Database Administration:

The Complete Guide to Practices and Procedures

Chapter 11

Database Performance

Agenda

- Techniques for Optimizing Databases
  - Partitioning
  - Raw Partitions
  - Indexing
  - Denormalization
  - Clustering
  - Interleaving Data
  - Free Space
  - Compression
  - File placement and allocation
  - Page size (Block size)
- Database Reorganization
- Questions

Techniques for Optimizing Databases

- The database, its structure, and the objects defined in the database can impact the performance of applications and queries accessing the database.
- No amount of SQL tweaking or system tuning can optimize the performance of queries run against a poorly designed or disorganized database.

Partitioning

- A database table is a logical manifestation of a set of data that physically resides on storage.
- Each DBMS provides different mechanisms for physical files to database tables.
- The DBA must decide from among the following mapping options for each table:
  - Single table to a single file
  - Single table to multiple files
  - Multiple tables to a single file

Parallelism

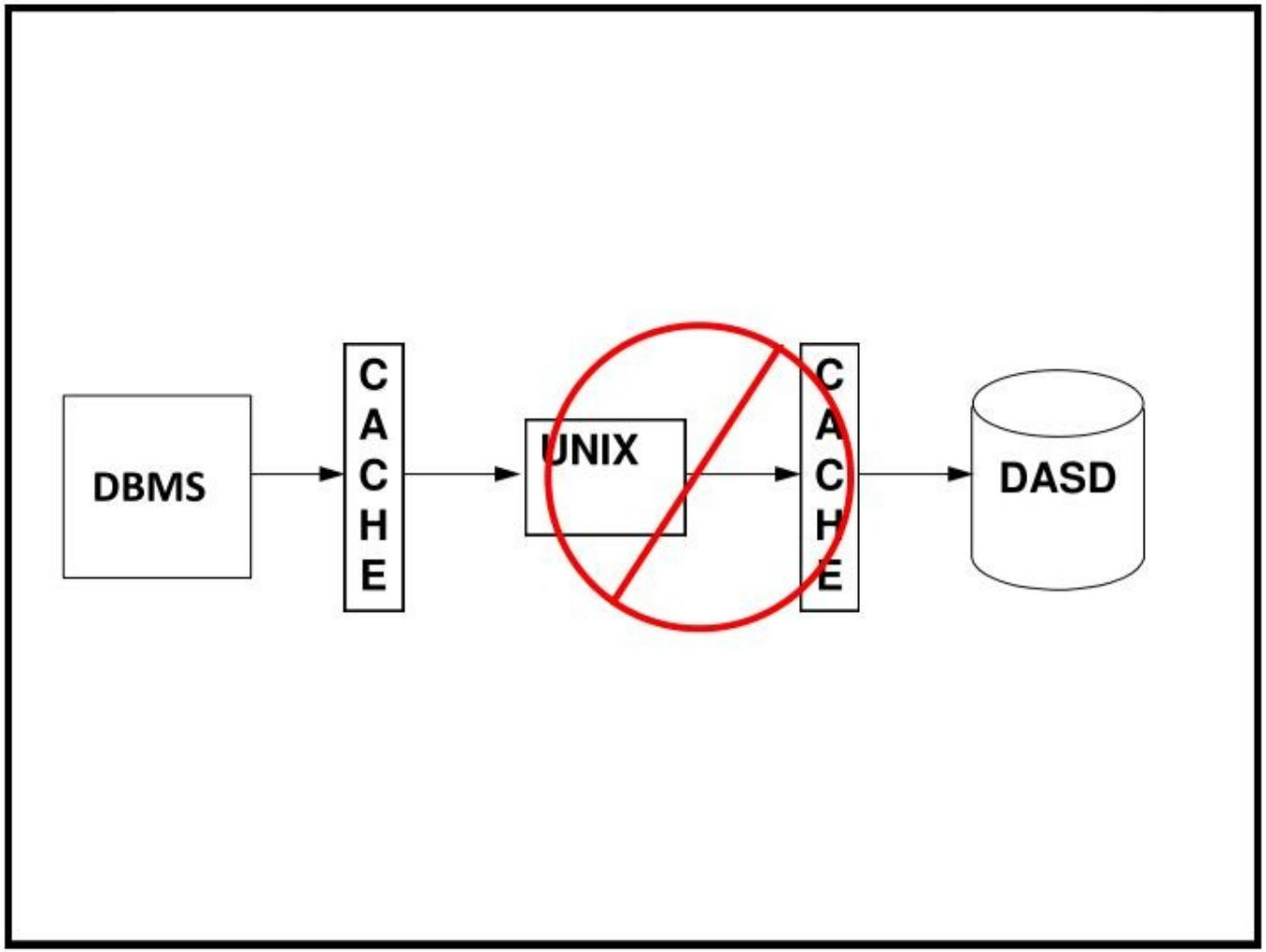
- *Parallelism* is the process of using multiple tasks to access the database in parallel.
- Partitioning helps to accomplish parallelism.
- A parallel request can be invoked to use multiple, simultaneous read engines for a single SQL statement.
  - Parallelism is desirable because it can substantially reduce the elapsed time for database queries.

