Speicheroptimierte Tabellen

Was sind speicheroptimierte Tabellen?

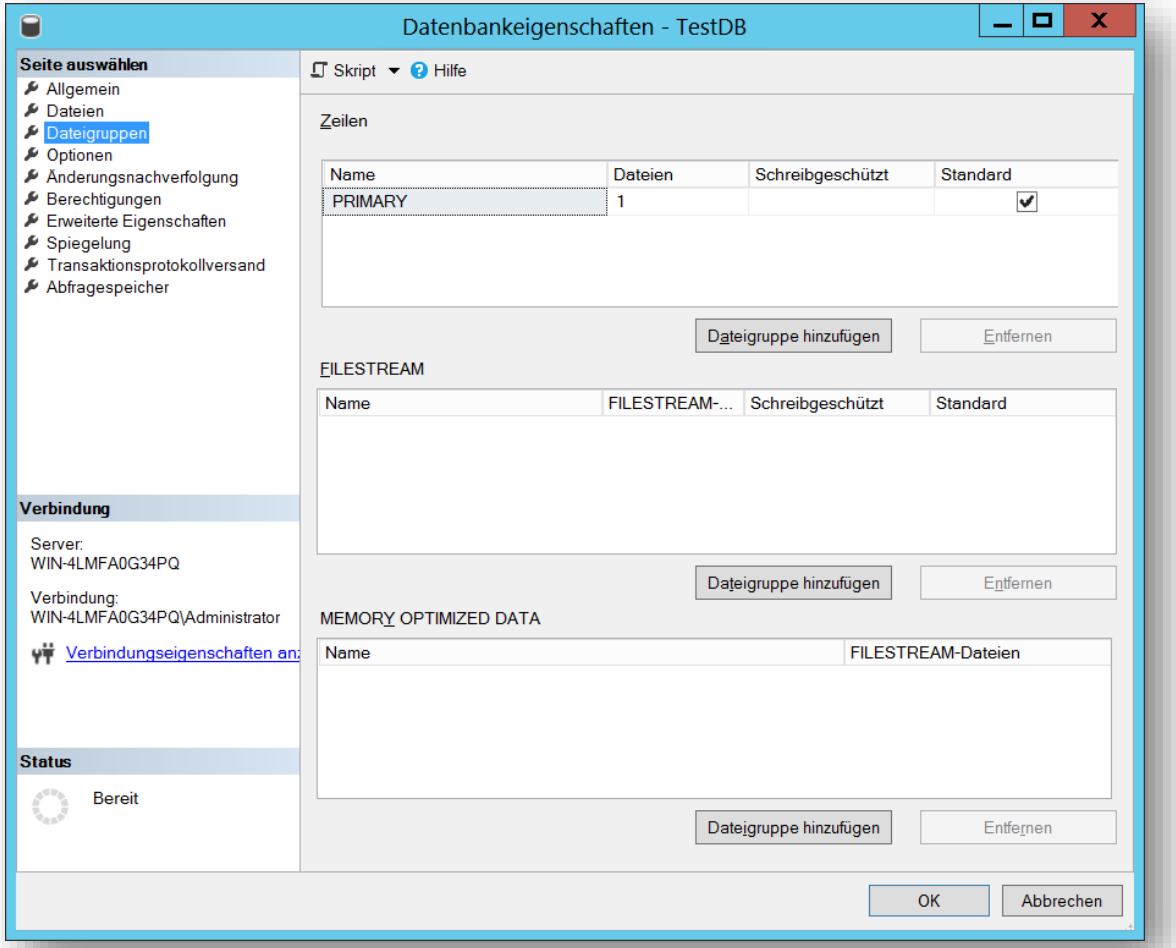
- Speicheroptimierte Tabellen befinden sich im Arbeitsspeicher
  - Zeilen werden aus dem Arbeitsspeicher gelesen und in diesen geschrieben
- Vorteile: weniger CPU-Belastung, reduzierter Festplattenzugriff
- Speicheroptimierte Tabellen können auch über Versionierung verfügen
- Benötigen den Zusatz `WITH (MEMORY_OPTIMIZED = ON)`
- Benötigen Primärschlüssel oder Index (nicht gruppiert!)
- Erfordern zwingend eine speicheroptimierte Dateigruppe
  - Direkt beim Erstellen der Datenbank anlegen oder nachträglich hinzufügen
  - Thema der Administrierung, daher hier nur grob erklärt

Exkurs Dateigruppen

- Dateigruppen zu einer Datenbank hinzufügen
  - Mit T-SQL oder per MMS

```
USE [master]
GO
ALTER DATABASE [MODB]
ADD FILEGROUP [MODBDG]
CONTAINS MEMORY_OPTIMIZED_DATA;
```

  - In der GUI eine Dateigruppe vom Typ **MEMORY_OPTIMIZED_DATA** hinzufügen
  - Die Dateigruppe alleine reicht noch nicht, man braucht auch eine Datei innerhalb der Dateigruppe

