10 个 T-SQL 提示和技巧

作者：微信公众号：【架构师老卢】 3-13 19:28 2

概述：.NET 和 T-SQL 共享一种明确无误的招带。从我记事起，我就一直使用 T-SQL 以及 C# 和 Entity Framework Core。然而，在所有这些抽象之下，人们很容易忽视原始SQL的细微差别以及深入理解它的重要性。因此，让我们深入了解一些 RAW SQL :). 通用表表达式（CTE）有没有发现自己迷失在了查询的海洋中？ CTE 可以成为救命稻草。它们允许您命名临时结果集，并在主查询中使用它。以下是如何使用它：WITH Sales_CTE AS (SELECT CustomerID, SUM(OrderTotal) AS TotalSales FROM Ord

.NET 和 T-SQL 共享一种明确无误的招带。从我记事起，我就一直使用 T-SQL 以及 C# 和 Entity Framework Core。

然而，在所有这些抽象之下，人们很容易忽视原始SQL的细微差别以及深入理解它的重要性。

因此，让我们深入了解一些 RAW SQL :)

1. 通用表表达式 （CTE）

有没有发现自己迷失在了查询的海洋中？ CTE 可以成为救命稻草。它们允许您命名临时结果集，并在主查询中使用它。以下是如何使用它：

```
WITH Sales_CTE AS (
  SELECT CustomerID, SUM(OrderTotal) AS TotalSales
  FROM Orders
  GROUP BY CustomerID
)
SELECT *
FROM Sales_CTE
WHERE TotalSales > 1000;
```

2. 窗口函数

从后的角度看待数据。

窗口函数允许您通过“窗口”查看数据并将相关行执行计算。考虑计算运行总数或对项目进行排名。

```
SELECT
  CustomerID,
  OrderDate,
  OrderTotal,
  SUM(OrderTotal) OVER (ORDER BY OrderDate) AS RunningTotal
FROM Orders;
```

3. 表变量和临时表

有时，您需要一个临时位置来存储查询中的数据。

表变量和临时表非常适合此目的。它们就像您在重新整理房间时使用的临时储物箱。

```
DECLARE @TempTable TABLE (CustomerID INT, TotalSales MONEY);
INSERT INTO @TempTable (CustomerID, TotalSales)
  SELECT CustomerID, SUM(OrderTotal)
  FROM Orders
  GROUP BY CustomerID;
```

4. 索引：使查询运行速度更快。

如果您的查询速度很慢，索引可能正是您需要的提升。它们就像帮助 SQL Server 更快地查找数据的快捷方式。但请记住，太多会减慢数据更新速度。所以这完全是关于平衡的。

有几种类型的索引，但让我们保持简单，并讨论您可能会使用的两种主要索引：

聚簇索引

将聚集索引视为组织数据的主要方式。它实际上按索引的顺序存储行数据。因此，当您在列上创建聚集索引时，SQL Server 会根据该列的值重新排列表中的数据。

下面是一个基本示例：

```
CREATE CLUSTERED INDEX IX_YourTable_YourColumn
ON YourTable(YourColumn);
```

非聚集索引

这些类似于附加指南，可帮助您根据您可能经常查询但不是主要排序机制的其他列查找数据。

与聚集索引不同，非聚集索引维护与实际表数据不同的结构，其中包括索引列和指向包含这些值的行的指针。

以下是您可以创建一个的方法：

```
CREATE NONCLUSTERED INDEX IX_YourTable_AnotherColumn
ON YourTable(AnotherColumn);
```

为什么要关心索引？

好吧，如果没有它们，SQL Server 必须扫描整个表才能找到您要求的数据，这可能会很慢且很痛苦，尤其是对于大型表。使用索引，它可以快速跳转到要的石例并获取所需的内容。

但是，要注意的是

虽然索引非常适合读取操作，但它们会减慢 INSERT、UPDATE 和 DELETE 等写入操作的速度，因为每次数据更改时都需要更新索引本身。这一切都是为了根据应用程序使用数据的方式找到正确的平衡点。

5. MERGE 语句

MERGE 语句是 SQL Server 中的一个强大工具。

它允许您更新现有记录、插入新记录或删除不再需要的记录，所有这些都只需一个命令即可完成。您可以设置如何在两个表之间匹配数据的规则，其余的由 MERGE 负责。

这使得使表保持最新变得更加容易，尤其是在处理大量更改时。

```
MERGE INTO Customers AS target
USING UpdatedCustomers AS source
ON target.CustomerID = source.CustomerID
WHEN MATCHED THEN
  UPDATE SET target.ContactName = source.ContactName
WHEN NOT MATCHED BY TARGET THEN
  INSERT (CustomerID, ContactName)
  VALUES (source.CustomerID, source.ContactName)
WHEN NOT MATCHED BY SOURCE THEN
  DELETE;
```

6. TRY_CAST和TRY_CONVERT：避免转换问题。

当您从不同位置提取数据或使用用户输入时，有时事情并不完全匹配。这就是TRY_CAST和TRY_CONVERT派上用场的地方。

这两个函数都尝试将值的数据类型更改为另一种类型，如果转换是可能的，那就太好了！它们以熟练式返回值。

但是，如果无法完成转换（可能是因为数据的格式意外），则它们将返回 NULL 而不是抛出错误。这非常有用，因为这意味着即使数据出现问题，您的查询也能保持平稳运行。

TRY_CAST

假设您有一列应该是日期，但有可能有一些非日期值混入其中。

```
SELECT TRY_CAST(YourColumn AS DATE) FROM YourTable;
```

