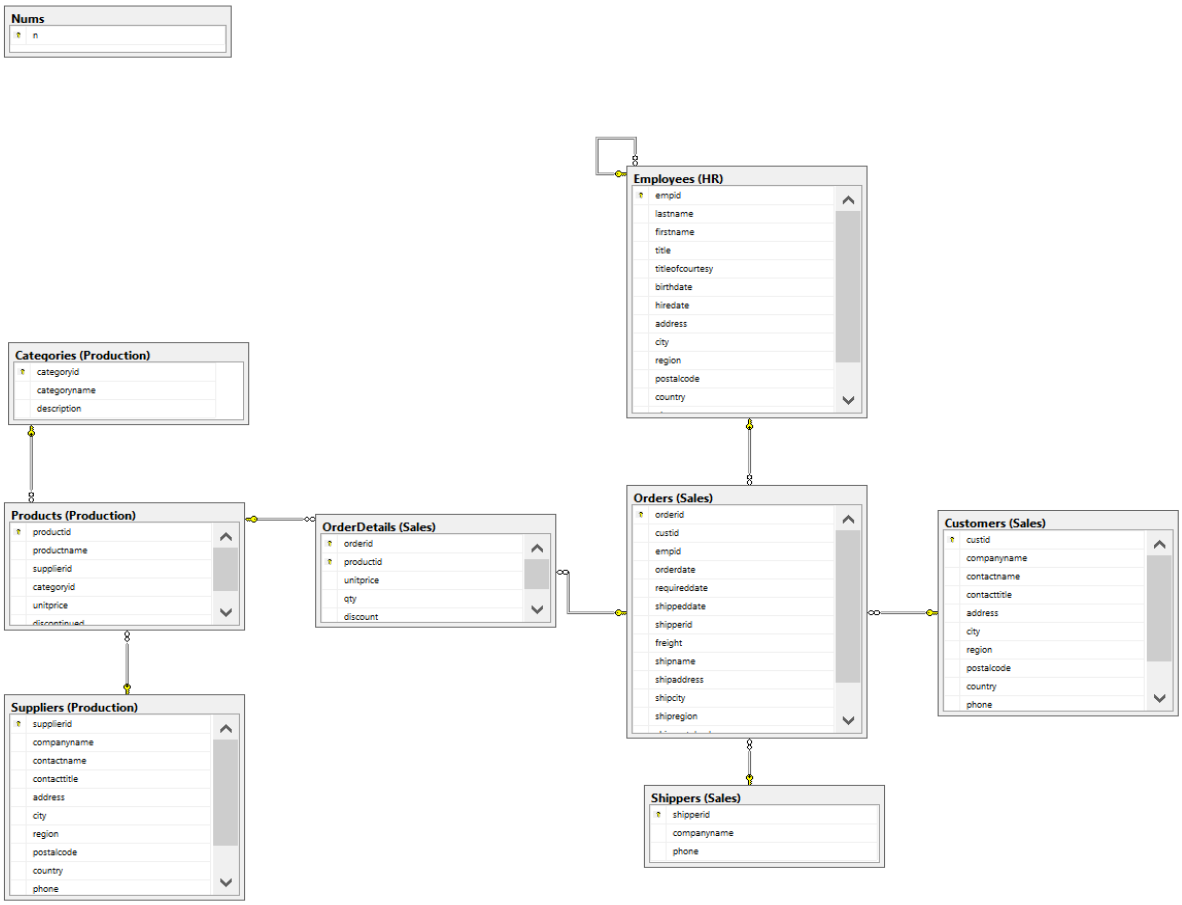
DATA TRAINING
TECH FOR BUSINESS
Contact: marcela.filip@datatraining.ro; +40 721 612 785; www.datatraining.ro

SQL PROGRAMMING FOR DEVELOPERS

VERSIUNEA: V3/2020.06.28

Contents

I. BAZA DE DATE TRAINING (DIAGRAMA)



Baza de date captureaza toate vanzarile realizate intr-o companie, clientii companiei, precum si tranzactiile dintre companie si furnizorii sai.

