



SQL Server

Performance Tuning

Christoph Müller

christoph.mueller@unilog.de

0178 / 88 66 204



- Performance, was gehört dazu?
 - Hardware
 - Netzwerk
 - Datenbankintern
 - Datenzugriff
 - Sperrverhalten
 - Verarbeitung in der Applikation
- Wo fang ich an, wo hör ich auf?
 - Identifikation von Flaschenhälsen
 - Beispielszenarien

Grundsätzlich:

- Je mehr, je besser

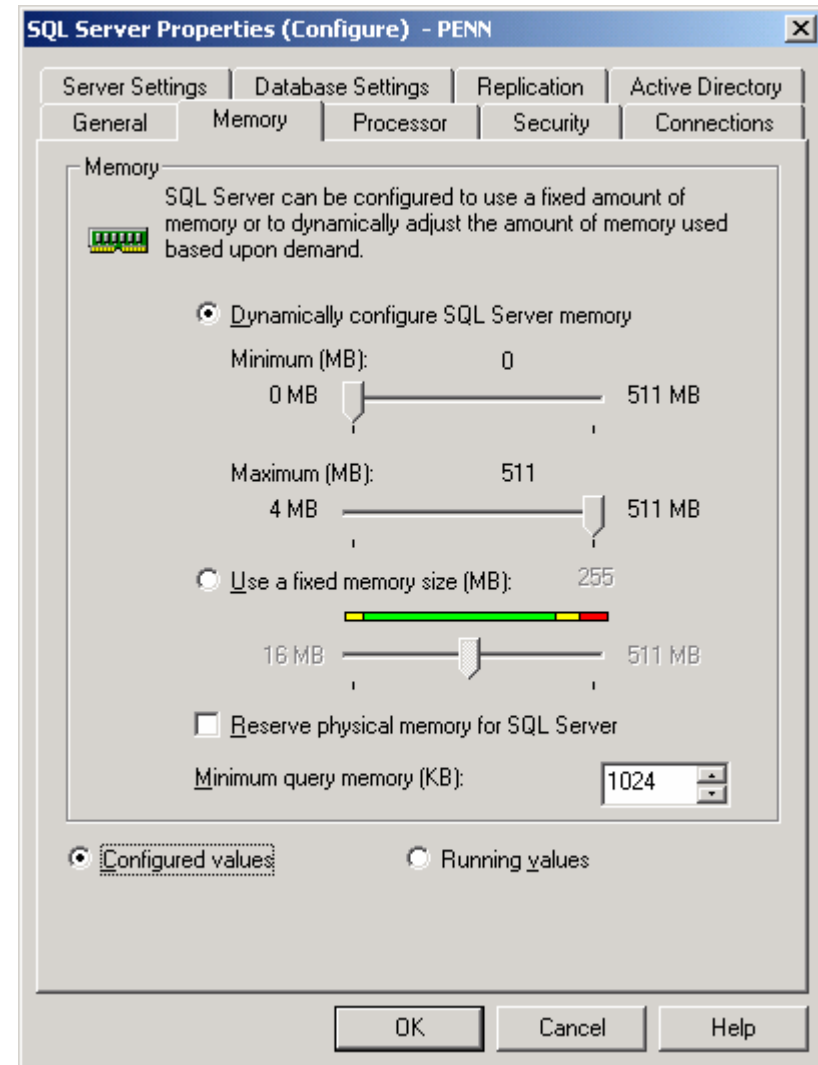
Aaaber

- Je mehr, je teurer

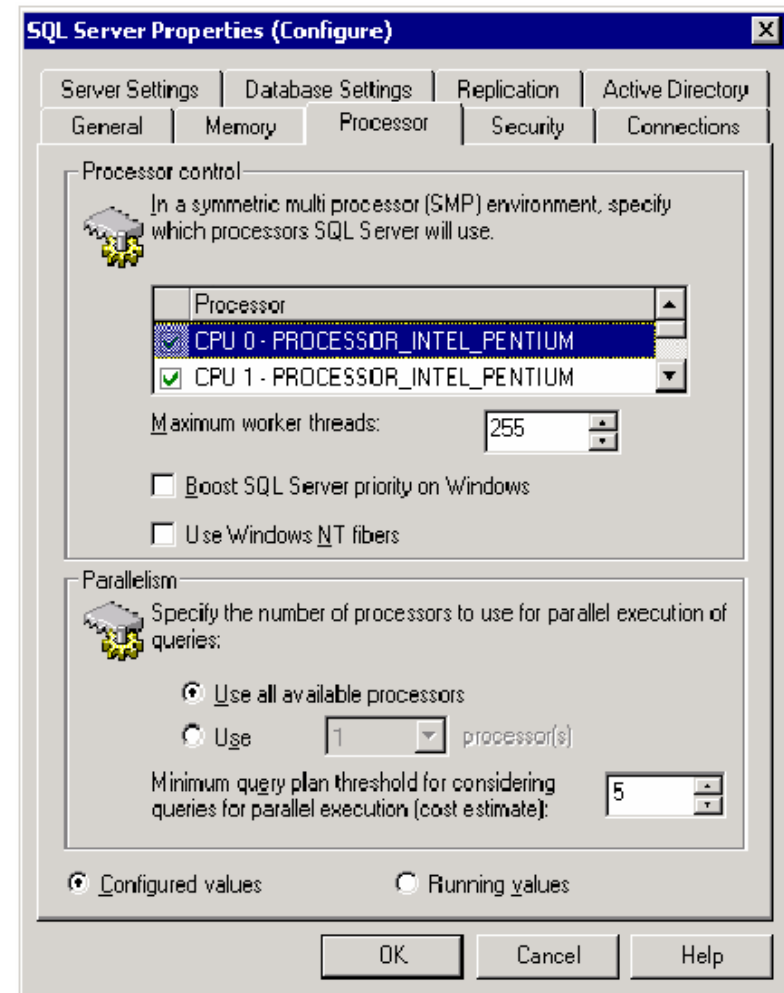
Mögliche Verbesserungen:

- Speicher
- Prozessor
- Platte

