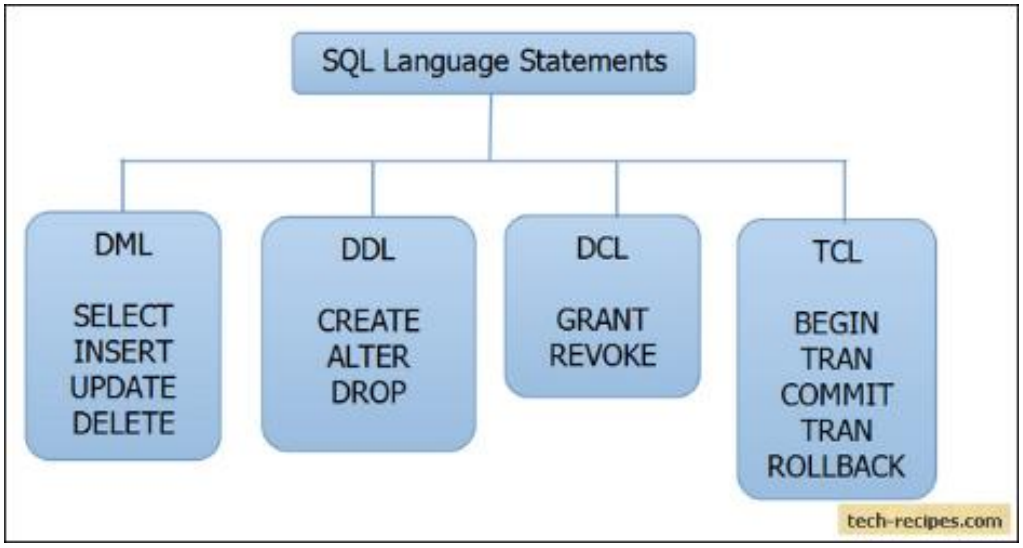
Napredni SQL

SQL stavke lahko razdelimo v 4 skupine

1. DML (Data Manipulation Language)
2. DDL (Data Definition Language)
3. DCL (Data Control Language)
4. TCL (Transaction Control Language)



1. Podatkovni tipi

Podatkovni tipi v SQL so našteti v tabeli:

Natančni numerični	Približni numerični	Datum/čas	Binarni	Znakovni	Ostali
tinyint	float	date	binary	char	cursor
smallint	real	time	varbinary	varchar	hierarchyid
int		datetime	image	text	sql_variant
bigint		datetime2		nchar	table
bit		smalldatetime		nvarchar	timestamp
decimal/numeric		datetimeoffset		ntext	uniqueidentifier
numeric					xml
money					georgaphy
smallmoney					geometry

Med posameznimi tipi lahko v T-SQL jeziku pretvarjamo s pomočjo

CAST/TRY_CAST → pretvarjanje med poljubnimi tipi (ANSI standard), try→ če ne uspe je rezultat NULL (velja za vse)

CONVERT/TRY_CONVERT → med katerimikoli tipi (specifičen za SQL strežnik)

PARSE/TRY_PARSE → iz niza v drug podatkovni tip

STR → iz vrednosti v niz

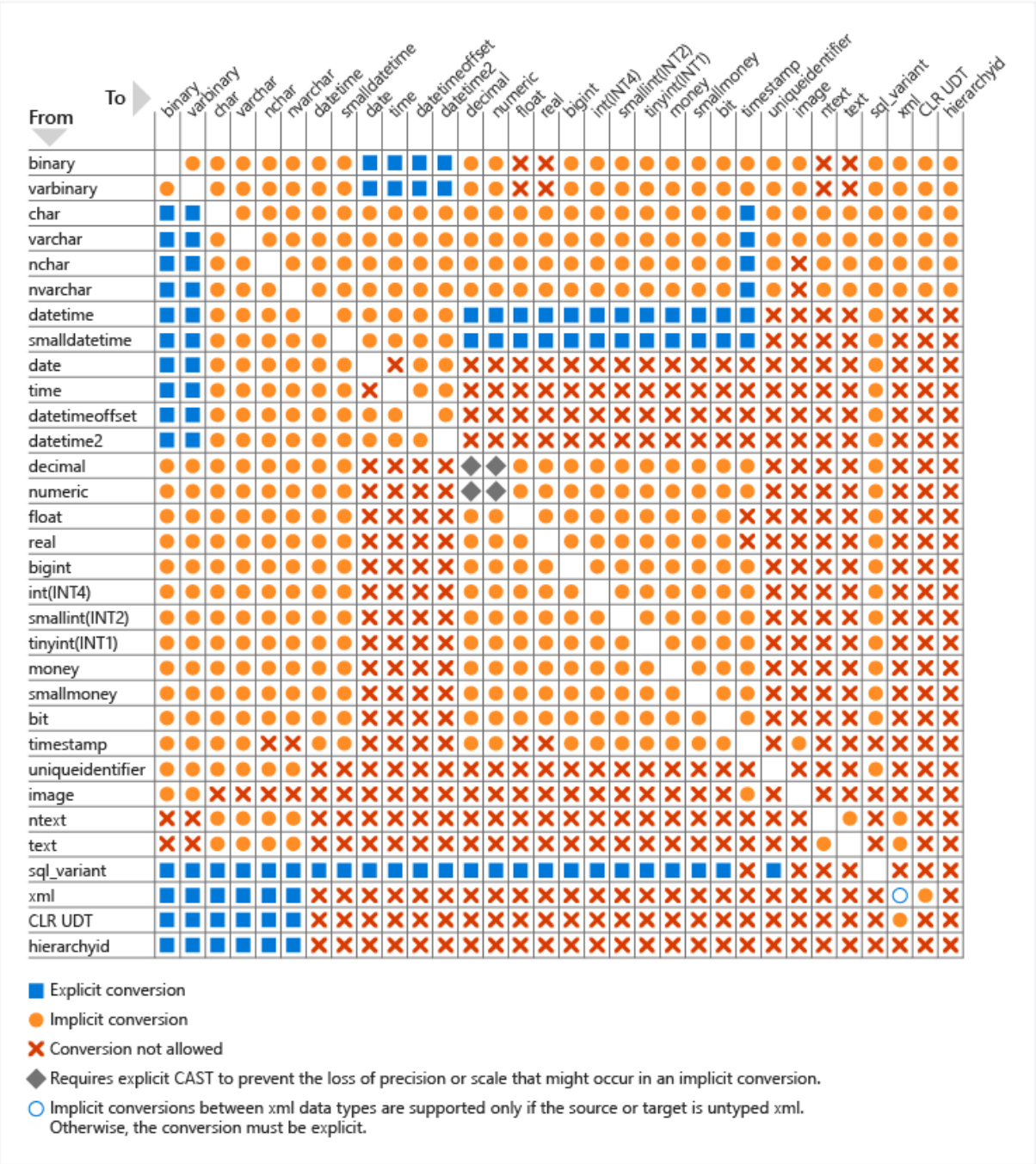
Za vrednost NULL velja ANSI standard, kar pomeni 2+NULL=NULL, NULL=NULL? False, NULL is NULL? True

