12. Set-Oriented DB-Interface

Goals
- Derivation of concepts for translation of descriptive queries and their
- Optimization and access module creation

Main references:

Realization of Database Systems – SS 2011

Logical Data Structures

Characterization of the mapping
SELECT Emp.Eno, Dept.Name
FROM Dept, Emp, Skill
WHERE Emp.Job = 'Programmer' &
Skill.Sno = Emp.Sno &
Emp.Dno = Dept.Dno

Mapping functions
- views <-> base relations
- relational expressions <-> logical access paths
- record sets <-> single records, currency indicators

 FETCH Skill USING ...
 FETCH NEXT Emp ...
 FETCH OWNER WITHIN ...

Properties of the upper interface
- Access-path-independent (relational) data model
- All facts and relationships are represented by values
- Non-procedural (descriptive) query languages
- Access to record sets
### Examples of Descriptive SQL Queries

- **Simple query**
  
  ```sql```
  ```
  SELECT Eno, EName, Salary/12  
  FROM Emp  
  WHERE Job = W  
  AND Bonus > Salary
  ```
  ```
  Replaced by
  ```
  DECLARE C1 CURSOR FOR SELECT Eno, EName, Salary/12  
  INTO :X, :Y, :Z  
  FROM Emp  
  WHERE Job = :W  
  AND Bonus > Salary
  ```
  ```
  With operators
  ```
  OPEN C1  
  FETCH C1 INTO :X, :Y, :Z  
  CLOSE C1
  ```
  ```

### Evaluation of DB Statements

- **Processing steps** for the evaluation of DB statements:

  1. **Lexical and syntactical analysis**
     - Creation of a query graph (QG) as reference structure for the subsequent compilation steps
     - Checking for correct syntax (parsing)

  2. **Semantic analysis**
     - Checking for the existence and validity of referenced tables, views, and attributes
     - Replacement of views in the QG by their view definitions
     - Replacement of external by internal names (name resolution)
     - Conversion of external formats into internal representations

  3. **Access control and integrity control**
     - should already be done for performance reasons, if possible, at compile time
     - Access control requires generation of runtime actions in case of value dependencies
     - Execution of simple integrity controls (control of formats and conversion of data types)
     - Generation of runtime actions for more complex controls
**Evaluation of DB Statements (2)**

4. **Standardization and simplification**
   - serve for more effective compilation and early error detection
   - Transformation of the QG into a normal form
   - Elimination of redundancies

5. **Restructuring and transformation**
   - Restructuring aims at global improvement of the QG; transformation inserts executable operations
   - Application of heuristic rules (*algebraic optimization* for QG restructuring)
   - Transformation leads to replacement and aggregation of logical operators by plan operators (*non-algebraic optimization*): In most cases, there are several plan operators available for the implementation of a logical operator
   - Determination of alternative access plans: Frequently, many execution sequences or access paths can be chosen
   - Cost estimation and selection of the cheapest execution plans
   - Steps 4 + 5 summarized as query optimization

6. **Code generation**
   - Generation of a tailor-made program for the given (SQL) statement
   - Creation of an executable access module
   - Management of access modules in a DBMS library

