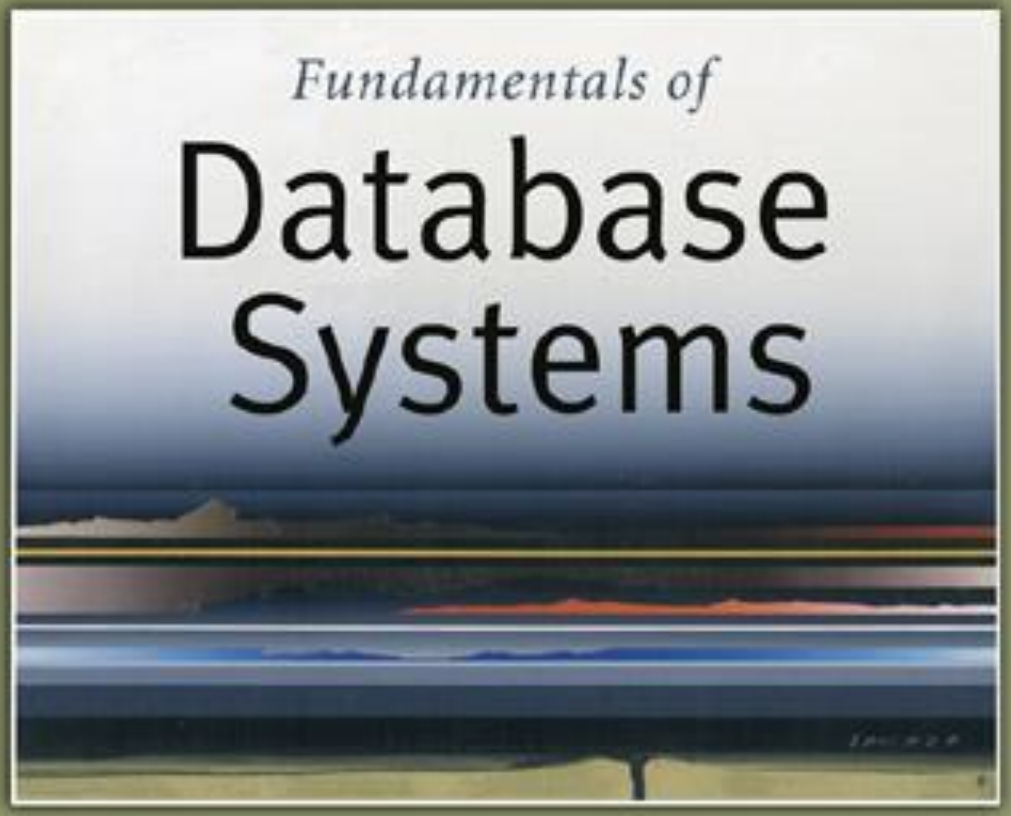
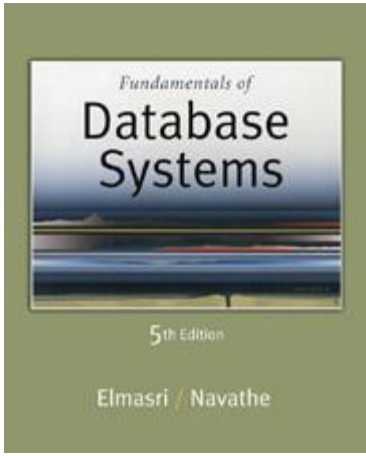