Za delo z vrednostjo NULL uporabljamo

ISNULL(stolpec,vr1) → če je v stolpcu NULL, vrednost nadomesti z vr1
NULLIF(stolpec, vr2)→če je v stolpcu vrednost vr2 jo nadomesti z NULL
COALESCE(st1,st2,…)→vrni vrednost v prvem stolpcu, kjer je le ta različna od NULL

Primeri:

```
--1. uporaba cast in conevrt
select cast(AddressID as varchar(5))+' :'+AddressLine1 as Naslov from SalesLT.Address
--2.
select convert(varchar(5),AddressID)+' : '+AddressLine1 as Naslov from SalesLT.Address
--3. različni datumi
select Name, convert(nvarchar(30),SellStartDate) as DatumProdaje,
       convert(nvarchar(30),SellEndDate,126) as ISOFormat,
from SalesLT.Product
--4. uporaba coalesce
SELECT Name,  Color, Size,
COALESCE( Color, Size) AS FirstNotNull
FROM SalesLT.Product; (če je Color različen od null izpiše vrednost Color, sicer pa
Size, če sta oba NULL izpiše NULL v stolpcu FirstNotNull)
--5. select case
select Name,
case when SellEndDate is null then 'Na razpolago'
     else 'Razprodano'
        end
        as SalesStaus
 from SalesLT.Product
--6.
select Name,
case Size
     when 'S'then 'Small'
        when 'M' then 'Medium'
     else ISNULL(Size,'n/a')
end
 from SalesLT.Product
```



Reši Vajo1

2. Različne poizvedbe (WHERE, ORDER BY)

Pri izvajanju SQL stavka SELECT moramo poznati vrstni red izvajanja, da lahko pravilno zapišemo poizvedbo. V poljubnem select stavku se najprej izvede FROM, nato WHERE, sledi GROUP BY, HAVING, SELECT in na koncu ORDER BY.

Select vedno pomeni izberi vse. Če želimo samo različne uporabimo besedico **DISTINCT**. Primer:

```
-- barva
select Color from SalesLT.Product
--različne barve
select distinct Color from SalesLT.Product
```

V poizvedbi lahko rezultat uredimo s pomočjo **ORDER BY**, lahko pa izberemo samo prvih nekaj rezultatov (na primer **TOP 10**) ali prvih nekaj % rezultatov (**TOP 10 PROCENT**).
Če nas zanima drugih 10 rezultatov ali želimo samo del podatkov iz rezultata, lahko uporabimo ostranjevanje.

```
--ostranjevanje
select Name,ListPrice from SalesLT.Product
order by ProductNumber offset 0 rows
fetch next 10 rows only
```

Rezultat

Name	ListPrice
LL Bottom Bracket	53.99
ML Bottom Bracket	101.24
HL Bottom Bracket	121.49
Mountain Bottle Cage	9.99
Road Bottle Cage	8.99
Mountain-500 Black, 40	539.99
Mountain-500 Black, 42	539.99
Mountain-500 Black, 44	539.99
Mountain-500 Black, 48	539.99
Mountain-500 Black, 52	539.99

```
select Name,ListPrice from SalesLT.Product
order by ProductNumber offset 50 rows
fetch next 10 rows only
```

Rezultat

Name	ListPrice
Road-650 Red, 58	782.99
Road-650 Red, 60	782.99
Road-650 Red, 62	782.99
Road-550-W Yellow, 38	1120.49
Road-550-W Yellow, 40	1120.49
Road-550-W Yellow, 42	1120.49
Road-550-W Yellow, 44	1120.49
Road-550-W Yellow, 48	1120.49
Road-450 Red, 44	1457.99
Road-450 Red, 48	1457.99

Operatorji primerjave v SQL-u so : =,<>,<, <=,>,>=, in, between, like
Logični operatorji: and, or, not
Posebej si poglejmo delovanje operatorja LIKE. S pomočjo like iščemo posamezne vzorce v tekstu, v smislu regularnih izrazov.
% nadomešča katerikoli znak ali zaporedje znakov: primer LIKE 'FR%' poišči vse, ki se začnejo na FR.
_ nadomešča natanko en znak
[a-z] – katerakoli mala črka
[0-9] – katerakoli številka
Primeri:

```
--izberi vse produkt, ki se začnejo na FR
select productNumber from SalesLT.Product
where ProductNumber like 'FR%'
--izberi vse produkte, ki se začnejo na FR, nato imajo znak -,
--nato katerikoli znak, nato dve številki pa spet katerikoli znak,
--sledi ponovno - in dve številki
select productNumber from SalesLT.Product
where ProductNumber like 'FR-_[0-9][0-9]_-[0-9][0-9]'
--izberi vse produkte, ki se začnejo na BK-, naslednji znak ne sme biti R
--sledi lahko več poljubnih znakov, končajo pa se na - in dve številki
```

```
select productNumber from SalesLT.Product
where ProductNumber like 'BK-[^R]%-[0-9][0-9]'
--izberi produkte, katerih kategorija je 5, 6 ali 7
select productcategoryID from SalesLT.Product
where productcategoryID in (5,6,7)
```

