**Tuntitehtävät vko 5**
Suoritusteho. Indeksit ja saantitavat. Statistiikan keruu. Optimoija ja suoritussuunnitelmat:

a) Miten saat selville käytetäänkö kyselyssä tietokannassa olevia indeksejä. Anna esimerkki tilanteesta, jossa indeksiä käytetään ja vastaavasti tilanteesta jolloin indeksiä ei käytetä.

Vastaus:
SQL Server Management Studiosta löytyy:
Query -> Include Actual Execution Plan
Tässä kohtaa voi suorittaa haluamansa kyselyn, jonka suoritussuunnitelmaa haluaa tarkastella, jonka jälkeen voi valita:
Query -> Display Estimated Execution Plan

Ajosuunnitelmassa on kerrottu onko indeksejä käytetty ja mikä niiden hyöty on ollut. Tästä on kerrottu tarkemmin osoitteessa: http://technet.microsoft.com/en-us/library/ms189562(v=sql.105).aspx.

b) Miten voit vaikuttaa SQL Server:ssä sen statistiikkaan tietokannan taulujen rivien lukumäärän osalta. Mihin statistiikkaa käytetään. Miten saat todellisen tilanteen palautettua takaisin SQL Server:iin?

Vastaus:

Kokeillaan ensin Projekti_db tietokantaan (jossa ei ole indeksejä kuin pääavaimissa).
SQL Server:in T-SQL kielestä löytyy update statistics lause, jonka avulla voi päivittää SQL Server:in statistiikan ajan tasalle tai muuttaa sen haluamakseen. Seuraavassa on ensin kokeiltu sen toimintaa Projekti_db tietokantaan ja sen jälkeen on jatkettu AdventureWorks tietokantaan. Projekti_db tietokantaa on tarkasteltu update statistics:lla ensin ilman indeksejä ulkoisissa viite-eheysavaimissa. Pääavaimissa toki on indeksit. Tyonkija ja osasto taulujen rivien ja sivujen lukumäärä on muutettu halutuiksi, jotta indekseistä on takuulla apua haun nopeuttamisessa. Lopuksi ne on palautettu todellisiksi päivittämällä kyseisten taulujen statistiikka ajan tasalle. Tämän jälkeen on tarkasteltu AdventureWorks tietokantaa (siitä löytyy indeksit valmiina moneen ulkoiseen viite-eheysavaimeen).

Aluksi tarkastellaan siis Projekti_db tietokantaa:

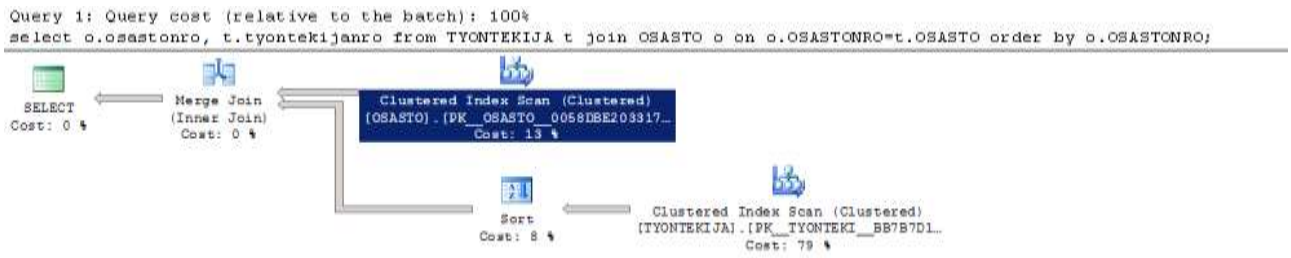
```
use Projekti_db;

select o.osastonro, t.tyontekijanro
from TYONTEKIJA t
join OSASTO o on o.OSASTONRO=t.OSASTO
order by o.OSASTONRO;

UPDATE STATISTICS osasto
        WITH ROWCOUNT = 10000, PAGECOUNT = 10000
UPDATE STATISTICS tyontekija
        WITH ROWCOUNT = 60000, PAGECOUNT = 60000
go

select o.osastonro, t.tyontekijanro
from TYONTEKIJA t
join OSASTO o on o.OSASTONRO=t.OSASTO
order by o.OSASTONRO
option (recompile);
```

