Query optimizer

- It automatically generates a query execution plan
  - It was formerly hard-coded by a programmer
  - The automatically generated execution plan is usually more efficient
    - It evaluates many different alternatives
    - It exploits statistics on data, stored in the system catalog, to make decisions
    - It exploits the best known strategies
    - It dynamically adapts to changes in the data distribution

Query optimizer

- It selects an efficient strategy for query execution
  - It is a fundamental building block of a relational DBMS
- It guarantees the data independence property
  - The form in which the SQL query is written does not affect the way in which it is implemented
  - A physical reorganization of data does not require rewriting SQL queries

Lexical, syntactic and semantic analysis

- Analysis of a statement to detect
  - Lexical errors
    - e.g., misspelled keywords
  - Syntactic errors
    - errors in the grammar of the SQL language
  - Semantic errors
    - references to objects which do not actually exist in the database (e.g., attributes or tables)
    - information in the data dictionary is needed
Lexical, syntactic and semantic analysis

- **Output**
  - Internal representation in (extended) relational algebra
- **Why relational algebra?**
  - It explicitly represents the order in which operators are applied
    - It is procedural (different from SQL)
  - There is a corpus of theorems and properties exploited to modify the initial query tree

Query optimizer

- **Selection of the “best” execution plan by evaluating execution cost**
  - Selection of
    - the best access method for each table
    - the best algorithm for each relational operator among available alternatives
  - Based on a cost model for access methods and algorithms
- **Generation of the code implementing the best strategy**

Algebraic optimization

- **Execution of algebraic transformations considered to be always beneficial**
  - Example: anticipation of selection with respect to join
  - Should eliminate the difference among different formulations of the same query
  - This step is usually independent of the data distribution
- **Output**
  - Query tree in “canonical” form

Cost based optimization

- **Output**
  - Access program in executable format
    - It exploits the internal structures of the DBMS
  - **Set of dependencies**
    - conditions on which the validity of the query plan depends
    - e.g., the existence of an index
**Query optimization**

**Query optimizer**

- **SQL QUERY**
- **LEXICAL, SYNTACTIC AND SEMANTIC ANALYSIS**
- **INTERNAL REPRESENTATION BASED ON RELATIONAL ALGEBRA**
- **ALGEBRAIC OPTIMIZATION**
  - "CANONICAL" QUERY TREE
- **COST BASED OPTIMIZATION**
  - DATA PROFILES (STATISTICS ON DATA)
  - DATA DICTIONARY
- **INTERNAL REPRESENTATION**
- **DATA BASED ON RELATIONAL ALGEBRA**

**Execution modes**

- **Compile and go**
  - Compilation and immediate execution of the statement
  - No storage of the query plan
  - Dependencies are not needed

**Compile and store**

- The access plan is stored in the database together with its dependencies
- It is executed on demand
- It should be recompiled when the data structure changes

**Algebraic optimization**

- It is based on equivalence transformations
  - Two relational expressions are *equivalent* if they both produce the same query result for any arbitrary database instance
  - Interesting transformations
    - reduce the size of the intermediate result to be stored in memory
    - prepare an expression for the application of a transformation which reduces the size of the intermediate result

---

Elena Baralis, Silvia Chiusano
Politecnico di Torino
Transformations

1. Atomization of selection
   \( \sigma_{F_1 \land F_2} \equiv \sigma_{F_2} (\sigma_{F_1}(E)) \)

2. Cascading projections
   \( \pi_X(E) \equiv \pi_X (\pi_{X,Y}(E)) \)

3. Anticipation of selection with respect to join (pushing selection down)
   \( \sigma_F (E_1 \times E_2) \equiv E_1 \times \sigma_F (E_2) \quad (F \text{ is a predicate on attributes in } E_2) \)

4. Anticipation of projection with respect to join
   \( \pi_L(E_1 \bowtie E_2) \equiv \pi_L (\pi_{L_1, J}(E_1 \bowtie \pi_{L_2, J}(E_2))) \)
   \( L_1 = L - \text{Schema}(E_2) \)
   \( L_2 = L - \text{Schema}(E_1) \)
   \( J = \text{set of attributes needed to evaluate join predicate } p \)