Types of Parallelism

- Multiple types of parallelism are based on the resources that can be invoked in parallel.
  - A single query can be broken down into multiple requests each utilizing a different CPU core in parallel.
  - In addition, parallelism can be improved by spreading the work across multiple database instances.
  - Each DBMS offers different levels of support for parallel database queries.
    - To optimize database performance, the DBA should be cognizant of the support offered in each DBMS being managed and exploit the parallel query capabilities

A multi-core processor is a single computing component with two or more independent actual processors (called "cores").

Raw Partition vs. File System



Indexing

Indexes are used to enhance performance.

Creating the correct indexes on tables in the database is perhaps the single greatest performance tuning technique that a DBA can perform.

Indexes are particularly useful for:

- Locating rows by value(s) in column(s)
- Making joins more efficient (when the index is defined on the join columns)
- Correlating data across tables
- Aggregating data
- Sorting data to satisfy a query

Indexes Straddle the Line Between Database and Application Tuning

- Without indexes, all access to data in the database would have to be performed by scanning all available rows.
- Scans are very inefficient for very large tables.
  - Covered in the next lesson.
- Designing and creating indexes for database tables actually crosses the line between database performance tuning and application performance tuning.
- Indexes are database objects created by the DBA with database DDL. However, an index is built to make SQL statements in application programs run faster.

Impact of Adding an Index

- The DBA should have an understanding of the access patterns of the table on which the index will be built.
- Useful information includes:
  - the percentage of queries that access rather than update the table
  - the performance thresholds set within any service level agreements for queries on the table
  - the impact of adding a new index to running database utilities such as loads, reorganizations, and recovery

Indexes Impact on Performance

- Indexes can improve the performance of database queries
- Indexes will degrade the performance of database inserts and deletes
- Indexes can degrade the performance of database updates

What to Index?

- Most indexes should be designed to support multiple queries.
  - Thoroughly test the performance of the queries each index supports.
- Experiment with different index combinations and measure the results.
- Creating an index to support a single query is acceptable if that query is important enough in terms of ROI to the business (or if it is run by your boss or the CEO).
  - If the query is run infrequently, consider creating the index before the process begins and dropping the index when the process is complete.

When to Avoid Indexing

- When **all** accesses retrieve every row of the table.
  - Because every row will be retrieved every time you want to use the table an index (if used) would just add extra I/O and would decrease, not enhance performance. Though not extremely common, you may indeed come across such tables in your organization.
- For a very small table with only a few pages of data and no primary key or uniqueness requirements.
  - A very small table (perhaps 10 or 20 pages) might not need an index because simply reading all of the pages is very efficient already.
- When performance doesn't matter and the table is only accessed very infrequently.
  - But when do you ever have those requirements in the real world?
- You may want to avoid indexing variable-length columns. Expansion can cause indexes to consume an inordinate amount of disk space.
  - However, if variable-length columns are used in SQL WHERE clauses, the cost of disk storage must be compared to the cost of scanning. Buying some extra disk storage is usually cheaper than wasting CPU resources to scan rows.
- Low cardinality columns.
  - For example: GENDER

http://db2portal.blogspot.com/2008/09/when-not-to-index.html

Index Overloading

- Sometimes query performance can be enhanced by overloading an index.
- Consider and index on SALARY and the following SQL statement:

```
select emp_no, last_name, salary

from   employee

where  salary > 15000.00;
```

- Adding EMP_NO and LAST_NAME all data can be retrieved from the index, thereby eliminating table I/O and improving performance.