---

**Embedding of a Set-Oriented Interface**

- **Specification of the desired record set** (*qualification operator*)
  ```sql```
  ```
  SELECT Eno, EName, Salary/12
  FROM Emp
  WHERE Job='Operator' AND Bonus>Salary
  ```
  ```
  DECLARE C1 CURSOR FOR
  SELECT Eno, EName, Salary/12
  FROM Emp
  WHERE Job=:W AND Bonus>Salary
  ```
  ```
  ```

- **Successive provision of qualified records** (*fetch operator*)
  ```java```
  ```
  OPEN C1;
  FETCH C1 INTO :X, :Y, :Z;
  CLOSE C1;
  ```
  ```
  Possible solution: replacement by pre-compiler
  ```java```
  ```
  DECLARE C1 ...
  OPEN C1
  FETCH C1 INTO ...
  ```
  ```
**Query Optimization**

- **From query (what?) to evaluation (how?)**
  - **Goal:** cost-effective evaluation plan

- **Use of a large number of methods and strategies**
  - Logical transformation of queries
  - Selection of access paths
  - Optimized storage of data on external memory

- **Key problem**
  - Exact optimization is not ‘computable’, in general
  - Lack of accurate statistical information
  - Broad use heuristics (rules of thumb)

- **Optimization goal**
  - **either**
    - maximization of output with given resources
    - minimization of resource usage for given output
  - **or**
    - throughput maximization?
  - response time minimization for given query language, mix of queries of different types and given system environment!

## Query Optimization (2)

### Which costs are to be considered?
- **Communication cost**: (# of messages, set of data to be transmitted)
- **Computation cost**: (CPU cost, path lengths)
- **I/O cost**: (# of physical references)
- **Storage cost**: (temporary storage occupancy in DB buffer and on disk)

- Variety of costs are not independent of one another
- In centralized DBMS often "weighted function of computation- and I/O-costs"

### What is the best approach?

1. **Step 1**: After compilation, find appropriate internal representation for the query (QG)
2. **Step 2**: Apply logical restructuring to the query graph
3. **Step 3**: Map restructured query onto alternative sequences of plan operators (transformation) (=? set of execution plans)
4. **Step 4**: Compute cost estimates for each QEP and select the cheapest one

---

### Query Processing – Cost Issues

#### How are the costs divided up for transaction processing?
DB system : communication system : application

#### When do costs incur for the query processing?

#### Analysis

- **query**
  - without any preparation
  - syntactic analysis
  - semantic analysis (access- and integrity control)

#### Optimization

- **query graph**
  - standardization
  - simplification
  - query restructuring
  - query transformation

#### Code generation

- **code generation**
  - for maximal preparation
  - compilation time
  - run time

#### Execution

- **execution plan**
  - execution control

#### Provision of result data

- **query result**
  - allocation in program environment
Realization of DBS

Evaluation of DB statements

Host language embedding

Query optimization

Estimation of execution plans

Creation of execution plans

Code creation

Ad-hoc queries

© 2011 AG DBIS

---

Standardization of a Query

- **Standardization**
  - Choice of a normal form
  - E.g., conjunctive normal form
  - (A1 OR ... OR A1n) AND ... AND (Am1 OR ... OR Amn)
  - Displacement of quantors

- **Transformation rules for Boolean expressions**
  - Commutative rules
    - A OR B ⇔ B OR A
    - A AND B ⇔ B AND A
  - Associative rules
    - (A OR B) OR C ⇔ A OR (B OR C)
    - (A AND B) AND C ⇔ A AND (B AND C)
  - Distributive rules
    - (A OR (B AND C)) ⇔ (A OR B) AND (A OR C)
    - (A AND (B OR C)) ⇔ (A AND B) OR (A AND C)
  - De Morgan rules
    - NOT (A AND B) ⇔ NOT (A) OR NOT (B)
    - NOT (A OR B) ⇔ NOT (A) AND NOT (B)
  - Double negation rule
    - NOT (NOT (A)) ⇔ A

- **Itempotence rules for Boolean expressions**
  - A OR A ⇔ A
  - A AND A ⇔ A
  - A OR NOT (A) ⇔ TRUE
  - A AND NOT (A) ⇔ FALSE
  - A AND (A OR B) ⇔ A
  - A OR (A AND B) ⇔ A
  - A OR FALSE ⇔ A
  - A OR TRUE ⇔ TRUE
  - A AND FALSE ⇔ FALSE

---

Simplification of a Query

- **Equivalent expressions**
  - Can have a varying degree of redundancy
  - Usage/elimination of common sub-expressions
    - (A1 = a11 OR A1 = a12) AND (A1 = a12 OR A1 = a11)
    - (Age > 25 OR (Age > 30 AND Job = 'Programmer'))
  - Simplification of expressions referring to an "empty table"

- **Constant propagation**
  - A op B AND B = const. → A op const.

- **Non-satisfiable expressions**
  - A ≥ B AND B > C AND C ≥ A → A > A → false

- **Use of integrity constraints (IC)**
  - ICs are true for all records of the related table
    - A is primary key: Π_A → no duplicate elimination required