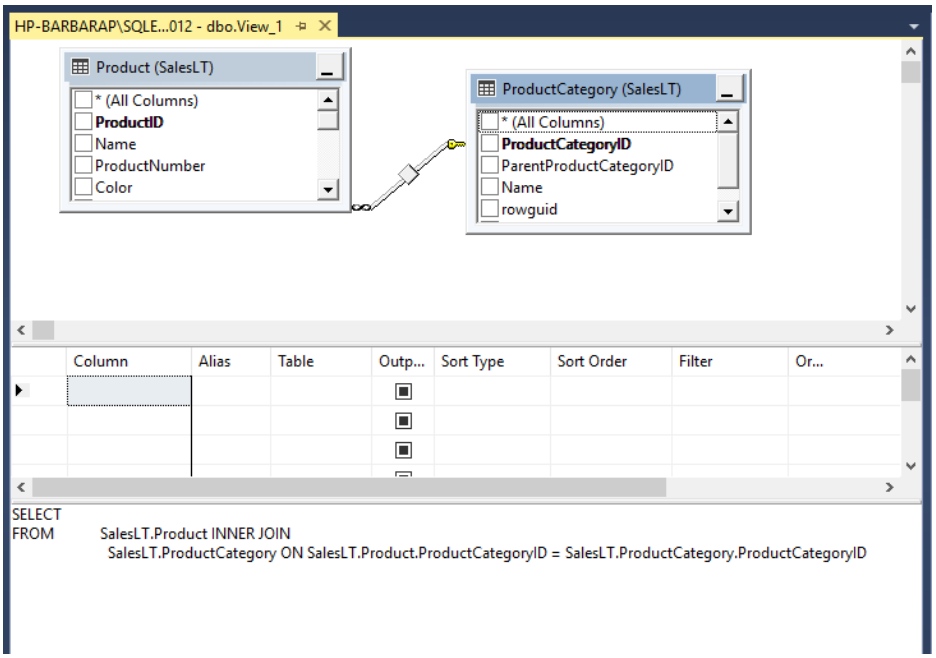
Reši vajo 2!

3. Stiki

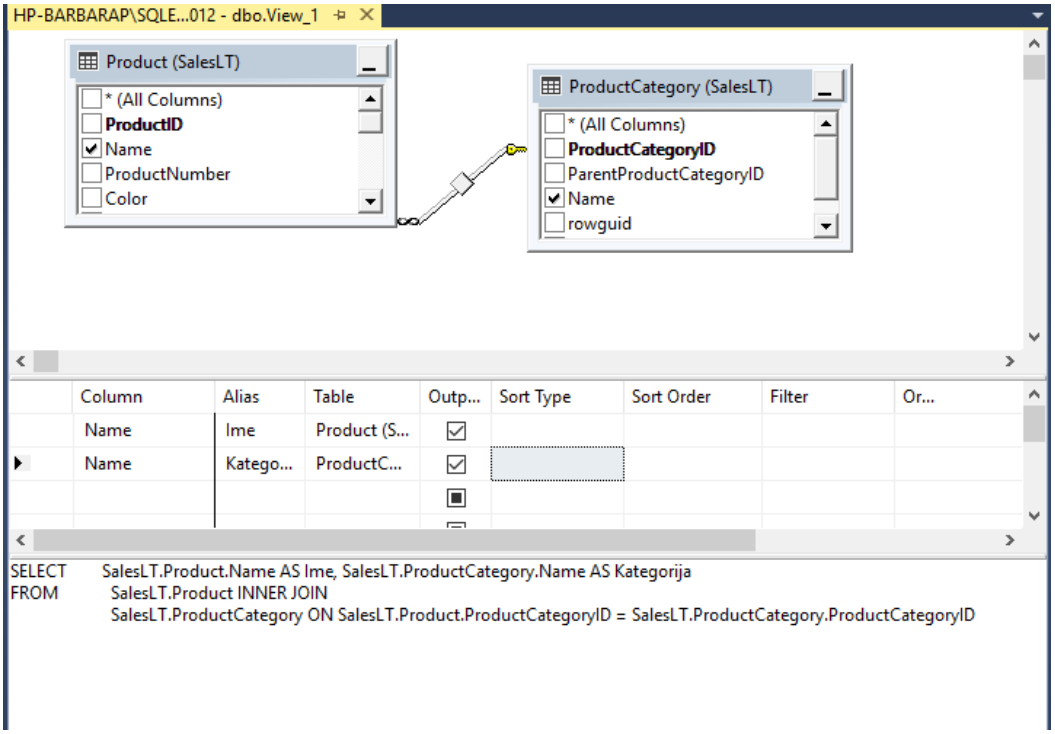
Imamo notranje (inner), leve, desne in polne zunanje (left outer, right outer, full outer) in kartezične stike (cross-join). Poleg tega lahko imamo tudi stike tabele same s seboj (self join)

```
--iste poizvedbe ANSI 92
select p.name as Ime, c.name as Kategorja from SalesLT.Product p
inner join SalesLT.ProductCategory c
on p.ProductCategoryID=c.ProductCategoryID
select p.name as Ime, c.name as Kategorja from SalesLT.Product p
join SalesLT.ProductCategory c
on p.ProductCategoryID=c.ProductCategoryID
--ANSI 89
select p.name as Ime, c.name as Kategorja from SalesLT.Product p,
SalesLT.ProductCategory c
where p.ProductCategoryID=c.ProductCategoryID
--zunanji stiki
select c.FirstName,c.LastName,oh.SalesOrderNumber from SalesLT.Customer as c
left outer join SalesLT.SalesOrderHeader as oh
on c.CustomerID=oh.CustomerID
order by c.CustomerID
select c.FirstName,c.LastName,oh.SalesOrderNumber from SalesLT.Customer as c
right outer join SalesLT.SalesOrderHeader as oh
on c.CustomerID=oh.CustomerID
order by c.CustomerID
select c.FirstName,c.LastName,oh.SalesOrderNumber from SalesLT.Customer as c
full outer join SalesLT.SalesOrderHeader as oh
on c.CustomerID=oh.CustomerID
order by c.CustomerID
--kartezični stik ni enko full outer!!
select c.FirstName,c.LastName,oh.SalesOrderNumber from SalesLT.Customer as c
cross join SalesLT.SalesOrderHeader as oh
order by c.CustomerID
```

Reši vajo 3! Namig: Pri stikih (tudi pri drugih poizvedbah) si lahko pomagaš z vizualnim načrtovanjem: Vzemimo prvo poizvedbo, kjer izpisujemo imena artiklov in imena kategorij. Izberemo mapo View, nato pa iz priročnega menija izberemo New View… Tu izberemo tabele, ki bi jih radi imeli.



Zgoraj imamo način načrtovanja, spodaj se izpisuje SQL. Dodamo še atribute, ki jih v izpisu želimo in dobimo izdelan SQL



4. Set operatorji: union, intersect, except

Union uporabljamo na dveh tabelah z istimi atributi. Izdela nam unijo obeh tabel, ponavljajoči se elementi so v rezultatu samo enkrat. Če želimo v rezultatu vse vrstice, uporabimo **union all**.