- ermöglicht, Daten für zukünftige Queries im Cache zu halten
- TempDB in den Cache,
 - besonders, falls selbst temporäre Tabellen verwendet werden.
- Empfehlung von Microsoft
 - 512MB für kleine Server
 - 2GB für mittlere (davon 1,5GB für SQL)
 - Clusterung, 64bit für sehr große



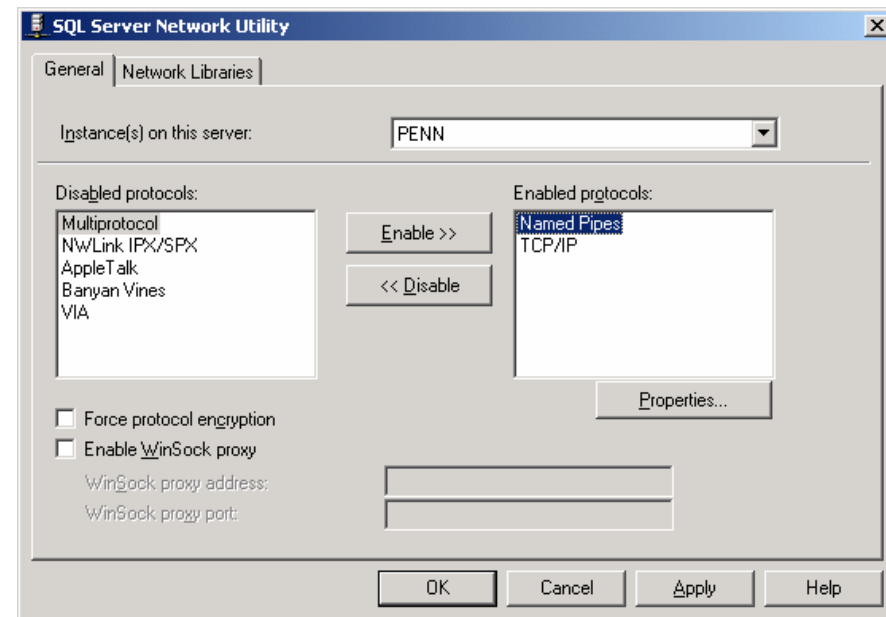
- SQL Server nutzt mehrere Prozessoren
 - Was Windows kann, kann SQL Server auch
- Echt parallele Ausführung von unterschiedlichen Queries
- Parallelisierung auch innerhalb eines einzigen Queries
- Bei kleinen Servern:
Achtung Lizenzkosten



