INDEX

Guidelines for SQL SERVER

Notice:

All rights reserved worldwide. No part of this book may be reproduced or copied or translated in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher, except for reading and browsing via the World Wide Web. Users are not permitted to mount this file on any network servers.

For more information send email to: pinal@sqlauthority.com

Important Guidelines for SQL SERVER

- Use "Pascal" notation for SQL server Objects Like Tables, Views, Stored Procedures. Also tables and views should have ending "s".
 Example: UserDetails
 Emails
- If you have big subset of table group than it makes sense to give prefix for this table group. Prefix should be separated by _.
 Example:
 Page_ UserDetails
 Page_ Emails
- Use following naming convention for Stored Procedure. sp<Application Name>_[<group name >_]<action type> Where action is: Get, Delete, Update, Write, Archive, Insert... i.e. verb Example: spApplicationName_GetUserDetails spApplicationName UpdateEmails
- Use following Naming pattern for triggers: TR_<TableName>_<action><description>
 Example: TR_Emails_LogEmailChanges TR_UserDetails_UpdateUserName
- Indexes : IX_<tablename>_<columns separated by_>
 Example:
 IX UserDetails UserID
- Primary Key : PK_<tablename> Example: PK_UserDetails PK_ Emails

- Foreign Key : FK_<tablename_1>_<tablename_2>
 Example:
 FK UserDetails Emails
- Default: DF__<column name> Example: DF_ UserDetails _UserName
- Normalize Database structure based on 3rd Normalization Form. Normalization is the process of designing a data model to efficiently store data in a database. (<u>Read More Here</u>)
- Avoid use of **SELECT** * in SQL queries. Instead practice writing required **column** names after **SELECT** statement.

Example: SELECT Username, Password FROM UserDetails

- Avoid using temporary tables and derived tables as it uses more disks I/O. Instead use CTE (Common Table Expression); its scope is limited to the next statement in SQL query. (Read More Here)
- Use **SET NOCOUNT ON** at the beginning of SQL Batches, Stored Procedures and Triggers. This improves the performance of Stored Procedure. (<u>Read More Here</u>)
- Properly format SQL queries using indents.

Example: Wrong Format SELECT Username, Password FROM UserDetails ud INNER JOIN Employee e ON e.EmpID = ud.UserID

Example: Correct FormatSELECTUsername, PasswordFROMUserDetails udINNER JOINEmployee e ON e.EmpID = ud.UserID

- Practice writing Upper Case for all SQL keywords.
 Example: SELECT, UPDATE, INSERT, WHERE, INNER JOIN, AND, OR, LIKE.
- There must be **PRIMARY KEY** in all the tables of database with column name ID. It is common practice to use Primary Key as **IDENTITY** column.
- If "One Table" references "Another Table" than the column name used in reference should use the following rule :

Column of Another Table : <OneTableName> ID

Example:

If User table references Employee table than the column name used in reference should be **UserID** where User is table name and ID primary column of User table and UserID is reference column of Employee table.

- Columns with **Default value** constraint should not allow NULLs.
- Practice using **PRIMARY** key in **WHERE** condition of **UPDATE** or **DELETE** statements as this will avoid error possibilities.
- Always create stored procedure in **same database** where its relevant table exists otherwise it will reduce network performance.
- Avoid server-side Cursors as much as possible, instead use SELECT statement. If you need to use cursor then replace it with WHILE loop (or read next suggestion).
- Instead of using LOOP to insert data from Table B to Table A, try to use SELECT statement with INSERT statement. (<u>Read More Here</u>)

INSERT INTO Table A (column1, column2) SELECT column1, column2 FROM Table B WHERE

- Avoid using spaces within the name of database objects; this may create issues with front-end data access tools and applications. If you need spaces in your database object name then will accessing it surround the database object name with square brackets. Example: [Order Details]
- Do not use **reserved words** for naming database objects, as that can lead to some unpredictable situations. (Read More Here)
- Practice writing comments in stored procedures, triggers and SQL batches, whenever something is not very obvious, as it won't impact the performance.
- Do not use wild card characters at the beginning of word while search using LIKE keyword as it results in Index scan.
- Indent code for better readability. (Example)
- While using **JOINs** in your SQL query always **prefix column name** with the table name. (Example). If additionally require then prefix Table name with ServerName, DatabaseName, DatabaseOwner. (Example)
- Default constraint must be defined at the **column level**. All other constraints must be defined at the table level. (Read More Here)
- Avoid using rules of database objects instead use constraints.
- Do not use the RECOMPILE option for Stored Procedure as it reduces the performance.
- Always put the **DECLARE** statements at the starting of the code in the stored procedure. This will make the query optimizer to reuse query plans. (Example)
- Put the SET statements in beginning (after DECLARE) before executing code in the stored procedure. (Example)
- Use BEGIN...END blocks only when multiple statements are present within a conditional code segment. (Read More Here)

To express apostrophe within a string, nest single quotes (two single quotes).
 Example:

SET @sExample = 'SQL''s Authority'

• When working with branch conditions or **complicated expressions**, use parenthesis to increase readability.

```
Example:
IF ((SELECT 1
FROM TableName
WHERE 1=2) ISNULL)
```

- To mark single line as comment use (--) before statement. To mark section of code as comment use (/*...*/).
- Avoid the use of cross joins if possible. (<u>Read More Here</u>)
- If there is no need of resultset then use syntax that doesn't return a resultset.