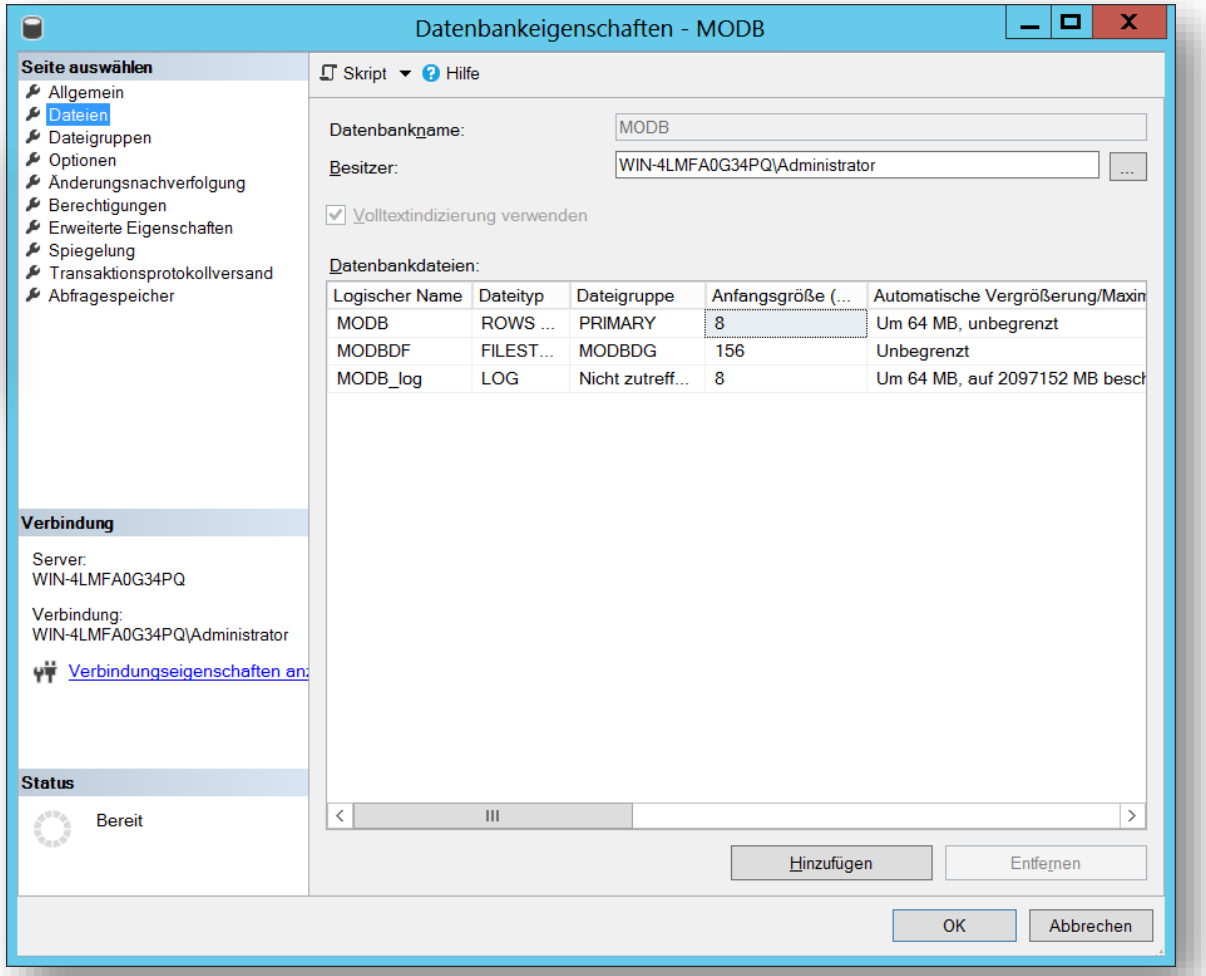


Exkurs Dateien

- Dateien zu einer Datenbank hinzufügen
  - Mit T-SQL oder per MMS

```
ALTER DATABASE [MODB]
ADD FILE
(    NAME = N'MODBDF',
     FILENAME = N'C:\Program Files\Microsoft SQL Server\MSS
TO FILEGROUP [MODBDG];
```

  - Der Dateityp muss FILESTREAM sein
  - SIZE und FILEGROWTH-Parameter dürfen bei speicheroptimierten Tabellen nicht angegeben werden



Anlegen einer speicheroptimierten Tabelle

```
CREATE TABLE Test
(
    Zahl INT PRIMARY KEY NONCLUSTERED
)
WITH (MEMORY_OPTIMIZED = ON,
      DURABILITY = SCHEMA_AND_DATA);
```

Mindestens Index muss vorhanden sein

Kennzeichnung der Tabelle als speicheroptimiert

Dauerhaftigkeit der Daten haben die Schalter SCHEMA_AND_DATA und SCHEMA_ONLY

- SCHEMA_AND_DATA (optionale Angabe) ist ein Default-Wert; er bedeutet, dass die Datenänderungen auch dauerhaft gespeichert werden
- SCHEMA_ONLY bedeutet, dass das Tabellenschema beibehalten wird, aber die Datenänderungen bei Neustart oder Failover verloren gehen