5th Edition

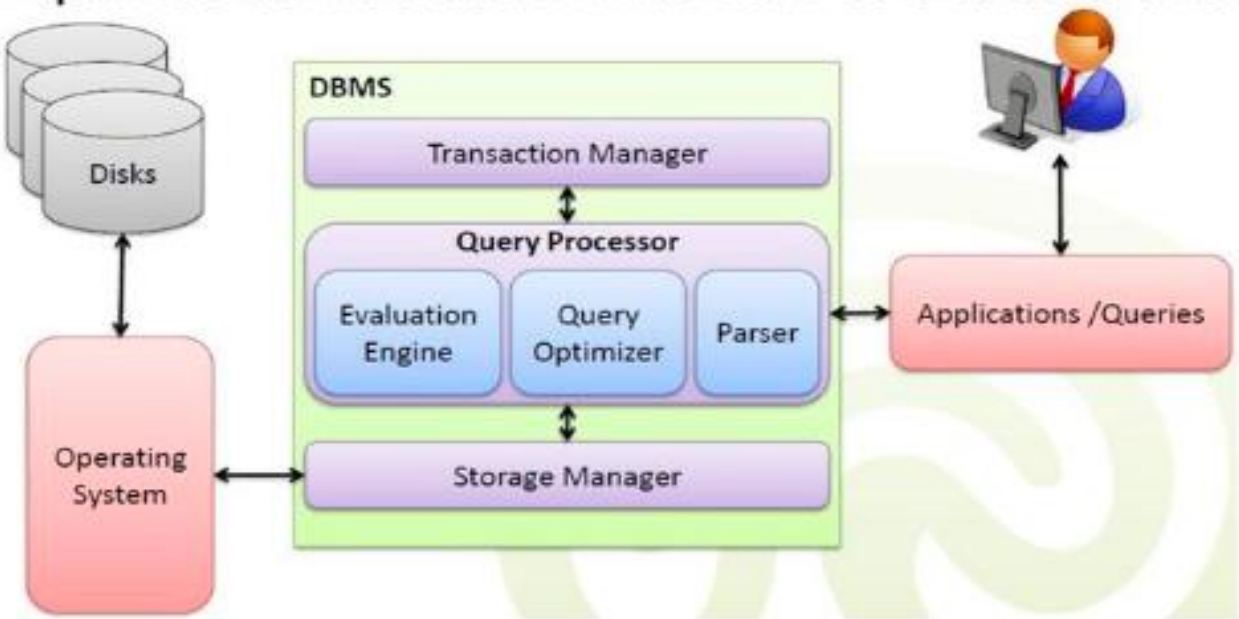
Elmasri / Navathe

Algorithms for Query Processing and Optimization



Introduction to Query Processing (1)

- Users/applications submit queries to the DBMS
- The DBMS processes queries before evaluating them
  - Recall: DBMS mainly use declarative query languages (such as SQL)
  - Queries can often be evaluated in different ways
  - SQL queries do not determine how to evaluate them



Introduction to Query Processing (2)

