Transactions, Concurrency, Recovery and Database Performance and Tuning

Objectives

- Understand the goals of database tuning
- Identify the two major areas in which tuning can done and the input to tuning process
- Understand index and database tuning, and how to de-normalize the relation by splitting into simpler relations
- Understand the process of optimizing and executing a high-level query
- Understand the concept of a transaction
- Appreciate the need for concurrency control
- Characterize transaction schedules as serial, non-serial and conflict-serializable
- Understand locking techniques as applied to concurrency control
- Understand the basic recovery concepts.

Transaction Processing Systems

- In today's world, large databases with hundreds of concurrent users are the norm (such as bank ATM networks or on-line flight booking systems).
- These are often known as **Transaction Processing Systems**.

Overview of Database Tuning

- Measuring the performance of a database is a very difficult task.
- Performance of database management depends on such factors as:
  - Database Software Algorithms
  - Operating Systems Overhead
  - Speed of Secondary Storage
  - Network Latency

Major Area in Database Tuning

- As a Database Designer rather than a Systems Programmer, two major areas in which tuning can be done are:
  - Considering Alternative Designs
  - Query Optimizations

Goals of Database Tuning

- To make applications run faster.
- To lower the response time of queries/transactions.
- To improve the overall throughput of the transactions.

Inputs to the tuning process

- The inputs to the tuning process include statistics related to the physical database design.
- In particular a DBMS can collect the following statistics internally:
  - Size of each table
  - Number of distinct (unique) values in a column
  - The number of times a particular query or transaction is submitted/executed in an interval of time
  - The time required for different query and transaction processes

Information to be Obtained

- Other information obtained from monitoring the database system activities and processes includes the following:
  - Storage statistics
  - I/O and device performance statistics
  - Query/transaction processing statistics
  - Locking/logging related statistics
  - Index statistics

Problem to be Deal with

- Tuning of database involves dealing with the following types of problems:
  - Avoid excessive lock contention and improve concurrency transactions
  - Minimize logging and unnecessary dumping of data
  - Optimizing buffer size and scheduling of processes
  - Allocate resources such as disks, RAM, and processes for efficient utilization

Revised of the Initial Index

- The initial indexes may have to be revised for the following reasons:
  - Queries may take more time for the lack of an index
  - Certain indexes may not be utilized at all
  - Indexes may causes overhead, if the index is on an attribute that undergoes frequent changes
  - Indexes may be dropped and may be created based on the tuning analysis
  - Dropping or creating indexes may cause overhead at the cost of improved performance
  - Beside dropping and creating indexes, rebuilding the index may improve performance

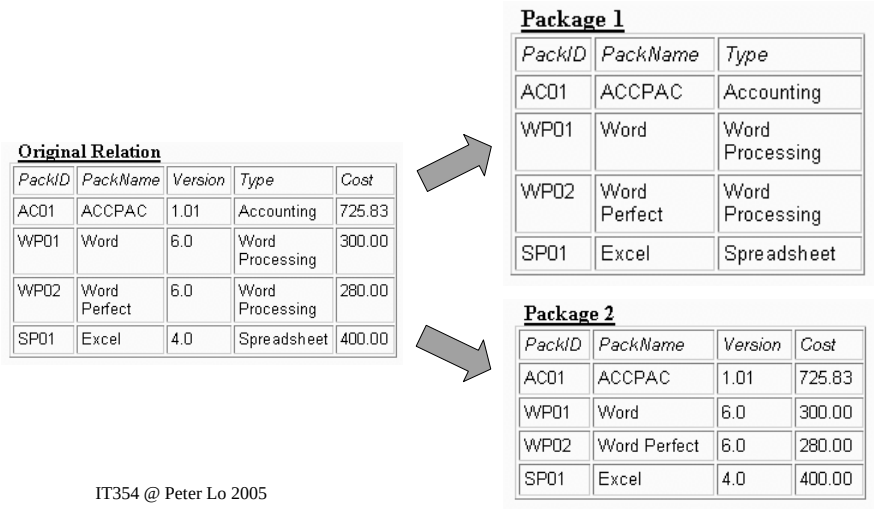
Database Tuning – Further Examples

- Database tuning can be achieved by making changes to the conceptual schema if necessary and then reflecting those changes in the logical schema and physical design.
- These changes may be one of the following:
  - Splitting Relations (Vertical Partitioning)
  - De-normalization (Joining Tables)
  - Different Normalization (Alternative Design)

Splitting Relations

- It is possible to split or combine relations in order to increase the performance.
- A relation may be split into two or more relations so that each has the same key.