- Je schneller, je besser
- Je mehr Platten, je besser
 - Partitionierung entsprechend Benutzung
 - Transaktions-Protokoll auf Extra-Platte
 - Stripe-Sets (RAID), parallelisieren Schreib/Lesezugriff
- Je mehr Controller, je besser

- Je schneller, je besser
- Problem bei
 - schlechter Bandbreite
 - Hohem Netzwerkverkehr, erzeugt durch Applikation
 - Hohem sonstigen Netzwerkverkehr

- WAN Verbindungen (VPN), RAS
- Mögliche Alternativlösungen
 - Terminal Server
 - Web Applikation
- TCP/IP anstelle von Named Pipes



- Applikationen holen sich alle Daten über's Netz und selektieren lokal
 - Beispiel: DATEV
 - Select * from Tabelle
 - Grund: Unkenntnis der Programmierer ?
- Applikationen setzen viele kleine SQL Statements ab, anstelle eines großen Joins, bzw. einer Menge von Statements in einer Übertragung
 - Oft nötig für Datenbankunabhängigkeit
- Overhead durch Verbindungsart
 - Oft nötig für Datenbankunabhängigkeit
 - Bspw. ODBC, ADO

- Physikalisches Datenbankdesign
 - Langsamer Dateizugriff
- Logisches Datenbankdesign
 - Auffinden der gewünschten Daten
 - Multi-Userprobleme

- Applikations-abhängiges Design
 - Verteilung von Daten auf Datenbanken und Tabellen
 - Aufgabe des Herstellers, nicht Thema dieses Vortrags
- Applikations-unabhängiges Design
 - Schrauben, an denen man drehen kann
 - Oft vom Anwendungsfall abhängig
 - Thema des Vortrags

- Im Normalfall zwei Dateien in einer File-Group
 - Daten-Datei (.mdf)
 - Transaktionsprotokoll (.ldf)
 - File-Group „PRIMARY“
- Weitere Filegroups und Dateien können definiert werden.
- Ziele:
 - Verteilung auf unterschiedliche Medien
 - Verteilung von strukturierten und Blob-Daten (Image, Text, Binary)
 - Verteilung von Programm, Daten und Log

- Eigene Filegroups möglich für
 - Jede Tabelle
 - zusammen mit Clustered Index
 - Text (Blob) und normale Daten separat
 - Jeder Index
- Wachsen von Datenbankdateien
 - Bei kleinen Dateien automatisch
 - Bei großen Dateien eher manuell wenn wenige User arbeiten
- Read/Write oder Read/Only
 - Bspw. Konfigurationstabelle oder Archiv in R/O Filegroup

- Erzeugt aus dem SQL Befehl einen Ausführungsplan
- Kostenbasiert
 - Mögliche Pläne werden bewertet
 - Der „preiswerteste“ wird angewandt
- Korrekte Bestimmung der Kosten
 - ist die Herausforderung
 - geschieht durch Bewertung von Indexwahl und Statistiken

- Datenstrukturen
 - B-Baum
 - Hash Tabelle
- Statistiken
 - Helfen bei Entscheidung des Optimierers
 - Selektivität von Feldern
 - `dbcc show_statistics [<Tabelle>] <PK>`
- Indexarten
 - Einfacher Index
 - Composite Index
 - Covering Index

- Index auf ein Feld in der Tabelle
- Pro Query kann mehr als ein Index zum Einsatz kommen
- Je kleiner ein Index Feld, umso mehr Records pro Page

