in the view
  - Grouping and aggregation
- Fast filtering of views
- Experimental results

Design objectives

- SPJG views and query expressions
- Single-view substitutes
- Fast algorithm
- Scale to hundreds, even thousands of views

Algorithm overview

1. Quickly dismiss most views that cannot be used
2. Detailed checking of remaining candidate views
3. Construct substitute expressions

When can a SPJ expression be computed from a view?

- View contains all required rows
- The required rows can be selected from the view
- All output expressions can be computed from the view output
- All output rows occur with the right duplication factor (not always required)

Column equivalence classes

- W = PE and PNE
  - PE = column equality predicates (R.Ci = S.Cj)
  - PNE = all other predicates
- Compute column equivalence classes using PE
- Columns in the same equivalence class interchangeable in PNE, output expressions, and grouping expressions
- Replace column references by references to equivalence classes

View contains all required rows?

- Assumption: query and view reference the same tables
- $Wq \Rightarrow Wv$ (containment)
- $Pq_1 \grave{U} Pq_2 \grave{U} \ldots \grave{U} Pq_m \Rightarrow Pv_1 \grave{U} Pv_2 \grave{U} \ldots \grave{U} Pv_n$
  - Convert predicates to CNF
  - Check that every *Pvi* matches some *Pqj*
  - Shallow or deep matching?
  - Too conservative – can do better

Exploiting column equivalences and range predicates

- *PEq Ù PRq ÙPUq Þ PEv Ù PRv ÙPuv*
  - *PE* = column equality predicates (R.Ci = S.Cj)
  - *PR* = range predicates (R.Ci < 50)
  - *PU* = residual (uninterpreted) predicates
- *PEq Þ PEv* (Equijoin subsumption)
- *PEq Ù PRq Þ PRv* (Range subsumption)
- *PEq ÙPUq Þ PUv* (Residual subsumption)

Equijoin subsumption test

- $PEq \supset PEv$
- Compute column equivalence classes for the query and the view
- Every view equivalence class must be a subset of some query equivalence class

Range subsumption test

- $PEq \land PRq \Rightarrow PRv$
- Compute range intervals for every column equivalence class (initially (-∞,+∞))
- Check that every query range interval is contained in a range interval of the corresponding view equivalence class

Residual subsumption test

- *PEq ÙPUq Þ PUv*
- Treat as uninterpreted predicates
  - Convert to CNF
  - Apply predicate matching algorithm, taking into account column equivalences
  - Currently using a shallow matching algorithm (convert to strings, compare strings)

Selecting rows from the view

- Compensating predicates
  - Unmatched column equality predicates from the query
  - Range predicates obtained when comparing query and view ranges
  - Unmatched residual predicates from the query
- All column references must map to an output column in the view (taking into account column equivalences)

Compute output expressions

- Map simple column references to view output columns (taking into account column equivalences)
- Complex scalar expressions
  - Check whether view outputs a matching expression
  - Otherwise, check whether all operand columns available in view output

