Advanced Query Optimization
Advanced Topics in Databases

Andreas Meister
Otto-von-Guericke University Magdeburg
Summer Semester 2015
Agenda I

1. Motivation
2. Query Processing Phases
3. Logical Query Optimization
4. Physical Query Optimization
5. Outlook
Part I

Motivation
Database Optimization Problems

- Several optimization problems in databases:

  - Query optimization
  - Physical database design
  - Maintenance tasks

- Logical
  - Physical

- In this lecture: Focus on query optimization
Physical Database Design

• Optimization of data storage and access
• Provides options for query optimization
• Consists of several subproblems:

- Replication strategies
- Materialized view selection
- Index selection
- Partitioning schema
Maintenance tasks

- Guarantee the efficiency during the runtime
- Several other optimization problems:

- Data compression
- Update merging
- Buffer management
- Self-Tuning
Basic principles - Query Optimization

• Basis language
  • SQL
  • Relational calculus
  • Here: Relational algebra

• Optimization goals
  • Fast query processing
    ⇒ Consider as few page accesses/rows as possible during query processing
    ⇒ Consider as few page accesses/rows as possible during all operations
Example

```
SELECT CUSTOMER.CNo, Last_Name
FROM CUSTOMER, ORDER
WHERE CUSTOMER.CNo = ORDER.CNo
    AND Date = DATE '22-NOV-13'
```

- Table CUSTOMER: 100 rows; one page: 5 rows
- Table ORDER: 10,000 rows; one page: 10 rows
- 50 orders per day
- 50 rows of (CNo, Last_Name) on one page
- 3 rows of R(CUSTOMER) × R(ORDER) on one page
- Puffer for every relation size: 1, no clamping sets
Direct evaluation

1. \( R_1 := R(\text{CUSTOMER}) \times R(\text{ORDER}) \)
   Page accesses:
   - \( r : (100/5 \cdot 10.000/10) = 20.000 \)
   - \( w : (100 \cdot 10.000)/3 = 333.000 \) (ca.)

2. \( R_2 := \sigma_{\text{SEL}}(R_1) \)
   - \( r : 333.000 \) (ca.)
   - \( w : 50/3 = 17 \) (ca.)

3. \( R_{\text{erg}} := \pi_{\text{PROJ}}(R_2) \)
   - \( r : 17 \)
   - \( w : 1 \)

Overall ca. 687.000 page accesses and ca. 333.000 pages for intermediate storage
Optimized evaluation

1. \( R_1 := \sigma_{\text{Date} = '22.11.13'}(R(\text{ORDER})) \)
   - \( r : 10.000/10 = 1.000 \)
   - \( w : 50/10 = 5 \)

2. \( R_2 := R(\text{CUSTOMER}) \bowtie_{\text{CNo} = \text{CNo}} R_1 \)
   - \( r : 100/5 \cdot 5 = 100 \)
   - \( w : 50/3 = 17 \)

3. \( R_{\text{erg}} := \pi_{\text{PROJ}}(R_2) \)
   - \( r : 17 \)
   - \( w : 1 \)

Ca. 1.140 page accesses (improved by factor about 600)
Evaluation with index usage

Indices $I(\text{ORDER(Date)})$ and $I(\text{CUSTOMER(CNo)})$

1. $R_1 := \sigma_{\text{Date}=22.11.13} (R(\text{ORDER}))$ over $I(\text{ORDER(Date)})$
   - $r$: minimal 5, maximal 50; $w = 50/10 = 5$

2. $R_2 := \text{sort } R_1$ over $\text{CNo}$
   - $r + w = 5 \cdot \log 5 = 15$ (ca.)