5. Join derivation from Cartesian product
   \( \sigma_F (E_1 \times E_2) \equiv E_1 \bowtie \pi_J (E_2) \)
   \( \pi_J \) predicate only relates attributes in \( E_1 \) and \( E_2 \)

6. Distribution of selection with respect to union
   \( \sigma_F (E_1 \cup E_2) \equiv (\sigma_F (E_1)) \cup (\sigma_F (E_2)) \)
### Transformations

5. Join derivation from Cartesian product
   - $\sigma_F(E_1 \times E_2) \equiv E_1 \bowtie E_2$
   - Predicate $F$ only relates attributes in $E_1$ and $E_2$

6. Distribution of selection with respect to union
   - $\sigma(F)(E_1 \cup E_2) \equiv (\sigma(F)(E_1)) \cup (\sigma(F)(E_2))$

7. Distribution of selection with respect to difference
   - $\sigma(F)(E_1 - E_2) \equiv (\sigma(F)(E_1)) - (\sigma(F)(E_2))$
   - All binary operators are commutative and associative except for difference.

8. Distribution of projection with respect to union
   - $\pi(X)(E_1 \cup E_2) \equiv (\pi(X)(E_1)) \cup (\pi(X)(E_2))$

9. Other properties
   - $\sigma_{F_1 \lor F_2}(E) \equiv (\sigma_{F_1}(E)) \cup (\sigma_{F_2}(E))$
   - $\sigma_{F_1 \land F_2}(E) \equiv (\sigma_{F_1}(E)) \cap (\sigma_{F_2}(E))$

10. Distribution of join with respect to union
    - $E(E_1 \cup E_2) \equiv (E \bowtie E_1) \cup (E \bowtie E_2)$

---

Elena Baralis, Silvia Chiusano  
Politecnico di Torino
Example

- Tables
  - EMP (Emp#, ………, Dept#, Salary)
  - DEPT (Dept#, DName,……………)

- SQL query
  SELECT DISTINCT DName
  FROM EMP, DEPT
  WHERE EMP.Dept#=DEPT.Dept#
  AND Salary > 1000;

Example: Algebraic transformations

\[ \pi_{\text{DName}}(\sigma_{\text{Salary}>1000}(\text{EMP} \times \text{DEPT})) \]

Prop #1

\[ \pi_{\text{DName}}(\sigma_{\text{Salary}>1000}(\text{EMP} \times \text{DEPT})) \]

Prop #5

Prop #2 and #4

Elena Baralis, Silvia Chiusano
Politecnico di Torino
Example: Query tree

Final query tree

Example: Cardinalities

- Cardinality (EMP) ≈ 10,000
- Cardinality (DEPT) ≈ 100
- Cardinality (EMP where Salary > 1000) ≈ 50

Cost based optimization

It is based on
- Data profiles
  - statistical information describing data distribution for tables and intermediate relational expressions
- Approximate cost formulas for access operations
  - Allow evaluating the cost of different alternatives for executing a relational operator

Database Management Systems

Cost based optimization

Data profiles

Elena Baralis, Silvia Chiusano
Politecnico di Torino
Table profiles

- Quantitative information on the characteristics of tables and columns
  - Cardinality (# of tuples) in each table T
  - Also estimated for intermediate relational expressions
  - Size in bytes of tuples in T
  - Size in bytes of each attribute Aj in T
  - Number of distinct values of each attribute in T
  - Cardinality of the active domain of the attribute
  - Min and max values of each attribute Aj in T

Table profiles are stored in the data dictionary. Profiles should be periodically refreshed by re-analyzing data in the tables.

Update statistics command
- Executed on demand
- Immediate execution during transaction processing would overload the system

Table profiles are exploited to estimate the size of intermediate relational expressions.
- For the selection operator
  \[ \text{Card} (\sigma_A T) = \frac{\text{Card} (T)}{\text{Val} (A) in T} \]
- \( \text{Val} (A) in T \) = # of distinct values of A in T (active domain)
- It holds only under the hypothesis of uniform distribution

Query tree

- Internal representation of the relational expression as a query tree
- Leaves correspond to the physical structures tables, indices
- Intermediate nodes are operations on data supported by the given physical structure
  - e.g., scan, join, group by
Sequential scan

- Executes sequential access to all tuples in a table
  - also called full table scan
- Operations performed during a sequential scan
  - Projection
  - discards unnecessary columns
  - Selection on a simple predicate \((A_i = v)\)
  - Sorting based on an attribute list
  - Insertion, update, delete