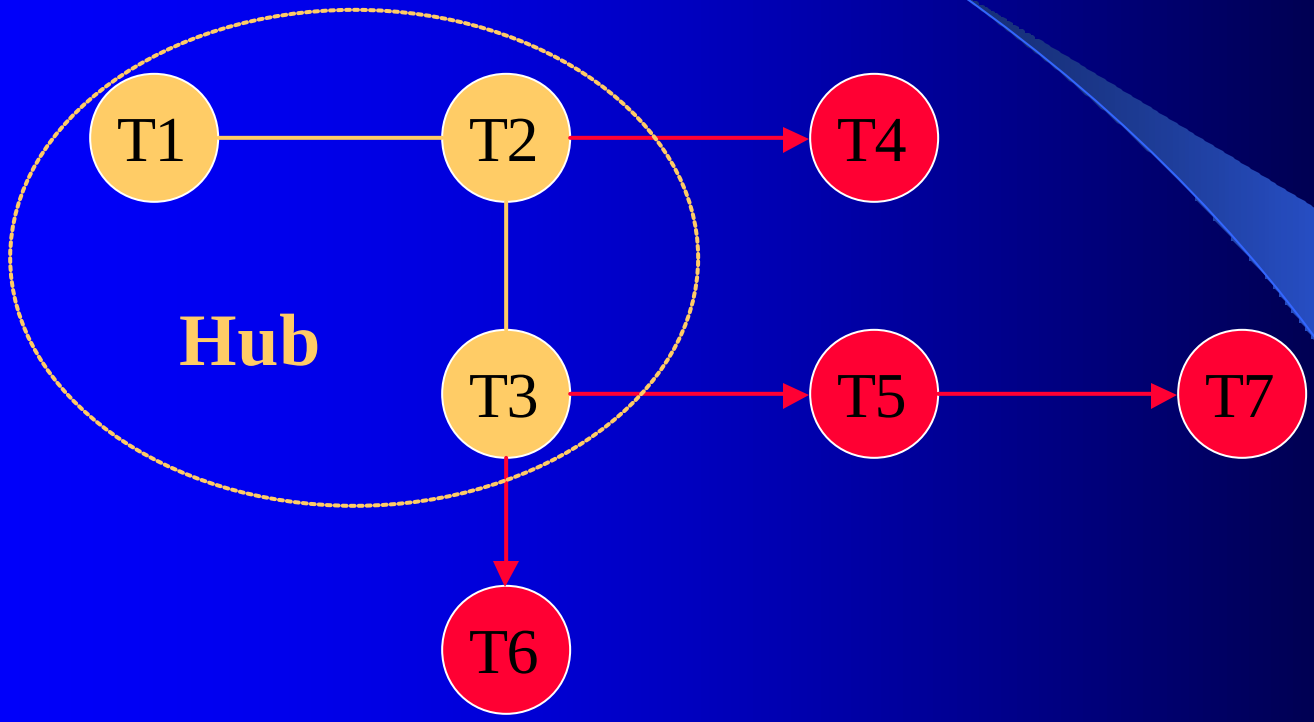
Correct duplication factor?

- Always true when query and view reference the same tables

Extra tables in the view

- View: R join S join T
- Query: R join S
- View usable if every row in (R join S) joins with exactly one row in T
- Row-extension join
  - Corresponds to a foreign key from S to T
  - Foreign key columns must be non-null
  - Referenced columns in T must be a unique key

View join graph and the hub



If view contains extra tables…

- Compute hub of view join graph
- Hub must be a subset of tables used in the query
- Logically add the extra tables to the query through row-extension joins
  - Just modify query's column equivalence classes
- Proceed normally because query and view now reference the same tables

Group-by queries and views

- SPJ part of view contains all required rows and with correct duplication factor
- Compensating predicates computable
- View less or equally aggregated
- Query grouping columns available if further grouping required
- Query output expressions computable

Further aggregation

- GB list of query must be a subset of GB list of view
- Query must use only partitionable aggregates
  - Count, sum, min, max

Example view and query

```
Create view SalesByCust with schemabinding as
  Select c_custkey, c_name, c_mktsegment, n_name,
         count_big(*) as cnt, sum(o_totalprice)as stp
  from orders, customer, nation
  where c_custkey between 1000 and 5000
    and o_custkey = c_custkey
    and c_nationkey = n_nationkey
  group by c_custkey, c_name, c_mktsegment, n_name


Select c_mktsegment, sum(o_totalprice)
from orders, customer
where c_custkey between 1000 and 2000
  and o_custkey = c_custkey
group by c_mktsegment
```

