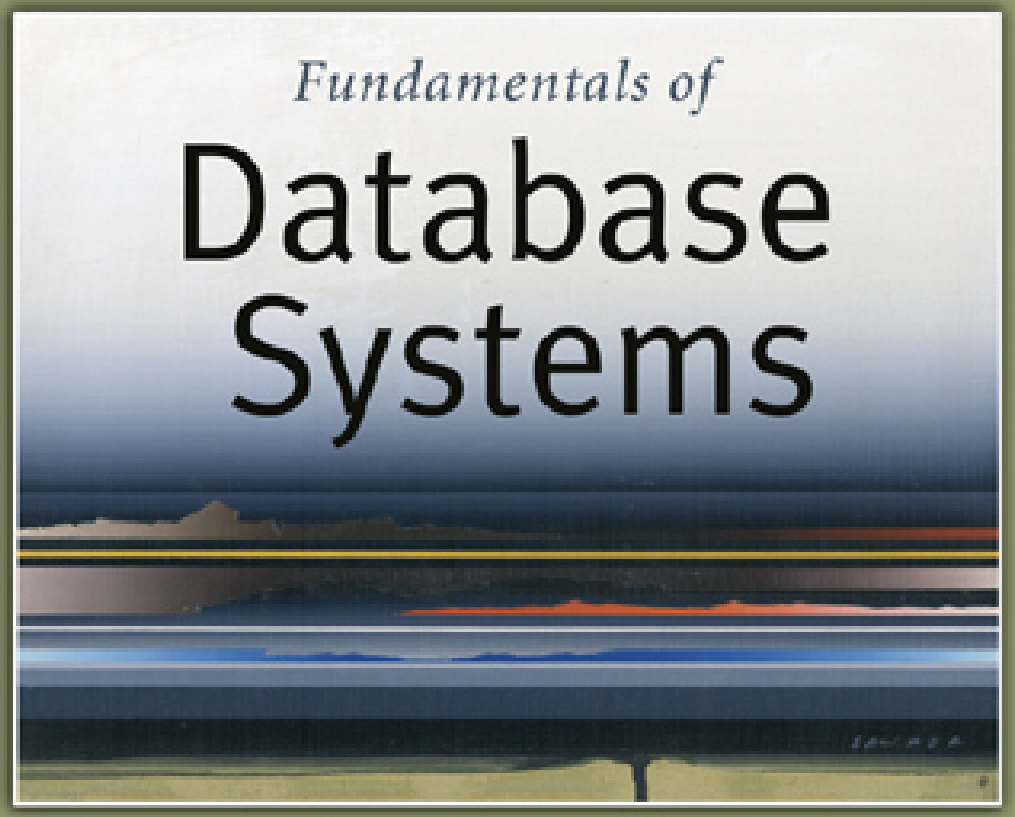
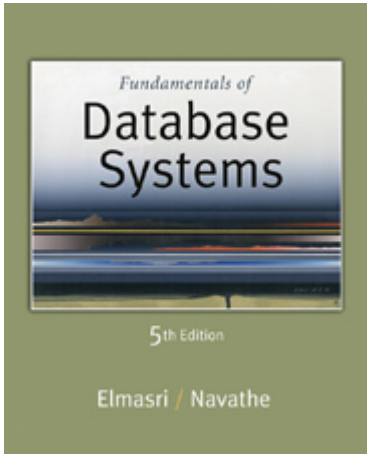


5th Edition

Elmasri / Navathe

Chapter 16

Practical Database Design and Tuning



Chapter Outline

1. Physical Database Design in Relational Databases

2. An Overview of Database Tuning in Relational Systems

1. Physical Database Design in Relational Databases(1)

- Factors that Influence Physical Database Design:
  A. **Analyzing the database queries and transactions**
  - For each query, the following information is needed.
    - The *files* that will be accessed by the query;
    - The *attributes* on which any *selection* conditions for the query are specified;
    - The *attributes* on which any *join* conditions or conditions to link multiple tables or objects for the query are specified;
    - The *attributes* whose values will be *retrieved* by the query.
  - Note: the attributes listed in items 2 and 3 above are candidates for definition of access structures.

Physical Database Design in Relational Databases(2)

- Factors that Influence Physical Database Design (contd.):
  A. **Analyzing the database queries and transactions** (contd.)
  - For each **update** transaction or operation, the following information is needed.
    1. The files that will be updated;
    2. The type of operation on each file (insert, update or delete);
    3. The attributes on which selection conditions for a delete or update operation are specified;
    4. The attributes whose values will be changed by an update operation.
  - Note: the attributes listed in items 3 above are candidates for definition of access structures. However, the attributes listed in item 4 are candidates for avoiding an access structure.

Physical Database Design in Relational Databases(3)

- Factors that Influence Physical Database Design (contd.):
  B. **Analyzing the expected frequency of invocation of queries and transactions**
  - The expected frequency information, along with the attribute information collected on each query and transaction, is used to compile a cumulative list of expected frequency of use for all the queries and transactions.
  - It is expressed as the expected frequency of using each attribute in each file as a selection attribute or join attribute, over all the queries and transactions.
  - **80-20 rule**
    - 20% of the data is accessed 80% of the time

Physical Database Design in Relational Databases(4)

- Factors that Influence Physical Database Design (contd.)
  C. **Analyzing the time constraints of queries and transactions**
  - Performance constraints place further priorities on the attributes that are candidates for access paths.
  - The selection attributes used by queries and transactions with time constraints become higher-priority candidates for primary access structure.