- Index mit mehreren Feldern
- Nur verwendet, wenn erste n Felder in Bedingung
- Selektivität des ersten Feldes für Optimierer ausschlaggebend bei Entscheidung für/gegen Index

- Composite Index
- Zweck: enthält alle Felder einer Tabelle, die im Query vorkommen
 - In select Klausel
 - In Join
 - In where Klausel
 - In group-by, order-by Klauseln
- Kein extra Zugriff auf Tabelle mehr notwendig
- Limit: Index-Seitengröße von 900 Byte

- Seit SQL 2000 Bit-Feld auch als Teil von Index möglich
 - Selektivität schlecht
 - Daher nur in Covering Index ratsam
 - Enterprise Manager läßt das noch nicht zu, aber in SQL direkt eingebbar
- FILLFACTOR und PAD_INDEX
 - bei Index nur dann, wenn Schreibvorgänge > 30% der Lesevorgänge

- Die Tabelle selbst ist der Index
- Falls vorhanden, wird der Schlüssel in jedem anderen Index als Zeiger auf die Daten verwendet
 - Kein weiterer Index anzupassen bei Insert
 - Jeder andere Index wird groß bei einem Clustered Index auf (var)char-Felder!!!
 - Falls nicht unique, werden eindeutige Dummy-Werte hinzugefügt (unsichtbar für User/Client)

- Verfügbar in SQL Server Enterprise Edition
- Auch ein View kann einen clustered Index haben
- Index kann verwendet werden bei
 - View
 - Ähnlich definiertem Query
 - Macht Sinn, selbst wenn Applikation keinen View definiert, sondern reines SQL
- Vor allem bei Joins, Aggregat-funktionen, Group-By-Klauseln

- Abbildung der Daten in Bits/Bytes
- Beispiel: Ganzzahl
 - im Programm wird codiert als
 - tinyint (1 Byte)
 - smallint (2 Byte)
 - int (4 Byte)
 - bigint (8 Byte)
 - Je mehr Bytes,
 - umso größere Zahlen sind möglich
 - umso größer wird auch der Platzbedarf

- Alle Daten eines Records müssen auf eine Datenseite passen
 - Seitengröße: 8KB
 - Ausnahme der Regel:
 - BLOB Daten haben eigene Seite(n)
- Mehr als ein Record kann auf eine Datenseite
- Je mehr Records pro Seite, umso geringer der I/O Aufwand

- Gibt Maximalgröße an
- Weniger Platz wird verbraucht, falls die tatsächlichen Daten geringer sind
- Maximalgröße eines Records kann > 8KB sein, tatsächliche Größe nicht
 - Beispiel:
Vorname varchar(4500)
Nachname varchar(4500)
Firma varchar(5000)
 - Fehler erst bei insert/update, falls Gesamtgröße > 8KB
- nvarchar: wie varchar, aber Unicode: 2 Bytes pro Zeichen