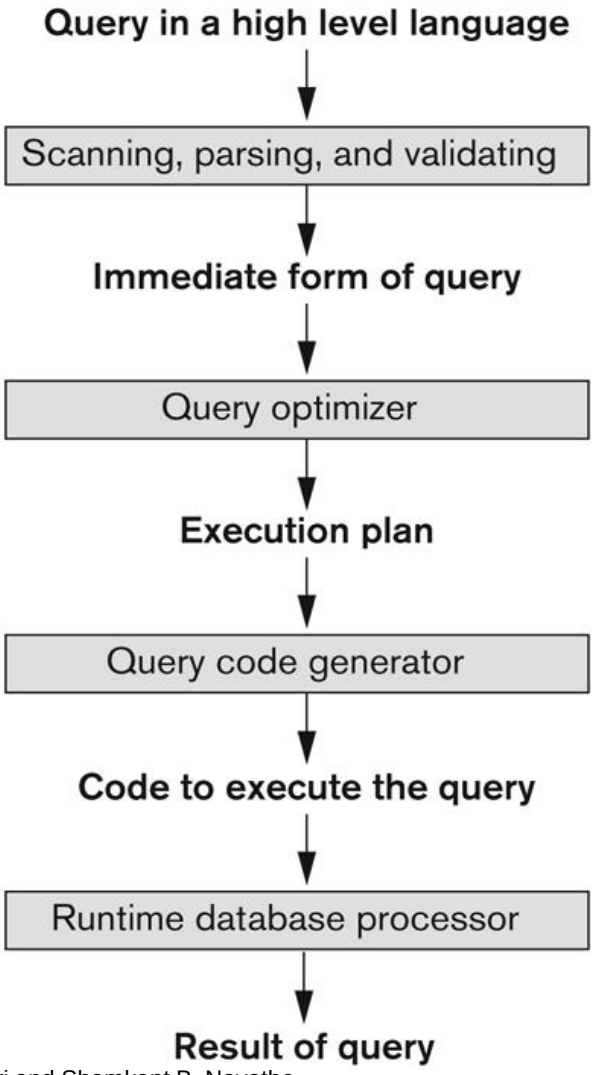


Figure 15.1
Typical steps when processing a high-level query.

Code can be:

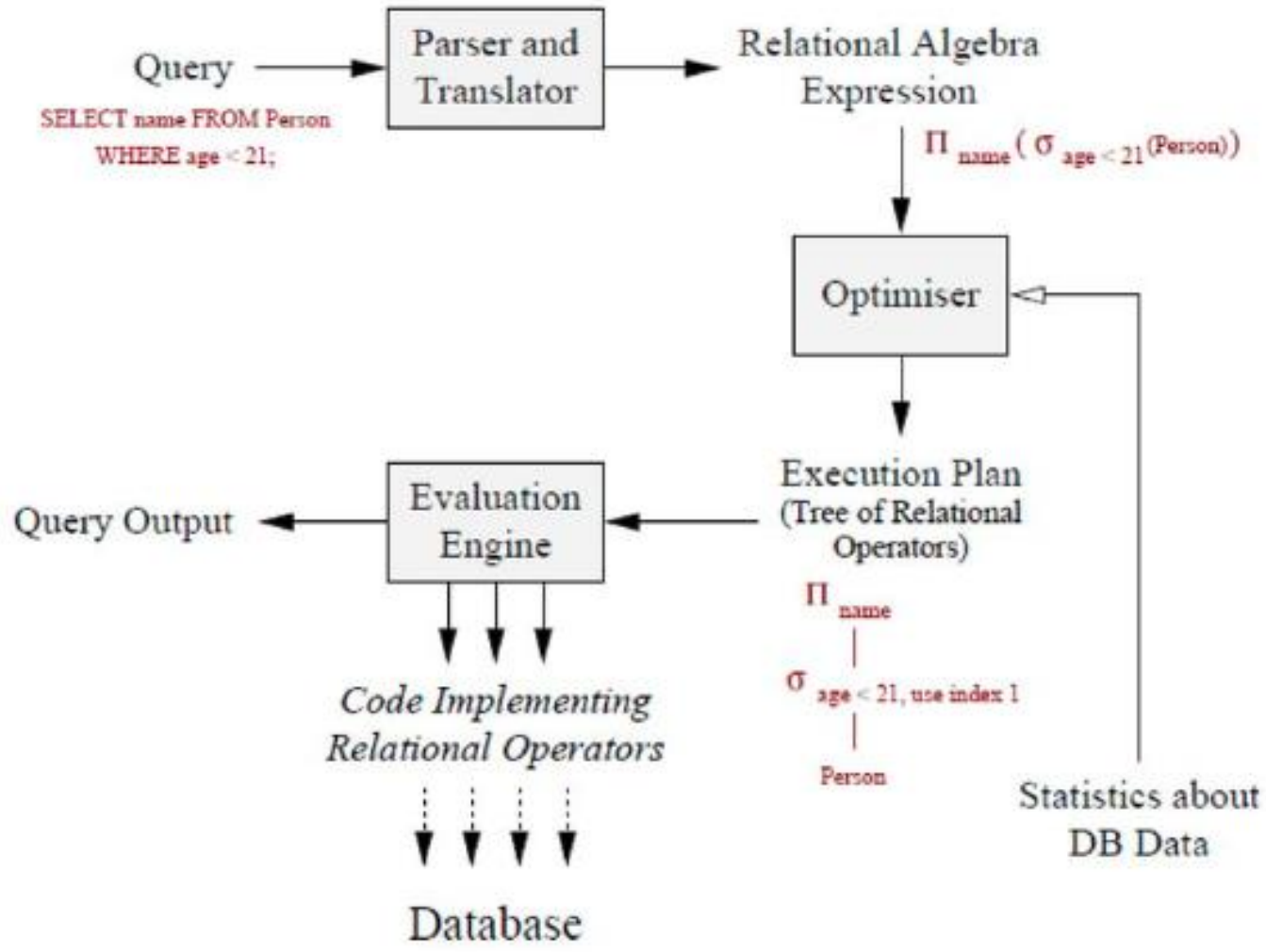
Executed directly (interpreted mode)