Valitse sen jälkeen:
Query -> Display Estimated Execution Plan


Query 1: Query cost (relative to the batch): 100%
select o.osastonro, t.tyontekijanro from TYONTEKIJA t join OSASTO o on o.OSASTONRO=t.OSASTO order by o.OSASTONRO;
SELECT
Cost: 0 %
Merge Join
(Inner Join)
Cost: 0 %
Clustered Index Scan (Clustered)
[OSASTO].[PK__OSASTO__0058DBE203317...
Cost: 13 %
Sort
Cost: 8 %
Clustered Index Scan (Clustered)
[TYONTEKIJA].[PK__TYONTEKI__BB7B7D1...
Cost: 79 %

Luomalla indeksi esimerkiksi tyontekija taulun osasto kenttään, joka on on ulkoiseen viite-eheysavaimeen, parannettaisiin yllä olevan haun nopeutta. Taulujen pääavaimista on jo olemassa indeksit:

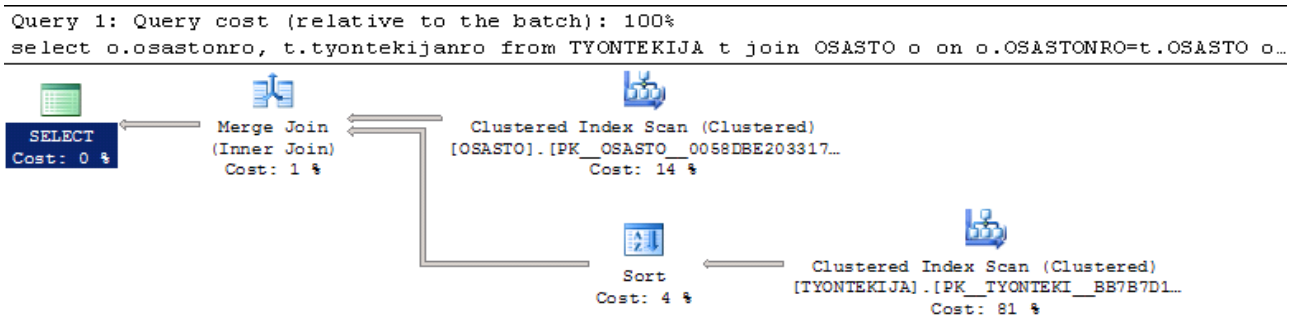
```
create index ix_id on tyontekija(osasto);
```

Taulussa pitää olla riittävästi data rivejä, jotta optimoija tietokantapalvelimella päättää käyttää tällaista indeksiä. Lisäksi indeksiä ei kannata luoda, jos sitä tarvitsevaa hakua käytetään harvoin. Indeksejä ei siis kannata ryhtyä luomaan ilman, että tietää sen todellisen käyttöasteen. Seuraavaksi muutamme statistiikan tiedoja projekti_db tietokannassa:

```
UPDATE STATISTICS osasto
        WITH ROWCOUNT = 100, PAGECOUNT = 100
UPDATE STATISTICS tyontekija
        WITH ROWCOUNT = 600, PAGECOUNT = 600
go

select o.osastonro, t.tyontekijanro
from TYONTEKIJA t
join OSASTO o on o.OSASTONRO=t.OSASTO
order by o.OSASTONRO
option (recompile);
```

Valitse sen jälkeen:
Query -> Display Estimated Execution Plan




Selvitetään ensin kuinka paljon data rivejä todellisuudessa on `Purchasing.PurchaseOrderDetail` ja `Sales.Customer` tauluissa:

```
use AdventureWorks
go
select COUNT(*) from Purchasing.PurchaseOrderDetail;
select COUNT(*) from Sales.Customer;
```

