Tuntitehtävät vko 6
Tarkistusajot. Lukot, Deadlock:

a) Haluat varautua mahdolliseen SQL Server:in systeemi tietokantojen (master, model, msdb, tai resource) korruptoitumiseen. Mitä toimenpiteitä on tehtävä ennen systeemi tietokantojen uudelleen luomista (rebuild)?

Vastaus:
Tietokannat voivat korruptoitua puutteellisen huollon, ohjelmisto tai laitevikojen takia. Siksi on tärkeää seurata tietokantojen fragmentoitumisastetta ja tilaa yleensäkin (ks. edellisten viikkojen tehtävät). Jos tietokannan järjestelmän tilaa sisältävät tiedot, eli ns. systeemi tietokannat menetetään korruptoitumisen takia, ei tietokanta palvelinta saa esimerkiksi käynnistettyä enää muusta puhumattakaan. SQL Server:ssä nämä systeemi tietokannat ovat master, model, msdb ja resource. Jos ne korruptoituvat eli niitä ei voi enää lukea, voidaan ne luoda uudelleen. Tässä yhteydessä menetetään tietysti kaikki ne user-defined oliot, jotka on luotu master tietokantaan. Samoin käy kaikille scheduler:iin ajastetuille eräajoille, jotka ovat msdb:ssä. Myös kaikki model:iin tehdyt oletus asetukset menetetään. Alla on esitelty ne työvaiheet, jotka on tehtävä ennen systeemi tietokantojen uudelleen luomista SQL Server:ssä. Asiaa on käsitelty tarkemmin osoitteessa: http://technet.microsoft.com/en-us/library/dd207003(v=sql.105).

- Selvitä mitkä ovat tietokanta palvelimen asetukset:
```
SELECT * FROM sys.configurations;
```
- Selvitä kaikki hot fix päivitykset, service pack päivitykset, jotka on tehty kyseiseen tietokanta instanssiin. Myös tietokannassa käytössä olevat collation on tärkeää tietää. Nämä kaikki nimittäin täytyy päivittää systeemi tietokantojen uudelleen luonnin jälkeen. Nämä asiat saa selville lauseella:
```
SELECT
SERVERPROPERTY('ProductVersion ') AS ProductVersion,
SERVERPROPERTY('ProductLevel') AS ProductLevel,
SERVERPROPERTY('ResourceVersion') AS ResourceVersion,
SERVERPROPERTY('ResourceLastUpdateDateTime') AS
ResourceLastUpdateDateTime,
SERVERPROPERTY('Collation') AS Collation;
```
- Selvitä systeemi tietokantojen data tiedostojen ja transaktio loki tiedostojen sijainti hakemistojärjestelmässä. Jos olet muuttanut niiden paikkaa, on se tehtävä uudelleen systeemi tietokantojen luonnin jälkeen. Asian saa selville lauseella:
```
SELECT name, physical_name AS current_file_location
FROM sys.master_files
WHERE database_id IN (DB_ID('master'), DB_ID('model'), DB_ID('msdb'),
DB_ID('tempdb'));
```
- Varmistusten olemassa olo on tärkeää systeemi tietokantojen osalta. Niitä tarvitaan nyt.

b) Jos muuttaisit SQL Server:ssä lukituksen READ UNCOMMITED tyyppiseksi, niin mitkä olisivat seuraukset. Testaa tilannetta luomalla pankkiA tietokanta ja sinne tilit taulu. Lisää kyseiseen tauluun sopivaa dataa testausta varten. Testauksen voi tehdä ottamalla kaksi eri yhteyttä tähän tietokantaan yhtä aikaa ja editoimalla ja select:oimalla siellä olevaa tietoa. Millaiset ovat tämän testin tulokset?

Vastaus:
READ UNCOMMITTED avaa käytännössä lukot tietokannassa. Sen vuoksi kesken olevan transaktion tilanne näkyy muille transaktioille. Jos esimerkiksi suoritettavasta tilisiirrosta on ehditty suorittaa vain ensimmäinen lause eli vähentää asiakkaan 1 tililtä laskun edellyttämä summa (esim. 40 euroa), ja heti tämän jälkeen suoritetaan jonkin toisen transaktion raportin ajo kyseisen pankin talletuksista, eivät seuraukset ole hyvät. Raportissa näkyy 40 euron vajaus tileissä. Jos lukkoja ei olisi avattu, ei tällainen tilanne olisi mahdollista. Ks. http://msdn.microsoft.com/en-us/library/ms175519(v=sql.105).aspx, http://msdn.microsoft.com/en-us/library/ms173763(v=sql.105).aspx.