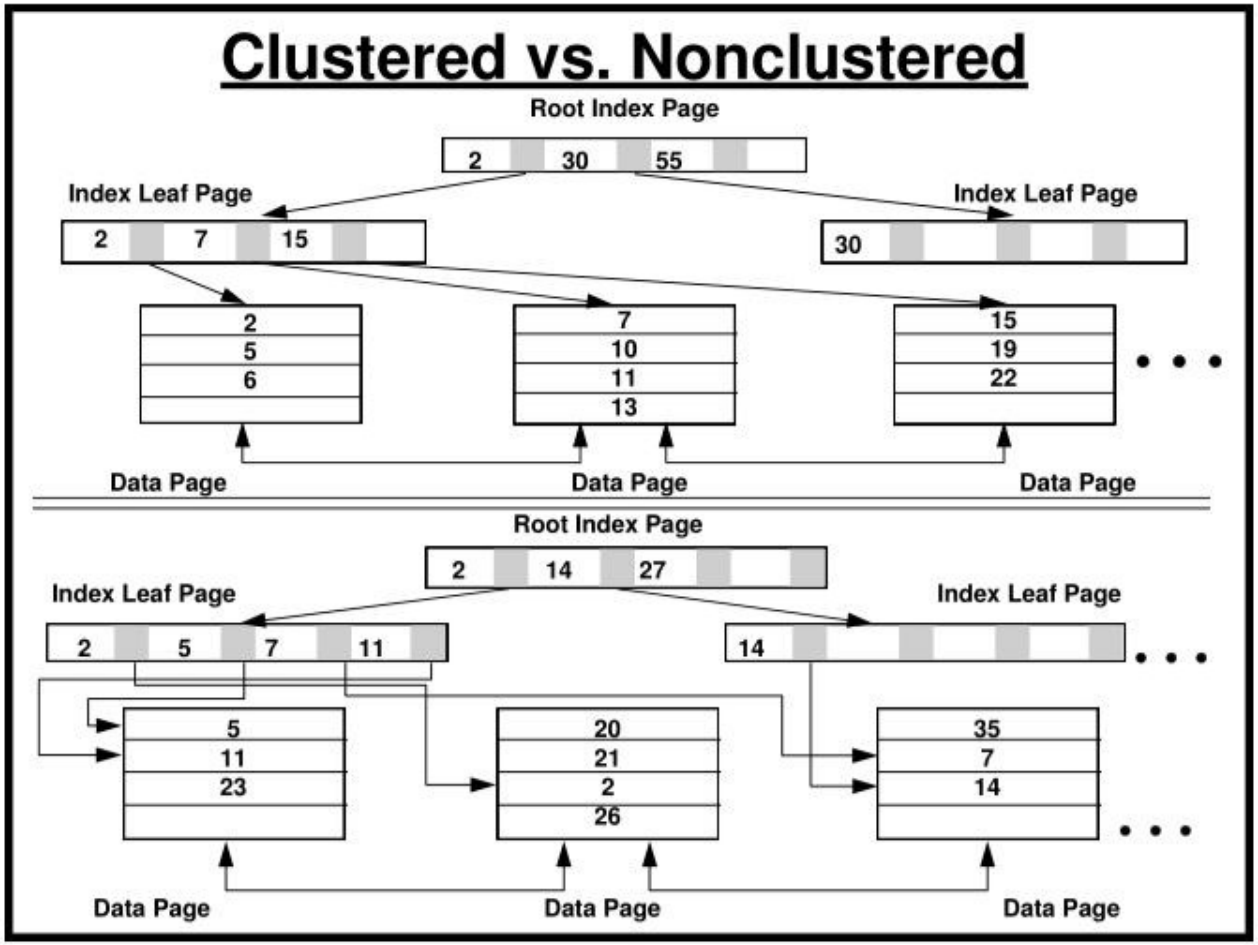
Denormalization

- We covered denormalization earlier in this course (lesson 4).