Muutetaan keinotekoisesti statistiikkaa simuloimaan suurempia rivimäärää (ks. SQL Server 2008 R2: http://msdn.microsoft.com/en-us/library/ms187348(v=sql.105).aspx):

```
UPDATE STATISTICS Purchasing.PurchaseOrderDetail
        WITH ROWCOUNT = 60000, PAGECOUNT = 60000
UPDATE STATISTICS Purchasing.PurchaseOrderHeader
        WITH ROWCOUNT = 10000, PAGECOUNT = 10000
go
```

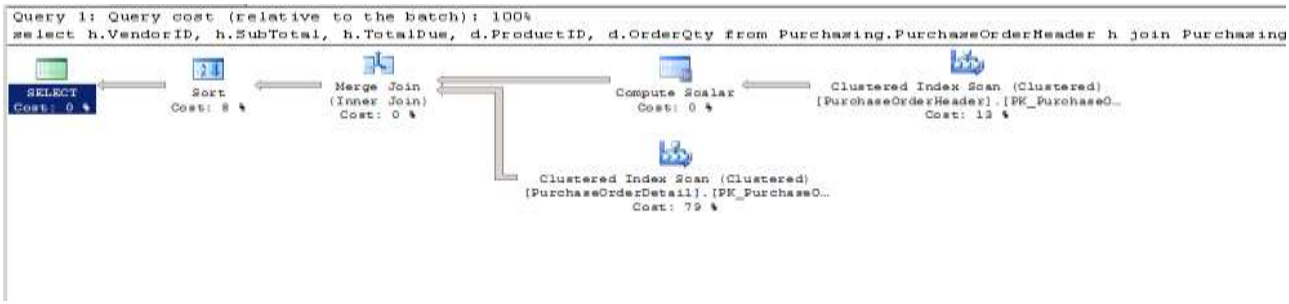
Jos volyymit olisivat tällaiset, niin vaikuttaako se suoritussuunnitelmiin?

Tutki asiaa SQL Server Management Studion Display Estimated Execution Plan:

Suorita ensin lause:

```
select h.VendorID, h.SubTotal, h.TotalDue, d.ProductID, d.OrderQty
from Purchasing.PurchaseOrderHeader h
join Purchasing.PurchaseOrderDetail d on h.PurchaseOrderID =
d.PurchaseOrderID
order by h.VendorID
option (recompile);
```

Valitse sen jälkeen:
Query -> Display Estimated Execution Plan




Lisätietoja tästä löytyy mm. osoitteesta:
http://blogs.msdn.com/b/queryoptteam/archive/2006/07/21/674350.aspx.

Statistiikkaa käytetään sql lauseiden ajosuunnitelman tekoon. Tällöin statiikka olisi hyvä olla kohtalaisen paikkaansa pitävä. Yleensä statistiikkaa ei päivitetä automaattisesti sillä siitä tule on ajo kustannuksensa palvelimen kuormituksena. Statistiikassa on tieto taulujen rivien lukumäärästä ja indekseistä joita voidaan käyttää. Jos data rivejä on vähän taulussa, niin query optimizer voi päättää olla käyttämättä indeksiä. Tällöin tauluun tehdään full table scan, eli etsitään suoraan taulusta. Jos query optimizer toteaa, että taulussa on niin paljon datarivejä, että tarjolla olevan indeksin käyttö kannattaa, tehdään ajo suunnitelma, jossa indeksin kautta etsitään tarvittavat data rivit taulusta tai tauluista. Ajo suunnitelmasta on kerrottu lisää osoitteessa: http://msdn.microsoft.com/en-us/library/ms191194(v=sql.105).aspx.

c) Miten indeksien ja taulujen ehetyttä pidetään yllä SQL Server:ssä (mitkä ovat komennot ja miten niitä käytetään ja miten työrutiinit on syytä käytännössä järjestää).

Vastaus:

SQL Server pitää automaattisesti yllä indeksejä ainakun insert, update, tai delete operaatioita suoritetaan indeksiin liittyviin dataan. Ajan myötä nämä muutokset indeksissä johtavat indeksoidun tiedon hajautumiseen indeksin ja tietokannan välillä. Indeksiin indeksoitu tieto on eri järjestyksessä kuin mitä se on tietokannassa. Tätä kutsutaan fragmentoitumiseksi. Pahoin fragmentoituneet indeksit voivat hidastaa ja siten vaikuttaa sovellusten toimintaa heikentävästi.

