ICS 434
Advanced Database Systems

Dr. Abdallah Al-Sukairi
sukairi@kfupm.edu.sa

Second Semester 2003 - 2004 (032)

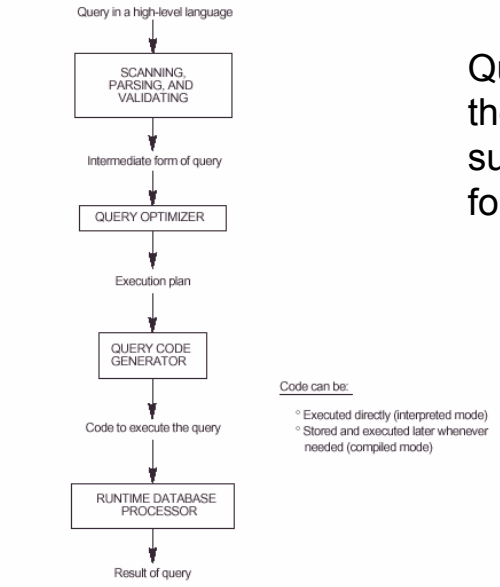
King Fahd University of Petroleum & Minerals
Information & Computer Science Department

Outline

1. The Relational Data Model: Version 2
2. Advanced Data Modeling
3. Client-Server Architecture
4. Client-Server Databases & Tools
5. Databases on the Web
6. The System Catalog
7. **Query Processing and Optimization**
8. Transaction Processing
9. Concurrency Control
10. Recovery
11. Administration & Security
12. Distributed Databases
13. Database Replication
14. Object-Oriented Databases
15. Data Warehousing and Data Mining
16. Other Emerging Database Technologies

7. Query Processing and Optimization

Processing a Query



Query optimization:
the process of choosing a suitable execution strategy for processing a query

Indexing for Performance

- Simple (individual) index
- B^+-tree index
- Matching index scan vs nonmatching index scan
- Unique index
  - one entry and one pointer (RID) at leaf pages
  - RID = page # + offset
- Nonuniqe index
  - one entry for each distinct values with the RIDs
- New levels are generated as needed and splitting keeps the B^+-tree balanced

... Indexing for Performance

- Hashed index
- Index compression
- Multiple clustering
- Distributing input/output

Columns to Index

- Primary key and foreign key columns
- Columns that have aggregates computed frequently
- Columns that are searched or joined over less than 5 to 10 percent of the rows
- Columns frequently used in an ORDER BY, GROUP BY, and DISTINCT clause

Clustering

- The clustering index holds the most potential for performance gains
- A clustered index allow for merge join
- Columns that benefit from clustering:
  - Primary key column
  - Columns with low index cardinality (# of distinct values) or skewed distribution
  - Column frequently processed in sequence (ORDER BY, GROUP BY, DISTINCT)
  - Columns frequently searched or joined over a range of values such as BETWEEN, <, >, LIKE

Composite Index

- If two or more columns are often used together, composite index should be considered
- If the first column of a composite index is not specified, matching scan can not be perform
- Can't be used to satisfy ORed predicates

When not to Index

- Avoid indexing columns
  - That are frequently updated
  - That have low cardinality, skewed distribution (unless clustering index is used)
  - That are longer than 40 bytes
  - That cause lock contention problems
- Avoid indexing tables
  - That are fewer than 7 pages unless used frequently in joins or in referential integrity checking

Algorithms for External Sorting

- External sorting: refers to sorting algorithms that are suitable for large files of records stored on disk that do not fit entirely in main memory, such as most database files.
- Sort-Merge strategy: starts by sorting small subfiles (runs) of the main file and then merges the sorted runs, creating larger sorted subfiles that are merged in turn.
  - Sorting phase: $n_R = \lceil (b/n_B) \rceil$
  - Merging phase: $d_M = \text{Min}\ (n_B\text{-}1, n_R)$; $n_P = \lceil (\log_{dM}(n_R)) \rceil$

  n_R: number of initial runs; b: number of file blocks;
  n_B: available buffer space; d_M: degree of merging;
  n_P: number of passes.