  - **Prejoined Tables** - when the cost of joining is prohibitive
  - **Report Tables** - for specialized critical reports (e.g. CEO)
  - **Mirror Tables** - when two types of environments require concurrent access to the same data **(OLTP vs DSS)**
  - **Split Tables** - when distinct groups/apps use different parts of the same table
    - Splitting **columns** across two tables for long variable character columns.
  - **Combined Tables** - to eliminate one-to-one relationships
  - **Redundant Data** - to reduce the number of joins for a single column (e.g. definitional, CA to California)
  - **Repeating Groups** - to reduce overall I/O (& possibly DASD)
  - **Derivable Data** - to eliminate calculations & aggregations
  - **Speed Tables** - to support hierarchies
  - **Physical Implementation Needs** – e.g.) to reduce page size

Clustering

- A clustered table will store its rows physically on disk in order by a specified column or columns.



Clustering Implementation

- Depending on the DBMS, the data may not always be physically maintained in exact clustering sequence.
- When a clustering sequence has been defined for a table, the DBMS will act in one of two ways to enforce clustering:
  1. When new rows are inserted, the DBMS will physically maneuver data rows and pages to fit the new rows into the defined clustering sequence; or
  2. When new rows are inserted, the DBMS will try to place the data into the defined clustering sequence, but if space is not available on the required page the data may be placed elsewhere.