- Luodaan ensimmäiseksi pankkiA niminen tietokanta SQL Server 2008 R2:een. Huomaa, erityisesti alla olevassa pankkiA tietokannan luonti scriptissä, että `cursor_close_on_commit` asetus on `FALSE`. Myös statistiikan automaattipäivitys on laitettu päälle, asetuksella `auto_update_statistics on`. Se voisi olla hyvä laittaa `off` asentoon tuotannollisessa tietokannassa:

```
CREATE DATABASE [pankkiA] ON  PRIMARY
( NAME = N'pankkiA', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\pankkiA.mdf' , SIZE = 2048KB ,
MAXSIZE = 1024000KB , FILEGROWTH = 102400KB )
 LOG ON
( NAME = N'pankkiA_log', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\pankkiA_log.ldf' , SIZE = 1024KB ,
MAXSIZE = 512000KB , FILEGROWTH = 10%)
GO
ALTER DATABASE [pankkiA] SET COMPATIBILITY_LEVEL = 100
GO
ALTER DATABASE [pankkiA] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [pankkiA] SET ANSI_NULLS OFF
GO
ALTER DATABASE [pankkiA] SET ANSI_PADDING OFF
GO
ALTER DATABASE [pankkiA] SET ANSI_WARNINGS OFF
GO
ALTER DATABASE [pankkiA] SET ARITHABORT OFF
GO
ALTER DATABASE [pankkiA] SET AUTO_CLOSE OFF
GO
ALTER DATABASE [pankkiA] SET AUTO_CREATE_STATISTICS ON
GO
ALTER DATABASE [pankkiA] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [pankkiA] SET AUTO_UPDATE_STATISTICS ON
GO
ALTER DATABASE [pankkiA] SET CURSOR_CLOSE_ON_COMMIT OFF
GO
ALTER DATABASE [pankkiA] SET CURSOR_DEFAULT  GLOBAL
GO
ALTER DATABASE [pankkiA] SET CONCAT_NULL_YIELDS_NULL OFF
GO
ALTER DATABASE [pankkiA] SET NUMERIC_ROUNDABORT OFF
GO
ALTER DATABASE [pankkiA] SET QUOTED_IDENTIFIER OFF
GO
ALTER DATABASE [pankkiA] SET RECURSIVE_TRIGGERS OFF
```

```
GO
ALTER DATABASE [pankkiA] SET  DISABLE_BROKER
GO
ALTER DATABASE [pankkiA] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
ALTER DATABASE [pankkiA] SET DATE_CORRELATION_OPTIMIZATION OFF
GO
ALTER DATABASE [pankkiA] SET PARAMETERIZATION SIMPLE
GO
ALTER DATABASE [pankkiA] SET  READ_WRITE
GO
ALTER DATABASE [pankkiA] SET RECOVERY FULL
GO
ALTER DATABASE [pankkiA] SET  MULTI_USER
GO
ALTER DATABASE [pankkiA] SET PAGE_VERIFY CHECKSUM
GO
USE [pankkiA]
GO
IF NOT EXISTS (SELECT name FROM sys.filegroups WHERE is_default=1 AND name
= N'PRIMARY') ALTER DATABASE [pankkiA] MODIFY FILEGROUP [PRIMARY] DEFAULT
GO
```

- Luodaan tämän jälkeen tilit taulu pankkiA tietokantaan:

```
USE pankkiA;
GO
CREATE TABLE tilit (
      tilinro int not null primary key,
      saldo decimal(8,2) not null,
);
```

- Lisää tarvittavat datarivit tilit tauluun:

```
insert into tilit (tilinro, saldo) values (111222, 1050);
insert into tilit (tilinro, saldo) values (999888, 567321);
```