Zwei Hauptprobleme

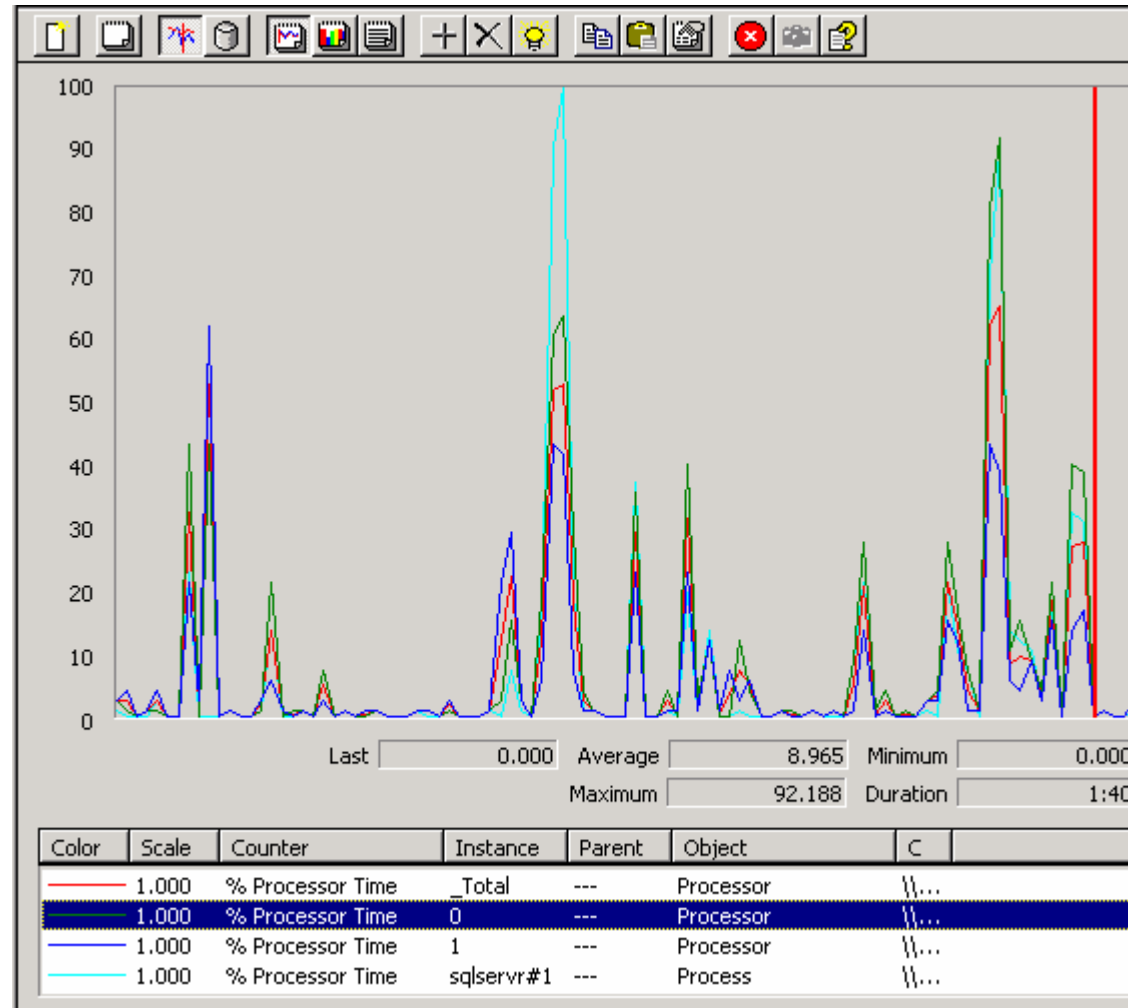
- Mehr Last für die Datenbank
 - Mehr Benutzer, mehr Arbeit
- Sperrverhalten
 - Mehr Verkehr
 - Mehr Verkehrsregeln
 - Benutzer müssen an der roten Ampel aufeinander warten

- Paralleler Zugriff auf Daten bedingt Sperren
 - Shared (S) Lock beim Lesezugriff
 - Exclusive (X) Lock beim Schreibzugriff
- Isolation Level
 - Regelt Freigabe von Sperren

- Granularität der Sperren
 - Wird vom Server festgelegt
 - Zeilenweise
 - Seitenweise
 - Tabellenweise
- Je mehr Records pro Seite, umso wahrscheinlicher Kollision von unterschiedlichen Queries
 - Direkter Widerspruch zur I/O Optimierung

- Performance Monitor
- Query Analyzer
- SQL Profiler
- Sonstige Tools
 - DBSelect
- Maintenance Skripte unter
 - <http://www.mssqlserver.com/scripts/>

- Zeigt Performance Zähler an, die Applikationen zur Verfügung stellen
 - Speicher und CPU-Last
 - Plattenzugriffe
 - Locks, Timeouts, Zugriffsmethoden, ...
- Problem:
 - Oft zu viel Information
 - Dokumentierte Performance Counter fehlen manchmal



- Details beschrieben unter
 - Administering SQL Server
 - Monitoring Server Performance and Activity
 - Monitoring with System Monitor
- Enthält Regeln, nach denen Flaschenhalse identifiziert werden können.