Fragmentoitumisen voi poistaa joko uudelleen organisoimalla indeksit tai luomalla ne kokonaan uudelleen. Dba:n kannattaa siis seurata säännöllisesti indeksien fragmentoitumisastetta. Tähän tarkoitukseen on SQL Server:ssä olemassa sys.dm_db_index_physical_stats systeemi funktio. Kyseisen funktion käyttöä on esitelty tarkemmin osoitteissa: SQL Server 2008 R2: http://technet.microsoft.com/en-us/library/ms189858(v=sql.105).aspx ja SQL Server 2012: http://technet.microsoft.com/en-us/library/ms189858.aspx. Sys.dm_db_index_physical_stats funktio kertoo fragmentoitumisasteen avg_fragmentation_in_procent kentässä, jonka tiedon perusteella valitaan millä tavalla edetään. Jos fragmentoitumisaste on välillä 5% … 30 %, on käytettävä `alter index reorganize` lausetta tilanteen korjaamiseksi indekseissä. Jos fragmentoitumisaste on enemmän kuin 30 %, on käytettävä `alter index rebuild with (online = on)` lausetta.

Indeksien uudelleen organisointi tai peräti uudelleen luonti on raskas operaatio, jonka suorittaminen kannattaa suunnitella huolella. Se kannattaa yleensä suorittaa käytön kannalta hiljaiseen aikaan.

Alla on esimerkki miten sys.dm_db_index_physical_stats funktiota voi käyttää AdventureWorks tietokannan `Purchasing.PurchaseOrderHeader` taulun indeksien fragementoitumisasteen tutkimiseen:

```
USE AdventureWorks;
GO
SELECT a.index_id, name, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats (DB_ID(),
OBJECT_ID(N'Purchasing.PurchaseOrderHeader'),
     NULL, NULL, NULL) AS a
    JOIN sys.indexes AS b ON a.object_id = b.object_id AND a.index_id =
b.index_id;
GO
```

Tulos on:

```
index_id    name                                    avg_fragmentation_in_percent
----------- ----------------------------------------------------------------
1           PK_PurchaseOrderHeader_PurchaseOrderID     0
2           IX_PurchaseOrderHeader_VendorID           33,3333333333333
3           IX_PurchaseOrderHeader_EmployeeID         16,6666666666667

(3 row(s) affected)
```

Indeksit siis näyttävät olevat melko fragmentoituneita, ja asialle pitää tehdä jotain. Tehtävät toimenpiteet riippuvat fragmentoitumisasteesta. Ensimmäiselle IX_PurchaseOrderHeader_VendorID indeksille on suositeltavaa Microsoftin mukaan käyttää alter index reorganize lausetta. Seuraaville indekseille on puolestaan käytettävä alter index rebuild lausetta.

```
USE AdventureWorks;
GO
ALTER INDEX IX_PurchaseOrderHeader_VendorID ON Purchasing.PurchaseOrderHeader
REBUILD;
GO
```

Tosin jos indeksissä on vähän sivuja voi fragmentoitumisaste kasvaa. Yllä olevasta on siis käytännön höytyä vasta kun indeksi on riittävän suuri.

```
USE AdventureWorks;
GO
UPDATE STATISTICS Purchasing.PurchaseOrderHeader
IX_PurchaseOrderHeader_VendorID;
GO
```

d) Mitä merkitystä indekseille on fill factor arvolla indeksiä luotaessa SQL Server:ssä? Millaista fill factoria kannattaisi esimerkiksi käyttää IDENTITY avaimelle (esim. create table henkilo (henkiloid int not null identity(1,1))?