- Avaa kaksi eri yhteyttä tietokanta instanssiin (avaa nämä kaksi yhteyttä SQL Server Management Studiolla):
- Avaamasi 1 istunto ikkuna on asiakkaan yhteys pankkiA tietokantaan. Asiakas, jonka tilinumero on 111222, haluaa suorittaa 40 euron maksun tilille 999888. Tähän tarvitaan kaksi update lausetta, jossa ensimmäinen lause vähentää 40 euroa asiakkaan tililtä. Toisena oleva update lause lisää tämän 40 euroa tilille 999888. Suoritetaan ensimmäiseksi vain ensimmäinen näistä lauseista, eli lause:

```
begin transaction
update tilit set saldo = saldo - 40 where tilinro = 111222;
```

- Jatketaan seuraavaksi avaamassasi 2 ikkunassa. Se on pankin kontrollerin ikkuna, jossa hän haluaa raportin pankin tileistä.

```
use pankkiA;
go
set transaction isolation level read uncommitted;
select * from dbo.tilit;

tilinro     saldo
----------- ---------------------------------------
111222      1010.00
999888      567321.00

(2 row(s) affected)
```

Pankin kontrolleri siis näkee keskeneräisen 40 euron tilisiirron asiakkaan tilillä, jota ei ole vielä ehditty lisätä 999888 tilille.

- Siirrytään seuraavaksi 1 ikkunaan, eli asiakkaan ikkunaan. Seuraavaksi jatketaan kesken eräistä tilisiirtoa eteenpäin lauseella:

```
update tilit set saldo = saldo + 40 where tilinro = 999888;
commit;
```

- Siirry takaisin 2 ikkunaan, eli pankin kontrollerin ikkunaan. Otetaan raportti uudelleen lauseella:

```
select * from dbo.tilit;
```

Set transaction isolation level lauseen syntaksi on:

```
SET TRANSACTION ISOLATION LEVEL
  { READ UNCOMMITTED
  | READ COMMITTED
  | REPEATABLE READ
  | SNAPSHOT
  | SERIALIZABLE
  }
[ ; ]
```

Siitä on kerrottu osoitteessa: http://msdn.microsoft.com/en-US/library/ms173763(v=sql.105).aspx.

c) Miten vastaavaa voisi kokeilla MySQL:ssä? Anna tarvittavat lauseet:

Vastaus:
Lukkoja ja transaktioiden eristystasojen muuttamista on käsitelty MySQL:n reference manual:ssa osoitteessa: http://dev.mysql.com/doc/refman/5.1/en/innodb-transaction-model.html
(MySQL:n Reference Manual 5.6:n pdf tiedostossa sivulla 1110):

```
SET [GLOBAL | SESSION] TRANSACTION
transaction_characteristic [, transaction_characteristic] ...
transaction_characteristic:
ISOLATION LEVEL level
| READ WRITE
| READ ONLY
level:
REPEATABLE READ
| READ COMMITTED
| READ UNCOMMITTED
| SERIALIZABLE
```

Ks. MySQL:n Reference Manual:sta s. 1102:

```
SET autocommit

START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

c) Kokeile SQL Server:ssä b tehtävän tilannetta kun transaktioiden eristystasona on read committed. Millaiseen tilanteeseen päädytään?

Vastaus:
Jos pankkiA tietokannan `CURSOR_CLOSE_ON_COMMIT FALSE` on siis false arvossa, aiheuttaa asiakkaan keskeneräinen transaktio pankin kontrollerin transaktion jumiutumisen deadlock tilanteeseen, jos kontrollerin transaktio tarvitsee asiakkaan transaktion lukitsemia resursseja. Tilanteen avaa asiakkaan transaktion jatkuminen eteenpäin ja lopuksi commit:ointi.

- Siirrytään 1 ikkunaan eli asiakkaan ikkunaan, ja suoritetaan lauseet:

```
begin transaction
update tilit set saldo = saldo - 40 where tilinro = 111222;
```

- Siirrytään 2 ikkunaan eli pankin kontrollerin ikkunaan:

```
set transaction isolation level read committed;
select * from dbo.tilit;
```

Päädytään deadlock tilanteeseen.

- Siirrytään seuraavaksi asiakkaan 1 ikkunaan, ja suoritetaan lauseet:

```
update tilit set saldo = saldo + 40 where tilinro = 999888;
commit;
```

- Siirrytään 2 ikkunaan eli pankin kontrollerin ikkunaan. Siellä on saatu nyt suoritettua loppuun deadlock:ssa ollut select –lause.

Tällä tavalla päästiin näkemään käytännössä miten lukot toimivat tietokannoissa ja mitä eritystasot tarkoittavat. Tämä tehtävä myös toi esille miten tärkeää on, että tietokannassa ei ole ns. auto commit asetusta päällä. Silloinhan jokainen transaktion lause commit:oidaan erikseen ja itse asiassa päädytään a tehtävän ongelmaan. Transaktion epäonnistuessa syystä tai toisesta on se aina oltava ohjelmoitu sovellukseen niin, että se rollback:tään eli peruutetaan.

d) Miten SQL Server:ssä voi tarkastaa tietokannan objektien loogisen ja fyysisen eheyden?

Vastaus:

Tietokannan objektien loogisen ja fyysisen eheyden voi tarkistaa komennolla (ks. http://msdn2.microsoft.com/en-us/library/ms176064.aspx):

```
dbcc checkdb ('AdventureWorks');
```

e) Miten SQL Server:ssä palautetaan tietokannan ja sen indeksien eheys.

Vastaus:

Seuraavassa on esitetty ohjeet tietokannan ja sen indeksien eheyden palauttamiseksi. Kyseessä on toimenpide, jonka aikana tietokanta ei tietenkään ole muiden käyttäjien käytettävissä. Tietysti myös varmistuskopioinnit olisi syytä olla tehtyinä koko tietokannasta (ks. http://msdn2.microsoft.com/en-us/library/ms174269.aspx):

1) Siirry ns. single-user tilaan:

```
alter database AdventureWorks
set single_user
go
```

2) Nopeiten pienimmät eheysehdot voi korjata komennolla:

```
dbcc checkdb ('AdventureWorks', repair_fast)
go
```

3) Edellinen komento ei korjaa indeksejä tietokannassa. Ne voi pyrkiä korjaamaan komennolla:

```
dbcc checkdb ('AdventureWorks', repair_rebuild)
go
```

4) Jos tilanne eheysehtojen korjaamisessa tietokannassa on niin huono, että kumpikaan yllä olevista komennoista ei niitä korjaa, voi yrittää seuraavaa dramaattisempaa komentoa:

```
dbcc checkdb ('AdventureWorks', repair_allow_data_loss)
go
```

Palautetaan AdventureWorks tietokanta lopuksi multi_user tilaan, jotta siihen pääsevät nyt muutkin käyttäjät:

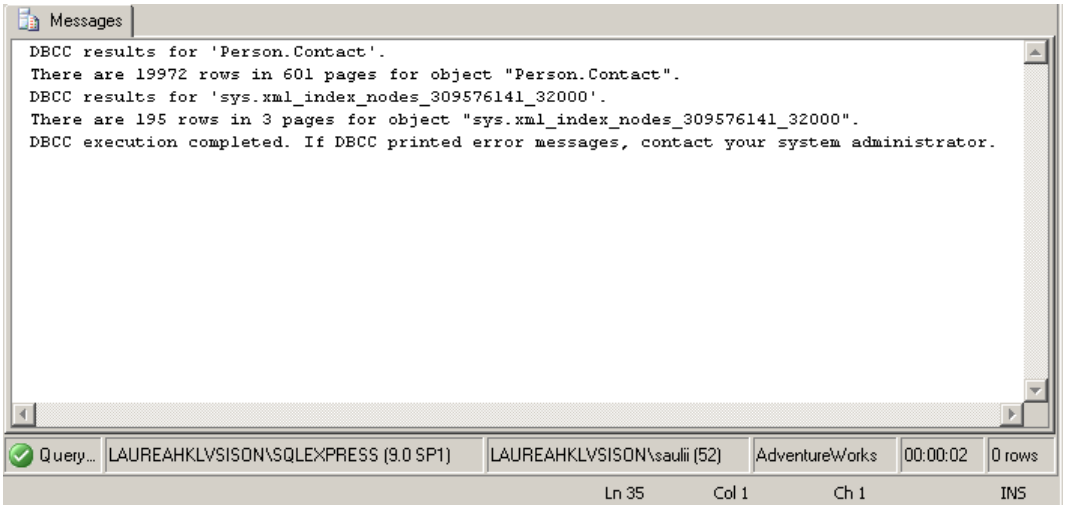
```
alter database AdventureWorks
set multi_user
go
```

f) SQL Server:ssä on hyvä aina tarkastaa, että varmistettavat tietokannan taulut ovat eheitä. Eli että käytännössä varmistus myös onnistuu, eikä varmistuksessa vain varmisteta tauluja, joiden eheys ei ole enää kunnossa. Miten tämä tehdään SQL Server:ssä?

Vastaus:

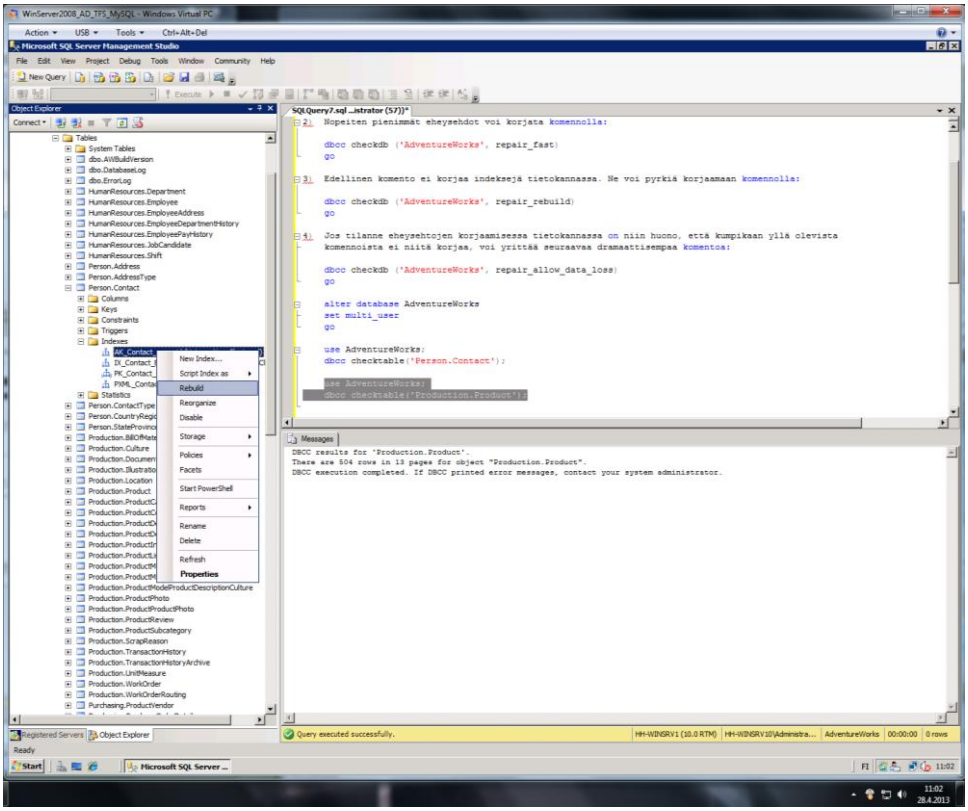
Silloin tällöin on syytä tarkistaa data- ja indeksisivujen linkitys ja oikea järjestys. Tämä on syytä tehdä erityisesti ennen varmistusnauhoitusta. Teet sen komennolla `dbcc checktable` (SQLServer v 6.5:ssä on ollut ongelmia B-puun muistiviittauksissa, joten