Query Processing and Optimization

Structure of a DBMS

Applications

DBMS

SQL Commands

Query Evaluation Engine

Files and Access Methods

Concurrency Control

Disk Space Manager

Buffer Manager

Recovery Manager

System Catalog

Index Files

Data Files

Web Forms

Application Front Ends

SQL Interface

Focus
Steps in Processing a Query

- SQL statement
- Scan, Parse, Validate
  - Intermediate form of query
  - Query Optimizer
    - Execution plan
  - Query Code Generator
    - Code to execute query
  - Runtime DB Processor
    - Results of running query

Conceptual Evaluation Strategy

Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:

1. Compute the cross-product of \textit{from-list}
2. Discard resulting tuples that fail \textit{qualifications}
3. Delete attributes that are not in \textit{select-list}
4. If \texttt{DISTINCT} is specified, eliminate duplicate rows

\begin{verbatim}
SELECT S.name, E.name
FROM Student S, Enrolls E
WHERE S sidewinformatics.E sidewinformatics. term=10-2;
\end{verbatim}
Conceptual Evaluation Strategy

- This strategy is probably the least efficient way to compute a query!

- An “optimizer” will find more efficient strategies to compute the same answers

Intermediate Query Form

- An SQL query is translated into equivalent:
  
  **Relational Algebra Expression**

- Represented as a:
  
  **Query Tree**
Query Tree

- **Query tree**: a data structure that corresponds to a relational algebra expression
- Input relations of the query as *leaf nodes*
- Relational algebra operations as *internal nodes*
- An *execution* of the query tree consists of executing internal node operations

```
SELECT S.sname
FROM Student S, Enrolled E
WHERE S.sid = E.sid
AND S.gpa > 2
AND E.cid = CS2550;
```

\[
\pi_{sname}\left(\sigma_{\text{gpa} > 2}\left(S\right) \times_{\text{id} = \text{sid}} \sigma_{\text{cid} = \text{CS2550}}(E)\right)
\]
Example Selection Queries:

1. $\sigma_{\text{salary}>25000}$ (Employee)
2. $\sigma_{\text{ssn}=123456789}$ (Employee)
3. $\sigma_{\text{dnum}=6}$ (Employee)
4. $\sigma_{\text{gender]='m'}}$ (Employee)
5. $\sigma_{\text{dnum=6 AND salary}>25000 AND gender='f'}}$ (Employee)
6. $\sigma_{\text{salary>25000 AND salary<35000}}$ (Employee)
7. $\sigma_{(\text{dnum=6 OR gender='f'}}) \text{ AND salary}>25000$ (Employee)

How to choose an Implementation?

- Index Type
  - What indexes are available for the given relations?

- Query Type
  - Do we have range query, point query, conjunction?

- Statistics
  - Selectivity… 0-1
    - # of selected tuples / cardinality
  - Examples: what is the (estimated) selectivity?
    - $\sigma_{\text{sex='m'}}$ (Employee)
    - $\sigma_{\text{ssn='123456789'}}$ (Employee)
    - $\sigma_{\text{salary\_level=30000}}$ (Employee)
Implementations for Selection

- A1: linear search (brute force)
  - Full scan
- A2: binary search
  - Assume file is ordered on attribute
- A3: using primary index (or hash key)
  - Equality on key attribute
- A4: using primary/clustering index – multiple records
  - Equality on non-key attribute

Implementations for Selection...

- A5: using secondary index
  - Most general method – key/non-key attribute
- A6: primary index, comparison conditions
- A7: secondary index, comparison conditions
Implementations for Selection...

- How to handle conjunction (AND) / disjunction (OR)?
  - A8: Conjunctive selection using individual index
    - Check simple condition first, if it has index
  - A9: Conjunctive selection using composite index
    - Composite on both attributes must exist
  - A10: Conjunctive selection by intersection of record ptrs
    - Evaluate simple conditions independently
    - Produce intersection of lists of RIDs/BIDs
  - A11: Disjunctive selection by union of record ptrs

External Sorting

- **External sorting**: algorithms for sorting large files of records stored on disk that do not fit entirely in main memory

- Sorting is a **primary algorithm** in query processing: ORDER BY, DISTINCT, ...