3. $R_3 := \text{CUSTOMER} \bowtie_{\text{CNo}=\text{CNo}} R_2$
   - $r = 100/5 + 5 = 25; w = 50/3 = 17$

4. $R_{\text{erg}} := \pi_{\text{PROJ}} (R_3)$
   - $r = 17; w = 1$

Maximal ca. 130 and minimal ca. 85 page accesses
Comparison of different variants

<table>
<thead>
<tr>
<th>Execution variants</th>
<th>Read and write operations</th>
<th>pages for intermediate results</th>
</tr>
</thead>
<tbody>
<tr>
<td>direct evaluation</td>
<td>ca. 687.000</td>
<td>ca. 333.000</td>
</tr>
<tr>
<td>optimized evaluation</td>
<td>ca. 1.140</td>
<td>17</td>
</tr>
<tr>
<td>evaluation with indices</td>
<td>min. 85, max. 130</td>
<td>17, 17</td>
</tr>
<tr>
<td>with pipelining</td>
<td>51 to 96</td>
<td>5 (plus sorting)</td>
</tr>
</tbody>
</table>
Part II

Phases
Phases of Query processing

Translation & View resolving

Translalorization & Simplification

Optimization

Access plan

Execution

Code-Generation

Plan parameterization

Translation time

Runtime
Phases of query processing

1. **Translation and view resolving**
   - Simplification of arithmetic expressions
   - Resolve subqueries
   - Insertion of view definitions

2. **Logical or algebraic optimization**
   - Transformation of queries independent of storage information
3. **Physical or internal Optimization**
   - Consider storage information (Indices, Clusters)
   - Choose algorithms
   - Consider several alternative internal plans

4. **Cost-based selection**
   - Use statistical information (table size, selectivity of attributes) for the selection of one internal plan

5. **Plan parameterization**
   - In pre-compiled queries (e.g., n embedded-SQL): Replace placeholders with values

6. **Code-Generation**
   - Transformation of the access plan in executable code
Phases of Query processing /3

- Representation of queries during the processing
  - Algebraic expression $\rightarrow$ **Operator tree**
    - Operators as nodes
    - Edges represent data flow
  - Later $\rightarrow$ **Query execution plan (QEP)**
    - Concrete algorithms as operator nodes
Part III

Logical Optimization
Logical Optimization

• Heuristic methods
  • E.g., algebraic optimization (Rewriting)
  • For relational algebra + grouping, . . .

• Exact methods
  • Reducing number of joins
  • For particular relational algebra queries
Algebraic Optimization

- Replacement of terms of the relational algebra based on algebraic equivalence
- Equivalence used as replacement rules
- Heuristic methods:
  - Move operations to reduce intermediate results
  - Identify redundancies
Removing of redundant operations

- Necessary in queries with views

\[ R(\text{ACT\_PRODUCTS}) = R(\text{PRODUCTS}) \Join \pi_{\text{ProdNo}, \text{Label}}(\ldots \sigma_{\text{Date}=\text{current\_date}}(R(\text{ORDER}))) \]

- Query on view:

\[ \pi_{\text{Label}, \text{Price}}(R(\text{PRODUCT}) \Join R(\text{ACT\_PRODUCTS})) \]

- View expansion:

\[ \pi_{\text{Label}, \text{Price}}(R(\text{PRODUCTS}) \Join R(\text{PRODUCTS}) \Join \pi_{\text{ProdNo}, \text{Label}}(\ldots \sigma_{\text{DATE}=\text{current\_date}}(R(\text{ORDER})))) \]

- Rules: idempotence

\[ R = R \Join R \quad \text{meaning} \Join \text{is idempotent} \]
Moving of selections

\[ \sigma_{\text{Price}>100}( R(\text{ORDER}) \bowtie R(\text{PRODUCT}) ) \]

more efficient:

\[ R(\text{ORDER}) \bowtie (\sigma_{\text{Price}>100}( R(\text{PRODUCT}) )) \]

Rules:

\[ \sigma \text{ and } \bowtie \text{ commute} \]

Only if the attributes of the selections predicate supporting this
Join order

- Statistical information of the catalog necessary

\[(R(CUSTOMER) \bowtie R(PRODUCT)) \bowtie R(ORDER)\]