Implementing the SELECT Operation

- Search Methods for Simple Selection
  - S1. Linear search (brute force)
  - S2. Binary search
  - S3. Using a primary index or hash key to retrieve a single record
  - S4. Using a primary index to retrieve multiple records
  - S5. Using a clustering index to retrieve multiple records
  - S6. Using a secondary (B^+-tree) index
- Search Methods for Complex Selection
  - S7. Conjunctive selection
  - S8. Conjunctive selection using a composite index
  - S9. Conjunctive selection by intersection of record pointers

Implementing the JOIN Operation

- Join (EQUIJOIN, NATURAL JOIN)
  - two–way join
    - $R \bowtie_{A=B} S$
  - multi-way joins
    - $R \bowtie_{A=B} S \bowtie_{C=D} T$
- Methods for implementing joins:
  - J1. Nested-loop join (brute force) *O(m x n)*
  - J2. Single-loop join *O(m + n)*
  - J3. Sort-merge join *O(n log n + m log m + n + m)*
  - J4. Hash-join
  - J5. Partition hash-join
  - J6. Hybrid hash-join

Selectivity

- The Selectivity (s) = the ratio of records satisfying the condition
  - C1 = V 1 / COLCARD
  - C1 IN (V1, V2, ..) # of values in list / COLCARD
  - C1 > V (HIGHKEY - V) / (HIGHKEY - LOWKEY)
  - C2 < V (V - LOWKEY) / (HIGHKEY - LOWKEY)
- Number of I/O needed = s * # of rows or pages (blocks)

Factors Affecting JOIN Performance

- Available buffer space
- Join selection factor
- Choice of inner VS outer relation

Combining Operations using Pipelining

- Motivation
  - A query is mapped into a sequence of operations
  - Each execution of an operation produces a temporary result
  - Generating and saving temporary files on disk is time consuming and expensive
- Alternative:
  - Avoid constructing temporary results as much as possible
  - Pipeline the data through multiple operations - pass the result of a previous operator to the next without waiting to complete the previous operation

… Combining Operations using Pipelining

- Example: For a 2-way join, combine the 2 selections on the input and one projection on the output with the Join
- Dynamic generation of code to allow for multiple operations to be pipelined
- Results of a select operation are fed in a "Pipeline" to the join algorithm
- Also known as stream-based processing

Query Optimization

- A challenge and an opportunity to DBMS vendors
- Access plan
- Execution method
- Two approaches
  - Heuristics-based
  - Cost-based
- Access path selection
  - Table scan
  - Index order scan
  - Indexed access

... Heuristics-based Optimization

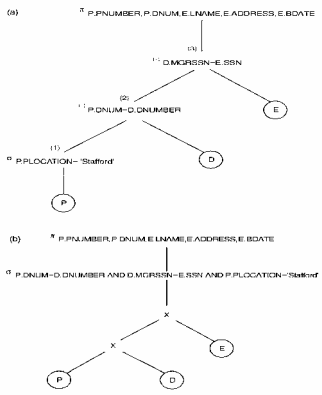
- Process for heuristics optimization
  - The parser of a high-level query generates an initial internal representation
  - Apply heuristics rules to optimize the internal representation
  - A query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query
- The main heuristic is to apply first the operations that reduce the size of intermediate results.
  - E.g., Apply SELECT and PROJECT operations before applying the JOIN or other binary operations.

... Heuristics-based Optimization

- Query tree: a tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as leaf nodes of the tree, and represents the relational algebra operations as internal nodes
- An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation
- Query graph: a graph data structure that corresponds to a relational calculus expression. It does not indicate an order on which operations to perform first. There is only a single graph corresponding to each query

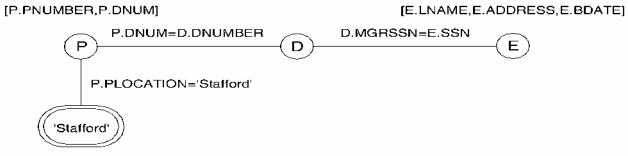
... Heuristics-based Optimization

Figure 18.4 Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2.



... Heuristics-based Optimization

Figure 18.4 (c) Query graph for Q2.

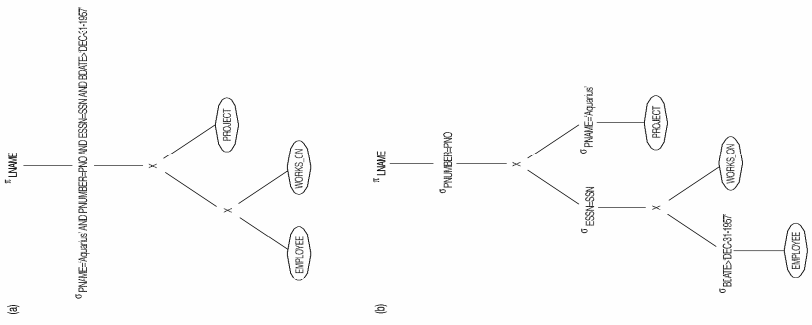


... Heuristics-based Optimization

- Heuristic Optimization of Query Trees:
  - The same query could correspond to many different relational algebra expressions — and hence many different query trees
  - The task of heuristic optimization of query trees is to find a final query tree that is efficient to execute

