**1. Some tips to improve the performance of SQL queries**

1. Instead of writing long queries use either views or stored procedures for minimizing network load.
2. It’s better to introduce constraints instead of triggers. They are more efficient than triggers and can increase performance.
3. Make use of table-level variables instead of temporary tables.
4. The UNION ALL clause responds faster than UNION. It doesn’t look for duplicate rows whereas the UNION statement does that regardless of whether they exist or not.
5. Prevent the usage of DISTINCT and HAVING clauses.
6. Avoid excessive use of SQL cursors.
7. Make use of SET NOCOUNT ON clause while building stored procedures. It represents the rows affected by a T-SQL statement. It would lead to reduced network traffic.
8. It’s a good practice to return the required column instead of all the columns of a table.
9. Create indexes for tables

**2. Bottlenecks that affect the performance of a database**

1. First of all need to identify system performance using CPU utilization (Processors).
2. Low memory is the next most common bottleneck. You can resolve it by expanding the physical RAM, but it won’t solve memory leaks if there is any. In such a case, you need to profile the application to identify the potential leaks within its code.
3. Disk Storage bottleneck –upgrading faster drives--Its impact gets visible while writing large data to the disk. If output operations are very slow, then it is a clear indication an issue becoming the bottleneck.

**3. Steps to Improving the SQL Performances**

1. **Discover –** First of all, find out the areas of improvement. Explore tools like Profiler, Query execution plans, SQL tuning advisor, dynamic views, and custom stored procedures.
2. **Review –** Brainstorm the data available to isolate the main issues.
3. **Propose –** Here is a standard approach one can adapt to boost the performance. However, you can customize it further to maximize the benefits.

1. Identify fields and create indexes.  
2. Modify large queries to make use of indexes created.  
3. Refresh table and views and update statistics.  
4. Reset existing indexes and remove unused ones.  
5. Look for dead blocks and remove them.

1. **Validate –** Test the SQL performance tuning approach. Monitor the progress at a regular interval. Also, track if there is any adverse impact on other parts of the application.
2. **Publish –** Now, it’s time to share the working solution with everyone in the team. Let them know all the best practices so that they can use it with ease.

**4. Difference between Heap table and clustered table? How can we identify if the table is a heap?**

A Heap table is table in which, the data rows are not stored in any particular order within each data pages. In addition, there is no particular order to control the data pages sequence. That is not linked in a linked list. This is due to the fact that the heap table contains no clustered index.

A clustered table is a table that has a predefined clustered index on a column or multiple columns of the table that defines the sorting order of the rows within the data pages and the order of the data pages within the table on the clustered index key.

The heap table can be identified by querying the [**sys.partitions**](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-partitions-transact-sql%22%20/t%20%22_blank) system object that has one row per each partition with index\_id value equal to 0. You can also query the **sys.indexes** system object also to show the heap table index details, which shows, the id of that index is 0 and the type of it is HEAP.

**5. Index Allocation Map (IAM):**

Sql Server Engine uses an Index Allocation Map (IAM) to keep an entry for each page to track the allocation of these available pages. The IAM is considered as the only logical connection between the data pages that the SQL Server Engine will use to move through the heap.

**6. Forwarding Pointers issue:-**

When a data modification operation is performed on heap table data pages. **Forwarding Pointers** will be inserted into the heap to point to the new location of the moved data. These forwarding pointes will cause the performance issues over time due to visiting the old/original location vs the new location specified by the forwarding pointers to get a specific value.

Starting from SQL Server 2008, a new method was introduced to overcome the forwarding pointers performance issues, by using the ALTER TABLE REBUILD command that will rebuild the heap table.

**7. SQL Server Index:-**

A SQL Server Index is a one of the most important factor in the SQL Server Performance tuning process. Indexes are created to speed up the data retrieval and the querying process operations from a database tables or views, by providing swift access to the database rows, without no need to scan all the table’s data in order to retrieve the requested data.

For example, if we need to search any particular information instead of searching page by page, it’s an easy to identify the index page.

**8. Structure of SQL Server Index:-**

A SQL Server index is created using the shape of B-Tree structure, that’s is made up of 8k pages, with each page in that structure, called an index node. The B-Tree structure provides the SQL Server engine with a fast way to move through the table rows based on the index key that decides to navigate left or right, to retrieve the requested values directly. Without scanning all the underlying table rows. You can imagine the potential performance degradation that may occur due to scanning large database table.

The B-Tree structure of the index consists of three main levels:

* the **Root Level**, the top node that contains a single index page, form which SQL Server starts its data search,
* the **Leaf Level**, the bottom level of nodes that contains the data pages we are looking for, with the number of leaf pages depends on the amount of data stored in the index,
* And finally the **Intermediate Level**, one or multiple levels between the root and the leaf levels that holds the index key values and pointers to the next intermediate level pages or the leaf data pages. The number of intermediate levels depends on the amount of data stored in the index.

**9. OLTP and OLAP workloads and how do they affect index creation decisions?**

On **Online Transaction Processing (OLTP)**databases, workloads are used for transactional systems, in which most of the submitted queries are data modification queries.

**On Online Analytical Processing (OLAP)** database workloads are used for data warehousing systems, in which most of the submitted queries are data retrieval queries that filter, group, aggregate and join large data sets quickly.

For **Online Transaction Processing (OLTP)**creating a large number of indexes on a database table affects data modification (e.g. Updates) operation performance. When you add or modify a row in the underlying table, the row will also be adjusted appropriately in all related table indexes. Because of that, you need to avoid creating a large number of indexes on the heavily modified tables and create the minimum possible number of indexes, with the least possible number of columns on each index.

For **Online Analytical Processing (OLAP)** workloads, in which tables have low modification requirements, you can create a large number of indexes that improve the performance of the data retrieval operations

## 10. Difference between PAD\_INDEX and FILLFACTOR?

* **FILLFACTOR** is used to set the percentage of free space that the SQL Server Engine will leave in the leaf level of each index page during index creation. The FillFactor should be an integer value from 0 to 100, with 0 or 100 is the default value, in which the pages will be filled completely during the index creation.
* **PAD\_INDEX** is used to apply the free space percentage specified by FillFactor to the index intermediate level pages during index creation.

## 11. Why it is not recommended to use GUID and CHARACTER columns as Clustered index keys?

For GUID columns, that are stored in UNIQUE IDENTIFIER columns, the main challenge that affects the clustered index key sorting performance is the nature of the GUID value that is larger than the integer data types, with 16 bytes size, and that it is generated in random manner, different from the IDENTITY integer values that are increasing continuously.

For the CHARACTER columns. The main challenges include limited sorting performance of the character data types, the large size, non-increasing values, non-static values that often tend to change in the business applications and not compared as binary values during the sorting process, as the characters comparison mechanism depends on the used collation.

## 12. When checking the index usage statistics information, retrieved by querying the sys.dm\_db\_index\_usage\_stats dynamic management view, explain the results of the returned number of seeks, scans, lookups and updates.

* The number of **Seeks** indicates the number of times the index is used to find a specific row,
* the number of **Scans**shows the number of times the leaf pages of the index are scanned,
* the number of **Lookups** indicates the number of times a Clustered index is used by the Non-clustered index to fetch the full row
* And the number of **Updates** shows the number of times the index data has modified.