Baza de date contine urmatoarele informatii:

- Furnizorii care livreaza produsele companiei
- Clientii care cumpara de la companie
- Angajatii companiei
- Produsele existente in companie
- Transportatorii care transmit produsele de la comercianti la clientii finali
- Comenzile si detaliile despre acestea

Toate aceste informatii sunt stocate in diferite tabele. Tabelele au denumiri, in functie de ce informatii detin. Scopul crearii mai multor tabele este de a da userului o imagine de ansamblu asupra procesului de business si ce rapoarte sunt necesare si pot fi extrase. Relatiile dintre tabele ne redau **flow-ul de business**.

Tabelele sunt organizate pe diferite scheme: dbo, HR, Production, Sales, Stats.

II. RECAPITULARE SQL QUERYING PENTRU BUSINESS

Exemple:

1. Sa se determine numarul de clienti din fiecare tara.
2. Sa se determine numarul de comenzi livrate si suma costului de transport la nivel de tara client.
3. Sa se determine la nivel de fiecare tara client:
    a. Numarul de client unici
    b. Numarul de comenzi unice livrate
    c. Suma incasarilor
4. Sa se modifice interogarea anterioara astfel incat:
    a. Se vor afisa doar tarile unde incasarile sunt peste 5000 si numarul de clienti unici cel putin 3
    b. Se vor lua in considerare doar comenzile livrate in 2007 si 2008
5. Sa se defineasca un view cu statementul de la punctul 4.

III. STORED PROCEDURES

O **procedura stocata** este un obiect in baza de date care inglobeaza un cod si care poate fi reutilizat.

O procedura stocata poate primi parametri, iar rezultatul executiei sale depinde de valorile parametrilor stabiliti la rulare.

Apelarea unei procedurii stocate se realizeaza cu comanda EXECUTE sau EXEC.

Executarea unei proceduri stocate are ca rezultat:

- Un set de date (rezultatul unui select)
- Poate modifica date in baza de date (insert, update, delete – DML statements)
  Nota: DML = Data Manipulation Language
- Poate crea, altera, sterge tabele sau indecsi (alter, drop, create)
  Nota: DDL = Data Definition Language

Avantaje pentru a utiliza proceduri stocate:

- Incapsuleaza intreg codul sql
- Poate fi apelata din diferite locuri cu diferiti parametri, iar rezultatul este diferit in functie de fiecare rulare
- Baza de date este securizata (un user primeste acces pe o procedura stocata, nu si pe tabelele utilizate in cadrul procedurii stocate)

Exemplu: Sa se determine toate comenzile clientului 37 din anul 2007.

```
Use Training;
go
SELECT
         orderid
       , orderdate
       , shippeddate
       , custid
from Sales.Orders
where
       custid=37
       and orderdate>='2007-01-01'
       and orderdate<='2007-12-31'
```

	orderid	orderdate	shippeddate	custid
1	10429	2007-01-29 00:00:00.000	2007-02-07 00:00:00.000	37
2	10503	2007-04-11 00:00:00.000	2007-04-16 00:00:00.000	37
3	10516	2007-04-24 00:00:00.000	2007-05-01 00:00:00.000	37
4	10567	2007-06-12 00:00:00.000	2007-06-17 00:00:00.000	37
5	10646	2007-08-27 00:00:00.000	2007-09-03 00:00:00.000	37
6	10661	2007-09-09 00:00:00.000	2007-09-15 00:00:00.000	37
7	10687	2007-09-30 00:00:00.000	2007-10-30 00:00:00.000	37
8	10701	2007-10-13 00:00:00.000	2007-10-15 00:00:00.000	37
9	10712	2007-10-21 00:00:00.000	2007-10-31 00:00:00.000	37
10	10736	2007-11-11 00:00:00.000	2007-11-21 00:00:00.000	37

Observatii:

- “Use Training;” – seteaza baza de date Training activa pentru aceasta sesiune
- “Go” – stabileste finalul unei instructiuni

Rezultatul selectului de mai sus este format din 10 comenzi.