Example



Original Relation

PackID	*PackName*	*Version*	*Type*	*Cost*
AC01	ACCPAC	1.01	Accounting	725.83
WP01	Word	6.0	Word Processing	300.00
WP02	Word Perfect	6.0	Word Processing	280.00
SP01	Excel	4.0	Spreadsheet	400.00

Package 1

PackID	*PackName*	*Type*
AC01	ACCPAC	Accounting
WP01	Word	Word Processing
WP02	Word Perfect	Word Processing
SP01	Excel	Spreadsheet

Package 2

PackID	*PackName*	*Version*	*Cost*
AC01	ACCPAC	1.01	725.83
WP01	Word	6.0	300.00
WP02	Word Perfect	6.0	280.00
SP01	Excel	4.0	400.00

Advantages and Disadvantages

- Advantages:
  - If certain columns are accessed heavily while the other columns are not then it saves on data access, data transfer time and sometimes also, space.
- Disadvantages:
  - If any transactions require both tables then we would need to do a join operator, which may not be efficient.

De-normalization

- Combining the relations is exactly the opposite of splitting them.
- Normalization reduces data duplications, but it may increase number of disk accesses.
- In some cases it is efficient to do de-normalization.
- **De-normalization** means bringing the data back to the previous normal form from the existing normal form.

Example

PackID	Tag Num
AC01	3208
AC01	3269
WP01	5771
WP01	2292
WP01	3333
WP02	2277
SP01	88923

PackID	PackName	Version	Type	Cost
AC01	ACCPAC	1.01	Accounting	725.83
WP01	Word	6.0	Word Processing	300.00
WP02	Word Perfect	6.0	Word Processing	280.00
SP01	Excel	4.0	Spread Sheet	400.00

PackID	PackName	Version	Type	Cost	Tag Num
AC01	ACCPAC	1.01	Accounting	725.83	3208
AC01	ACCPAC	1.01	Accounting	725.83	32961
WP01	Word	6.0	Word Processing	300.00	5771
WP01	Word	6.0	Word Processing	300.00	3333
WP01	Word	6.0	Word Processing	300.00	2292
WP01	Word Perfect	6.0	Word Processing	280.00	2277
SP01	Excel	4.0	Spreads Sheet	400.00	88923

Advantages and Disadvantages

- Advantages:
  - If the data is in 1NF we need to join two tables first and then do the selection.
  - But using the above table it is a simple retrieval.
- Disadvantages:
  - Handling repeating groups is difficult.

Different Normalization

- This can happen when for the same relation and functional dependencies there are many alternate normal forms available.
- Again the criteria for selecting the optimal design are the query patterns and updates.
- An advantage of using different normalization rather than de-normalization is that it does not affect the non-redundancy property whereas de-normalization will.
- In tuning the database both different normalization and de-normalization techniques can be used.
- It is always advisable to try different normalization first since it will not affect the non-redundant property.

Example

- Consider the relation:
  - SOFTWARE (Pack#, Tag#, P-Name, P-Ver, P-Cost, Soft-Cost)
- The following functional dependencies apply to this relation:
  - (Pack#, Tag#) → P-Name, P-Ver, P-Cost, Soft-Cost
  - Tag# → P-Cost, Soft-Cost
  - Pack# → P-Cost, Soft-Cost

Example

- So we can normalize this relation into:
  - SOFTWARE (Pack#, Tag#, P-Name, P-Ver)
  - TAG (Tag#, P-Cost, Soft-Cost)
- OR
  - SOFTWARE (Pack#, Tag#, P-Name, P-Ver)
  - PACKAGE (Pack#, P-Cost, Soft-Cost)

Example

- Which design should we choose?
  - You have to look at the query patterns.
  - If there are more queries involved with Pack#, P-Cost and Soft-Cost than that of Tag#, P-Cost, Soft-Cost then it is better to select the second design relation

Translating an SQL Query into Relation Algebra

- An SQL query is first translated into an equivalent extended relational algebra expression that is represented as a query tree data structure and it is then optimized.
- Typically, SQL queries are decomposed into query blocks, which form basic blocks and are then optimized.

Basic Query Operations

- An RDBMS must include basic algorithms for implementing the different types of relational operations.

External Sorting

- Sorting is one of the primary algorithms used in query processing.
- Sorting is also a key component in sort-merge algorithms used for JOIN and other operations.
- External sorting refers to sorting algorithms that are suitable for large files of records stored in disk that do not fit entirely in main memory.

Sort-merge Strategy

- The sorting algorithm sorts the small sub files known as **Runs** of the main files and then merges the sorted runs (sub files) into large sorted sub files that are merged in turn in to files.
- This strategy consists of two phases:
  - The Sorting Phase
  - The Merging Phase

Sorting Phase

- Runs (sub files) of files are read into main memory, sorted and written back into disk as temporary sorted runs (or sub files).
- The size of the run is indicated by the available buffer space (nB).
- The number of initial run (nR) are indicated by the number of file blocks (b) and the available buffer space (nB).
- Then nR = (b / nR).

Merging Phase

- The sorted runs are merged during one or more passes.
- The degree of merging (dM) is the number of runs that can be merged together in each pass.

