Running Head: DATABASE INDEXING

Database Indexing: Performance Considerations

Jered McClure

Walden University

Database Indexing: Performance Considerations

While database indexing is meant to increase the performance of lookups across large datasets, they can actually decrease performance if the index is not managed correctly. For instance, if a composite index were created between two columns

E.g.:

**CREATE** **INDEX** TWOCOLUMNS **ON** TABLE\_1(COLUMNA,CLOUMNB);

When a lookup containing the order COLUMNA + COLUMNB is done, the index works exactly as intended. However, in the case where the order is COLUMNB + COLUMNA there is no logical index construct in the system, therefore the index is rendered useless.

Additionally, when creating indexes, the total overall storage capacity required to store the database and index is increased. For instance, if an index is placed on COLUMNA and this column contains 1GB of data, then that 1GB of data must be duplicated in whole in the index. If then, the database is backed up that 1GB of data becomes 4GB of data. Then there is the three different development environments, the mirrored location server, and the disaster recovery server…soon enough we are looking at 14GB of data for a one column index. While storage is definitely cheap, when this is expanded across multiple indexes throughout a database, the storage requirements expand exponentially.

However, indexes are definitely required in transaction databases where lookups are constantly occurring across multiple application connections (think point of sale systems). An index on a STOCK\_ID column is a near requirement for looking up information about the latest and greatest “super plushie” for the customer standing in front of the customer service rep. Otherwise, a simple search could take ages as the system slogs through every single stock item looking for the one called “super plushie.”

That being said, this STOCK\_ID index has some throw back problems which may cause issues for the stock taking department of the store. If stock is constantly coming in being updated and/or deleted then having an index on the STOCK\_ID column will cause these changes to take longer, as the database must reorganize the index each time one of these changes occurs. Furthermore, if the database does not have enough system resources, then these changes may slow down the entire system, thereby reducing performance further than not having an index at all would have caused.

In such a situation, the best choice would be to bring both situations to the business managers and let them decide how they would like to proceed. It may be that a simple operational change of stock updates/deletes takes place in a batch file each night after the store has closed. This would cause all changes to the index to flow through during non-peak hours and, therefore, cause no slowdown in the system’s operation. Alternatively, the business may decide that a one minute lookup is not an issue when searching for specific stock items, and nix the whole index idea altogether.

In the end, an index can increase performance, but only at a cost of storage increase and possible system performance degradation elsewhere. This does not mean that the index is a bad thing, only that one must apply indexes in a knowledgeable fashion. Haphazard database indexing can actually be detrimental to the overall perceived performance boost. Lastly, for major system changes always bring the final decision to the appropriate stake holders for confirmation.

Reference

Coronel, C., Morris, S., & Rob, P. (2012). *Database Systems: Design, Implementation, and Management* (10th ed.). Boston, MA, USA: Cengage Learning.