Physical Database Design in Relational Databases(4)

- Factors that Influence Physical Database Design (contd.)
  D. **Analyzing the expected frequencies of update operations**
  - A minimum number of access paths should be specified for a file that is updated frequently.

Physical Database Design in Relational Databases(4)

- Factors that Influence Physical Database Design (contd.)
  E. **Analyzing the uniqueness constraints on attributes**
  - Access paths should be specified on all candidate key attributes — or set of attributes — that are either the primary key or constrained to be unique.

Physical Database Design in Relational Databases(5)

- Physical Database Design Decisions
- Design decisions about indexing
  - Whether to index an attribute?
  - What attribute or attributes to index on?
  - Whether to set up a clustered index?
  - Whether to use a hash index over a tree index?
  - Whether to use dynamic hashing for the file?

Physical Database Design in Relational Databases(6)

- Physical Database Design Decisions (contd.)
- Denormalization as a design decision for speeding up queries
  - The goal of normalization is to separate the logically related attributes into tables to minimize redundancy and thereby avoid the update anomalies that cause an extra processing overheard to maintain consistency of the database.
  - The goal of denormalization is to improve the performance of frequently occurring queries and transactions. (Typically the designer adds to a table attributes that are needed for answering queries or producing reports so that a join with another table is avoided.)
  - Trade off between update and query performance

2. An Overview of Database Tuning in Relational Systems (1)

- **Tuning**:
  - The process of continuing to revise/adjust the physical database design by monitoring resource utilization as well as internal DBMS processing to reveal bottlenecks such as contention for the same data or devices.

- Goal:
  - To make application run faster
  - To lower the response time of queries/transactions
  - To improve the overall throughput of transactions

An Overview of Database Tuning in Relational Systems (2)

- Statistics internally collected in DBMSs:
  - Size of individual tables
  - Number of distinct values in a column
  - The number of times a particular query or transaction is submitted/executed in an interval of time
  - The times required for different phases of query and transaction processing
- Statistics obtained from monitoring:
  - Storage statistics
  - I/O and device performance statistics
  - Query/transaction processing statistics
  - Locking/logging related statistics
  - Index statistic

An Overview of Database Tuning in Relational Systems (3)

- Problems to be considered in tuning:
  - How to avoid excessive lock contention?
  - How to minimize overhead of logging and unnecessary dumping of data?
  - How to optimize buffer size and scheduling of processes?
  - How to allocate resources such as disks, RAM and processes for most efficient utilization?

An Overview of Database Tuning in Relational Systems (4)

- Tuning Indexes
  - Reasons to tuning indexes
    - Certain queries may take too long to run for lack of an index;
    - Certain indexes may not get utilized at all;
    - Certain indexes may be causing excessive overhead because the index is on an attribute that undergoes frequent changes
  - Options to tuning indexes
    - Drop or/and build new indexes
    - Change a non-clustered index to a clustered index (and vice versa)
    - Rebuilding the index

An Overview of Database Tuning in Relational Systems (5)

- Tuning the Database Design
  - Dynamically changed processing requirements need to be addressed by making changes to the conceptual schema if necessary and to reflect those changes into the logical schema and physical design.

An Overview of Database Tuning in Relational Systems (6)

- Tuning the Database Design (contd.)
  - Possible changes to the database design
    - Existing tables may be joined (denormalized) because certain attributes from two or more tables are frequently needed together.
    - For the given set of tables, there may be alternative design choices, all of which achieve 3NF or BCNF. One may be replaced by the other.
    - A relation of the form R(K, A, B, C, D, …) that is in BCNF can be stored into multiple tables that are also in BCNF by replicating the key K in each table.
    - Attribute(s) from one table may be repeated in another even though this creates redundancy and potential anomalies.
    - Apply **horizontal partitioning** as well as **vertical partitioning** if necessary.

An Overview of Database Tuning in Relational Systems (7)

- Tuning Queries
  - Indications for tuning queries
    - A query issues too many disk accesses
    - The query plan shows that relevant indexes are not being used.

An Overview of Database Tuning in Relational Systems (8)

- Tuning Queries (contd.): Typical instances for query tuning
  - In some situations involving using of correlated queries, temporaries are useful.
  - If multiple options for join condition are possible, choose one that uses a clustering index and avoid those that contain string comparisons.
  - The order of tables in the FROM clause may affect the join processing.
  - Some query optimizers perform worse on nested queries compared to their equivalent un-nested counterparts.
  - Many applications are based on views that define the data of interest to those applications. Sometimes these views become an overkill.

An Overview of Database Tuning in Relational Systems (8)

- Tuning Queries (contd.): Typical instances for query tuning
  - In some situations involving using of correlated queries, temporaries are useful.
  - If multiple options for join condition are possible, choose one that uses a clustering index and avoid those that contain string comparisons.
  - The order of tables in the FROM clause may affect the join processing.
  - Some query optimizers perform worse on nested queries compared to their equivalent un-nested counterparts.
  - Many applications are based on views that define the data of interest to those applications. Sometimes these views become an overkill.

An Overview of Database Tuning in Relational Systems (10)

- Additional Query Tuning Guidelines
  - A query with multiple selection conditions that are connected via OR may not be prompting the query optimizer to use any index. Such a query may be split up and expressed as a union of queries, each with a condition on an attribute that causes an index to be used.
  - Apply the following transformations
    - NOT condition may be transformed into a positive expression.
    - Embedded SELECT blocks may be replaced by joins.
    - If an equality join is set up between two tables, the range predicate on the joining attribute set up in one table may be repeated for the other table
  - WHERE conditions may be rewritten to utilize the indexes on multiple columns.

Summary

- Physical Database Design in Relational Databases
- Database Tuning in Relational Systems