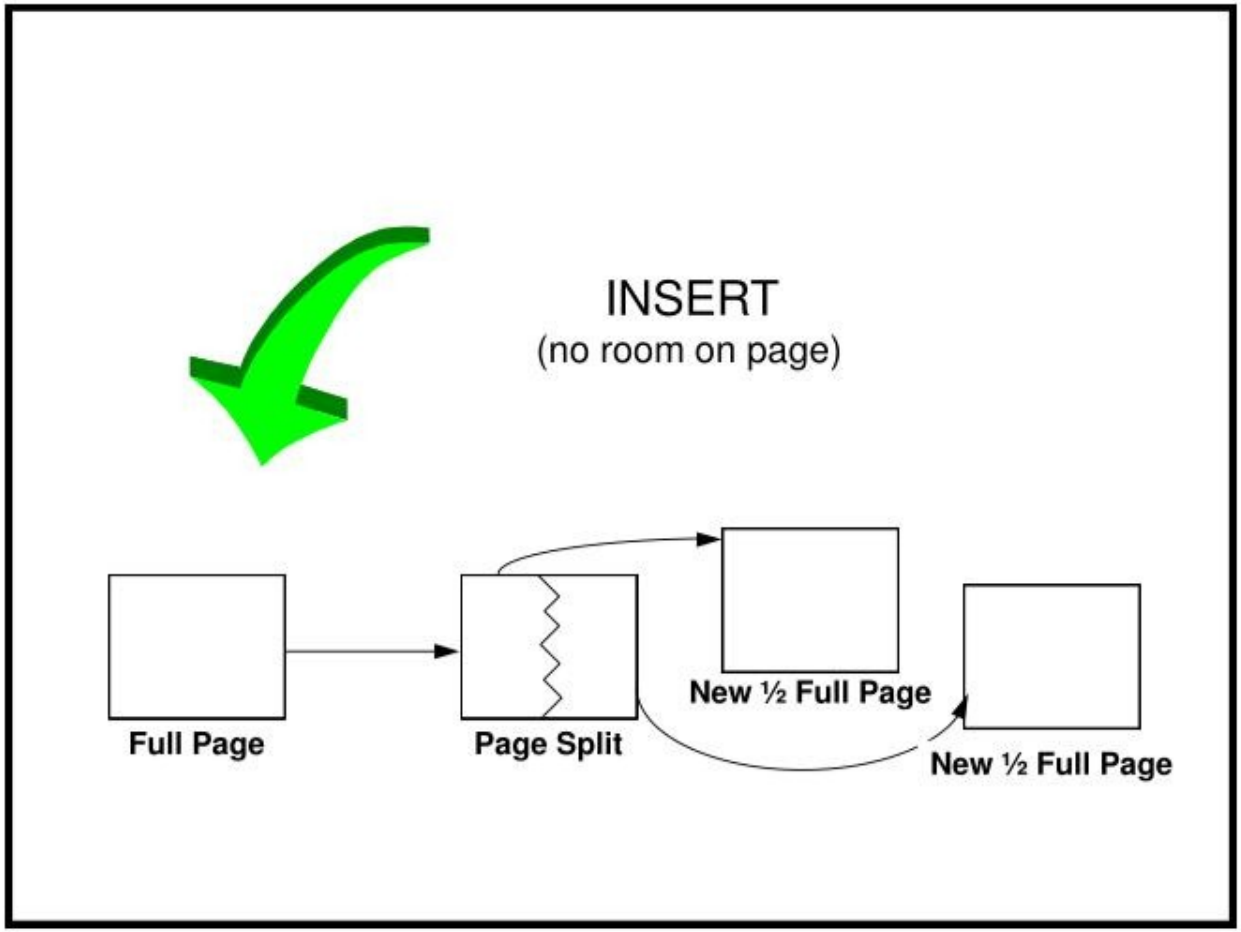
When Clustering Improve Performance

- Cluster on join columns, to optimize SQL joins where multiple rows match for one or both tables participating in the join
- Cluster on foreign key columns because they are frequently involved in joins and the DBMS accesses foreign key values during declarative referential integrity checking
- Cluster on predicates in a WHERE clause
- Cluster on range columns
- Cluster on columns that do not change often
  - reduces physically reclustering
- Cluster on columns that are frequently grouped or sorted in SQL statements

General Advice on Clustering

- As a rule of thumb: if the DBMS supports clustering, it is usually a good practice to define clustering for each table that is created.
  - Unless the table is very small.
- You can only have one clustering sequence.
  - When a table has multiple candidates for clustering, weigh the cost of sorting against the performance gained by clustering for each candidate key.
- Clustering is generally not recommended for primary key columns because the primary key is, by definition, unique.
  - However, if ranges of rows frequently are selected and ordered by primary key value, a clustering index may be beneficial.

Page Splitting



The process of creating new pages to store inserted data is called *page splitting*.

Monotonic Page Splitting

- A monotonic page split is a much simpler process, requiring only two steps. The DBMS
  1. Creates a new page in between the full page and the next page.
  2. Inserts the new values into the fresh page.
- Useful when rows are inserted strictly in ascending sequence.
  - Avoid wasted space

Interleaving Data

- Also covered earlier (lesson 4).
- When data is interleaved, rows from two tables are combined physically in a single physical sequence by a join key.
- Interleaving can be used to improve the performance of joins.
- Interleaving can be viewed as a specialized form of clustering.

Free Space

- Free space is used to leave a portion of a table space or index empty and available to store newly added data.
- Supplying appropriate free space can:
  - reduce the frequency of reorganization
  - reduce contention
  - Increase insert efficiency
- Each DBMS provides methods of specifying free space for a database object in the CREATE and ALTER statements.

Pros and Cons of Free Space

Ensuring a proper amount of free space for each database object provides the following benefits:

- Inserts are faster when free space is available.
- As new rows are inserted, they can be properly clustered.
- Variable-length rows and altered rows have room to expand, potentially reducing the number of relocated rows.
- Fewer rows on a page results in better concurrency because less data is unavailable to other users when a page is locked.

However, free space also has several disadvantages.

- Disk storage requirements are greater.
- Scans take longer.
- Fewer rows on a page can require more I/O operations to access the requested information.
- Because the number of rows per page decreases, the efficiency of data caching can decrease because fewer rows are retrieved per I/O.

Determining Correct Free Space

The correct amount of free space must be based on the following criteria:

- Frequency of inserts and modifications
- Amount of sequential versus random access
- Impact of accessing unclustered data
- Type of processing
- Likelihood of row chaining, row migration, and page splits

Static Tables and Free Space?

- Don’t do it!
- If the data does not change and no more data is being added then there is no reason for free space.
  - Defining free space just degrades performance

Compression

- Compression can be used to shrink the size of a database.
  - When compression is specified, data is algorithmically compressed upon insertion into the database and decompressed when it is read.
- Compression always requires a trade-off that the DBA must analyze.
  - On the positive side, we have disk savings and the potential for reducing I/O cost (because more rows can be stored on a single page or block).
  - On the negative side, we have the additional CPU cost required to compress and decompress the data.