Stored and executed later whenever needed (compiled mode)

Introduction to Query Processing (2)

- A query expressed in a high-level query language such as SQL must first be scanned, parsed, and validated
- The **scanner - identifies the query** tokens—such as SQL keywords, attribute names, and relation names
- The **parser - checks the query syntax to determine** whether it is formulated according to the syntax rules
- The query must also be **validated by checking that all attribute and** relation names are valid and semantically meaningful names
- An internal representation of the query is then created, usually as a tree data structure called a **query tree**
- The DBMS must then devise an **execution strategy or query plan for retrieving the results of the** query from the database files
- A query typically has many possible execution strategies, and the process of choosing a suitable one for processing a query is known as **query optimization**
- The **runtime database processor** has the task of running (executing) the query code, whether in compiled or interpreted mode, to produce the query result

Query processing in Database



Cost-based query optimization Measures of Query Cost:

- **Cost-based query optimization**:
  - The main aim of query optimization is to choose the most efficient way of implementing the relational algebra operations at the lowest possible cost.
  - Estimate and compare the costs of executing a query using different execution strategies and choose the strategy with the lowest cost estimate.
  - The method of optimizing the query by choosing a strategy those results in minimum cost is called cost-based query optimization

Cost-based query optimization Measures of Query Cost:

- **Cost Components for Query Execution**
  - Access cost to secondary storage
    - Disk I/O Cost – cost of transferring blocks to and from disk to memory
    - The cost of searching for records in a disk file (access structure)
    - Contiguous/non-contiguous blocks allocation
  - Storage cost
    - cost of storing on disk any intermediate files that are generated by an execution strategy for the query
  - Computation cost
    - Cost of performing in-memory operations on the records within the data buffers during query execution

Cost-based query optimization
Measures of Query Cost:

- **Cost Components for Query Execution (contd..)**
  - Memory usage cost
    - Cost pertaining to the number of main memory buffers needed during query execution
  - Communication cost
    - The cost of shipping the query and its results from the database site to the site or terminal where the query originated

**Note: Different database systems may focus on different cost components.

Translating SQL Queries into Relational Algebra (1)

- **Query block**: An SQL query is first translated into an equivalent extended relational algebra expression—represented as a query tree data structure—that is then optimized.
  - The basic unit that can be translated into the algebraic operators and optimized.
- A query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clause if these are part of the block.
- **Nested queries** within a query are identified as separate query blocks.
- Aggregate operators in SQL must be included in the extended algebra.

Translating SQL Queries into Relational Algebra (2)

```
SELECT   LNAME, FNAME
FROM     EMPLOYEE
WHERE    SALARY > (  SELECT   MAX (SALARY)
                     FROM     EMPLOYEE
                     WHERE    DNO = 5);
```

```
SELECT   LNAME, FNAME
FROM     EMPLOYEE
WHERE    SALARY > C
```

```
SELECT   MAX (SALARY)
FROM     EMPLOYEE
WHERE    DNO = 5
```

$$\pi_{\text{LNAME, FNAME}} (\sigma_{\text{SALARY}>C}(\text{EMPLOYEE}))$$

$$\mathcal{F}_{\text{MAX SALARY}} (\sigma_{\text{DNO}=5} (\text{EMPLOYEE}))$$

Translating SQL Queries into Relational Algebra (3)

- The *query optimizer* would then choose an execution plan for each query block. Notice that in the above example, the inner block needs to be evaluated only once to produce the maximum salary of employees in department 5, which is then used—as the constant c—by the outer block.