SELECT Operation

- There are several options for executing SELECT operations and depends on the file having a specific access path.
- Some of the search algorithms used to implement SELECT operations:
  - Linear Search, Binary Search, Primary Index (Hash Key), Primary Index to Retrieve Multiple Records, Clustering Index to Retrieve Multiple Records, Secondary (B+ Tree) Index, Conjunctive Selection using an Individual Index, Conjunctive Selection using a Composite Index, Conjunctive Selection by Intersection of Record Pointers

Linear Search

- Retrieve every record in the file, and test whether its attributes values satisfy the selection condition.

Binary Search

- If the selection condition involves an equality comparison on a key attribute, binary search is more efficient than linear search.

Primary Index (Hash Key)

- Uses primary index, if the selection condition involves an equality comparison on key attribute. This condition retrieves a single record (at most).

Primary Index to Retrieve Multiple Records

- Use the index to find the record satisfying the corresponding equality condition, the retrieve all the subsequent or the preceding records.

Clustering Index to Retrieve Multiple Records

- Used if the selection condition involves an equality comparison on a non-key attribute.

Secondary (B+ Tree) Index

- Used to retrieve a single record or multiple records if the indexing field is a key or a non-key respectively.

Conjunctive Selection using an Individual Index

- Used when an attribute in any single simple condition has an access path that allows the use of one of the last five methods above, use that method then check to see if the record satisfies the remaining conditions.

Conjunctive Selection using a Composite Index

- Used when two or more attributes are involved in equality conditions in the condition and a composite index exists on the combined attributes.

Conjunctive Selection by Intersection of Record Pointers

- Used when secondary indexes are available on more than one attribute and the indexes include record pointers.

JOIN Operation

- Nested-loop Join
- Single-loop Join
- Sort-merge Join

Nested-loop Join

- For every record t in R (outer loop), retrieve every record s from S (inner loop) and test whether the two records satisfy the join condition.

Single-loop Join

- If an index exists for one of the two join attributes, retrieve each record t in R, one at a time (single loop), and then retrieve directly all matching records s from S that satisfy the join condition.

Sort-merge Join

- If the records of R and S are physically sorted by value of the join attributes A and B.
- Both files are scanned concurrently in order of the join attributes, matching the records that have the same values for A and B.

PROJECT and Set Operations

- A project operation is straightforward to implement if the <attribute list> includes the key of relation R.
- The result of the operation will have the same number of tuples as R, but only the values for the <attributes list> in each tuple.
- If <attribute list> does not include the key of R, duplicate tuples must be eliminated.
- Hashing can also be used to eliminate duplicates.
- It is useful to recall here that in SQL queries, only if the DISTINCT is included are duplicates eliminated from the query result.

Indexes

- The two types of indexes used for aggregate operations are:
  - **Dense Index**: an index entry for every record in the main file.
  - **Non Dense Index**: the actual number of records associated with each index entry is used for correct computation.

GROUP BY Operation

- When GROUP BY clause is used in a query, aggregate operator must be applied separately to each group of tuples.
- The usual technique for such queries is to first use either sorting or hashing on the grouping attributes to partition the file into the appropriate groups.

Heuristics in Query Optimization

- Used heuristics rules to modify the internal presentation of a query, which is the form of a query tree or a query graph data structure to improve the performance.
- One of the main heuristics rule is to apply SELECT and PROJECT operations before applying the JOIN or other binary operations.
- The SELECT and PROJECT operations reduce the size of a file and hence should be applied before a JOIN or other binary operation.

Heuristics in Query Optimization

- The query tree and query graph notations are used as basis for the data structures that are used for internal representation of queries.
- A query tree is used to represent relation algebra or extended relational algebra expression.
- Whereas a query graph is used to represent a relational calculus expression.

Selectivity and Cost Estimates in Query Optimization

- A query optimization should not depend solely on heuristic rules; it should also estimate and compare the costs of executing a query using different execution strategies and should choose the strategy with the lowest cost estimate.
- It is important to limit the number of execute strategies to be considered; otherwise, too much time will be spent making cost estimates for the many possible executions strategies.

Selectivity and Cost Estimates in Query Optimization

- This approach is more suitable for compiled queries where the optimization is done at compile time and the resulting execution strategy code is stored and executed directly at runtime.
- This approach is known as **Cost-based Query Optimization**. The cost functions used in the query optimization are estimates and not exact cost functions.

Cost Components for Query Execution

- The cost of executing a query includes the following components:
  - Access cost to secondary storage
  - Storage cost
  - Computation cost
  - Memory usage cost
  - Communication cost

Catalog Information Used in the Cost Functions

- Catalog information used in the cost functions are:
  - Number of records (Tuples)
  - Record size
  - Number of blocks
  - The number of levels of each multilevel indexes
  - Number of distinct values of an attribute