Predicate evaluation

- Conjunction of predicates \(A_i = v_1 \land A_j = v_2\)
  - The most selective predicate is evaluated first
  - Table is read through the index
  - Next the other predicates are evaluated on the intermediate result
- Optimization
  - First compute the intersection of RIDs (Record-Ids) coming from available indices
  - Next table read and evaluation of remaining predicates

Sorting

- Classical algorithms in computer science are exploited
  - e.g., quick sort
- Size of data is relevant
  - memory sort
  - sort on disk

Predicate evaluation

- Disjunction of predicates \(A_i = v_1 \lor A_j = v_2\)
  - Index access can be exploited only if all predicates are supported by an index
  - otherwise full table scan

Join operation

- A critical operation for a relational DBMS
  - connection between tables is based on values
  - instead of pointers
  - size of the intermediate result is typically larger than the smaller table
- Different join algorithms
  - Nested loop
  - Merge scan join
  - Hash join

Elena Baralis, Silvia Chiusano
Politecnico di Torino
A single full scan is done on the outer table
- For each tuple in the outer table
  - A full scan of the inner table is performed, looking for corresponding values
- Also called "brute force"

Both tables are sorted on the join attributes
- The two tables are scanned in parallel
  - Tuple pairs are generated on corresponding values
- Execution cost
  - The merge scan technique is symmetric
  - Requires sorting both tables
    - May be sorted by a previous operation
    - May be read through a clustered index on join attributes
- More used in the past
  - Efficient for large tables, because sorted tables may be stored on disk

Efficient when
- Inner table is small and fits in memory
- Optimized scan
- Join attribute in the inner table is indexed
- Index scan

The nested loop join technique is not symmetric
- The execution cost depends on which table takes the role of inner table
### Hash join

- Application of the same hash function to the join attributes in both tables
- Tuples to be joined end up in the same buckets
- Collisions are generated by tuples yielding the same hash function result with different attribute value
- A local sort and join is performed into each bucket
- Fastest join technique

### Group by

- Sort based
  - Sort on the group by attributes
  - Next compute aggregate functions on groups
- Hash based
  - Hash function on the group by attributes
  - Next sort each bucket and compute aggregate functions

### Cost based optimization

- Inputs
  - Data profiles
  - Internal representation of the query tree
- Output
  - "Optimal" query execution plan
  - Set of dependencies
- It evaluates the cost of different alternatives for
  - Reading each table
  - Executing each relational operator
- It exploits approximate cost formulas for access operations

### General approach to optimization

- The search for the optimal plan is based on the following dimensions
  - The way data is read from disk
    - e.g., full scan, index
  - The execution order among operators
    - e.g., join order between two join operations
  - The technique by means of which each operator is implemented
    - e.g., the join method
  - When to perform sort (if sort is needed)

### Execution plan selection

- The optimizer builds a tree of alternatives in which
  - Each internal node makes a decision on a variable
  - Each leaf represents a complete query execution plan
Example

Given 3 tables
- R, S, T
- Compute the join $R \bowtie S \bowtie T$

Execution alternatives
- 4 join techniques to evaluate (for both joins)
- 3 join orders
- In total, at most
  - $4 \times 4 \times 3 = 48$ different alternatives

Best execution plan selection

The final execution plan is an approximation of the best solution.
- The optimizer looks for a solution which is of the same order of magnitude of the “best” solution.
  - For compile and go
    - It stops when the time spent in searching is comparable to the time required to execute the current best plan.

Example

- Given 3 tables $R$, $S$, $T$
- Compute the join $R \bowtie S \bowtie T$
- Execution alternatives
  - 4 join techniques to evaluate (for both joins)
  - 3 join orders
  - In total, at most
    - $4 \times 4 \times 3 = 48$ different alternatives

Best execution plan selection

The optimizer selects the leaf with the lowest cost.
- General formula
  - $C_{\text{Total}} = C_{\text{IO}} \times n_{\text{IO}} + C_{\text{CPU}} + n_{\text{CPU}}$
    - $n_{\text{IO}}$ is the number of I/O operations
    - $n_{\text{CPU}}$ is the number of CPU operations
- The selection is based on operation research optimization techniques
  - e.g., branch and bound