Se cere un nou raport, pentru clientul 53, din anul 2008. In acest caz, se vor modifica doar conditiile.

```
Use Training;
go
SELECT
          orderid
        , orderdate
        , shippeddate
        , custid
from Sales.Orders
where
        custid=53
        and orderdate>='2008-01-01'
        and orderdate<='2008-12-31'
```

Rezultatul este:

	orderid	orderdate	shippeddate	custid
1	11057	2008-04-29 00:00:00.000	2008-05-01 00:00:00.000	53

Daca acelasi raport este solicitat unui alt coleg, cel de-al doilea coleg trebuie sa scrie si el codul sql. Acest lucru se va evita creand o procedura stocata cu 3 parametri de intrare: id-ul de client, data de inceput si data de sfarsit.

Procedura stocata va putea fi executata de diferiti useri cu diferiti parametri.

Creare procedura stocata:

- CREATE PROCEDURE nume (parametri) sau CREATE PROC nume (parametri)

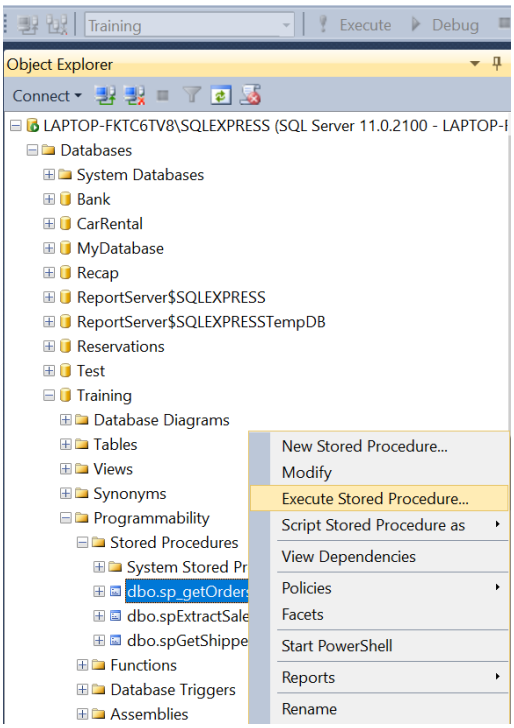
```
create procedure sp_getOrders (@idclient int, @start_date date, @end_date date)
as
```

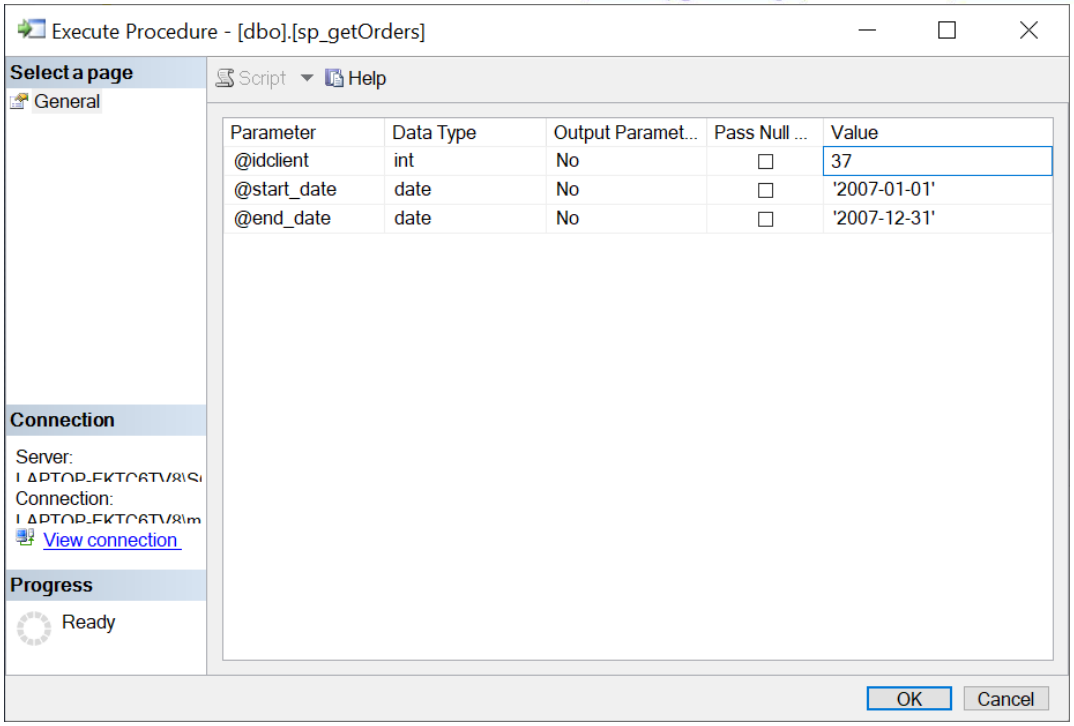
```
SELECT
        orderid
      , orderdate
      , shippeddate
      , custid
from Sales.Orders
where
      custid=@idclient
      and orderdate>=@start_date
      and orderdate<=@end_date
```