Primer:

```
select countryregion,city from SalesLt.Address
union
select CountryRegion,city from SalesLT.Employees
```

Število stolpcev se mora ujemati. Enako velja za presek, vrača skupne elemente obeh tabel:

```
select countryregion,city from SalesLt.Address
intersect
select CountryRegion,city from SalesLT.Employees
```

Razlika (except) vrača razliko, to je elemente, ki so v prvi tabeli in niso hkrati v drugi.

```
select countryregion,city from SalesLt.Address
except
select CountryRegion,city from SalesLT.Employees
```

Reši Vajo4!

5. Funkcije v SQL podatkovni bazi

a. Skalarne

Vrnejo eno vrednost in delujejo na eni vrednosti. Imamo deterministične (vračajo vedno isto vrednost) in ne-deterministične, ki ne vračajo vedno istega rezultata (na primer getdate() vrne vedno trenutni datum/čas). Najbolj pogosto uporabljamo funkcije za delo z datumi in za upravljanje z nizi. Celoten spisek si lahko ogledate na https://docs.microsoft.com/en-us/sql/t-sql/functions/functions

Naštejmo jih nekaj:

year(datum)
datename(mm,datum) – ime meseca
datename(dw,datum) - ime dneva v tednu
day(datum)
datediff(yy,datum1,datum2)
upper(niz)
concat(niz1,niz2)
left(niz,2)
substring(niz,začetek, dolžina)
len(niz)
reverse(niz)
right(niz,število)
Primer:

```
SELECT ProductID,
            UPPER(Name) AS ProductName, -- v velike črke
            ROUND(Weight, 0) AS ApproxWeight, --zaokroži na 0 decimalk--> na cele
            YEAR(SellStartDate) as SellStartYear,--vzemi samo leto začetka
            DATENAME(m, SellStartDate) as SellStartMonth, --ime meseca začetka
            LEFT(ProductNumber, 2) AS ProductType --leva 2 znaka številke produkta
FROM SalesLT.Product;
```

ProductID	ProductName	ApproxWeight	SellStartYear	SellStartMonth	ProductType
680	HL ROAD FRAME - BLACK, 58	1016.00	1998	June	FR
706	HL ROAD FRAME - RED, 58	1016.00	1998	June	FR

b. Logične funkcije

Tako kot skalarne delujejo na eni vrednosti(vrstici) vračajo eno vrednost.

Naštejmo jih nekaj:

isnumeric('101.99')
iif(listprice>50,'high','low')

choose(ProductCategoryID,'Bikes','Components','Other')= če je ProductCategory=1 postane 'Bikes', če je 2 posatne vrednost 'Components', če je 3 postane vrednost 'Other', sicer je vrednost NULL
Primer:

```
SELECT ProductID,
              UPPER(Name) AS ProductName, -- v velike črke
              IIF(ProductCategoryID in (5,6,7),'Kolesa','Ostalo')  as Kategorija
              --če je katgorija 5,6,7 gre za kolo, sicer gre za ostalo opremo
FROM SalesLT.Product;

SELECT ProductID,
UPPER(Name) AS ProductName,
YEAR(SellStartDate) as SellStartYear,
iif((sellEndDate is not null),datediff(mm,sellstartdate,sellenddate),0) as
MesecevVProdaji
FROM SalesLT.Product;
```

c. Okenske funkcije

Delujejo na več vrsticah. Denimo, da bi želeli razvrstiti produkte po ceni in jim dodati številko, kjer se nahajajo:

```
--rangiraj produkte od najdražjega navzdol
select ProductID,Name, ListPrice,
rank() over (order by Listprice desc) as RangPoCeni
from SalesLT.Product
```

vrne rezultat:

ProductID	Name	ListPrice	RangPoCeni
749	Road-150 Red, 62	3578.27	1
750	Road-150 Red, 44	3578.27	1
751	Road-150 Red, 48	3578.27	1
752	Road-150 Red, 52	3578.27	1
753	Road-150 Red, 56	3578.27	1
771	Mountain-100 Silver, 38	3399.99	6
772	Mountain-100 Silver, 42	3399.99	6

……

Rangiramo lahko tudi glede na nek stolpec:

```
select  ProductID,Name, ListPrice,
rank() over (partition by productCategoryID order by Listprice desc) as RangPoCeni
from SalesLT.Product;
```

Poizvedba bo rangirala cene v okviru ene kategorije (ProductCategoryID). Če uporabimo funkcijo **dense_rank**() namesto rank(), bo poizvedba delovala enako, le oštevilčevanje bo teklo 1,2,3,… in ne kot prej 1,6,…
Tako poizvedba:

```
select top 7 ProductID,Name, ListPrice,
dense_rank() over (order by Listprice desc) as RangPoCeni
from SalesLT.Product
```

Vrne rezultat:

ProductID	Name	ListPrice	RangPoCeni
749	Road-150 Red, 62	3578.27	1
750	Road-150 Red, 44	3578.27	1
751	Road-150 Red, 48	3578.27	1
752	Road-150 Red, 52	3578.27	1
753	Road-150 Red, 56	3578.27	1
771	Mountain-100 Silver, 38	3399.99	2
772	Mountain-100 Silver, 42	3399.99	2

ntile(n) rangira izdelke v n razredov. Na primer:

```
select ProductID,Name, ListPrice,
ntile(10) over (order by Listprice desc) as RangPoCeni
from SalesLT.Product;
```

Bo vse izdelke razvrstilo v razrede od 1 do 10, tako, da jih bo v vsakem enako število.
row_number() rangira enako kot rank(), le da oštevilči vsako vrstico z drugo zaporedno številko.

```
select ProductID,Name, ListPrice,
ROW_NUMBER() over (order by Listprice desc) as RangPoCeni
from SalesLT.Product;
```

d. Agregati

Agregat uporabi več vrstic, da izračuna rezultat. Agregatne funkcije so sum, count, min, max in avg.
Primer

```
SELECT count(*)as [Št artiklov],count(distinct ProductCategoryID)as [Št kategorij],
       avg(ListPrice) as [Povprečna cena]
FROM SalesLT.product AS SOD
```

Rezultat:

Št artiklov	Št kategorij	Povprečna cena
296	37	742.1235

Reši Vajo 5!