- First join: Cartesian product, therefore:

\[R(CUSTOMER) \bowtie (R(PRODUCT) \bowtie R(ORDER))\]

- Rules:

  \[\bowtie\text{ is associative and commutative}\]

- No unique preferable application of this rule (therefore, internal optimization, cost-based)
Algebraic rules - Example

• **CommJoin**: Operator $\bowtie$ is commutative:

$$r_1 \bowtie r_2 \iff r_2 \bowtie r_1$$

• **AssocJoin**: Operator $\bowtie$ is associative:

$$(r_1 \bowtie r_2) \bowtie r_3 \iff r_1 \bowtie (r_2 \bowtie r_3)$$

• **ProjProj**: For operator $\pi$ the outer dominates the inner parameter in the combination:

$$\pi_X(\pi_Y(r_1)) \iff \pi_X(r_1)$$

• Plenty more rules available [Saake et al., 2012]
Further rules

• **Idempotences**
  
  **IdemUnion:** \[ r_1 \cup r_1 \iff r_1 \]
  
  **IdemIntersect:** \[ r_1 \cap r_1 \iff r_1 \]
  
  **IdemJoin:** \[ r_1 \Join r_1 \iff r_1 \]
  
  **IdemDiff:** \[ r_1 - r_1 \iff \emptyset \]

• **Association with empty relation:**

  **EmptyUnion:** \[ r_1 \cup \emptyset \iff r_1 \]
  
  **EmptyIntersect:** \[ r_1 \cap \emptyset \iff \emptyset \]
  
  **EmptyJoin:** \[ r_1 \Join \emptyset \iff \emptyset \]
  
  **EmptyDiffRight:** \[ r_1 - \emptyset \iff r_1 \]
  
  **EmptyDiffLeft:** \[ \emptyset - r_1 \iff \emptyset \]

• for \( \Join, \cup \) and \( \cap \): **commutative** and **associative law**
Algebraic optimization: Algorithm

- Simple optimization algorithm
  1. Resolve complex selection predicate, if applicable, resolving of \( \neg \) and \( \lor \)
  2. Remove redundant operators
  3. Pushing down selections as near as possible to the leaf
  4. Resolve cross joins
  5. Pushing projections in leaf direction
- Single steps will be executed in the stated order until no replacement can be performed
Algebraic optimization: example

\[ \pi \text{OrderNo, CNo} \left( \sigma \text{Date<'}18.2.14' \land \text{Label='}Arabica Black' \right) \]

\[
\left( \pi \text{OrderNo, CNo, Date, Label} \left( R(\text{PRODUCT}) \bowtie R(\text{ORDER}) \bowtie R(\text{CUSTOMER}) \right) \right)
\]
Unoptimized query plan

\[ \pi \text{OrderNo, CNo} \]
\[ \sigma \text{Date > '18.02.14' \land Label = 'Arabica Black} \]
\[ \pi \text{OrderNo, CNo, ProdNo, Label, Date} \]

\[ \bowtie \]
\[ \text{CUSTOMER} \]
\[ \text{PRODUCTS} \]
\[ \text{ORDER} \]
Query plan

- Remove redundant operations

\[
\begin{align*}
\pi \text{OrderNo, CNo} \\
\sigma \text{ Date > '18.02.14' } \land \text{ Label = 'Arabica Black'}
\end{align*}
\]
Query plan /2

- Move of the selections

```
π ᵇ ᵇ σ ᵇ

ORDERPRODUCT
CUSTOMER

Label = 'Arabica Black'
Date > '18.02.14'

OrderNo, CNo

PRODUCT
ORDER

Customer
```
Query plan /3