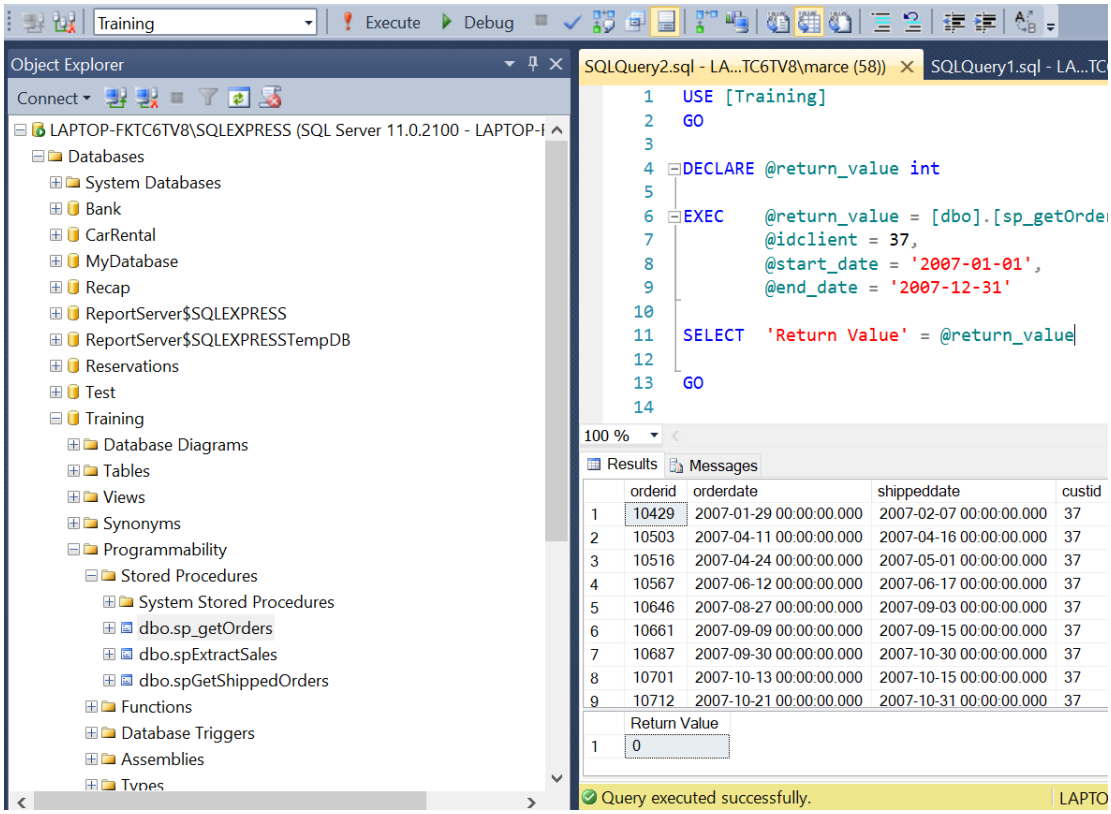
Executare procedura stocata:

Procedura poate fi executata astfel:

- din Object Explorer/ baza de date Training / Programmability / Stored Procedures click dreapta pe numele ei / Execute Stored Procedure







- utilizand comanda EXECUTE:

```
Use Training;
go

execute sp_getOrders @idclient=37, @start_date='2007-01-01', @end_date='2007-12-31'
```

Results | Messages

	orderid	orderdate	shippeddate	custid
1	10429	2007-01-29 00:00:00.000	2007-02-07 00:00:00.000	37
2	10503	2007-04-11 00:00:00.000	2007-04-16 00:00:00.000	37
3	10516	2007-04-24 00:00:00.000	2007-05-01 00:00:00.000	37
4	10567	2007-06-12 00:00:00.000	2007-06-17 00:00:00.000	37
5	10646	2007-08-27 00:00:00.000	2007-09-03 00:00:00.000	37
6	10661	2007-09-09 00:00:00.000	2007-09-15 00:00:00.000	37
7	10687	2007-09-30 00:00:00.000	2007-10-30 00:00:00.000	37
8	10701	2007-10-13 00:00:00.000	2007-10-15 00:00:00.000	37
9	10712	2007-10-21 00:00:00.000	2007-10-31 00:00:00.000	37
10	10736	2007-11-11 00:00:00.000	2007-11-21 00:00:00.000	37

- utilizand comanda EXEC

```
Use Training;
go

exec sp_getOrders @idclient=37, @start_date='2007-01-01', @end_date='2007-12-31'
```

Results | Messages

	orderid	orderdate	shippeddate	custid
1	10429	2007-01-29 00:00:00.000	2007-02-07 00:00:00.000	37
2	10503	2007-04-11 00:00:00.000	2007-04-16 00:00:00.000	37
3	10516	2007-04-24 00:00:00.000	2007-05-01 00:00:00.000	37
4	10567	2007-06-12 00:00:00.000	2007-06-17 00:00:00.000	37
5	10646	2007-08-27 00:00:00.000	2007-09-03 00:00:00.000	37
6	10661	2007-09-09 00:00:00.000	2007-09-15 00:00:00.000	37
7	10687	2007-09-30 00:00:00.000	2007-10-30 00:00:00.000	37
8	10701	2007-10-13 00:00:00.000	2007-10-15 00:00:00.000	37
9	10712	2007-10-21 00:00:00.000	2007-10-31 00:00:00.000	37
10	10736	2007-11-11 00:00:00.000	2007-11-21 00:00:00.000	37

- direct F5 sau prin accesarea butonului Execute

```
Use Training;
go

sp_getOrders @idclient=37, @start_date='2007-01-01', @end_date='2007-12-31'
```

Results | Messages

	orderid	orderdate	shippeddate	custid
1	10429	2007-01-29 00:00:00.000	2007-02-07 00:00:00.000	37
2	10503	2007-04-11 00:00:00.000	2007-04-16 00:00:00.000	37
3	10516	2007-04-24 00:00:00.000	2007-05-01 00:00:00.000	37
4	10567	2007-06-12 00:00:00.000	2007-06-17 00:00:00.000	37
5	10646	2007-08-27 00:00:00.000	2007-09-03 00:00:00.000	37
6	10661	2007-09-09 00:00:00.000	2007-09-15 00:00:00.000	37
7	10687	2007-09-30 00:00:00.000	2007-10-30 00:00:00.000	37
8	10701	2007-10-13 00:00:00.000	2007-10-15 00:00:00.000	37
9	10712	2007-10-21 00:00:00.000	2007-10-31 00:00:00.000	37
10	10736	2007-11-11 00:00:00.000	2007-11-21 00:00:00.000	37