Rewritten example query

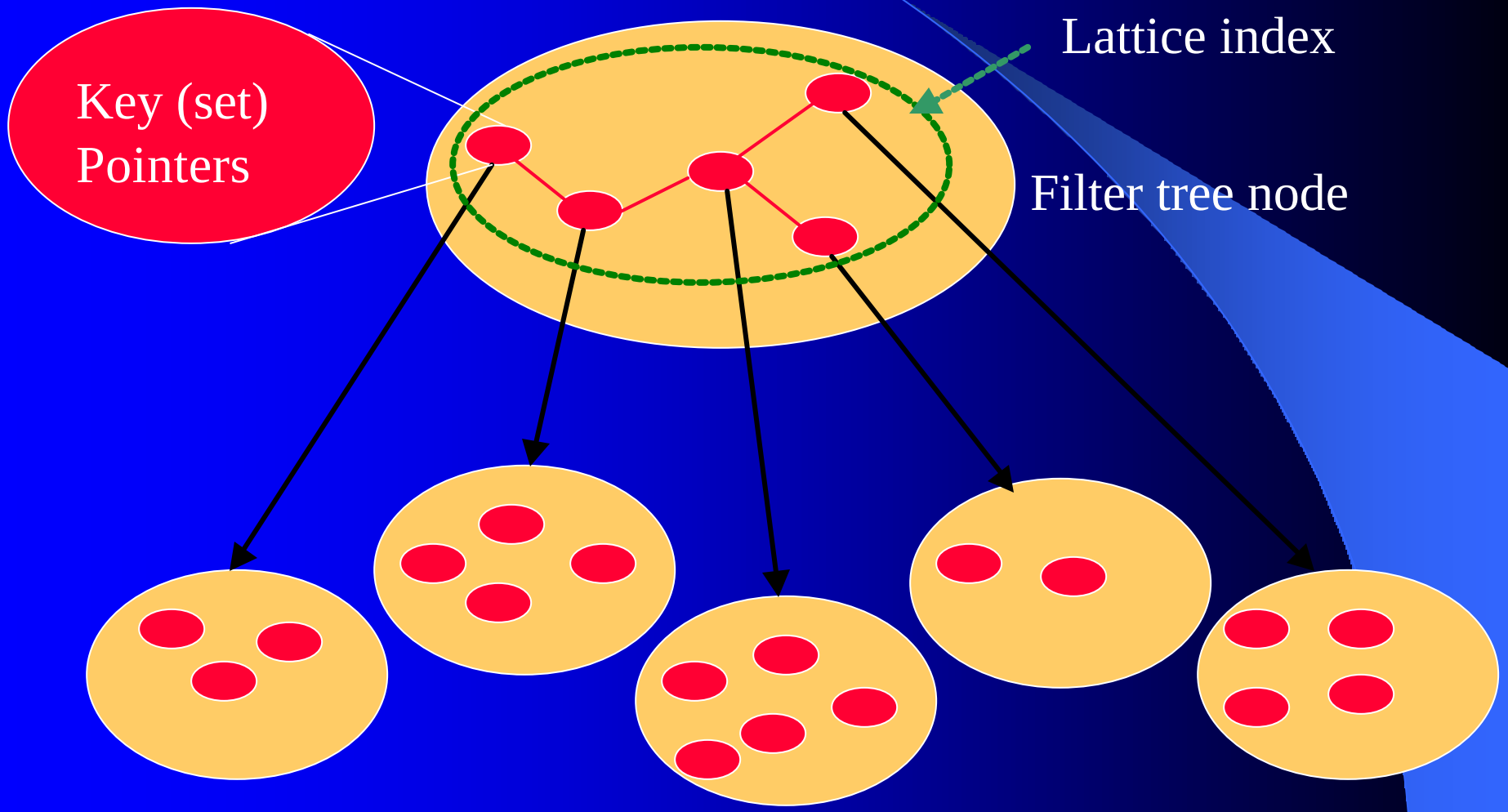
- View hub {orders} subset of {orders, customer}
- Compensating predicate (c_custkey <= 2000) computable
- Query GB-list subset of view GB-list
- Output expressions computable

```
Select c_mktsegment, sum(stp)
from SalesByCust
where c_custkey <= 2000
group by c_mktsegment
```

Fast filtering of views

- View descriptions in memory
- Too expensive to check all views each time
- Filter tree – index on view descriptions
- Tree subdivides views into smaller and smaller subsets
- Locating candidate views by traversing down the tree

Filter tree structure



Source table condition

- TSv = set of tables referenced in view
- TSq must be a subset of TSv
- Subdivide views based on set of tables referenced
- Filter tree node with key = table set

