Optimizing Queries Using Materialized Views

Qiang Wang
CS848
Answering queries using views

Cost-based rewriting
(query optimization and physical data independence)
- System-R style
- Transformational approaches

Logical rewriting
(data integration and ontology integration)
- Rewriting algorithms
- Query answering algorithms
Transformational approach

- Transformation-based optimizer
- Related work
  - Oracle 8i DBMS
  - IBM DB2
  - Microsoft SQL Server
A practical and scalable solution

- Microsoft SQL Server 2000
- Limitations on the materialized view
  - A single-level SQL statement containing selections, inner joins and an optional group-by
  - It must reference base tables
  - Aggregation view must output all grouping columns and a count column
  - Aggregation functions are limited to SUM and COUNT
- A solution by Goldstein and Larson
  - An efficient view-matching algorithm
  - A novel index structure on view definitions
Problem definition

- View matching with single-view substitution: Given a relational expression in SPJG form, find all materialized views from which the expression can be computed and, for each view found, construct a substitute expression equivalent to the given expression.
Create view v1 as

\[
\text{select } p\_\text{partkey}, p\_\text{name}, p\_\text{retailprice}, \\
\quad \text{count_big(\text{*}) as cnt} \\
\quad \text{sum(l\_\text{extendedprice}*l\_\text{quantity}) as gross\_\text{revenue}} \\
\text{from } \text{dbo.lineitem, dbo.part} \\
\text{where } p\_\text{partkey} < 1000 \\
\quad \text{and } p\_\text{partkey} = l\_\text{partkey} \\
\quad \text{and } p\_\text{name} \text{ like '%steel%'} \\
\text{group by } p\_\text{partkey}, p\_\text{name}, p\_\text{retailprice} \\
\]

Create unique clustered index v1\_cidx on v1(p\_partkey)

Materialized View

\[
\text{select } p\_\text{partkey}, p\_\text{name}, \\
\quad \text{sum(l\_\text{extendedprice}*l\_\text{quantity})}, \\
\text{from } \text{dbo.lineitem, dbo.part} \\
\text{where } p\_\text{partkey} < 500 \\
\quad \text{and } p\_\text{partkey} = l\_\text{partkey} \\
\quad \text{and } p\_\text{name} \text{ like '%steel%'} \\
\text{group by } p\_\text{partkey}, p\_\text{retailprice} \\
\]

Query
substitutability requirements

1. The view contains all rows needed by the query expression
2. All required rows can be selected from the view
3. All output expressions can be computed from the output of the view
4. All output rows occur with the correct duplication factor
Do all required rows exist in the view (1)

- Check whether the source tables in the view are superset of the source tables in the query
- Check whether the query selection predicates can imply the view selection predicates

\[(PE_q \Rightarrow PE_v) \land (PE_q \land PR_q \Rightarrow PR_v) \land (PE_q \land PU_q \Rightarrow PU_v)\]

\[(PE_q \land PR_q \land PU_q \Rightarrow PE_v) \land (PE_q \land PR_q \land PU_q \Rightarrow PR_v) \land (PE_q \land PR_q \land PU_q \Rightarrow PU_v)\]

\[PE_q \land PR_q \land PU_q \Rightarrow PE_v \land PR_v \land PU_v\]
Equijoin subsumption test

- Test
  - Step1: equivalence class construction
    - \{p\_partkey,l\_partkey\} as Vc1 and Qc1
  - Step2: check whether every nontrivial view equivalence class is a subset of some query equivalence class

- Compensating equality constraints
- Missing cases
Range subsumption test

- Test
  - Associate (view/query) equivalence classes with lower and upper bound according to the range predicates; uninitialized bounds are assigned infinitum
    - Upperbound($Vc_1$) = 1000
    - Upperbound($Qc_1$) = 500
  - Check whether the view is more tightly constrained than the query on ranges
- Compensating range constraints
- Missing cases
Residual subsumption test

- Test (a shallow matching algorithm)
  - An expression is represented by a text string and a list of equivalence class referenced
    - Text string: like ‘%steel%’
    - Attached equivalence class: Vc1
  - Check whether every conjunct in the view residual predicate matches a conjunct in the query residual predicate
- Compensating residual constraints
- Missing cases
Can the required rows be selected (2)

- Check whether the view covers output columns for compensating equality, range, and residual predicates
Can output expressions be computed (3)

- Check whether all output expressions of the query can be computed from the view
  - “l_expendedprice * l_price”
Do rows occur with correct duplication factor(4)

- Cardinality-preserving joins (table extension joins)

- A stronger condition is employed: an equijoin between all columns in a non-null foreign key in T and a unique key in S
Test

- Given the query references tables $T_1, \ldots T_n$ and the view reference tables $T_1, \ldots T_n, T_{n+1}, \ldots T_{n+m}$
- Construct foreign key join graph
- Build cardinality-preserving relationships among tables
- Check whether the extra tables can be eliminated

Compensation (not needed)
Substitution with aggregation views

- **Special requirements**
  - The view contains no aggregation or is less aggregated than the query
    - Exact subsumption
    - Functionally determination
  - All columns required to perform further grouping are available in the view output
    - COUNT, SUM, AVG

- **Compensation**
  - Add extra grouping and aggregation expression computation
A clever index ---- filter tree

View definition list

A1

.....

An

G1

.....

Gm

G1' 

.....

Gm'

View definition list
Internal structure: lattice index
Partitioning conditions

- Hub condition
- Source table condition
- Output expression
- Output column
- Residual constraint
- Range constraint
- Grouping expression
- Grouping column
Experiment & extension work

- Experiment
  - Concerning 1000 views, candidate set is reduced to less than 0.4% of the views on average; Optimization time is 0.15 second per query where around 10 different substitutions are generated

- Extension
  - Constraint test refinement
  - Relaxation on single-table substitutes
Consideration

- How about a consolidated lattice structure over views?
  - Construction && maintenance
  - Introducing DL for reasoning, but how about the aggregation support?