- with additional projections

$\pi_{\text{OrderNo, CNo}}$

$\text{CUSTOMER}$

$\sigma_{\text{Label = 'Arabica Black'}}$

$\text{PRODUCT}$

$\sigma_{\text{Date > '18.02.14'}}$

$\pi_{\text{OrderNo, CNo}}$

$\pi_{\text{CNo}}$
Algebraic optimization: example from CoGaDB

```
SELECT  d_year, s_nation, p_category,
       SUM(lo_revenue - lo_supplycost) AS profit
FROM    lineorder, dates, customer, supplier, part
WHERE   lo_orderdate = d_datekey AND
        lo_custkey = c_custkey AND
        lo_suppkey = s_suppkey AND
        lo_partkey = p_partkey AND
        c_region = 'AMERICA' AND s_region = 'AMERICA'
        AND (d_year = 1997 OR d_year = 1998) AND
        (p_mfgr = 'MFGR# 1' OR p_mfgr = 'MFGR# 2')
Group by d_year, s_nation, p_category
Order by d_year, s_nation, p_category
```
Algebraic optimization: example from CoGaDB

- Transform sql query in relational algebra/ operator tree
Algebraic optimization: example from CoGaDB

- Resolve complex selection predicate
Algebraic optimization: example from CoGaDB

- Push down selections

```
\gamma \sum(LO_REVENUE - LO_SUPPLYCOST) \cdot D_YEAR, S_NATION, P_CATEGORY
\delta D_YEAR, S_NATION, P_CATEGORY
\sigma LO_PARTKEY = P_PARTKEY
\delta P_MFGR = MFGR1 OR P_MFGR = MFGR2
\delta S_REGION = AMERICA
\sigma LO_SUPPKEY = S_SUPPKEY
\delta S_REGION = AMERICA
\sigma LO_CUSTKEY = C_CUSTKEY
\delta C_REGION = AMERICA
\sigma LO_LISTPRICE = D_DATEKEY
\delta D_YEAR = 1997 OR D_YEAR = 1998
```
Algebraic optimization: example from CoGaDB

- Resolve cross joins

\[
\sum(\text{LO_REVENUE} - \text{LO_SUPPLYCOST}) ; \text{D_YEAR}, \text{S_NATION}, \text{P_CATEGORY}
\]

\[
\sigma \quad \text{P_MFGR} = \text{MFGR}1 \text{ OR P_MFGR} = \text{MFGR}2
\]

\[
\sigma \quad \text{S_REGION} = \text{AMERICA}
\]

\[
\sigma \quad \text{C_REGION} = \text{AMERICA}
\]

\[
\sigma \quad \text{D_YEAR} = 1997 \text{ OR D_YEAR} = 1998
\]

\[
\sigma \quad \text{D_YEAR}, \text{S_NATION}, \text{P_CATEGORY}
\]

\[
\sigma \quad \text{LO_PARTKEY} = \text{P_PARTKEY}
\]

\[
\sigma \quad \text{LO_SUPPKEY} = \text{S_SUPPKEY}
\]

\[
\sigma \quad \text{LO_CUSTKEY} = \text{C_CUSTKEY}
\]

\[
\sigma \quad \text{LO_ORDERDATE} = \text{D_DATEKEY}
\]

\[
\text{Part} \quad \supseteq \quad \text{SUPPLIER} \quad \supseteq \quad \text{CUSTOMER} \quad \supseteq \quad \text{LINEORDER} \quad \supseteq \quad \text{DATES}
\]
Possibilities for logical query optimization on modern hardware

- Application of different rules in parallel not possible
  - Deterministic application of rules
  - One optimal result

- Evaluate several plans in parallel

- Determine redundant operations faster
Part IV

Physical Optimization
From logical to physical optimization

- What to do next?

```
\text{ORDER} \bowtie \text{PRODUCT} /
\text{CUSTOMER}
\pi \text{OrderNo, CNo}
\pi \text{OrderNo, CNo}
\pi \text{CNo}
\sigma \text{Label = 'Arabica Black'}
\sigma \text{Date > '18.02.14'}
\text{PRODUCT}
\text{ORDER}
```

Andreas Meister
Advanced Query Optimization
Last Change: June 1, 2015
38/113
From logical to physical optimization

• What to do next?