6. Pod-poizvedbe (subquery)

Pod poizvedba je ugnezdena v neko drugo poizvedbo ali v drugi sql stavek. V spodnjem primeru nam pod poizvedba vrača en stolpec poizvedbe

```
SELECT Ord.SalesOrderID, Ord.OrderDate,
    (SELECT MAX(OrdDet.UnitPrice)
     FROM SalesLT.SalesOrderDetail AS OrdDet
     WHERE Ord.SalesOrderID = OrdDet.SalesOrderID) AS MaxUnitPrice
FROM SalesLT.SalesOrderHeader AS Ord
```

Veliko poizvedb, ki vsebujejo stike lahko oblikujemo v obliki pod poizvedb, nekatere poizvedbe pa lahko realiziramo samo s pod poizvedbami. V spodnjem primeru vidimo poizvedbo, ki da isti rezultat, a je izvedena na dva različna načina (izpis vseh artiklov, ki imajo isto ceno kot 'Touring Tire Tube')

```
/* SELECT statement built using a subquery. */
SELECT Name
FROM SalesLT.Product
WHERE ListPrice =
    (SELECT ListPrice
     FROM SalesLT.Product
     WHERE Name = 'Touring Tire Tube' );
```

```
/* SELECT statement built using a join that returns
   the same result set. */
SELECT Prd1. Name
FROM SalesLT.Product AS Prd1
     JOIN SalesLT.Product AS Prd2
       ON (Prd1.ListPrice = Prd2.ListPrice)
WHERE Prd2. Name = 'Touring Tire Tube';
```

Običajno pod poizvedbe uporabljamo v naslednjih primerih

- WHERE izraz [NOT] IN **(**subquery**)**
- WHERE izraz operator primerjave [ANY | ALL] **(**subquery**)**
- WHERE [NOT] EXISTS **(**subquery**)**

Pod poizvedba je lahko povezana (korelirana) z glavno poizvedbo. Recimo, da bi želeli podatke o zadnjem naročilu za vsako stranko:

```
select CustomerId,SalesOrderID,orderDate from SalesLT.SalesOrderHeader as s01
where orderdate=(select max(orderdate) from SalesLT.SalesOrderHeader as s02
                 where s02.CustomerID=s01.CustomerID)
order by CustomerID
```

Operator APPLY omogoča klic tabelaričnih funkcij za vsako vrnjeno vrstico zunanje poizvedbe. CROSS APPLY aplicira rezultat funkcije na vsako vrstico leve (glavne) poizvedbe. Podoben je CROSS JOIN, le da sta tabeli povezani med seboj. OUTER APPLY doda vrstice skupaj z vrednostmi NULL v stolpcih desne tabele. Konceptualno podoben LEFT OUTER JOIN.
Primer:

```
select SOH.SalesOrderID,MUP.MaxUnitPrice
from SalesLT.SalesOrderHeader as SOH
cross apply SalesLT.udfMaxUnitPrice(SOH.SalesOrderID) as MUP
Order by SOH.SalesOrderID;
```

V primeru kličemo funkcijo dbo.udfMaxUnitPrice(SOH.SalesOrderID), ki vrača podatke najvišji ceni za vsako naročilo (iz vseh poziciji vrstice izbere tisto, ki ima najvišjo ceno). Funkcija sprejme en parameter (SalesOrderID), ki ga dobi iz poizvedbe na levi (iz tabele z aliasem SOH) in vrača tabelo v kateri je najvišja cena na naročilu. Isto poizvedbo bi lahko naredili s pomočjo operatorja JOIN, a je za veliko ljudi ta rešitev bolj razumljiva.
Funkcija je zapisana s spodnjo SQL kodo

```
CREATE FUNCTION SalesLT.udfMaxUnitPrice
(
            @SalesOrderId int
)
RETURNS TABLE
AS
RETURN
(
      SELECT SalesOrderID,Max(unitPrice) as MaxUnitPrice from salesLT.SalesOrderDetail
      where SalesOrderID=@SalesOrderId
      group by SalesOrderID
)
GO
```

Reši vajo 6!

7. Spremenljivke, pogledi, začasne tabele

Pogled (View) je virtualna tabela, ki jo definiramo s pomočjo poizvedbe. Prav tako kot tabela ima stolpce in vrstice podatkov. Če pogled ni indeksiran, ne obstaja kot shranjena zbirka v podatkovni bazi. Vrstice podatkov se dinamično napolnijo, ko se na pogled sklicujemo. Pogled deluje kot filter osnovnih tabel. Uporabljamo ga za poenostavitev pogleda na podatke za posameznega uporabnika. Pogled omogoča dostop do podatkov, ne da bi uporabnik imel pravice do tabel, iz katerih je pogled zgrajen. Pogled uporabljamo tudi v primeru, ko tabeli spremenimo shemo. Prek pogleda lahko omogočimo dostop do starejše strukture tabele.

View izdelamo tako, da dodamo DDL stavek create view. Na primer

```
create view Nov as
select * from SalesLT.Address
```

V SQL-u lahko deklariramo tudi spremenljivke. Običajno spremenljivke uporabljamo

- Kot zančne števce
- V njih hranimo vrednosti za kontrolo toka programa
- Za shranjevanje vrednosti, ki jo vrne shranjena procedura ali funkcija

Pred uporabo moramo spremenljivke deklarirati. Pri deklaraciji povemo ime spremenljivke, kjer je prvi znak obvezno @, nato povemo podatkovni tip , prevajalnik nastavi vrednost na NULL.

```
DECLARE @MyCounter int;
DECLARE @LastName nvarchar(30), @FirstName nvarchar(20), @StateProvince nchar(2);
```

Spremenljivki nastavimo vrednost s stavkom SET.

```
SET @MyCounter=1;
```

Primer uporabe

```
-- Declare variables.
DECLARE @FirstNameVariable nvarchar(50);
-- Set their values.
SET @FirstNameVariable = N'Amy';
-- Use them in the WHERE clause of a SELECT statement.
SELECT LastName, FirstName
FROM SalesLT.Customer
WHERE FirstName = @FirstNameVariable;
```

Doseg spremenljivke je v delu kode, kje je definirana.