- SQL Editor zur ad-hoc Ausführung von SQL Befehlen
- Möglichkeit, Execution Plan für auszuführende SQL Befehle anzuzeigen
- Liefert Information, ob Index verwendet wurde (und welcher), oder nicht

Zwei Ansätze zur Performance Analyse

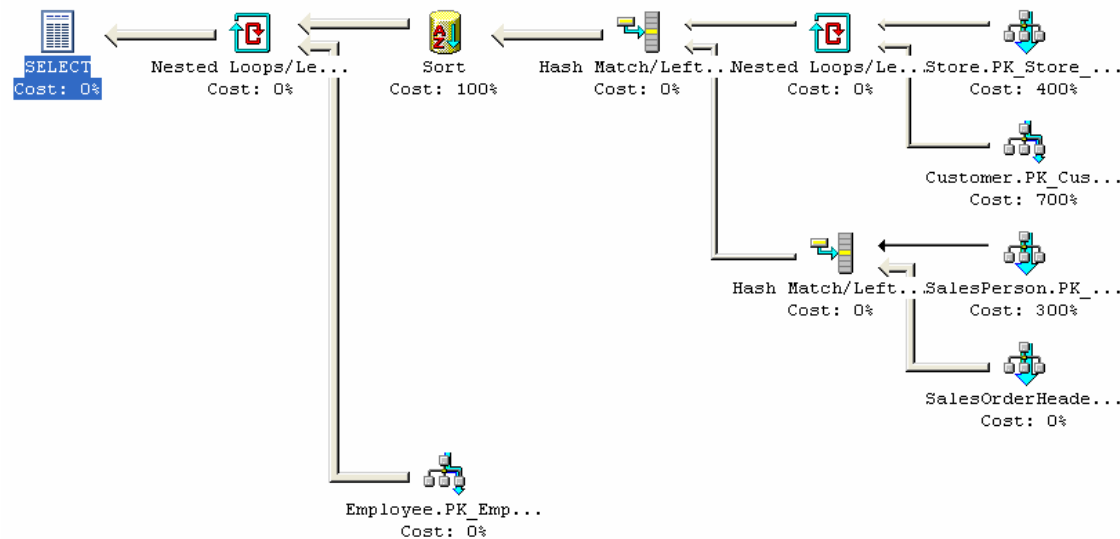
- SQL Statement bekommen, um dann im Query Analyzer damit zu „spielen“
 - Schönere Optik
 - Geht nicht immer (falls Applikation Cursor verwendet, ...)
- Direkt den Execution Plan im Profiler anzeigen
 - Als Event noch „Performance, Execution Plan“ auswählen

- Analysiert SQL Befehle
 - .sql Datei
 - .trc Protokoll des SQL Profilers
- Sucht nach Indexdefinitionen, die ideal für gegebene SQL Befehle sind
- Schlägt keinen Covering Index vor

```

select s.name as StoreName, ep.FirstName, ep.Lastname,
       soh.subtotal, soh.comment
from store s
      left join customer cu on s.customerid=cu.customerid
      left join salesperson sp on cu.salespersonid=sp.salespersonid
      left join employee ep on ep.employeeid=sp.salespersonid
      left join salesorderheader soh on soh.salespersonid=sp.salespersonid
order by storename
  