Hub condition

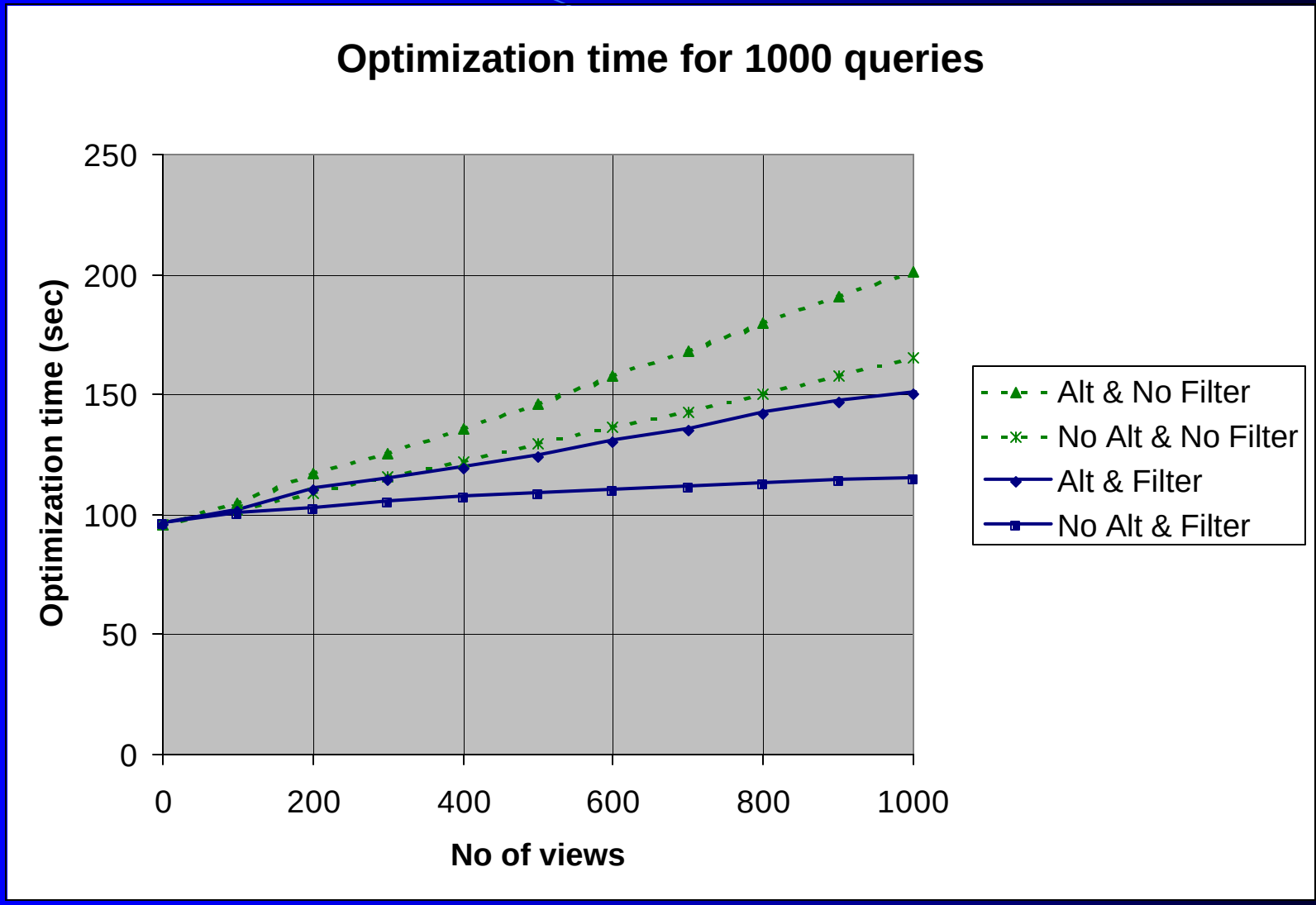
- View hub must be a subset of query's source tables
- Add another level to the tree
- One tree node for each subset of views
- Further subdivide each set of views based on view hubs

Other partitioning conditions

- Output columns
  - View's output columns must be a superset of query's output columns
- Grouping columns
  - View's GB list must be a subset of view's GB list
- Range constrained columns
  - View's RC columns must be a subset of query's RC columns
- Residual predicates
  - View's RP set must be a subset of query's RP set
- Must consider column equivalences everywhere