Spremenljivke so lahko tudi tabele. Na primer:

```
declare @varProdukti as table
(ProductID integer, productName varchar(50));
select  * from @varProdukti
```

Doseg take tabele je v enem »batchu« (stavkih, ki jih poganjamo skupaj), zato lahko namesto, da tabelo deklariramo kot spremenljivko, uporabimo začasno tabelo. Deklariramo jo s # pred imenom (ali ##, če želimo globalno tabelo).

```
create table #tempProdukti
(ProductID integer,ImeProdukta varchar(50));
insert into #tempProdukti
select ProductID,Name from SalesLT.Product
select * from #tempProdukti
```

Tabela bo veljavna v eni uporabniški seji. Če deklariramo tabelo z ##tempProdukti, bo le ta imela doseg toliko časa, dokler obstaja še kakšna odprta seja, ki dela s to tabelo. Začasne tabele pridejo v poštev, ko delamo z manjšim naborom podatkov. Primer:

```
create table #Barve
(Barva varchar(15));
insert into #Barve
select distinct Color from SalesLT.Product;
select * from #Barve
```

Izpeljane tabele so podobne pod poizvedbam. Njihov doseg je v okviru ene poizvedbe. Primeri:

```
--izpeljana tabela derived_year
select orderyear,count(distinct customerid) as cust_count
from
(select year(orderdate) as orderyear, customerid from SalesLT.SalesOrderHeader)
as derived_year
group by orderyear;
--izpeljana tabela derived_year definirana malo drugače
select orderyear,count(distinct custid) as cust_count
from
(select year(orderdate) , customerid from SalesLT.SalesOrderHeader)
as derived_year(orderyear,custid)
group by orderyear;
```

S stavkom WITH določimo poimenovano začasno tabelo poznano kot »pogosti izrazi na tabelah« (Common table expression). Tabelo izdelamo s poizvedbo in je uporabna v samo enem SELECT, INSERT, UPDATE ali DELTE stavku. Stavek WITH lahko uporabljamo tudi skupaj s CREATE VIEW, lahko uporabimo tudi rekurzijo (tabela se sklicuje sama nase)
Primeri:

```
--koliko različnih kupcev smo imeli v posameznem letu
with cte_leto (LetoNar,KupId)
as (select year(orderdate), CustomerID from SalesLT.SalesOrderHeader)
select letoNar,count(distinct KupID) as c from cte_leto group by letoNar ;
-- Ustvari tabelo Employee.
CREATE TABLE dbo.MyEmployees
(
EmployeeID smallint NOT NULL,
FirstName nvarchar(30)  NOT NULL,
LastName  nvarchar(40) NOT NULL,
Title nvarchar(50) NOT NULL,
DeptID smallint NOT NULL,
ManagerID int NULL,
 CONSTRAINT PK_EmployeeID PRIMARY KEY CLUSTERED (EmployeeID ASC)
);
-- Napolni tabelo z vrednostmi.
INSERT INTO dbo.MyEmployees VALUES
 (1, N'Ken', N'Sánchez', N'Chief Executive Officer',16,NULL)
,(273, N'Brian', N'Welcker', N'Vice President of Sales',3,1)
,(274, N'Stephen', N'Jiang', N'North American Sales Manager',3,273)
,(275, N'Michael', N'Blythe', N'Sales Representative',3,274)
,(276, N'Linda', N'Mitchell', N'Sales Representative',3,274)
,(285, N'Syed', N'Abbas', N'Pacific Sales Manager',3,273)
,(286, N'Lynn', N'Tsoflias', N'Sales Representative',3,285)
,(16,  N'David',N'Bradley', N'Marketing Manager', 4, 273)
,(23,  N'Mary', N'Gibson', N'Marketing Specialist', 4, 16);
-- Poišči neposrednega nadrejenega.
WITH DirectReports(ManagerID, EmployeeID, Title, EmployeeLevel) AS
```

```
(
    SELECT ManagerID, EmployeeID, Title, 0 AS EmployeeLevel
    FROM dbo.MyEmployees
    WHERE ManagerID IS NULL
    UNION ALL
    SELECT e.ManagerID, e.EmployeeID, e.Title, EmployeeLevel + 1
    FROM dbo.MyEmployees AS e
        INNER JOIN DirectReports AS d
        ON e.ManagerID = d.EmployeeID
)
SELECT ManagerID, EmployeeID, Title, EmployeeLevel
FROM DirectReports
ORDER BY ManagerID;

--omeji rekurzijo na 2 nivoja
WITH DirectReports(ManagerID, EmployeeID, Title, EmployeeLevel) AS
(
    SELECT ManagerID, EmployeeID, Title, 0 AS EmployeeLevel
    FROM dbo.MyEmployees
    WHERE ManagerID IS NULL
    UNION ALL
    SELECT e.ManagerID, e.EmployeeID, e.Title, EmployeeLevel + 1
    FROM dbo.MyEmployees AS e
        INNER JOIN DirectReports AS d
        ON e.ManagerID = d.EmployeeID
)
SELECT ManagerID, EmployeeID, Title, EmployeeLevel
FROM DirectReports
WHERE EmployeeLevel <= 2 ;
--nariši hierarhično drevo
WITH DirectReports(Name, Title, EmployeeID, EmployeeLevel, Sort)
AS (SELECT CONVERT(varchar(255), e.FirstName + ' ' + e.LastName),
        e.Title,
        e.EmployeeID,
        1,
        CONVERT(varchar(255), e.FirstName + ' ' + e.LastName)
    FROM dbo.MyEmployees AS e
    WHERE e.ManagerID IS NULL
    UNION ALL
    SELECT CONVERT(varchar(255), REPLICATE ('|    ' , EmployeeLevel) +
        e.FirstName + ' ' + e.LastName),
        e.Title,
        e.EmployeeID,
        EmployeeLevel + 1,
        CONVERT (varchar(255), RTRIM(Sort) + '|    ' + FirstName + ' ' +
                LastName)
    FROM dbo.MyEmployees AS e
    JOIN DirectReports AS d ON e.ManagerID = d.EmployeeID
    )
SELECT EmployeeID, Name, Title, EmployeeLevel
FROM DirectReports
ORDER BY Sort;
```

Reši vajo 7.

8. Hierarhično grupiranje

Ko uporabljamo v poizvedbah agregate, želimo velikokrat imeti podatke grupirane v skupine. Poznamo že osnovno uporabo stavka GROUP BY, ki omogoča, da računamo na primer vsote po nekem atributu.