如果包含不是日期的内容，TRY_CAST 将为该值返回 NULL，而不是因错误而停止查询。YourColumn

TRY_CONVERT

与 TRY_CAST 非常相似，但它允许您指定转换的样式。这对于格式可能发生变化的日期和时间特别有用。

```
SELECT TRY_CONVERT(DATE, YourColumn, 101) FROM YourTable;
```

这将尝试使用样式“101”（美国样式，mm/dd/yyyy）转换为 DATE。如果转换不起作用，则会收到 NULL 而不是错误。YourColumn

7. SET NOCOUNT ON：跳过不必要的信息。

在 SQL 脚本或存储过程中打开就像告诉 SQL Server，“嘿，让我们保持聊天最少。SET NOCOUNT ON

通常，SQL Server 喜欢在每次运行 INSERT、UPDATE、DELETE 或 SELECT 语句时通过说明受影响的行数来提醒您一些提示。此消息乍一看似乎很有帮助，但是当您运行大量语句或处理复杂的过程时，所有这些更新都会减慢速度。

性能提升

特别是在复杂的程序中或运行大量小操作时，删除这些消息可以稍微加快速度。它可以减少 SQL Server 和应用程序之间的网络流量，这在高容量环境中特别有用。

更干净的应用程序日志

如果您的应用程序或工具记录了这些消息，则关闭它们可以使您的日志更干净，并更专注于重要的事情。

这是你如何做到的：

```
SET NOCOUNT ON;
-- Your T-SQL code goes here
```

只需将此行放在脚本或存储过程的开头即可。这是一个很小的改变，但可以在保持事情顺利进行和专注于重要的事情方面产生很大的影响。

请记住，仅影响有关受影响行的消息。它不会使错误消息或其他重要信息静音，因此您仍然可以了解引擎盖下发生的事情。SET NOCOUNT ON

8. 动态 SQL：当您需要灵活性时。

动态 SQL 允许您动态构建 SQL 语句。这就像能够根据您目前拥有的零件（数据）随心所欲地组装您的乐高套装。

```
DECLARE @Table NVARCHAR(100) = N'Customers';
EXECUTE sp_executesql N'SELECT * FROM ' + @Table;
```

9. OUTPUT 参数：密切关注变化。

当您执行更改数据的操作时，OUTPUT 子句可以准确地显示所做的更改。

它可以告诉您添加、更新或删除了哪些行，甚至可以显示更新之前和之后的值。

下面是一个基本示例，用于说明如何将其与 INSERT 操作一起使用：

```
INSERT INTO Employees (Name, Role)
OUTPUT inserted.EmployeeID, inserted.Name, inserted.Role
VALUES ('Jane Doe', 'Developer');
```

在这种情况下，OUTPUT 子句指示 SQL Server 返回刚刚添加的新记录的 EmployeeID、Name 和 Role，对于 UPDATE 或 DELETE 操作，同样可以使用来查看新值或查看已删除的内容。inserted和deleted

这为您提供了一种强大的方法来审核更改，甚至将这些更改馈送到另一个进程或表中。

考虑一个场景，其中你正在更新员工记录，并希望保留更改日志：

```
UPDATE Employees
SET Role = 'Senior Developer'
OUTPUT deleted.EmployeeID, deleted.Role AS OldRole, inserted.Role AS NewRole
INTO EmployeeRoleChanges (EmployeeID, OldRole, NewRole)
WHERE EmployeeID = 1;
```

这不仅会更新员工的角色，还会捕获更改并将其记录到表中，显示旧角色、新角色以及角色已更改的员工的 ID。EmployeeRoleChanges

10. 在游标上选择基于 SET 的操作。

SQL Server 中的游标允许您一次处理一行。

这在某些情况下非常有用，因为您需要对每一行执行复杂的逻辑。但是，游标通常速度较慢，因为它们将一个可能是单个快速操作的过程转换为一长串步骤。

想象一下，你正在分发传单。您可以一次将它们分发给房间里的每个人（基于 SET 的操作），或者您可以走到每个人面前，一次递给他们一张传单（光标），第一种方法显然要快得多。

另一方面，基于 SET 的操作可以同时处理整个数据集。SQL Server 的设计非常擅长于此。

假设您要更新表中某些行的列。使用光标，您可以编写如下内容（为清楚起见，请简化）：

```
DECLARE @ID INT, @Value VARCHAR(100)
DECLARE myCursor CURSOR FOR
SELECT ID, Value FROM MyTable WHERE Condition = 'Yes'

OPEN myCursor
FETCH NEXT FROM myCursor INTO @ID, @Value

WHILE @@FETCH_STATUS = 0
BEGIN
    UPDATE MyTable SET AnotherColumn = @Value WHERE ID = @ID
    FETCH NEXT FROM myCursor INTO @ID, @Value
END

CLOSE myCursor
DEALLOCATE myCursor
```

使用基于 SET 的方法，您可以以更简单、更快速的方式获得相同的结果：

```
UPDATE MyTable
SET AnotherColumn = Value
WHERE Condition = 'Yes'
```

在这种情况下，基于 SET 的操作不仅更易于读取和写入，而且速度可能更快。

SQL Server 在处理数据集时可以利用索引、并行处理和其他优化，而不是单独处理每一行。