Vastaus:

SQL Server:ssä käytetään fill factor:ia indeksien käytön tehostamisessa. Fill factor kertoo miten täyteen indeksin lehtitason sivut täytetään. Tyhjäksi jätettävä osa lehtitason sivuista käytetään tulevaisuudessa indeksin kasvattamiseen. Fill factor:lle on omat ohjeensa SQL Server 2008 R2:lle osoitteessa: http://technet.microsoft.com/en-us/library/ms177459(v=sql.105).aspx ja SQL Server 2012:lle osoitteessa: http://technet.microsoft.com/en-us/library/ms177459.aspx. Ne poikkeavat toisistaan. Alla olevassa esimerkissä on muutettu AdventureWorks tietokannan indeksien fill factor arvo 100, eli indeksien lehtitason sivut täytetään aina täyteen asti. Tästä tietysti syntyy myöhemmin ongelmia tehon kannalta kun indeksi täytyy ja sitä pitää kasvattaa. Esimerkiksi fill factor:in arvo 80 olisi jo parempi arvo. Tilanne on täysin toinen jos indeksin avain muodostetaan IDENTITY arvona. Tällöin indeksin avaimen arvo on aina kasvava arvo, joka lisätään indeksin loppuun, ei siis olemassa olevan indeksin tyhjiin lehtitason sivuihin. Tässä tapauksessa on parempi käyttää tällaisen indeksin fill factor:ina arvoa 100 (ks. http://technet.microsoft.com/en-us/library/ms177459.aspx). Alla oleva esimerkki on SQL Server R2:lle:

```
Use AdventureWorks;
GO
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'fill factor', 100;
GO
RECONFIGURE;
GO
```

```
Configuration option 'show advanced options' changed from 0 to 1. Run the RECONFIGURE statement
to install.
Configuration option 'fill factor (%)' changed from 0 to 100. Run the RECONFIGURE statement to
install.
```

Nyt tietokannan kaikkien indeksien fill factor:ien arvot ovat 100. Tämä tosin on hyvä asia jos kyseessä on siis pääavaimen indeksi, jonka arvo muodostetaan IDENTITY:nä. Muiden indeksien kohdalla on tehtävä muutuksia, kuten alla on tehty parin indeksin osalta (kaikke ei IDENTITY indeksit pitäisi hoitaa kuntoon vastaavalla tavalla).

Indeksin fill factor arvon voi muuttaa haluamakseen alter index lauseella:

```
USE AdventureWorks;
GO
ALTER INDEX IX_PurchaseOrderHeader_EmployeeID ON
Purchasing.PurchaseOrderHeader
REBUILD WITH (FILLFACTOR = 80, SORT_IN_TEMPDB = ON,
              STATISTICS_NORECOMPUTE = ON);
GO

USE AdventureWorks;
GO
ALTER INDEX IX_PurchaseOrderHeader_VendorID ON Purchasing.PurchaseOrderHeader
REBUILD WITH (FILLFACTOR = 80, SORT_IN_TEMPDB = ON,
              STATISTICS_NORECOMPUTE = ON);
GO
```

e) Tutkitaan nyt SQL Server Management Studiolla alla olevia kyselyjä (ks. t-sql:n inner join http://msdn.microsoft.com/en-us/library/ms190014(v=sql.105).aspx):

```
-- eka
USE AdventureWorks

SELECT C.CustomerID, S.Name, O.SalesOrderID, O.OrderDate
FROM Sales.SalesOrderHeader O
INNER JOIN Sales.Customer C ON O.CustomerID = C.CustomerID
INNER JOIN Sales.Store S ON S.CustomerID = C.CustomerID
WHERE C.CustomerID = 117
UNION
SELECT C.CustomerID, S.Name, O.SalesOrderID, O.OrderDate
FROM Sales.SalesOrderHeader O
INNER JOIN Sales.Customer C ON O.CustomerID = C.CustomerID
INNER JOIN Sales.Store S ON S.CustomerID = C.CustomerID
WHERE C.CustomerID = 98

-- toka
SELECT C.CustomerID, S.Name, O.SalesOrderID, O.OrderDate
FROM Sales.SalesOrderHeader O
INNER JOIN Sales.Customer C ON O.CustomerID = C.CustomerID
INNER JOIN Sales.Store S ON S.CustomerID = C.CustomerID
WHERE C.CustomerID IN (98, 117)
ORDER BY C.CustomerID;
GO
```