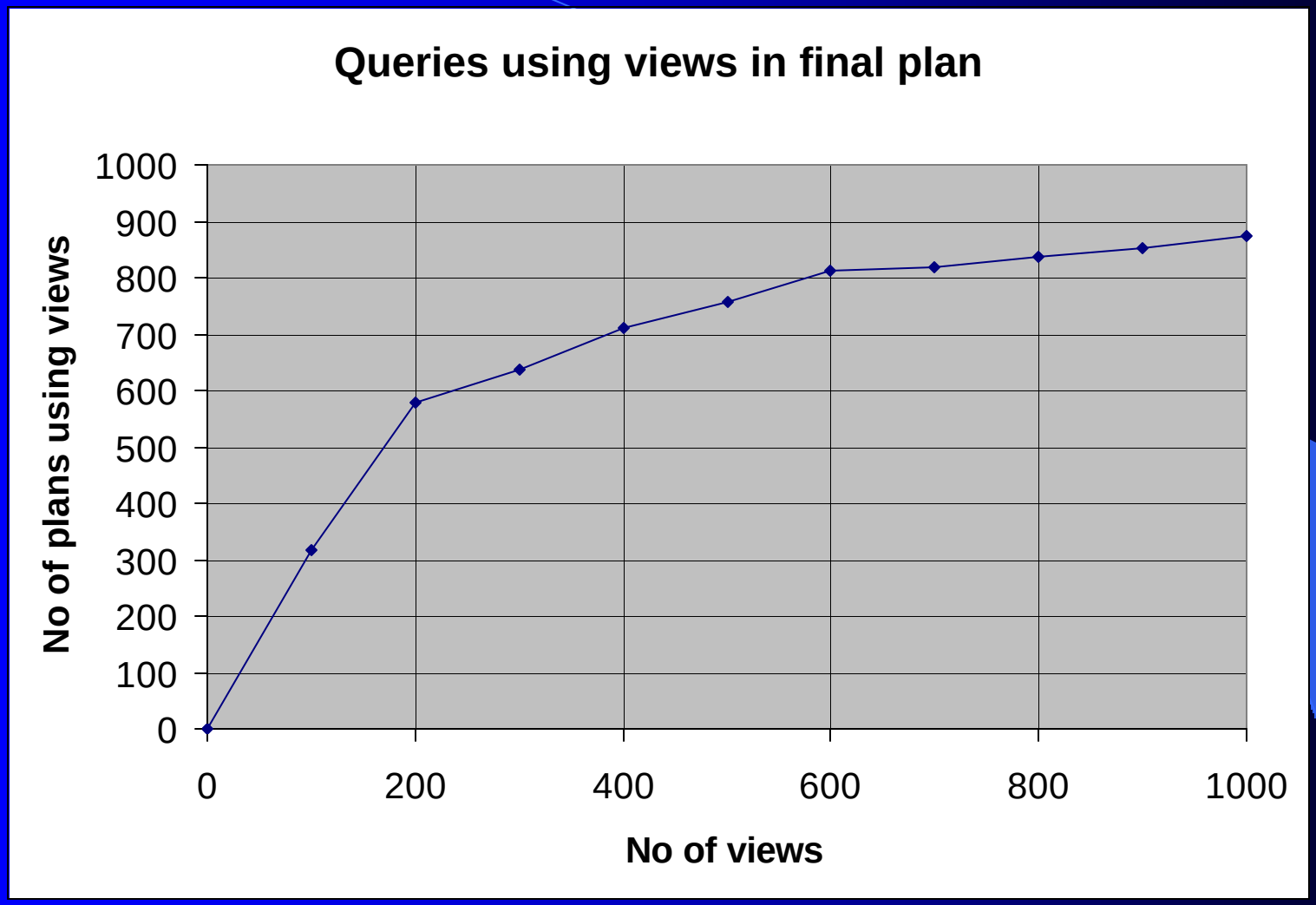
Experimental results

- Prototyped in SQL Server code base
- Database: TPC-H/R at 500MB
- Views: up to 1000 views
  - Randomly generated, 75% with grouping
- Queries: 1000 queries
  - Randomly generated, 75% with grouping
  - 2:40%, 3:20%, 4:17%, 5:13%, 6:8%, 7:2%
- Machine: 700 MHz Pentium, 128MB


Optimization time for 1000 queries
Optimization time (sec)
250
200
150
100
50
0
0
200
400
600
800
1000
No of views
Alt & No Filter
No Alt & No Filter
Alt & Filter
No Alt & Filter

Statistics

- About 17.8 invocations per query
- Filter tree was highly effective
- Average fraction of views in candidate set
  - 100 views 0.29%, 1000 views 0.36%
- 15-20% of candidates produced substitutes
- Avg. no of substitutes produced per query
  - 100 views 0.7, 1000 views 10.5


Queries using views in final plan
No of plans using views
1000
900
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1000
No of views

Conclusion

- Our view matching algorithm is
  - Flexible
    - column equivalences, range predicates, hubs
  - Fast and scalable
  - But limited to SPJG expressions and single-view substitutes

Possible extensions

- Additional substitutes
  - Back-joins to base tables
  - Union of views
- Additional view types
  - Self-joins
  - Grouping sets, cube and rollup
  - Outer joins
  - Union views