    - Rule: Family-Status = 'married' AND Tax-Class ≥ 3
    - Expression: (Family-Status = 'married' AND Tax-Class = 1) → false

- **Improvement of evaluation**
  - Adding of an IC to the WHERE clause does not change its truth value
  - Simpler evaluation structure, however more efficient heuristics needed

---

© 2011 AG DBIS
Query Restructuring

Most important rules for restructuring and transformation

- Early execution of Selection ($\sigma$) and Projection ($\pi$) without duplicate elimination
- Aggregation of unary operator sequences (as $\sigma$ and $\pi$) to a single operation
- Evaluation of equal subtrees in the QG only once
- Binary operator sequences (as $\land$, $\lor$, $\neg$, $\times$): minimization of intermediate results

- Selective operations ($\sigma$, $\pi$) before constructive operations ($x$, $\Join$)

Aggregation of operator sequences

- $R1: \pi_A1(...\pi_{AI}(\text{Tab})...) \Leftrightarrow \pi_{A1}(\text{Tab})$
- $R2: \sigma_{p1}(...\sigma_{p2}(\sigma_{p1}(\text{Tab})...)) \Leftrightarrow \sigma_{p1} \land p2 ... \land p_m(\text{Tab})$

Restructuring algorithm

1. Decompose complex join predicates such that they can be assigned to binary joins
2. Divide Selections with several predicate terms in separate Selections with a single predicate term each.
3. Execute Selections as early as possible, i.e., push down Selections to the leaves of the QG
4. Aggregate simple Selections such that subsequent Selections (of the same table) are executed at a time
5. Execute Projections without duplicate elimination as early as possible, i.e., push down Projections to the leaves as far as possible
6. Aggregate simple projections (on a table) to a single operation

Query Transformation

Task

aggregation of logical operators (one- and two-variable expressions) and their replacement by plan operators

Typical plan operators in relational systems

- On a single table:
  - Selection, Projection, Sorting, Aggregation, Update operations (IUD) and ACCESS to base tables
  - Extensions: recursion, grouping . . .
- On two tables:
  - Join- and set-operations, Cartesian product

Adjustments in QG for the effective use of plan operators

1. Grouping of adjacent operators (if possible);
2. Determination of processing sequence for binary operations;
   (minimize the size of intermediate tables)
3. Detection of common subtrees (compute them only once).
Realization of DBS

Evaluation of DB statements

Descriptive DB languages

Host language embedding

Query optimization

Estimation of execution plans

Creation of execution plans

Code creation

Ad-hoc queries

© 2011 AG DBIS

12-15

Estimation of Execution Plans – Fundamental Problems

- Query optimization rests on “fatal” assumptions, in general
  1. All data elements and all attribute values are uniformly distributed
  2. Independence of Predicates:
     - \( \text{sel}(X = a \text{ and } Y = b) = \text{sel}(X = a) \times \text{sel}(Y = b) \)
  3. Principle of Inclusion
     - \( \text{sel}(T1.X = T2.Y) = \frac{1}{\max\{|X|, |Y|\}} \)
  ➤ These assumptions are wrong (in the general case)

- Example
  (Family_Status = 'Married') AND (Age < 20)

➤ Linear interpolation, multiplication of probabilities
Although cost estimates are mostly wrong . . .

What is the Major Problem?

- Correlation in predicates!

EXAMPLE:
  Assume a query with the following WHERE clause:
  WHERE make = 'Honda' AND model = 'Accord',
  suppose
  - 10 makes \( \Rightarrow \) selectivity(make) = 1/10
  - 100 models \( \Rightarrow \) selectivity(model) = 1/100

  So selectivity of both = 1/10 \times 1/100 = 1/1000
  - But only Honda makes an Accord model!
  - We assumed the predicates were independent by multiplying the selectivities!
  - In fact, model and make are heavily correlated (predicate on make really adds no information)!

  ➤ Effect: We underestimate cardinality by an order of magnitude!
Estimation of Execution Plans – Fundamental Problems

- **Solution?**
  - Improvement of statistics/heuristics
  - Histograms, Sampling

- **Random samplings**
  - Speed: large data set, complex algorithms
  - **Example**: estimation "Sales in Europe" in a TPC-H application:
    - 1%: 8.46 Mio. in 4 sec.
    - 10%: 8.51 Mio. in 52 sec.
    - 100%: 8.54 Mio. in 200 sec.

- **Areas of sampling usage**
  - Approximate query evaluation, estimation of response time
  - Query optimization
  - Load balancing
  - Data mining
  - Interactive query design

---

**Use of Histograms**

- **Equi-width histogram of T.a**

<table>
<thead>
<tr>
<th>value</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>30</td>
<td>50</td>
<td>70</td>