Poglejmo primer. Najprej si definiramo tabelo Sales, nato jo napolnimo s podatki:

```
CREATE TABLE Sales ( Country varchar(50), Region varchar(50), Sales int );

INSERT INTO sales VALUES (N'Canada', N'Alberta', 100);
INSERT INTO sales VALUES (N'Canada', N'British Columbia', 200);
INSERT INTO sales VALUES (N'Canada', N'British Columbia', 300);
INSERT INTO sales VALUES (N'United States', N'Montana', 100);
```

V tabeli so spodnji podatki

```
Country             Region       Sales

Canada              Alberta      100

Canada British      Columbia     200

Canada British      Columbia     300

United States       Montana      100
```

Poizvedba

```
SELECT Country, Region, SUM(sales) AS TotalSales
FROM Sales
GROUP BY Country, Region;
```

Vrača

```
Country       Region              TotalSales

Canada        Alberta             100

Canada        British Columbia    500

United States Montana             100
```

Podatke smo grupirali po državi in regiji. Kaj pa če želimo delne vsote? V našem primeru bi želeli skupno vsoto vseh prodaj, skupno vsoto za Kanado in skupno vsoto za ZDA, pa še delne vsote po regijah. V tem primeru uporabimo GROUP BY ROLLUP(col1,col2,col3,col4), ki nam za ta prier vrne vse kombinacije:

- col1, col2, col3, col4
- col1, col2, col3, NULL
- col1, col2, NULL, NULL
- col1, NULL, NULL, NULL
- NULL, NULL, NULL, NULL --This is the grand total

Na našem primeru, bi stavek

```
SELECT Country, Region, SUM(Sales) AS TotalSales
FROM Sales
GROUP BY ROLLUP (Country, Region);
```

vračal

```
Country       Region              TotalSales
Canada        Alberta             100
Canada        British Columbia    500
Canada        NULL                600
United States Montana             100
United States NULL                100
NULL   NULL                      700
```

Če nas zanimajo vse možne kombinacije delnih vsot, uporabimo GROUP BY CUBE(a,b), ki vrača (a, b), (NULL, b), (a, NULL), and (NULL, NULL). Za naš primer:

```
SELECT Country, Region, SUM(Sales) AS TotalSales
FROM Sales
GROUP BY CUBE (Country, Region);
Country       Region              TotalSales
Canada        Alberta             100
```

```
NULL          Alberta             100
Canada        British Columbia    500
NULL          British Columbia    500
United States Montana             100
NULL          Montana             100
NULL          NULL                700
Canada        NULL                600
United States NULL                100
```

Uporabimo pa lahko tufi GROUP BY GROUPING SETS (st1,st2,….). Če želimo skupno vsoto v množici, jo označimo z (). Primer:

```
SELECT Country, SUM(Sales) AS TotalSales
FROM Sales
GROUP BY GROUPING SETS ( Country, () );
```

Vrača rezultat

```
Country       TotalSales
Canada        600
United States 100
NULL          700
```

GROUPING_ID vrača številko nivoja v hierarhiji, kjer smo. Primer

```
SELECT Country, Region, SUM(Sales) AS TotalSales,GROUPING_ID(Country) as IDCo,
GROUPING_ID(Region) as IDReg
FROM Sales
GROUP BY ROLLUP (Country, Region);
```

```
Country       Region              TotalSales    IDCo   IDReg
Canada        Alberta             100           0      0
Canada        British Columbia    500           0      0
Canada        NULL                600           0      1
United States Montana             100           0      0
United States NULL                100           0      1
NULL          NULL                700           1      1
```

Kjer je Id=1, imamo delne vsote (IDReg=1, kjer imam vsoto za Kanado in ZDA), IDCo in IDReg=1 kjer je skupna vsota. S tem lahko izboljšamo izpis. Na primer:

```
SELECT
IIF(GROUPING_ID(Country) =1 and GROUPING_ID(Region) =1,'Vse skupaj',
    IIF(GROUPING_ID(Region)=1,'Skupaj '+Country,Region)) as Level,
       SUM(Sales) AS TotalSales
FROM Sales
GROUP BY ROLLUP (Country, Region);
```

```
Level                TotalSales
Alberta              100
British Columbia     500
Skupaj Canada        600
Montana              100
Skupaj United States 100
Vse skupaj           700
```

9. Vnos, popravljanje in brisanje podatkov

a. Vstavljanje (INSERT)

Imamo več možnosti. Vstavimo lahko **eno vrstico** ali **več vrstic**. Uporabimo tabelo MyEmployees, ki smo jo generirali v prejšnjem poglavju:

```
insert into MyEmployees
values (2345,'Janez','Novak','Gospod',1,1);
insert into MyEmployees
values (2346,'Meri','Novak','Gospa',1,1),
       (2347,'Pepi','Novak','Gospod',1,1),
       (2348,'Marija','Novak','Gospa',1,1);
```

Vstavimo lahko tudi podatke, ki niso v istem **vrstnem redu** kot so našteti atributi v definiciji naše relacije. V tem primeru naštejemo imena atributov in vrednosti atributov.

```
insert into MyEmployees (EmployeeID,Title,LastName,FirstName, DeptID)
values (2349,'Gospod','Novak','Janez',1);
```

Pri vstavljanju podatkov, ki so definirani kot **identiteta (identity), privzeta vrednost (default)** ali imajo podatkovni tip kot na primer **uniqueidentifier**, moramo vrednosti za vstavljanje našteti nekoliko drugače. Izdelajmo si tabelo T1 s stavkom:

```
CREATE TABLE dbo.T1
(
    column_1 AS 'Computed column ' + column_2,
    column_2 varchar(30)
        CONSTRAINT default_name DEFAULT ('my column default'),
    column_3 rowversion,
    column_4 varchar(40) NULL
);
```

Column_1 je izračunana vrednost, ki nizu 'Computed column' doda niz, ki se nahaja v column_2. Column_2 ima privzeto vrednots 'my column default', ki se uporabi, če ne vnesem nobene vrednosti. Column_3 je tipa rowversion, ki se napolni avtomatsko z enoličnim binarnim številom, column_4 je navaden niz, če zanj ne navedemo vrednosti, bo imel vrednost null. Izdelajmo sedaj stavke za vstavljanje in poglejmo rezultat:

```
INSERT INTO dbo.T1 (column_4)
    VALUES ('Explicit value');
INSERT INTO dbo.T1 (column_2, column_4)
    VALUES ('Explicit value', 'Explicit value');
INSERT INTO dbo.T1 (column_2)
    VALUES ('Explicit value');
INSERT INTO T1 DEFAULT VALUES;

SELECT column_1, column_2, column_3, column_4
FROM dbo.T1;
```

```
column_1                          column_2            column_3            column_4
Computed column my column default my column default   0x00000000000007D1  Explicit
value
Computed column Explicit value    Explicit value      0x00000000000007D2  Explicit
value
Computed column Explicit value    Explicit value      0x00000000000007D3  NULL
Computed column my column default my column default   0x00000000000007D4  NULL
```