- **Two-phase: Sort-Merge Strategy**
Sort-Merge

- **Phase 1: Sorting Phase**
  - Sorts main-memory sized sub-files (runs)
  - Size of a run = available memory blocks ($N_b$)
  - # of runs ($N_r$) = $B/N_b$ (B=#file blocks)
  - **Example:**
    - $B=1024$ and $N_b=5$ blocks, then $N_r=205$ runs

- **Phase 2: Merge Phase**
  - Merges the sorted runs in one or more pass, creating larger sorted sub-files
  - **Degree of merging** ($D_m$): the # of runs merged in each pass
  - $D_m = \min(N_b-1, N_r)$
  - **Example:**
    - $\min(4, 205)$ then $D_m = 4$
Sort-Merge

- **Passes:**
  - Example: $D_m = 4$ then 205 runs to 52, 13, 4, 1
  - Number of passes = $\log_{D_m} N_r$

- **Total block access:**
  - $(2*B) + ((2*(B * \log_{D_m} N_r)))$
J1: Nested-loop join

- Join relations R and S
  - A is the common attribute in R, B is the common attribute in S
- For each record t in R (=outer loop)
  - For each record s in S (=inner loop)
    - Test if t[A] = s[B]
- In practice, we are accessing an entire disk block at a time rather than a record at a time.
- Is there any difference which relation will be inner/outer?

J1: Nested-loop join performance

- Let buffer size be \( N_B \)
  - 1 block for result
  - 1 block for inner file
  - \( N_B-2 \) for outer file \( \Rightarrow \) # times outer loads: \( B_o/(N_B-2) \)
  - Total inner block accesses: \( B_i \times (B_o/(N_B-2)) \)
- Total number of block accesses:
  \[ B_o + B_i \times (B_o/(N_B-2)) \]
- Join relations R and S with \( N_B = 7 \) blocks
  - R: \( N_R=50 \) records stored in \( B_R=10 \) disk blocks
  - S: \( N_S=6000 \) records stored in \( B_S=2000 \) disk blocks
J2: Single-loop join

- Join relations $R$ and $S$
  - $A$ is the common attribute in $R$, $B$ is the common attribute in $S$
- Must use an access structure to retrieve the matching records
- Only works if an index (hash) key exists for one of the two join attributes ($A$ or $B$), say $B$ of $S$
- For each record $t$ in $R$
  - Locate tuples $s$ from $S$, that satisfy $s[B] = t[A]$

J2: Single-loop join performance

- Suppose indexes on both $E$.SSN (4 levels) & $D$.Head (2 levels)
  - Load $D$.Head Index and scan $E$
  - Load $E$.SSN and scan $D$
- Cost in # block accesses: $B_O + (N_O \times (L_{IR} + 1))$
- Join relations $D$ (Department) and $E$ (Employee)
  - $D$: $N_R=50$ records stored in $B_R=10$ disk blocks
  - $E$: $N_S=6000$ records stored in $B_S=2000$ disk blocks
  - $B_E + (R_E \times (L_{D}.Head +1)) = 2000 + (6000 \times 3) = 20,000$
  - $B_D + (R_D \times (L_{E}.SSN +1)) = 10 + (50 \times 5) = 260$
J2: Single-loop join performance

- Rule 1: Smaller file should be the outer join loop
- Join selection factor of a file wrt another file
  - depends on the equi-join condition
- Join relations $D$ (Department) and $E$ (Employee)
  - $E.SSN = D.Head$
  - $JSF_{DE} = 50/50 = 1$
  - $JSF_{ED} = 6000/50 = 0.008$
- Rule 2: The file that has high JSF should be the outer join loop

J3: Sort-merge join

- Join relations $R$ and $S$
  - $A$ is the common attribute in $R$,
    $B$ is the common attribute in $S$
- IF relations $R$ and $S$ are physically sorted (ordered)
  by the value of the join attributes
  - we simply have to scan the relations
    - Pairs of the file blocks are brought in memory
    - produce match or advance pointer
- Q: what happens if the relations are not sorted?
  - If both sorted then total # blocks: $B_R + B_S$
  - If sort is required, plus $\sim (B \log_2 B)$ for each sort
J4: Hash-join

- Join relations R and S
  - A is the common attribute in R,
  - B is the common attribute in S
- 1) Single pass through relation with fewer records (R)
  - Partitioning phase (into hash buckets)
- 2) Single pass through other relation (S)
  - Probing phase (use hash to find matching records of R)

Q: Memory requirements (N_b)? What is the cost?
Q: Will this work if R does not fit in memory?