File Placement and Allocation

- Data placement optimizes access by reducing contention on physical devices.
- Must understand the access patterns associated with each piece of data in the system
- Specifically allocate the data on physical disk devices in such a way as to optimize performance
- However, with modern disk systems such as RAID devices, precise file placement is often difficult, if not impossible to achieve.
  - More details on RAID is forthcoming in lesson 18.

Database Log Placement

- Placing the transaction log on a separate disk device from the actual data allows the DBA to back up the transaction log independently from the database.
  - It also minimizes dual writes to the same disk.
  - Writing data to two files on the same disk drive at the same time will degrade performance even more than reading data from two files on the same disk drive at the same time.
  - Remember, too, every database modification (write) is recorded on the database transaction log.

Distributed Data Placement

- Reduce transmission costs by placing data effectively for access.
- Data should reside at the database server where it is most likely, or most often, to be accessed.
- For example:
  - Chicago data should reside at the Chicago database server
  - Los Angeles–specific data should reside at the Los Angeles database server
  - And so on.

Disk Allocation

- The DBMS may require disk devices to be allocated for database usage.
- Specific commands are provided by each DBMS to initialize physical disk devices.
  - The disk initialization command will associate a logical name for a physical disk partition or OS file.
  - After the disk has been initialized, it is stored in the system catalog and can be used for storing table data.
- Use meaningful device names to facilitate more efficient usage and management of disk devices.

Page Size (Block Size)

- Rows are stored physically by databases in a table space page or block.
- Most DBMSs limit the size of the page/block that can be chosen.
- Examples:
  - DB2 limits pages sizes to 4K, 8K, 16K, or 32K
  - SQL Server only supports 8K
- The DBA must calculate the best page size based on row size, the number of rows per page, and free space requirements.

Page Size Example

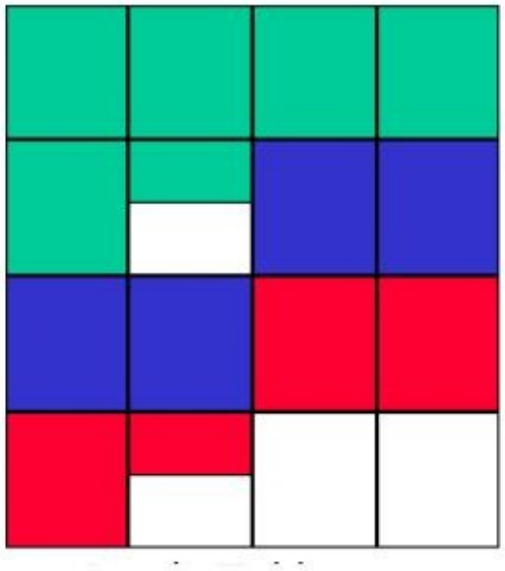
- In DB2, what page size should be chosen if 0% free space is required and the record size is 2500 bytes?
- The simplistic answer is 4K, but it might not be the best answer.
- A 4K page would hold one 2500-byte record per page, but an 8K page would hold three 2500-byte records.
- The 8K page can provide for more efficient I/O, especially if data is accessed sequentially (instead of randomly).
  - This is so because reading 8K of data would return 3 rows, whereas reading 8K of data using two 4K pages would return only two rows.

Database Reorganization

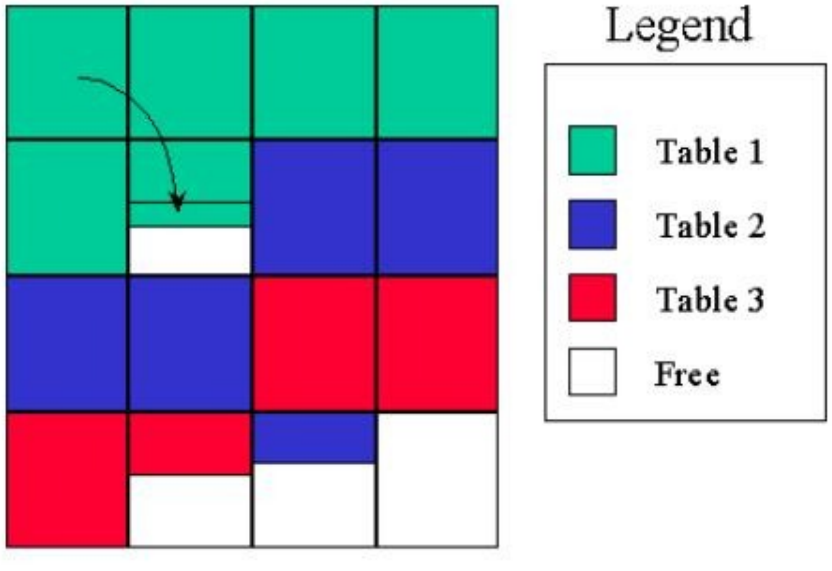
Causes of database disorganization:

- *Unclustered data*. A clustered table or index can become unclustered as data is added and changed. If the data becomes significantly unclustered, queries that were optimized to access data cannot take advantage of the clustering sequence causing performance to suffer.
- *Fragmentation* is a condition in which there are many scattered areas of storage in a database that are too small to be used productively. It results in wasted space, which can hinder performance because additional I/Os are required to retrieve the same data.
- *Row chaining* or *row migration* occurs when updated data does not fit in the space it currently occupies, and the DBMS must find space for the row. In each case, a pointer is used to locate either the rest of the row or the full row., causing performance to suffer.
  - With row chaining, the DBMS moves a part of the new, larger row to a location within the tablespace where free space exists.
  - With row migrations, the full row is placed elsewhere in the tablespace.
  - I*Page splits* can cause disorganized databases, too. If the DBMS performs monotonic page splits when it should perform normal page splits, or vice versa, space may be wasted. When space is wasted, fewer rows exist on each page, causing the DBMS to issue more I/O requests to retrieve data. Therefore, once again, performance suffers.
- *File extents* can negatively impact performance. An *extent* is an additional file that is tied to the original file and can be used only in conjunction with the original file. When the file used by a tablespace runs out of space, an extent is added for the file to expand.

Organized vs. Unorganized Tablespace



Organized Tablespace



Disorganized Tablespace

http://www.craigsmullins.com/ou_deal.htm

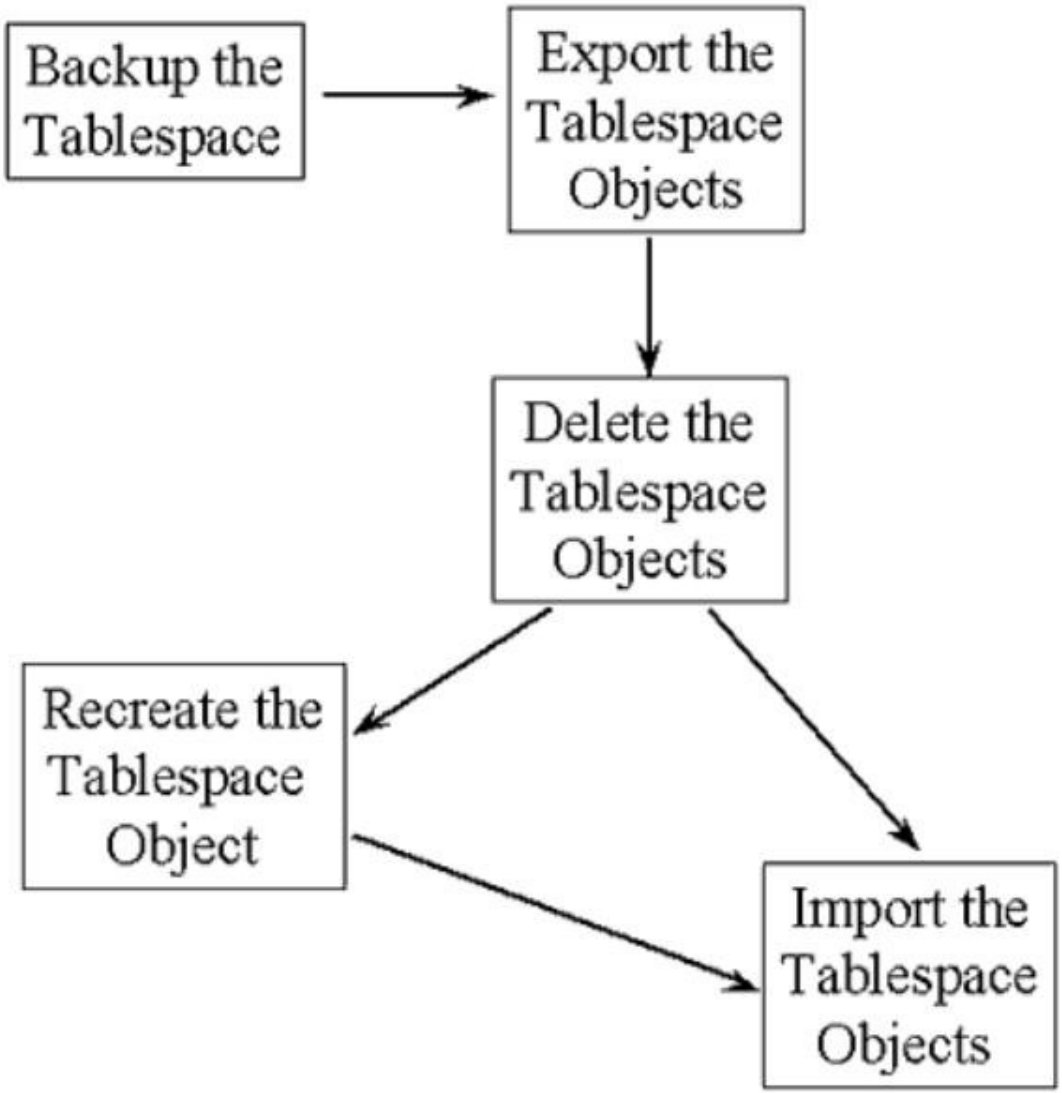
Reorganization Fixes Disorganized Tablespaces