Prvo vstavljanje vstavi v column_4 navedeno vrednost, column_2 dobi privzeto vrednost, column_1 dobi izračunano vrednost, column_3 dobi avtomatično vrednost. Pri drugem vstavljanju se prepiše

vrednost column_2 (ni več privzeta vrednost, ampak vrednost, ki smo jo navedli). Tretje vstavljanje nastavi column_4 na NULL, ker vrednosti nismo vnesli. Četrto vstavljanje napolni vrstico s privzetimi vrednostmi.
Vstavljanje **identitete** (samoštevilo) si poglejmo na primeru. Denimo, da imamo tabelo

```
CREATE TABLE dbo.T1 ( column_1 int IDENTITY, column_2 VARCHAR(30));
```

Vstavljanje v tabelo z identiteto poteka tako, da vrednosti identitete ne naštejamo

```
INSERT T1 VALUES ('Row #1');
INSERT T1 (column_2) VALUES ('Row #2');
```

Če želimo identiteto vnesti, moramo prej to dovoliti

```
SET IDENTITY_INSERT T1 ON;

INSERT INTO T1 (column_1,column_2)
    VALUES (-98, 'Explicit identity value');
```

Vstavljanje vrednost stolpca, ki ima oznako **uniqueindetifier** poteka prek funkcije NEWID(). Če vrednosti ne vnesemo, je privzeta vrednost NULL. Primer:

```
CREATE TABLE dbo.T1
(
    column_1 int IDENTITY,
    column_2 uniqueidentifier,
);

INSERT INTO dbo.T1 (column_2)
    VALUES (NEWID());
INSERT INTO T1 DEFAULT VALUES;

SELECT column_1, column_2
FROM dbo.T1;
```

Vrednosti lahko vstavljamo tudi iz druge tabele, lahko pa jih vstavimo tudi v tabelarično spremenljivko.

b. Posodabljanje (UPDATE)

Stolpec v tabeli posodobimo s stavkom UPDATE. Na primer:

```
USE AdventureWorks2012;
UPDATE Person.Address
SET ModifiedDate = GETDATE();
```

SQL strežnik odgovori s številom posodobljenih vrstic. Več stolpcev posodobimo s stavkom

```
UPDATE Sales.SalesPerson
SET Bonus = 6000, CommissionPct = .10, SalesQuota = NULL;
```

Običajno ne želimo posodobiti vseh vrstic, zato uporabimo stavek WHERE

```
UPDATE Production.Product
SET Color = N'Metallic Red'
WHERE Name LIKE N'Road-250%' AND Color = N'Red';
```

Pri posodabljanju lahko uporabimo tudi **CTE (common table expressions**). Denimo, da bi želeli posodobiti vse dele, ki sestavljajo produkt s kodo ProductAssembly=800. V tabeli Parts bomo imeli vse dele, ki sestavljajo proizvod 800, nato vse dele, ki so sestavni deli prvega nivoja itd.
Nato lahko posodobimo vse sestavne dele produkta 800:

```
WITH Parts(AssemblyID, ComponentID, PerAssemblyQty, EndDate, ComponentLevel) AS
(
    SELECT b.ProductAssemblyID, b.ComponentID, b.PerAssemblyQty,
        b.EndDate, 0 AS ComponentLevel
    FROM Production.BillOfMaterials AS b
    WHERE b.ProductAssemblyID = 800
          AND b.EndDate IS NULL
    UNION ALL
    SELECT bom.ProductAssemblyID, bom.ComponentID, p.PerAssemblyQty,
        bom.EndDate, ComponentLevel + 1
    FROM Production.BillOfMaterials AS bom
        INNER JOIN Parts AS p
        ON bom.ProductAssemblyID = p.ComponentID
        AND bom.EndDate IS NULL
)
```

Results | Messages

	AssemblyID	ComponentID	PerAssemblyQty	EndDate	ComponentLevel
1	800	518	2.00	NULL	0
2	800	806	2.00	NULL	0
3	800	812	2.00	NULL	0
4	800	819	2.00	NULL	0
5	800	827	2.00	NULL	0
6	800	835	2.00	NULL	0
7	800	894	2.00	NULL	0
8	800	907	2.00	NULL	0
9	800	939	2.00	NULL	0
10	800	945	2.00	NULL	0
11	800	948	2.00	NULL	0
12	800	950	2.00	NULL	0
13	800	952	2.00	NULL	0
14	800	994	2.00	NULL	0
15	994	3	2.00	NULL	1
16	994	525	2.00	NULL	1
17	3	2	2.00	NULL	2
18	3	461	2.00	NULL	2
19	3	504	2.00	NULL	2

```
UPDATE Production.BillOfMaterials
SET PerAssemblyQty = c.PerAssemblyQty * 2
FROM Production.BillOfMaterials AS c
JOIN Parts AS d ON c.ProductAssemblyID = d.AssemblyID
WHERE d.ComponentLevel = 0;
```

Vrednosti, ki jih nastavimo po SET so lahko **izračunane.** Primer:

```
UPDATE Production.Product
SET ListPrice = ListPrice * 2;
```

Vrednosti lahko dobimo tudi kot rezultat pod poizvedbe. Na primer:

```
UPDATE Sales.SalesPerson
SET SalesYTD = SalesYTD +
    (SELECT SUM(so.SubTotal)
     FROM Sales.SalesOrderHeader AS so
     WHERE so.OrderDate = (SELECT MAX(OrderDate)
                           FROM Sales.SalesOrderHeader AS so2
                           WHERE so2.SalesPersonID = so.SalesPersonID)
     AND Sales.SalesPerson.BusinessEntityID = so.SalesPersonID
     GROUP BY so.SalesPersonID);
```

Lahko pa uporabimo vrednosti iz druge tabele. Na primer:

```
--subtotal je stolpec v salesorderheader
UPDATE Sales.SalesPerson
SET SalesYTD = SalesYTD + SubTotal
FROM Sales.SalesPerson AS sp
JOIN Sales.SalesOrderHeader AS so
    ON sp.BusinessEntityID = so.SalesPersonID
```

```
    AND so.OrderDate = (SELECT MAX(OrderDate)
                        FROM Sales.SalesOrderHeader
                        WHERE SalesPersonID = sp.BusinessEntityID);
```

c) Brisanje (DELETE)

Podatke brišemo s

```
DELETE FROM Sales.SalesPersonQuotaHistory;
```

Bolj običajno pa je, da ne brišemo celotne vsebine tabele, ampak le del tabele. Na primer:

```
DELETE FROM Production.ProductCostHistory
WHERE StandardCost > 1000.00;
```

Reši vajo 9!