Query Processing: Joins

- J1: Nested-loop join
- J2: Single-loop join
- J3: Sort-merge join
- J4: Hash-join
  - Partition Hash Join
  - Hybrid Hash Join
Partition Hash Join

- Join relations $R$ and $S$

- **Partitioning Phase**
  - Partition hash function
  - $R$ into $M$ partitions: $R_1, R_2, \ldots, R_M$
  - $S$ into $M$ partitions: $S_1, S_2, \ldots, S_M$
    - IDEA: $R_i$ only needs to be joined with $S_i$

- **Probing or Joining Phase**
  - Perform $M$ iterations
    - Join partitions $R_i$ and $S_i$
    - Can use nested-loop join or hash-join
      - If hash-join, must use different hash function. WHY?

**Partition Hash Join - Discussion**

- Q: What is the cost? ($\#$ disk blocks)
  - A: How many times each block is read/written?
    - Partitioning Phase: $R$: read once, write once
      - $S$: read once, write once
    - Probing Phase
      - $R$: read once
      - $S$: read once
      - write results once
    - Total cost = $3^*(b_R + b_S) + b_{\text{results}}$

- Q: What is the main difficulty of the algorithm?
  - A: What is the partitioning phase relying on?
    - Hash function is uniform!
      - partition sizes are nearly equal in size
Hybrid Hash Join

- Variation of Partition Hash Join
- Main idea:
  - Get a “free-ride” for joining the first partition during first pass
- Differences:
  - Partition Hash Join:
    - M partitions, at least single-block in memory for each one
  - Hybrid Hash Join
    - M partitions, store first one fully, M-1 with single-block
  - Partitioning Phase completely joins first partition
  - Probing Phase applied to M-1 partitions

Heuristic Optimization of Query Trees

- Get initial Query Tree
  - Apply Cartesian Product of relations in FROM
  - Selection and Join conditions of WHERE is applied
  - Project on attributes in SELECT
- Transform it into an Equivalent Query Tree
  - Represents a different relational expression
  - Gives the same result
  - More efficient to execute
- How ?
  - Push selections down the tree
  - Identify Joins
  - Push project operations down the tree
Example of Query Tree

\[ \Pi \ P.PNumber, P.DNum, E.Last, E.Address, E.DOB \]

\[ \sigma \ P.Location = "PGH" \ AND \ P.DNum=D.DNumber \ AND \ D.MgrSSN=E.SSN \]

Example of Query Tree – 2\(^{nd}\) version

\[ \Pi \ P.PNumber, P.DNum, E.Last, E.Address, E.DOB \]

\[ \sigma \ P.Location = "PGH" \]

\[ \Pi \ P.PNumber, P.DNum, E.Last, E.Address, E.DOB \]

\[ (((\sigma \ P.Location = "PGH")) \ AND \ Dnum=DNumber) \ AND \ (D) \ AND \ (D) \ AND \ MgrSSN=SSN(E)) \]
General Transformation Rules - 1

- Cascade of $\sigma$
  - $\sigma_{c_1 \text{ AND } c_2 \text{ AND } \ldots \text{ AND } c_n} (R) = \sigma_{c_1} (\sigma_{c_2} (\ldots (\sigma_{c_n} (R)) \ldots ))$

- Commutativity of $\sigma$
  - $\sigma_{c_1} (\sigma_{c_2} (R)) = \sigma_{c_2} (\sigma_{c_1} (R))$

- Cascade of $\Pi$
  - $\Pi_{\text{list}_1} (\Pi_{\text{list}_2} (\ldots (\Pi_{\text{list}_n} (R)) \ldots )) = \Pi_{\text{list}_1} (R)$

- Commuting $\sigma$ with $\Pi$
  - $\Pi_{A_1, A_2, \ldots, A_n} (\sigma_c (R)) = \sigma_c (\Pi_{A_1, A_2, \ldots, A_n} (R))$