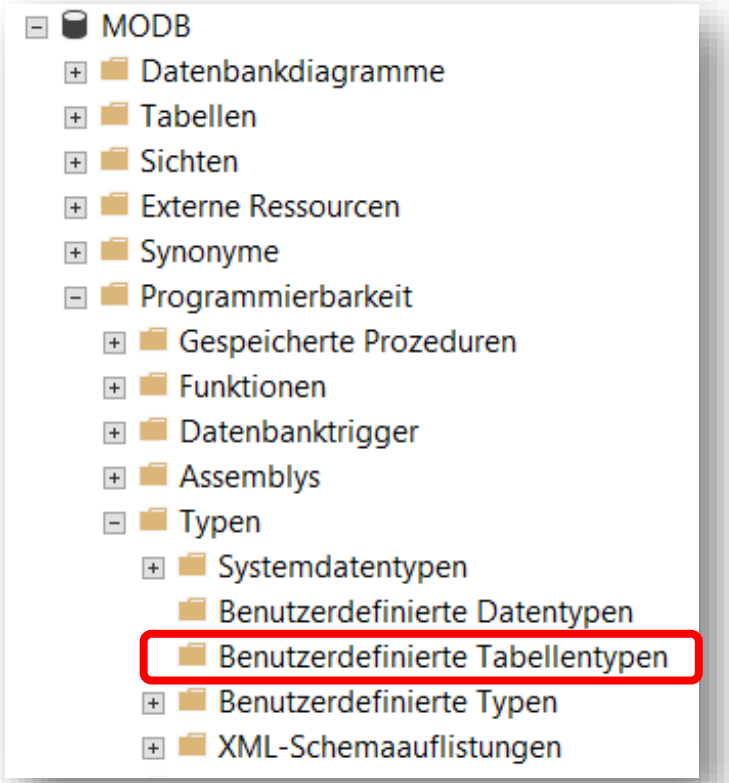
Eigenschaften speicheroptimierter Tabellen

- Änderungen mit ALTER TABLE sind nur sehr eingeschränkt nutzbar
  - https://docs.microsoft.com/de-de/sql/relational-databases/in-memory-oltp/altering-memory-optimized-tables

- Können die Abfrageleistung sehr stark erhöhen (bis Faktor 99)
- Die **CREATE**-Anweisung und Tabelleninformationen werden in die Datenbankmetadaten geschrieben
- Tabellen- und Indexstrukturen werden im Speicher erstellt
- Die Tabelle wird außerdem zu einer *.dll kompilliert
  - „Dynamic Link Library“: Programmbibliotheken, gleichzeitig mehrfach nutzbar, ähnlich *.exe

Speicheroptimierte Tabellenvariable

- Verfügt über **keine** Komponente auf dem Datenträger
  - Keine I/O-Aktivität
- Keine Nutzung oder Konflikte in Bezug auf die tempdb
- Kann an eine gespeicherte Prozedur als Tabellenwertparameter übergeben werden
- Benötigt mindestens einen Index (Hash-Index oder nicht gruppierter Index)
  - https://docs.microsoft.com/de-de/sql/relational-databases/sql-server-index-design-guide#hash_index
- Wird als datenbankeigener Typ erstellt



Beispiel

```
CREATE TYPE EinTabellenTyp AS TABLE
(
    ID INT PRIMARY KEY NONCLUSTERED,
    Daten NVARCHAR(max) NOT NULL
)
WITH (MEMORY_OPTIMIZED = ON);
```

- Schalter **WITH (MEMORY_OPTIMIZED = ON)**
- Typ kann jetzt wie ein Datentyp verwendet werden

Verwendungsbeispiel:

```
DECLARE @tab EineTabelle;

INSERT INTO @tab
VALUES
(1, 'haha');

SELECT * FROM @tab;
```

	ID	Daten
1	1	haha

Speicheroptimierte Stored Procedures

- Werden in systemeigenen Code kompiliert
  - „nativ“ kompiliert
  - Benötigen das Schlüsselwort **NATIVE_COMPILATION**
  - Unterstützen nur eine eingeschränkte Auswahl an T-SQL Schlüsselwörtern
- Greifen auf speicheroptimierte Tabellen zu
- Ermöglichen effiziente Ausführung der Abfragen der Geschäftslogik der Prozedur
- Übergabeparameter haben eine **NOT NULL** Einschränkung
- Schemabindung + Schlüsselwort **ATOMIC**
  - https://docs.microsoft.com/de-de/sql/relational-databases/in-memory-oltp/creating-natively-compiled-stored-procedures

Beispiel

```
CREATE PROCEDURE usp_beispiel(@zahl int)
WITH
    NATIVE_COMPILATION, SCHEMABINDING
AS
    BEGIN ATOMIC
    WITH
        (TRANSACTION ISOLATION LEVEL = SNAPSHOT,
        LANGUAGE = 'german')

            INSERT INTO dbo.Test
            VALUES (@zahl);
    END;
```

```
DECLARE @return_value int

EXEC    @return_value = [dbo].[usp_beispiel]
        @zahl = 12345

SELECT  'Return Value' = @return_value

SELECT * FROM Test;
```

	Zahl
1	4711
2	815
3	1234
4	12345