```

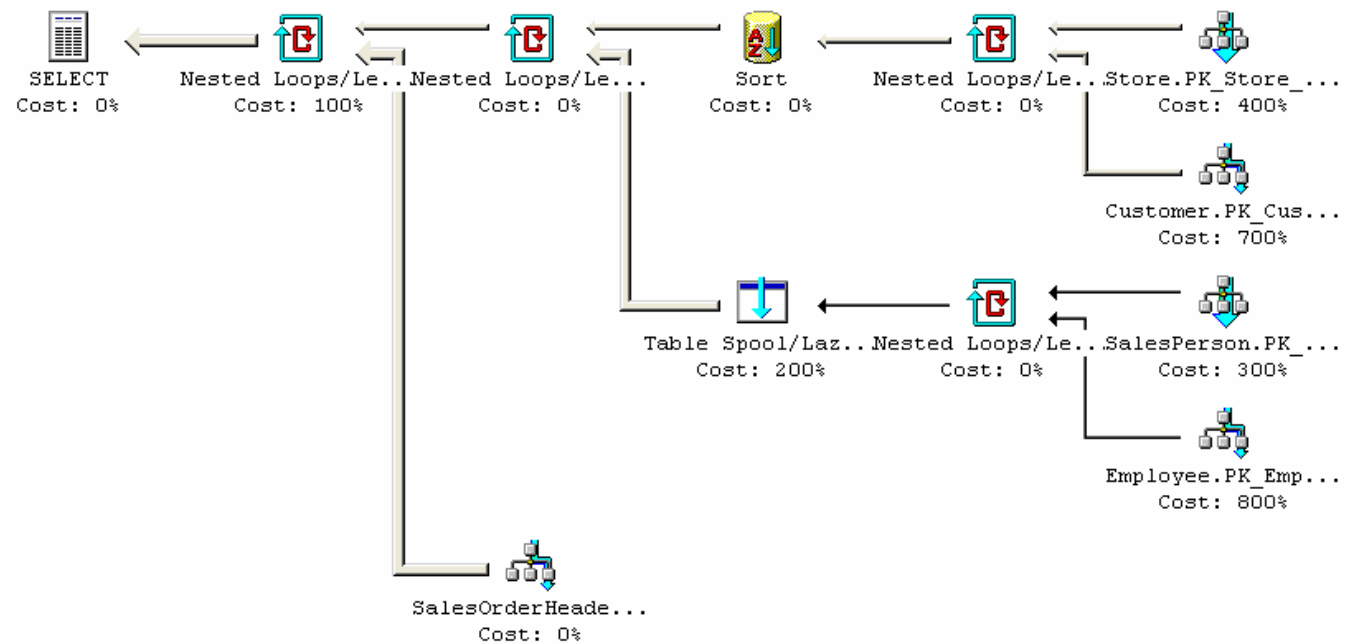
Dauer: 8 Sekunden
211025 Zeilen



- Statistiken erstellen/aktualisieren (im Beispiel schon gemacht)
- Covering Index erstellen für Tabellen
 - SalesOrderHeader (wegen der Felder)
- Jetzt: 3sec