IF EXISTS (SELECT	1
FROM	UserDetails
WHERE	UserID = 50)

Rather than,

IF EXISTS (SELECT	COUNT (UserID)
FROM	UserDetails
WHERE	UserID = 50)

- Use graphical execution plan in Query Analyzer or SHOWPLAN_TEXT or SHOWPLAN_ALL commands to analyze SQL queries. Your queries should do an "Index Seek" instead of an "Index Scan" or a "Table Scan". (<u>Read More Here</u>)
- Do not prefix stored procedure names with "SP_", as "SP_" is reserved for system stored procedures.
 Example:

SP<App Name>_ [<Group Name >_] <Action><table/logical instance>

- Incorporate your frequently required, complicated joins and calculations into a **view** so that you don't have to repeat those joins/calculations in all your queries. Instead, just select from the view. (<u>Read More Here</u>)
- Do not query / manipulate the data directly in your front end application, instead **create stored procedures**, and let your applications to access stored procedure.
- Avoid using ntext, text, and image data types in new development work. Use <u>nvarchar</u> (<u>max</u>), <u>varchar (max</u>), and <u>varbinary (max</u>) instead.
- Do not store **binary or image files (Binary Large Objects or BLOBs)** inside the database. Instead, store the path to the binary or image file in the database and use that as a pointer to the actual file stored on a server.
- Use the **CHAR** datatype for a non-nullable column, as it will be the fixed length column, NULL value will also block the defined bytes.
- Avoid using dynamic SQL statements. Dynamic SQL tends to be slower than static SQL, as SQL Server generate execution plan every time at runtime.
- Minimize the use of Nulls. Because they incur more complexity in queries and updates. **ISNULL** and **COALESCE** functions are helpful in dealing with NULL values
- Use Unicode datatypes, like NCHAR, NVARCHAR or NTEXT if it needed, as they use twice as much space as non-Unicode datatypes.
- Always use column list in **INSERT** statements of SQL queries. This will avoid problem when table structure changes.
- Perform all **referential integrity** checks and **data validations** using **constraints** instead of **triggers**, as they are faster. Limit the use of triggers only for auditing, custom tasks, and validations that cannot be performed using constraints.
- Always access tables in the same order in all stored procedure and triggers consistently. This will avoid deadlocks. (<u>Read More Here</u>)

- Do not call functions repeatedly in stored procedures, triggers, functions and batches, instead call the function once and store the result in a variable, for later use.
- With Begin and End Transaction always use global variable @@ERROR, immediately after data manipulation statements (INSERT/UPDATE/DELETE), so that if there is an Error the transaction can be rollback.
- Excessive usage of **GOTO** can lead to hard-to-read and understand code.
- Do not use **column numbers** in the ORDER BY clause; it will reduce the readability of SQL query.

Example: Wrong StatementSELECTUserID, UserName, PasswordFROMUserDetailsORDER BY2

Example: Correct Statement

SELECT UserID, UserName, Password FROM UserDetails ORDER BY UserName

- To avoid **trips from application to SQL Server**, we should retrive multiple resultset from single Stored Procedure instead of using output param.
- The **RETURN** statement is meant for returning the execution status only, but not data. If you need to return data, use **OUTPUT** parameters.
- If stored procedure always returns single row resultset, then consider returning the resultset using **OUTPUT** parameters instead of **SELECT** statement, as ADO handles OUTPUT parameters faster than resultsets returned by SELECT statements.
- Effective **indexes** are one of the best ways to improve performance in a database application.
- **BULK INSERT** command helps to import a data file into a database table or view in a user-specified format.

- Use **Policy Management** to make or define and enforce your own policies fro configuring and managing SQL Server across the enterprise, eg. Policy that Prefixes for stored procedures should be sp.
- Use **sparse columns** to reduce the space requirements for null values. (<u>Read More Here</u>)
- Use MERGE Statement to implement multiple DML operations instead of writing separate INSERT, UPDATE, DELETE statements.
- When some particular records are retrieved frequently, apply **Filtered Index** to improve query performace, faster retrieval and reduce index maintenance costs.
- Using the NOLOCK query optimizer hint is considered good practice in order to improve concurrency on a busy system.
- **EXCEPT** or **NOT EXIST** clause can be used in place of LEFT JOIN or NOT IN for better peformance.

Example:

umpic.		
SELECT	EmpNo, EmpName	
FROM	EmployeeRecord	
WHERE	Salary > 1000 AND Salary	
	NOT IN (SELECT	Salary
	FROM	EmployeeRecord
	WHERE	Salary > 2000);
(Recomended)	
SELECT	EmpNo, EmpName	
FROM	EmployeeRecord	
WHERE	Salery > 1000	
EXCEPT		
SELECT	EmpNo, EmpName	
FROM	EmployeeRecord	
WHERE	Salery > 2000	

EmpName;

ORDER BY