<td>50</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>30</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Cardinality of result for Query 1?**
  - sum of bucket values for \( T.a \leq 50 \):
    - \( 20 + 30 + 50 + 40 = 140 \)

- **Cardinality of result for Query 2?**
  - Cardinality for \( T.a \leq 50 \): 140
  - Cardinality for \( 50 < T.a \leq 52 \):
    - uniform distribution assumption within buckets:
      - \( ((52-50) / (60-50)) \times 70 = 1/5 \times 70 = 14 \)
      - \( 140 + 14 = 154 \)
Cost Model - Problems

- **Optimizer task**
  - Obtains cost estimate for each “promising” execution plan
  - Use of a weighted cost formula:
    \[ C = \text{#physical page accesses} + W \times (\text{#calls of the access system}) \]
  - Desired: weighted measure for I/O- and CPU utilization
  - \( W \) is the cost ratio of \( AS \) call to page access

- **Permanent problem**
  - In 1985, SQL was not standardized
  - SQL2 and SQL3 are essentially more complex
    - UDTs
    - Type and table hierarchies
    - Recursion, Constraints, Triggers, ...

- **Compilation and optimization**
  - Cost-based optimizer
    - Histograms
    - but: UDTs need their own cost model
    - “Optimizing the XXX optimizer”
  - Dynamic QEPs
    - alternative plans depending on resource availability
    - “reduce the braking distance”
  - Seduction to gambling

Cost Model - Statistical Quantities

- **Statistical quantities for segments**
  - \( M_s \): number of data pages of segment \( S \)
  - \( L_S \): number of empty pages in \( S \)

- **Statistical quantities for tables**
  - \( N_R \): number of records of table \( R \) (\( \text{Card}(R) \))
  - \( T_{RS} \): number of pages in \( S \) with records of \( R \)
  - \( C_{sl} \): clustering factor (number of records per page)

- **Statistical quantities per index** on attributes \( A \) of a table \( R \):
  - \( j_i \): number of attribute values / key values in the index
    (\( \text{Card}(s_i(R)) \))
  - \( B_i \): number of leaf pages (B*-tree)

Statistics must be maintained in the DB catalog

- **Updating for each modification very expensive**
  - Additional write- and log operations
  - DB catalog would be the lock bottleneck

- **Alternative**
  - Initialization of statistical quantities at load- or generation time of tables and index structures
  - Periodical re-calculation of statistics by special command/service program (DB2: RUNSTATS)
Realization of DBS

Cost Model – Computational Basis

- **Selectivity Factor SF for**

  \[
  A_i = a_i \quad SF = \begin{cases} 
  1/j_i & \text{if index on } A_i \\
  1/10 & \text{otherwise}
  \end{cases}
  \]

  \[
  A_i = A_k \quad SF = \begin{cases} 
  1/\text{Max}(j_i, j_k) & \text{if index on } A_i, A_k \\
  1/10 & \text{if index on } A_i
  \end{cases}
  \]

  \[
  A_i \geq a_i \text{ (or } A_i > a_i) \quad SF = \begin{cases} 
  (a_{\text{max}} - a_i) / (a_{\text{max}} - a_{\text{min}}) & \text{if index on } A_i \text{ and interpolatable} \\
  1/3 & \text{otherwise}
  \end{cases}
  \]

  \[
  A_i \text{ BETWEEN } a_i \text{ AND } a_k \quad SF = \begin{cases} 
  (a_k - a_i) / (a_{\text{max}} - a_{\text{min}}) & \text{if index on } A_i \text{ and interpolatable} \\
  1/4 & \text{otherwise}
  \end{cases}
  \]

  \[
  A_i \text{ IN } (a_1, a_2, ..., a_r) \quad SF = \begin{cases} 
  r / j_i & \text{if index on } A_i \text{ and } SF < 0.5 \\
  1/2 & \text{otherwise}
  \end{cases}
  \]

- **Computation of expressions**

  - \(SF(p(A) \land p(B)) = SF(p(A)) \cdot SF(p(B))\)
  - \(SF(p(A) \lor p(B)) = SF(p(A)) + SF(p(B)) - SF(p(A)) \cdot SF(p(B))\)
  - \(SF(\neg p(A)) = 1 - SF(p(A))\)

- **Join Selectivity Factor (JSF)**

  - \(\text{Card}(R \times S) = \text{JSF} \cdot \text{Card}(R) \cdot \text{Card}(S)\)
  - for (N:1)-joins (loss-free): \(\text{Card}(R \times S) = \text{Max}(\text{Card}(R), \text{Card}(S))\)

- **Estimation of Execution Plans**

  - **Input:**
    - Optimized query graph (QG)
    - Existing storage structures and access paths
    - Cost model
  
  - **Output:** optimal execution plan (or at least “good”)

  - **Approach:**
    1. Generate all “reasonable” logical execution plans for the evaluation of the query
    2. Make the execution plans complete by adding information for physical data representation (sort sequence, access path properties, statistical information)
    3. Select the cheapest execution plan corresponding to the given cost model

- **Generation by the query optimizer**

  - Small set plans containing the optimal plan
  - Confinement by heuristics
  - Hierarchical generation based on the concept of nesting of SQL
  - Decomposition in a set of subqueries with at most two-variable expressions
Creation and Selection of Execution Plans (2)*

Interplay of components