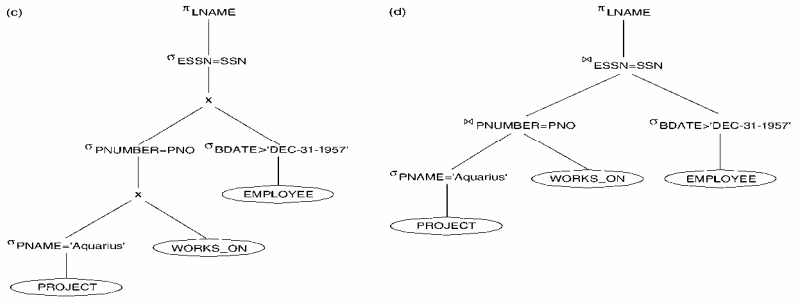
... Heuristics-based Optimization

Figure 18.5 Steps in converting a query tree during heuristic optimization. (a) Initial (canonical) query tree for SQL query Q. (b) Moving SELECT operations down the query tree.



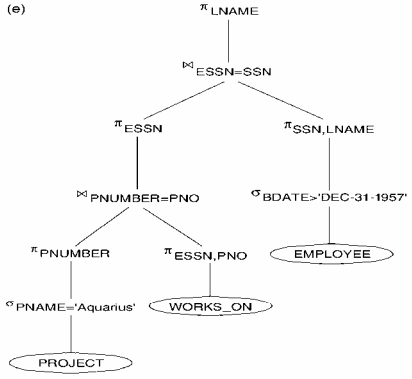
... Heuristics-based Optimization

Figure 18.5 Steps in converting a query tree during heuristic optimization. (c) Applying the more restrictive SELECT operation first. (d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.



... Heuristics-based Optimization

Figure 18.5 Steps in converting a query tree during heuristic optimization. (e) Moving PROJECT operations down the query tree.



... Heuristics-based Optimization

- Query Execution Plans
  - An execution plan for a relational algebra query consists of a combination of the relational algebra query tree and information about the access methods to be used for each relation as well as the methods to be used in computing the relational operators stored in the tree
  - Materialized evaluation: the result of an operation is stored as a temporary relation
  - Pipelined evaluation: as the result of an operator is produced, it is forwarded to the next operator in sequence.

... Heuristics-based Optimization

- ORACLE chooses an execution plan based on the following criteria:
  - The syntax of the SQL statement
  - The conditions of the WHERE clause
  - The structure and definition of the tables involved
  - Any index that exist on these tables

Cost-based Optimization

- Cost-based query optimization: Estimate and compare the costs of executing a query using different execution strategies and choose the strategy with the lowest cost estimate
- Issues
  - Cost function
  - Number of execution strategies to be considered
- Optimization steps
  - Semantic and syntax checking
  - Code transformation
  - Building a tree of alternate access paths (Pruning obviously expensive paths)
  - Using catalog statistics to choose and efficient plan

… Cost-based Optimization

- Code Transformation
  - Adding predicates, transforming code, flattening nested select
- Examples of adding predicates
  - WHERE TA.C1 = TB.C5
AND TA.C1 > 10
AND TB.C5 > 10 -- added by the optimizer
  - WHERE TA.C1 = TB.C5
AND TB.C5 = TC.C10
AND TC.C10 = TA.C1 -- added by the optimizer
- Examples of transforming code
  - WHERE TA.C1 = 10 OR TA.C1 = 30

    transformed to:

    WHERE TA.C1 IN (10, 30)

… Cost-based Optimization

- Cost Components for Query Execution
  - Access cost to secondary storage
  - Storage cost
  - Computation cost
  - Memory usage cost
  - Communication cost
- Different database systems may focus on different cost components

… Cost-based Optimization

- Estimating Costs
  - Cost = (w1 * I/O cost) + (w2 * CPU cost)
    where w1 and w2 are weighting factors
  - Selectivity and cost estimates
  - Number of I/O needed = FF * # of rows or pages

Using Selectivity and Cost Estimates

- Catalog Information Used in Cost Functions
  - Information about the size of a file
    - number of records (tuples) (r),
    - record size (R),
    - number of blocks (b)
    - blocking factor (bfr)
  - Information about indexes and indexing attributes of a file
    - Number of levels (x) of each multilevel index
    - Number of first-level index blocks (bI1)
    - Number of distinct values (d) of an attribute
    - Selectivity (sl) of an attribute
    - Selection cardinality (s) of an attribute. (s = sl * r)