\[
\begin{align*}
\pi^\text{OrderNo, CNo} &\quad \pi^\text{CNo} \\
\sigma^\text{Label = 'Arabica Black'} &\quad \sigma^\text{Date > '18.02.14'} \\
\text{PRODUCT} &\quad \text{ORDER} \\
\end{align*}
\]
Algorithm selection

• Selection
  - $\sigma^\text{REL}_\varphi$: Selection through relation-scan
  - $\sigma^\text{IND}_\varphi$: Selection over index

• Join
  - $\bowtie^\text{LOOP}$: Nested-loop join
  - $\bowtie^\text{MERGE}$: Merge join (Requirement: Input relations are sorted on joint attribute(s))
  - $\bowtie^\text{HASH}$: Hash-Join
Algorithm selection /2

- Projection
  - $\pi_{\text{AttList}}^{\text{REL/with}}$: Projection through relation-scan; with duplicate removal
  - $\pi_{\text{AttList}}^{\text{REL/without}}$: Projection through relation-scan; without duplicate removal
  - $\pi_{\text{AttList}}^{\text{SORT/with}}$: Projection through scan over relation sorted by AttList with duplicate removal
  - $\pi_{\text{AttList}}^{\text{SORT/without}}$: Projection through scan over relation sorted by AttList without duplicate removal
Algorithm selection /3

- Grouping
  - $\gamma^{\text{SORT}}_{F;\text{AttList}}$ Grouping on $\text{AttList}$ and application of the aggregation $F$ through sorting
  - $\gamma^{\text{HASH}}_{F;\text{AttList}}$ Grouping on $\text{AttList}$ and application of the aggregation $F$ through hashing
Algorithm selection /4

- Index access on predicate $A \Theta a$
  
  $$\sigma_{A \Theta a}^{\text{IND}}(I(R(A))) \rightarrow \text{list}(\text{tid})$$

- Special case
  
  $$\sigma_{\text{true}}^{\text{IND}}(I(R(A)))$$

- Combination of projection
  
  $$\pi_{\text{AttList}}^{\text{IND/with}} \text{ or } \pi_{\text{AttList}}^{\text{IND/without}}$$

- Without access of base relation
  
  $$\pi_{A}^{\text{IND}}(I(R(A)))$$
New operators

- For TID-lists: ""Realization""-operator $\rho$

  $$\rho(\langle\text{TID-List for } R\text{-Tuple}\rangle, R(R))$$

- Set operators $\cup$, $\cap$ and $-$ on TID-Lists

- **Sorting** of tuples $\omega$

  $$\omega_{\text{AttList}}(\langle\text{Tupel-order}\rangle)$$
Example for execution plans

```sql
SELECT * 
FROM ORDER 
WHERE ProdNo = 42 AND 
  (SName = 'Kaffeebude' OR 
   SName = 'CoffeeShop') AND 
Amount < 10
```
Example for execution plans /2
Algebraic optimization: example from CoGaDB
Cost-based Optimization

query

span search space

equivalent plans

search strategy

"best" plan

transformation rules

cost model
Spanning of the search space

- Search space: Set of all equivalent query plans
- Spanning through **transformation rules** (algebraic rules)
- Focus: **join trees**

  - for \( n \) relations:
    - Alone \( n! \) different left- or right-deep trees (permutation)
    - \( n = 10: 3.628.800 \) left-deep and \( 17.643.225.600 \) trees overall!