Index name: IX_SalesOrderHeader_CoveringIndex

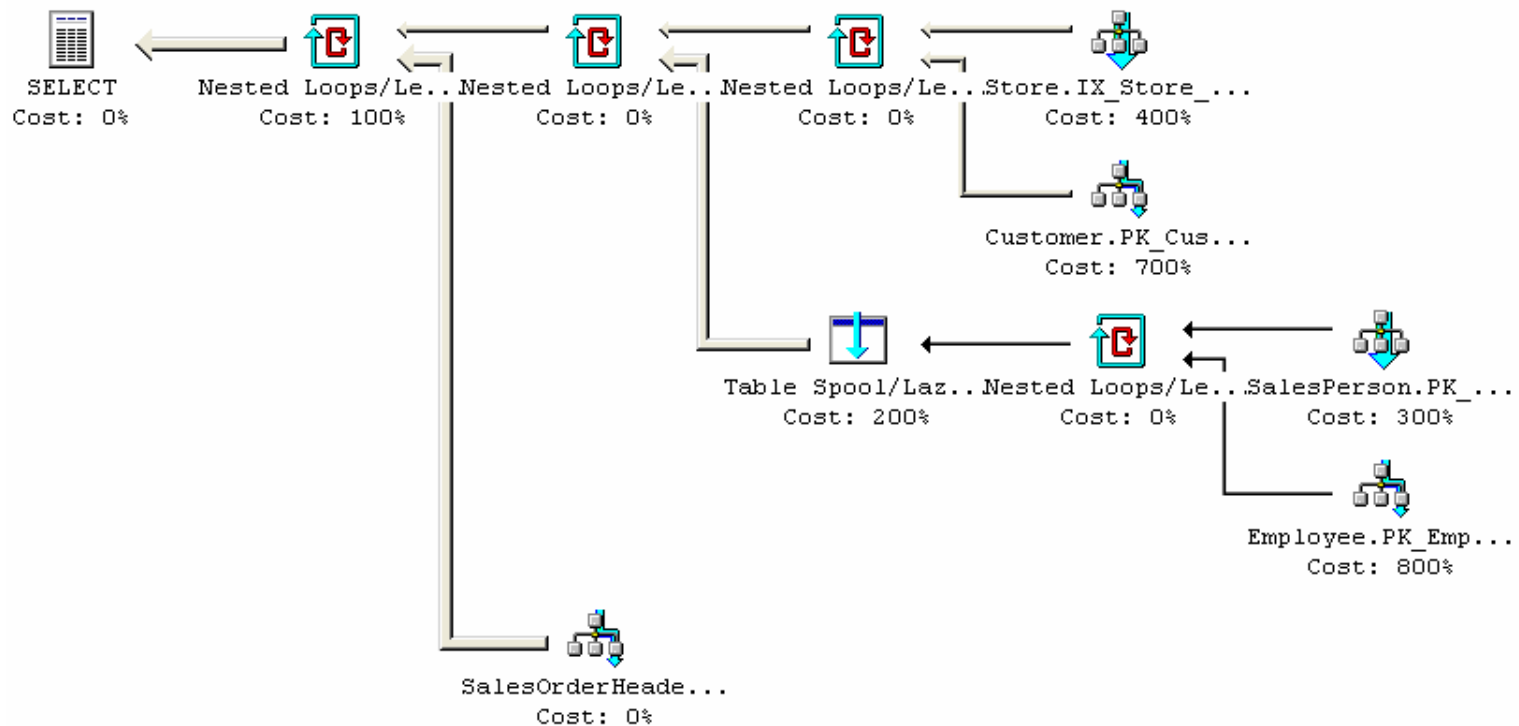
Column name	Order
SalesPersonID	Ascending
SalesOrderID	Ascending
SubTotal	Ascending
Comment	Ascending



- Covering Index erstellen für Tabellen
 - Store (wegen der Felder, Order By)
- Jetzt: 2sec

Index name: IX_Store_CoveringIndex

Column name	Order
Name	Ascending
CustomerID	Ascending
StoreID	Ascending



- Backup und Recovery
 - Recovery Modelle
 - Simple
 - Full Recovery
 - Bulk-Logged Recovery
- Statistiken
 - Manuell mit Create/Update Statistics
 - Automatisch über DB-Option
 - Am besten manuell per Maintenance Plan
- Reorganize
 - Reorganisiert Daten auf den Pages

Tipp: alles machbar mit SQL

- Maintenance Pläne sind SQL Befehle
 - Einfach mal anschauen und ggf. modifizieren
- Enterprise Manager kommuniziert mit dem Server auch in SQL
 - Reorganisiert Daten auf den Pages

- Service Packs lohnen fast immer
 - Software Update Service meist auch
- Integrierte Sicherheit ist sicherer, als SQL Server Sicherheit
 - Schwierig für Applikationen, die mehrere Datenbanken unterstützen
 - Wichtig: sa-Passwort
- Port 1433 zu wechseln bringt nicht viel
 - Jeder Port Scan lüftet das Geheimnis
 - Besser: direkten Zugriff übers Internet komplett verbieten (notfalls per VPN)
- Extended Stored Procedure xp_cmdShell verbieten

- SQL Server Books Online
 - SQL Server Architecture
 - Administering SQL Server
 - Optimizing Database Performance
- SQL Server Resource Kit / Technische Referenz
- <http://www.sql-server-performance.com/>
- <http://www.sqlmag.com/>

Fragen?