- Plan generation rules for plan generation
- Evaluation of execution plans
- Creation and selection of execution plans
- Cost model

- Search strategies
  - Full enumerative
  - Restricted enumerative
  - Randomized
  - Reduction: certain search paths are not considered further (pruning)

- Cost estimate
  - Requires sufficiently accurate cost model
  - Is incrementally performed for all search methods

Problem representation - example

SQL:
SELECT E.Eno, E.Job, J.PName
FROM Emp E, Dept D, Proj J
WHERE D.In > 1000000
AND D.Loc = 'KL'
AND D.Dno = E.Dno
AND D.Dno = J.Dno;

related query graph

© 2011 AG DBIS

12-23

**Creation and Selection of Execution Plans (4)**

- **Possible access path for the single tables**

- **Solution tree for single tables: reduction by pruning of subtrees**

- **Extended solution tree** for the Nested-Loop join with the second table

- **Cost estimate per path:** e.g. by $C(C(D.Dno) + c(E.Dno) + \text{join cost})$
### Code Creation

- **Optimized query graph**
  - Result of optimization phase
  - Input data structure for code generator

- **Use of the operations of the access system**
  - Direct operations (e.g., INSERT <record>)
  - Scan operations (example SYSTEM R)
    - CALL RSS (OPEN, SCAN_STRUCTURE, RETURN_CODE)
    - CALL RSS (NEXT, SCAN_STRUCTURE, RETURN_CODE)
    - SCAN_STRUCTURE is complex data structure for the handing-over of in-/output values, search arguments, etc.

  These operations are base operations for the code generation

- **Classification of SQL statements**
  - Each class is described by a base process (e.g., by means of a cursor)
  - Skeleton of a base process is called ‘model’
  - Processing step in the model is called ‘fragment’ (as code stored in a library)

  Classification happens according to the kind of access actions

- **Provision of models and fragments**
  - 4 models for simple queries (query blocks)
  - In total: 30 models with 5-10 fragments each (<100 fragments)

---

### Flow Diagram for an Access Module

Model for the selection of a record set by means of a cursor

- **Prolog**
- **FETCH**
  - OPEN or FETCH
    - OPEN
      - binding of input variables
      - RSS call for OPEN
        - OK
          - RSS call for NEXT
            - OK
              - evaluation of the WHERE clause
                - result
                  - assignment to output variables
                    - setting of the RETURN code
                      - back jump
                    - N
                      - evaluation of the output record
                        - result
                          - assignment to output variables
                            - setting of the RETURN code
                              - back jump
### Spectrum of Binding Times in System R

<table>
<thead>
<tr>
<th>Statement type</th>
<th>Section type</th>
<th>Analysis</th>
<th>Optimization</th>
<th>Code generation</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal operations (Query, Insert, Delete, Update)</td>
<td>COMPILESECT</td>
<td>compile time</td>
<td>run time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>non-optimizable operations (Create/ Drop Table, etc.)</td>
<td>INTERPSECT</td>
<td>compile time</td>
<td>run time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>operations on temporary objects</td>
<td>PARSEDSECT</td>
<td>compile time</td>
<td>run time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dynamically defined statements (Prepare, Execute)</td>
<td>INDEFSECT</td>
<td>run time</td>
<td>run time</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Summary

#### Interpretation of a DB statement
- General program (Interpreter) accepts statements of the DB language as input and creates result by means of calls to the access system
- High overhead at run time (esp. for repeated statement executions)

#### Compilation, code creation, and execution of a DB statement
- For each DB statement, a tailor-made program is created (compile time) which is evaluated at run time and thereby derives the result by means of calls to the access system
- Compilation overhead is avoided as far as possible at run time

#### Query optimization: core problem
- of compilation of set-oriented DB languages
  - “Fatal” assumptions
    - uniform distribution of all attribute values
    - independence of predicates, principle of inclusion
  - Cost estimates for execution plans
    - CPU time and I/O overhead
    - no. of messages and data volumes to be transmitted (distributed case)
  - Good heuristics for the selection of execution plans is very important

#### Cost model
- Minimization of cost in dependency of the system state
- Problem: Update of statistical quantities