- To correct disorganized database structures, the DBA can run a database or tablespace reorganization utility, or REORG, to force the DBMS to restructure the database object, thus removing problems such as unclustered data, fragmentation, and row chaining.
- The primary benefit is the resulting speed and efficiency of database functions because the data is organized in a more optimal fashion on disk.
- In short, reorganization maximizes availability and reliability for databases.

Reorganize Indexes and Tablespaces

- Tablespaces and indexes both can be reorganized.
- How the DBA runs a REORG utility depends on the DBMS.
  - Some DBMS products ship with a built-in reorganization utility.
  - Others require the customer to purchase the utility.
  - Still others claim that the customer will not need the utility at all when using their DBMS.
    - In practice, this last claim is not true. Every DBMS incurs some degree of disorganization as data is added and modified… and can therefore benefit from reorganization.

Rebuilding to Reorganize



Reorganization and Availability

- A traditional reorganization requires the database to be down.
- Some REORG utilities are available that perform the reorganization while the database is online.
  - Such a reorganization is accomplished by making a copy of the data. The online REORG utility reorganizes the copy while the original data remains online.
  - When the copied data has been reorganized, an online REORG uses the database log to “catch up” by applying to the copy any data changes that occurred during the process.
  - When the copy has caught up to the original, the online REORG switches the production tablespace from the original to the copy.

Determining When to Reorganize

- System catalog statistics can help to determine when to reorganize a database object.
- Each DBMS provides a method of reading through the contents of the database and recording statistical information about each database object.
- Depending on the DBMS, this statistical information is stored either in the system catalog or in special pages within the database object itself.

Using Statistics to Schedule Tablespace Reorganization – Cluster Ratio

- Consider cluster ratio, the percentage of rows in a table that are actually stored in clustering sequence
- The closer the cluster ratio is to 100%, the more closely the actual ordering of the rows on the data pages matches the clustering sequence.
- A low cluster ratio indicates bad clustering, and a reorganization may be required.

Using Statistics to Schedule Tablespace Reorganization – Others

- Fragmentation
- Row chaining
- Row migration
- Space dedicated to dropped objects
- Number of page splits

Using Statistics to Schedule Index Reorganization

- Index levels
- Leaf distance
  - Distance between leaf pages

Automating Reorganization

- Look into using the database utilities or third-party tools to automate reorganizations.
- Base reorganization on thresholds established on statistics gathered by the DBMS or the tools.
- The wise DBA will plan for and schedule reorganizations to resolve disorganization problems in their database systems.
  - The wiser DBA will automate the process as much as possible!

Questions