… Using Selectivity and Cost Estimates

- Examples of Cost Functions for SELECT
  - Linear search (brute force) approach
  - Binary search
  - Using a primary index or hash key to retrieve a single record
  - Using an ordering index to retrieve multiple records
  - Using a clustering index to retrieve multiple records
  - Using a secondary (B+-tree) index
  - Conjunctive selection
  - Conjunctive selection using a composite index

… Using Selectivity and Cost Estimates

- Examples of Cost Functions for JOIN
  - Join selectivity (js)
  - Size of the result file after join operation
  - Nested-loop join
  - Single-loop join
  - Sort-merge join

… Using Selectivity and Cost Estimates

- Multiple Relation Queries and Join Ordering
  - A query joining n relations will have n-1 join operations, and hence can have a large number of different join orders when we apply the algebraic transformation rules
  - Current query optimizers typically limit the structure of a (join) query tree to that of left-deep (or right-deep) trees
  - Left-deep tree: a binary tree where the right child of each non-leaf node is always a base relation
    - Amenable to pipelining
    - Could utilize any access paths on the base relation (the right child) when executing the join

Statistics Used by DB2 Optimizer

SYSIBM.SYSTABLES	
CARD	Number of rows in the table (cardinality)
NPAGES	Number of pages that contain rows of the table
PCTPAGES	Percentage of tablespace pages that contain rows of tables
EDPROC	blank = EDITPROC not used
SYSIBM.SYSINDEXES	
CLUSTERRATIO	Percent of rows in clustering sequence
CLUSTERED	Y if 95 percent of rows are in clustered sequence
FIRSTKEYCARD	Number of distinct values in indexed column
FULLKEYCARD	Number of distinct values in composite index
NLEAF	Number of active leaf pages in the index
NLEVELS	Number of levels in the index tree
SYSIBM.SYSCOLUMNS	
COLCARD	Number of distinct values in column
HIGH2KEY	Second highest value in the column (first 8 bytes)
LOW2KEY	Second lowest value of the column (first 8 bytes)
FOREIGNKEY	B if bit data, S if SBCS data, M if mixed mode
SYSIBM.SYSTABLESPACE	
NACTIVE	Number of pages up to the high preformatted page
SYSIBM.SYSFIELDS	
EXITPARM	Indexed value for 1 of 10 most frequently occurring value (first column of a composite index)
EXITPARML	Percentage of rows * 100 that contain the value in EXITPARM

Most columns contain −1 until RUNSTATS is executed

… Statistics Used by DB2 Optimizer

- If RUNSTATS has not been executed,
  - Default statistics are used by the optimizer

SYSIBM.SYSTABLES	
CARD	10,000
NPAGES	501 = CEILING (1 + CARD/20)
EDPROC	blank = EDITPROC not used
SYSIBM.SYSINDEXES	
CLUSTERRATIO	0 (95% if CLUSTERED = Y)
CLUSTERED	Y (if CLUSTER used when index was created)
FIRSTKEYCARD	25
FULLKEYCARD	25
NLEAF	33.33 = CARD/300
NLEVELS	2
SYSIBM.SYSCOLUMNS	
COLCARD	25
FOREIGNKEY	S for SBCS data
SYSIBM.SYSTABLESPACE	
NACTIVE	501 = CEILING (1 + CARD/20)

- Uniform distribution of data assumed

Overview of Query Optimization in Oracle

- Oracle DBMS V8
  - Rule-based query optimization: the optimizer chooses execution plans based on heuristically ranked operations. (Currently it is being phased out)
  - Cost-based query optimization: the optimizer examines alternative access paths and operator algorithms and chooses the execution plan with lowest estimate cost. The query cost is calculated based on the estimated usage of resources such as I/O, CPU and memory needed.
  - Application developers could specify hints to the ORACLE query optimizer. The idea is that an application developer might know more information about the data.

Semantic Query Optimization

- Semantic Query Optimization: Uses constraints specified on the database schema in order to modify one query into another query that is more efficient to execute.
- Consider the following SQL query

```
SELECT    E.LNAME, M.LNAME
FROM      EMPLOYEE E M
WHERE     E.SUPERSSN=M.SSN AND E.SALARY>M.SALARY
```

- Explanation: Suppose that we had a constraint on the database schema that stated that no employee can earn more than his or her direct supervisor. If the semantic query optimizer checks for the existence of this constraint, it need not execute the query at all because it knows that the result of the query will be empty. Techniques known as theorem proving can be used for this purpose.