Valitse ensin Query – Display Estimated Execution Plan (ks. http://msdn.microsoft.com/en-us/library/ms191194.aspx). Käytettyjen graafisten ikonien merkitys on kerrottu osoitteessa: http://msdn.microsoft.com/en-us/library/ms175913(v=sql.105).aspx. Ohjeet tulosten tulkitsemiseen on kerrottu osoitteessa: http://msdn.microsoft.com/en-us/library/ms178071(v=sql.105).aspx.
ja talleta “Execution Plan” näytöt kyselyistä.
Päivitä näiden taulujen statistiikka komennoilla (ks. http://msdn.microsoft.com/en-us/library/dd535534%28v=SQL.100%29.aspx)

```
UPDATE STATISTICS Sales.SalesOrderHeader
UPDATE STATISTICS Sales.Customer
UPDATE STATISTICS Sales.Store
```

Valitse sitten Query –Include Actual Execution Plan – Execute
ja talleta “Execution Plan” näytöt kyselyistä.
Onko Estimated ja Actual suoritussuunnitelmissa eroja?.
Kumpi kyselyistä eka ja toka on tehokkaampi ?
Mikä matalan tason operaatio toteuttaa UNION-operaation?
Mitkä välivaiheet ovat raskaimpia ja mitkä keveimpiä?
Pitävätkö estimoidut rivimäärät paikkansa?

Vastaus:
Toka on tehokkaampi.

UNION operaatiosta on kerrottu osoitteessa: http://msdn.microsoft.com/en-us/library/ff848745.aspx ja http://msdn.microsoft.com/en-us/library/ms180026.aspx. Se käyttää concatenation operaatiota.

Raskaimpia ovat aina lajittelut kun ei ole indeksiä. Esimerkiksi liitoksessa, jossa liitoksen toiselle avaimelle ei ole olemassa indeksiä. Näin voi helposti olla kun liitoksessa mukana olevassa pääavaimessa on indeksi, mutta toisella foreign key:lle ei ole luotu indeksiä.

Kevein vaihe on aina kun rivejä on taulussa vähän tai voidaan käyttää olemassa olevaa indeksiä esimerkiksi liitoksen yhteydessä tai lajittelussa (order by).

Raskain on AddressID pääavaimen etsiminen Address taulusta. Myös CustomerID pääavaimen etsiminen Customer taulusta on raskas.

Keveintä on inner join liitoksen teko (0%). Myös union concatenation:ia käyttäen on kevyt (0%).

f) JOIN-muotojen uskotaan yleensä olevan nested-kyselyjä tehokkaampia.
Tutki asiaa seuraavalla kyselyparilla

```
USE AdventureWorks

SELECT CustomerID, AccountNumber, CustomerType
FROM Sales.Customer
WHERE CustomerID IN
    (SELECT CustomerID
     FROM Sales.SalesOrderHeader
     WHERE SalesPersonID = 275)
     ORDER BY CustomerID;


SELECT DISTINCT C.CustomerID, C.AccountNumber, C.CustomerType
FROM Sales.Customer C
JOIN Sales.SalesOrderHeader H ON C.CustomerID = H.CustomerID
WHERE SalesPersonID = 275
ORDER BY C.CustomerID;

GO
```

Mistähän tämä johtuu?
Miltä liitos-operaatiot vaikuttavat?
Onko kyselyjä mahdollista virittää esim uusilla indekseillä tai vihjeillä

Vastaus:
Yllä olevien kyselyiden kustannus on sama.
Indekseillä on vaikutusta ajo kustannukseen, jos kyselyn tauluissa on riittävän paljon datarivejä. Vain tällaisessa tapauksessa query optimizer päättää käyttää tarjolla olevia indeksejä. AdventureWorks tietokannassa on luotu indeksejä foreign key kentille ja myös muille kentille. Yllä olevien kyselyiden kentille, joita käytetään liitosehdoissa, löytyy indeksit.