Algorithms for SELECT Operations (1)

- Implementing the SELECT Operation
  - A number of search algorithms are possible for selecting records from a file. These are also known as **file scans, because** they scan the records of a file to search for and retrieve records that satisfy a selection condition
  - If the search algorithm involves the use of an index, the index search is called an **index scan**
- Examples:
  - (OP1): $\sigma_{SSN='123456789'}(EMPLOYEE)$
  - (OP2): $\sigma_{DNUMBER>5}(DEPARTMENT)$
  - (OP3): $\sigma_{DNO=5}(EMPLOYEE)$
  - (OP4): $\sigma_{DNO=5\ AND\ SALARY>30000\ AND\ SEX=F}(EMPLOYEE)$
  - (OP5): $\sigma_{ESSN=123456789\ AND\ PNO=10}(WORKS\_ON)$

Algorithms for SELECT Operations (2)

Implementing the SELECT Operation (contd.):

- Search Methods for Simple Selection:
  - S1 **Linear search** (brute force):
    - Retrieve every record in the file, and test whether its **attribute values satisfy the selection condition**.
    - Since the records are grouped into disk blocks, each disk block is read into a main memory buffer, and then a search through the records within the disk block is conducted in main memory.

Algorithms for SELECT Operations (3)

Implementing the SELECT Operation (contd.):

- Search Methods for Simple Selection:
  - S2 **Binary search**:
    - If the selection condition involves an equality comparison on a key attribute on which the file is ordered, binary search (which is more efficient than linear search) can be used.
    - For ex. (OP1): $\sigma_{SSN='123456789'}(EMPLOYEE)$

Algorithms for SELECT Operations (4)

Implementing the SELECT Operation (contd.):

- Search Methods for Simple Selection:
  - S3 **Using a primary index or hash key to retrieve a single record**:
    - If the selection condition involves an equality comparison on a key attribute with a primary index (or a hash key), use the primary index (or the hash key) to retrieve the record.
    - For ex. (OP1): $\sigma_{SSN='123456789'}(EMPLOYEE)$

(Note*** : Single record = equality condition)

Algorithms for SELECT Operations (5)

Implementing the SELECT Operation (contd.):

- Search Methods for Simple Selection:
  - S4 **Using a primary index to retrieve multiple records**:
    - If the comparison condition is >, ≥, <, or ≤ on a key field with a primary index, use the index to find the record satisfying the corresponding equality condition, then retrieve all subsequent records in the (ordered) file.
    - For ex. (OP2): $\sigma_{DNUMBER>5}$(DEPARTMENT) – use initially index to satisfy equality condition $\sigma_{DNUMBER} = 5$ then find records which are $\sigma_{DNUMBER > 5}$

Algorithms for SELECT Operations (6)

Implementing the SELECT Operation (contd.):

- Search Methods for Simple Selection:
  - S5 **Using a clustering index to retrieve multiple records**:
    - If the selection condition involves an equality comparison on a non-key attribute with a clustering index, use the clustering index to retrieve all the records satisfying the selection condition.
    - For ex. (OP3): $\sigma_{DNO=5}$(EMPLOYEE) - use the index to retrieve all the records satisfying the condition.