on hyvä tehdä edelleen, ks. http://msdn2.microsoft.com/en-us/library/ms174338.aspx). Taulujen eheyden tarkistaminen voi olla keino korjata tietokanta, jos edellisen tehtävän `checkdb` ei auttanut.

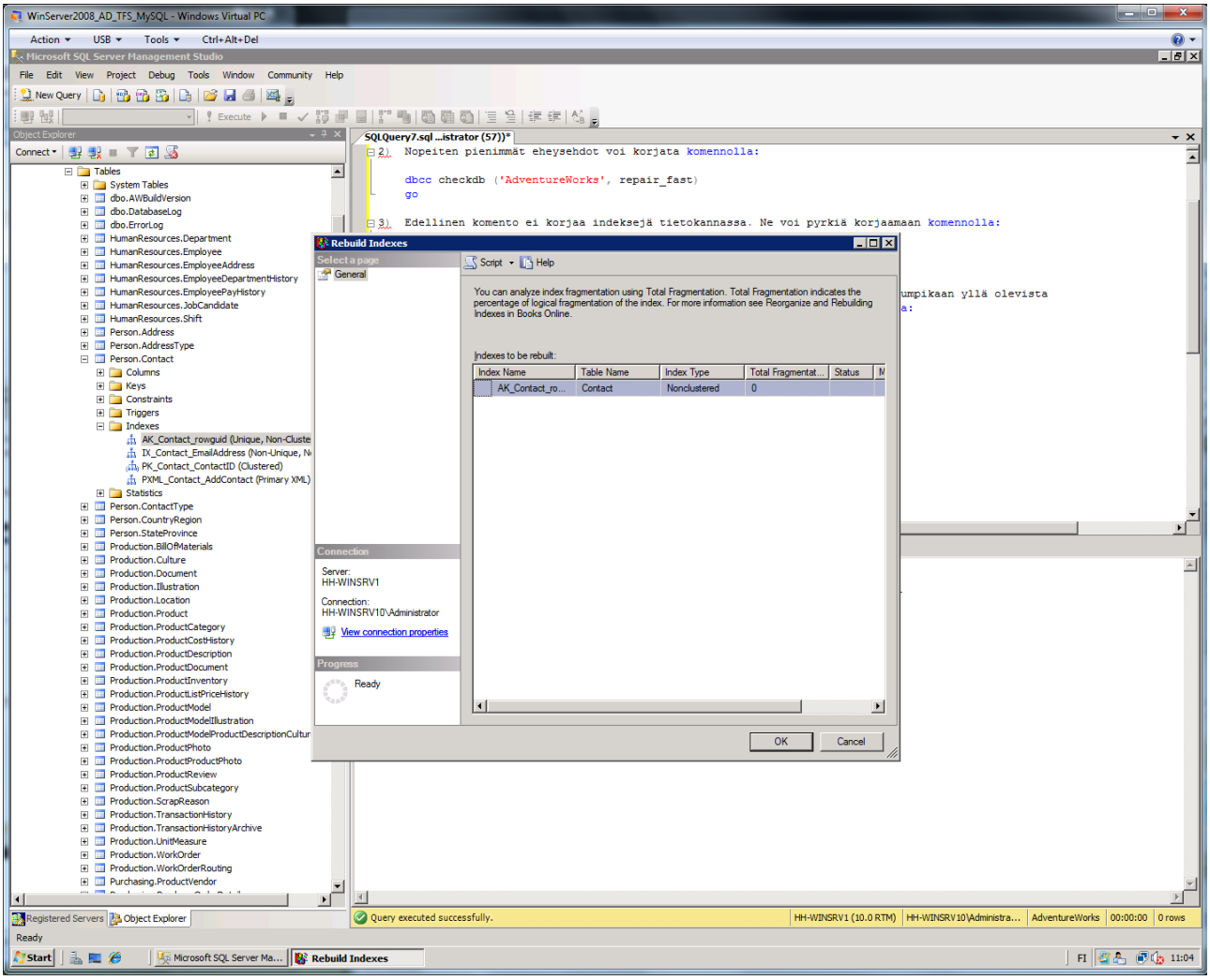
```
use AdventureWorks;
dbcc checktable('Person.Contact');
```



Jos tulee ilmoitus, että indeksi pitää luoda uudestaan, jotta sen saa takaisin online tilaan, niin luo kyseiset indeksi uudelleen, esimerkiksi valitsemalla Management Studiossa alla olevan kuvan mukaisesti:

Esimerkiksi `Person.Contact` taulun indeksin kanssa
Klikkaa uudelleen muodostettavaan indeksia alla olevan kuvan mukaisesti hiiren oikealla ja valitse Rebuild:





Ja luo indeksi uudelleen klikkaamalla OK.

Tämän jälkeen `dbcc checktable('Person.Contact');` lauseen saa ajettua ilman virheilmoituksia indekseistä.