General Transformation Rules - 2

- Commutativity of join and cartesian product
  - $R \text{ join } S = S \text{ join } R$
  - $R \times S = S \times R$
    - Note: order of attributes will not be the same

- Commuting $\sigma$ with join (or cartesian product)
  - If $c$ is applied only to $R$
    - $\sigma_c (R \text{ join } S) = (\sigma_c (R)) \text{ join } S$
  - If $c = c_1 \text{ AND } c_2$, with $c_1$ referring to $R$, $c_2$ referring to $S$:
    - $\sigma_c (R \text{ join } S) = (\sigma_{c_1} (R)) \text{ join } (\sigma_{c_2} (S))$
  - Same rules apply for cartesian product
General Transformation Rules - 3

- Commuting $\Pi$ with join or Cartesian product
  - Project list $L = \{A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m\}$
  - Suppose $A_1, A_2, \ldots, A_n$ are attributes of relation $R$, and that $B_1, B_2, \ldots, B_m$ are attributes of relation $S$
  - $\Pi_L (R \Join_c S) = (\Pi_{A_1, A_2, \ldots, A_n}(R)) \Join_c (\Pi_{B_1, B_2, \ldots, B_m}(S))$
    - Note: $c$ must only involve attributes in $L$
    - What if $c$ contains additional attributes?

- Commutativity of set operations:
  - Union?
  - Intersection?
  - Set difference?

General Transformation Rules - 4

- Associativity of join, $\times$, $\cup$, $\cap$
  - $(R \times S) \times T = R \times (S \times T)$
    - where $\times$ can be any of join, $\times$, $\cup$, $\cap$

- Commuting $\sigma$ with set operations
  - $\sigma_c (R \# S) = (\sigma_c (R)) \# (\sigma_c (S))$
    - where $\#$ can be any of $\setminus$, $\cup$, $\cap$

- Commuting of $P$ operation:
  - $\Pi_L (R \cup S) = (\Pi_L (R)) \cup (\Pi_L (S))$
**General Transformation Rules - 5**

- Converting a \((\sigma, x)\) sequence into a join operation
  - If \(c\) corresponds to a join condition, then
    \[ \sigma_c (R \times S) = R \text{ join}_c S \]

- Other transformations:
  - Any boolean transformation can still be applied, e.g.:
    - \(\neg (C_1 \text{ and } C_2) = (\neg C_1) \text{ or } (\neg C_2)\)
    - \(\neg (C_1 \text{ or } C_2) = (\neg C_1) \text{ and } (\neg C_2)\)

---

**Example of Query Tree**

\[
\begin{align*}
\pi & \quad \text{P.PNumber, P.DNum, E.Last, E.Address, E.DOB} \\
\sigma & \quad \text{P.Location = "PGH" AND P.DNum=D.DNumber AND D.MgrSSN=E.SSN} \\
\times & \quad \text{E} \\
\times & \quad \text{D} \\
\times & \quad \text{P}
\end{align*}
\]

- Select P.PNumber, P.DNum, E.Last, E.Address, E.DOB
  From Employee as E, Department as D, Project as P,
  Where P.DNum = D.DNumber and D.MgrSSN = E.SSN and
  P.Location = ‘PGH’
Example of Query Tree (Step 2)

\[ \Pi \ P.PNumber, P.DNum, E.Last, E.Address, E.DOB \]
\[ \sigma \ P.DNum=D.DNumber \]
\[ \times \]
\[ \sigma \ D.MgrSSN=E.SSN \]
\[ \sigma \ P.Location = “PGH” \]

Example of Query Tree (Step 3)

\[ \Pi \ P.PNumber, P.DNum, E.Last, E.Address, E.DOB \]
\[ \times \ P.DNum=D.DNumber \]
\[ \times \ D.MgrSSN=E.SSN \]
\[ \sigma \ P.Location = “PGH” \]
Example of Query Tree (Step 4)

```
Π P.PNumber, P.DNum, E.Last, E.Address, E.DOB

⋈ D.MgrSSN=E.SSN

⋈ P.DNum=D.DNumber

σ P.Location = "PGH"

P

D

E
```