(Note*** : Multiple records = comparison condition)

Algorithms for SELECT Operations (7)

Implementing the SELECT Operation (contd.):

- Search Methods for Simple Selection:
  - S6 **Using a secondary (B+-tree) index**:
    - On an equality comparison, this search method can be used to retrieve a single record if the indexing field has unique values (is a key) or to retrieve multiple records if the indexing field is not a key.
    - In addition, it can be used to retrieve records on conditions involving >,>=, <, or <=. (FOR RANGE QUERIES)

Algorithms for SELECT Operations (8)

Implementing the SELECT Operation (contd.):

- Search Methods for Simple Selection:
  - S7 **Conjunctive selection**:
    - If an attribute involved in any single simple condition in the conjunctive condition has an access path that permits the use of one of the methods S2 to S6, use that condition to retrieve the records and then check whether each retrieved record satisfies the remaining simple conditions in the conjunctive condition.
    - For ex. (OP4): $\sigma_{\text{DNO=5 AND SALARY>30000 AND SEX=F}}(\text{EMPLOYEE})$

Algorithms for SELECT Operations (9)

Implementing the SELECT Operation (contd.):

- Search Methods for Simple Selection:
  - S8 **Conjunctive selection using a composite index**
    - If two or more attributes are involved in equality conditions in the conjunctive condition and a composite index (or hash structure) exists on the combined field, we can use the index directly.
    - for example, if an index has been created on the composite key (Essn, Pno) of the WORKS_ON file for OP5—we can use the index directly.
    - For ex. (OP5): $\sigma_{ESSN=123456789\ AND\ PNO=10}(WORKS\_ON)$

Algorithms for SELECT Operations (10)

- Implementing the SELECT Operation (contd.):
- Search Methods for Complex Selection:
  - S9 **Conjunctive selection by intersection of record pointers**:
    - This method is possible if secondary indexes are available on all (or some of) the fields involved in equality comparison conditions in the conjunctive condition and if the indexes include record pointers (rather than block pointers).
    - Each index can be used to retrieve the record pointers that satisfy the individual condition.
    - The intersection of these sets of record pointers gives the record pointers that satisfy the conjunctive condition, which are then used to retrieve those records directly.
    - If only some of the conditions have secondary indexes, each retrieved record is further tested to determine whether it satisfies the remaining conditions.

Algorithms for SELECT Operations (11)

- Implementing the SELECT Operation (contd.):

  In brief:

  - Whenever a **single condition** specifies the selection, we can only check whether an access path exists on the attribute involved in that condition.
    - If an access path exists, the method corresponding to that access path is used; otherwise, the "brute force" linear search approach of method S1 is used. (See OP1, OP2 and OP3)
  - For **conjunctive selection conditions**, whenever *more than one* of the attributes involved in the conditions have an access path, query optimization should be done to choose the access path that *retrieves the fewest records* in the most efficient way.

Algorithms for SELECT Operations (12)

- Implementing the SELECT Operation (contd.):
  - **Disjunctive selection conditions**
    - For ex. (OP4): $\sigma_{\text{DNO=5 OR SALARY>30000 OR SEX=F}}(\text{EMPLOYEE})$
    - The disjunctive condition are the *union of the records satisfying the individual* conditions
    - If any one of the conditions does not have an access path, use linear search approach
    - If an access path exists on *every simple condition in the disjunction,* optimize the selection by retrieving the records satisfying each condition—or their record ids—and then applying the *union operation to eliminate duplicates.*

Algorithms for PROJECT Operations (1)

- Algorithm for PROJECT operations (Figure 15.3b) $\pi_{\text{<attribute list>}}(R)$
  1. If <attribute list> has a key of relation R, extract all tuples from R with only the values for the attributes in <attribute list>.
  2. If <attribute list> does NOT include a key of relation R, duplicated tuples must be removed from the results.
- Methods to remove duplicate tuples
  1. Sorting
  2. Hashing

Algorithms for PROJECT Operations (2)

- Methods to remove duplicate tuples
  1. Sorting

     the result of the operation and then eliminating duplicate tuples, which appear consecutively after sorting.

  2. Hashing

     search record is hashed and inserted into a bucket of the hash file in memory, it is checked against those records already in the bucket; if it is a duplicate, it is not inserted in the bucket.

Reference

- Navathe