- Therefore: Pruning of the search space through
  - Heuristics (algebraic optimization)
  - Predefined tree "form"
Join trees

- Linear order of joins: left- or right-deep trees
- Bushy trees
Spanning of the search space /2

- Number of bushy join tree between \( n \) relations

\[
S(1) = 1
\]

\[
S(n) = \sum_{i=1}^{n-1} S(i)S(n - i)
\]

- For number \( i \) from 1 \( \ldots \) \( n - 1 \) leafs of on part tree: \( S(i) \) different tree forms
- For the remaining \( n - i \) leafs: also \( S(n - i) \) forms
- For every form: assign \( n \) relations in \( n! \) variants as leaf
- Overall: for \( n \) relations \( S(n) \cdot n! \) variants
Spanning of the search space /3

\[ R(\text{ORDER}) \bowtie R(\text{PRODUCT}) \bowtie R(\text{CUSTOMER}) \bowtie R(\text{SUPPLIER}) \]
Search strategies

• Walking through the search space

• Selection of the minimum-cost plan

• Basis: cost model

• Search strategy determines
  • which plans will be considered (complete / partial search)
  • in which order the alternatives evaluated

• Variants: deterministic, randomized
Cost model: Components

- **Cost function**: To estimate the execution cost of operations or queries

- **Statistics**: Over size of the relations (cardinality, tuple size), value ranges and distributions

- **Formulas**: To calculate the size of (intermediate) results based on the statistics
Cost function

- Cost types:
  - I/O-costs: Caused by reading and writing of blocks from or to the external storage
  - CPU-costs: for internal calculations, comparison etc.,
  - Communication costs: in case of distributed database systems

- Usually:
  \[
  \text{cost} = \text{cost}_{IO} + W \cdot \text{cost}_{CPU}
  \]

- Factor $W$ to calibrate regarding hardware
Cost formulas

• Idea:
  • Estimate total expense through the cardinality of intermediate results
  • Cardinality over **Selectivity** of the operators

• Selectivity $sel$:
\[
  sel = \frac{\text{Expected size of the result}}{\text{Cardinality of the input relation}}
\]

• Assumption: equal distribution, independence of the attribute
Cost formulas: Selection - Example

\[ |\sigma_F(R(R))| = sel(F, R) \cdot |r| \]

- Estimation (for interpolateable, arithmetic values):

\[ sel(A = v, R) = \frac{1}{val_{A,r}} \]
\[ sel(A < v, R) = \frac{v - A_{min}}{A_{max} - A_{min}} \]
\[ sel(A > v, R) = \frac{A_{max} - v}{A_{max} - A_{min}} \]
\[ sel(A \text{ between } v_1 \text{ and } v_2, R) = \frac{v_2 - v_1}{A_{max} - A_{min}} \]

- Further cost formulas presented in [Saake et al., 2012]
Improvement of the estimation

- **Histograms**: Approximation of the real distribution

- **Parametrised functions**: Function parameter for approximation of the data distribution (e.g., normal or Zipf distribution)

- **Sample**: Estimate selectivity based on a random selected sample
Histograms: Principle

- Attribute values
- Frequency
- Real distribution
- Approximate distribution

Andreas Meister

Advanced Query Optimization

Last Change: June 1, 2015

58/113
Equi-width-Histograms

- Equi-sum: sum of the source values of the bucket is equal; corresponds to the $\beta$ part of the sum of all source values.

- Particularly: Equi-sum($V, S$)
  - Sorting parameter: Attribute values
  - Source parameter: Span

- Aggregate of connected ranges of attribute values in one equi-width-bucket meaning for a bucket with $[v_{\text{min}}, v_{\text{max}}]$:

$$|v_{\text{max}} - v_{\text{min}}| \approx \frac{1}{\beta} (\max(V) - \min(V))$$
Equi-depth-Histograms

- Particularly: **Equi-sum(V, F)**
  - Sorting parameter: Attribute values
  - Source parameter: frequency

- Equal frequency ("height") in all buckets through adaption of the width or amount of accounted buckets (range)
- If $f_i$ of $v_i$ is greater than the maximal frequency: distribute over multiple buckets
Equi-width vs. Equi-depth-Histogramm

<table>
<thead>
<tr>
<th>Price</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>9</th>
<th>30</th>
<th>50</th>
<th>99</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount</td>
<td>100</td>
<td>75</td>
<td>30</td>
<td>20</td>
<td>15</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Andreas Meister
Advanced Query Optimization
Last Change: June 1, 2015