Example of Query Tree (Step 5)

```
Π P.PNumber, P.DNum, E.Last, E.Address, E.DOB

⋈ D.MgrSSN=E.SSN

⋈ P.DNum=D.DNumber

Π P.PNumber, P.Dnum

σ P.Location = "PGH"

P

D

Π E.SSN, E.Last, E.Address, E.DOB

E
```
Example of Query Tree (Step $5\frac{1}{2}$)

$\Pi$ P.PNumber, P.DNum, E.Last, E.Address, E.DOB

$\bowtie$ D.MgrSSN=E.SSN

$\bowtie$ P.DNum=D.DNumber

$\Pi$ P.PNumber, P.DNumber

$\sigma$ P.Location = “PGH”

$\Pi$ D.MgrSSN, D.DNumber

$\Pi$ E.SSN, E.Last, E.Address, E.DOB

$\Pi$ P.PNumber, P.Dnum

$\Pi$ E.SSN, E.Last, E.Address, E.DOB

$\Pi$ D.MgrSSN, D.DNumber

Example of Query Tree (Step 6)

$\Pi$ P.PNumber, P.DNum, E.Last, E.Address, E.DOB

$\bowtie$ D.MgrSSN=E.SSN

$\bowtie$ P.DNum=D.DNumber

$\Pi$ P.PNumber, P.Dnum

$\sigma$ P.Location = “PGH”

$\Pi$ D.MgrSSN, D.DNumber

$\Pi$ E.SSN, E.Last, E.Address, E.DOB

$\Pi$ E.SSN, E.Last, E.Address, E.DOB

$\Pi$ D.MgrSSN, D.DNumber

P

D

E
Outline of algebraic optimization

1. Break up selections (with conjunctive conditions) into a cascade of selection operators
2. Push selection operators as far down in the tree as possible
3. Convert Cartesian products into joins
4. Rearrange leaf nodes to:
   - Execute first the most restrictive select operators
   - What is restrictive? (Fewest tuples or Smallest size)
   - Make sure we don’t have Cartesian products
5. Move projections as far down as possible
6. Identify subtrees that represent groups of operations which can be executed by single algorithm

Query Execution Plans

- Query tree annotated with algorithms and access methods

- Q: Is single-operator-at-a-time appropriate?
  - Materialized evaluation or processing
  - A: NO. Why?
  - A: We would need temporary relations (and extra disk space) to store intermediate results

- What is the alternative?
  - Combine operators (and their execution) into a sequence
  - Pipelining or Stream-based Processing
Cost-based query optimization

- Compiled queries VS interpreted queries
- Cost-based query optimization
  - Full-scale optimization, taking costs & selectivity into account to produce the lowest estimated cost plan
  - Usually happens only for compiled queries
- Costs:
  - Access cost to secondary storage
  - Storage cost (for intermediate results)
  - Computation cost
  - Memory usage cost
  - Communication cost (data/query shipping)

How to estimate costs?

- How can we determine costs without running the query?
  - Cost model
  - Sizes (record, block, index …)
    - cardinality (r), #blocks, blocking-factor…
    - Type of index, #levels, # first level index blocks
  - Selectivity of operations (sl)
    - Selection cardinality: \( S = sl \times r \)
    - Number of distinct values
- Histograms: Distribution of records over the data values for an attribute
Cost Functions for Selection

- **A1**: linear search (brute force)
  - \( B/2 \) if found, \( B \) otherwise

- **A2**: binary search
  - \( \log_2 B \) if unique key
  - \( \log_2 B + s/bfr -1 \) if not key

- **A3**: using primary index (or hash key)
  - 1 for static or linear hashing, 2 for extensible
  - Levels + 1

- **A4**: using primary/clustering index – multiple records
  - Levels + \( (B/2) \) for primary index
  - Levels + \( (s/bfr) \) for clustering index

Cost functions for Selection

- **A5**: using secondary (B+-tree) index
  - Levels + \( s \) (worst case) on equality (not a key)
  - Levels + \( \left( B_{\text{index}}/2 \right) + r/2 \) on comparison,
    - half of the records will be selected \( \Rightarrow \) half of the index block
Multiple relation queries with joins

- The algebraic transformation rules lead to many equivalent expressions (query trees)
- # alternative trees grow rapidly as the # joins increases
- Pruning the deep trees