- direct F5 sau prin accesarea butonului Execute pastrand ordinea parametrilor

```
Use Training;
go

sp_getOrders  37, '2007-01-01', '2007-12-31'
```

Results | Messages

	orderid	orderdate	shippeddate	custid
1	10429	2007-01-29 00:00:00.000	2007-02-07 00:00:00.000	37
2	10503	2007-04-11 00:00:00.000	2007-04-16 00:00:00.000	37
3	10516	2007-04-24 00:00:00.000	2007-05-01 00:00:00.000	37
4	10567	2007-06-12 00:00:00.000	2007-06-17 00:00:00.000	37
5	10646	2007-08-27 00:00:00.000	2007-09-03 00:00:00.000	37
6	10661	2007-09-09 00:00:00.000	2007-09-15 00:00:00.000	37
7	10687	2007-09-30 00:00:00.000	2007-10-30 00:00:00.000	37
8	10701	2007-10-13 00:00:00.000	2007-10-15 00:00:00.000	37
9	10712	2007-10-21 00:00:00.000	2007-10-31 00:00:00.000	37
10	10736	2007-11-11 00:00:00.000	2007-11-21 00:00:00.000	37

- direct F5 sau prin accesarea butonului Execute inversand ordinea parametrilor, insa specificand numele lor

```
Use Training;
go

sp_getOrders @start_date='2007-01-01', @end_date='2007-12-31', @idclient=37
```

Results | Messages

	orderid	orderdate	shippeddate	custid
1	10429	2007-01-29 00:00:00.000	2007-02-07 00:00:00.000	37
2	10503	2007-04-11 00:00:00.000	2007-04-16 00:00:00.000	37
3	10516	2007-04-24 00:00:00.000	2007-05-01 00:00:00.000	37
4	10567	2007-06-12 00:00:00.000	2007-06-17 00:00:00.000	37
5	10646	2007-08-27 00:00:00.000	2007-09-03 00:00:00.000	37
6	10661	2007-09-09 00:00:00.000	2007-09-15 00:00:00.000	37
7	10687	2007-09-30 00:00:00.000	2007-10-30 00:00:00.000	37
8	10701	2007-10-13 00:00:00.000	2007-10-15 00:00:00.000	37
9	10712	2007-10-21 00:00:00.000	2007-10-31 00:00:00.000	37
10	10736	2007-11-11 00:00:00.000	2007-11-21 00:00:00.000	37

Verificarea daca procedura exista deja in baza de date se realizeaza astfel:

```
select * from sys.all_objects where NAME='sp_getOrders' and type='P'
```

	name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date
1	sp_getOrders	1525580473	NULL	1	0	P	SQL_STORED_PROCEDURE	2019-11-19 16:49:58.317	2019-11-19 16:49:58.317

Daca procedura exista, inainte de a rula comanda de CREATE, aceasta va fi stearsa.

```
if (select object_id from sys.all_objects where NAME='sp_getOrders' and type='P') is
not null
drop procedure sp_getOrders
```

Acelasi lucru poate fi realizat cu ajutorul functiei OBJECT_ID, iar sintaxa de mai jos este cea mai des utilizata.

```
IF OBJECT_ID('sp_getOrders', 'P') IS NOT NULL
DROP PROC sp_getOrders;
```

Executarea procedurii are un rezultat si un mesaj:

```
Use Training;
go

exec sp_getOrders @idclient=37, @start_date='2007-01-01', @end_date='2007-12-31'
```

Rezultat:

Results | Messages

	orderid	orderdate	shippeddate	custid
1	10429	2007-01-29 00:00:00.000	2007-02-07 00:00:00.000	37
2	10503	2007-04-11 00:00:00.000	2007-04-16 00:00:00.000	37
3	10516	2007-04-24 00:00:00.000	2007-05-01 00:00:00.000	37
4	10567	2007-06-12 00:00:00.000	2007-06-17 00:00:00.000	37
5	10646	2007-08-27 00:00:00.000	2007-09-03 00:00:00.000	37
6	10661	2007-09-09 00:00:00.000	2007-09-15 00:00:00.000	37
7	10687	2007-09-30 00:00:00.000	2007-10-30 00:00:00.000	37
8	10701	2007-10-13 00:00:00.000	2007-10-15 00:00:00.000	37
9	10712	2007-10-21 00:00:00.000	2007-10-31 00:00:00.000	37
10	10736	2007-11-11 00:00:00.000	2007-11-21 00:00:00.000	37

Mesaj:

```
(10 row(s) affected)
```

Daca se doreste ca acest mesaj sa nu mai apara, atunci, in corpul procedurii se insereaza comanda: SET NOCOUNT, imediat dupa AS.

```
IF OBJECT_ID('sp_getOrders', 'P') IS NOT NULL
DROP PROC sp_getOrders;
GO
create procedure sp_getOrders (@idclient int, @start_date date, @end_date date)
as
SET NOCOUNT ON;
SELECT
         orderid
       , orderdate
       , shippeddate
       , custid
from Sales.Orders
where
       custid=@idclient
       and orderdate>=@start_date
       and orderdate<=@end_date


Use Training;
go

exec sp_getOrders @idclient=37, @start_date='2007-01-01', @end_date='2007-12-31'
```

Rezultat:

Results | Messages

	orderid	orderdate	shippeddate	custid
1	10429	2007-01-29 00:00:00.000	2007-02-07 00:00:00.000	37
2	10503	2007-04-11 00:00:00.000	2007-04-16 00:00:00.000	37
3	10516	2007-04-24 00:00:00.000	2007-05-01 00:00:00.000	37
4	10567	2007-06-12 00:00:00.000	2007-06-17 00:00:00.000	37
5	10646	2007-08-27 00:00:00.000	2007-09-03 00:00:00.000	37
6	10661	2007-09-09 00:00:00.000	2007-09-15 00:00:00.000	37
7	10687	2007-09-30 00:00:00.000	2007-10-30 00:00:00.000	37
8	10701	2007-10-13 00:00:00.000	2007-10-15 00:00:00.000	37
9	10712	2007-10-21 00:00:00.000	2007-10-31 00:00:00.000	37
10	10736	2007-11-11 00:00:00.000	2007-11-21 00:00:00.000	37

Mesaj:

```
Command(s) completed successfully.
```

Parametri obligatorii sau optionali:

O procedura stocata poate avea parametri obligatorii sau optionali.

In momentul crearii procedurii, daca se mentioneaza o initializare a parametrilor, atunci, in momentul executarii procedurii, completarea acestora este optionala. In acest caz parametrii sunt optionali.

Exemplu: Sa se construiasca o procedura stocata care sa determina lista comenzilor plasate de un anumit client intr-o perioada. In cazul in care se omite perioada sau se doreste afisarea tuturor comenzilor unui client, atunci perioada sa nu fie nevoie sa fie completata.

```
IF OBJECT_ID('sp_getOrders', 'P') IS NOT NULL
DROP PROC sp_getOrders;
GO
create procedure sp_getOrders (@idclient int, @start_date date='1900-01-01', @end_date
date='2999-12-31')
as
SET NOCOUNT ON;
SELECT
         orderid
       , orderdate
       , shippeddate
       , custid
from Sales.Orders
where
       custid=@idclient
       and orderdate>=@start_date
       and orderdate<=@end_date
```

Executarea procedurii cu parametri optionali:

```
Use Training;
go

exec sp_getOrders 37
```

Observatii:

Parametrii @start_date si @end_date cu valorile implicite 1 ianuarie 1900 si 31 decembrie 2999 inseamna ca nu este specificata perioada, iar rezultatul procedurii ar trebui sa fie intreaga tabela Orders pentru clientul 37.

Results | Messages

	orderid	orderdate	shippeddate	custid
1	10298	2006-09-05 00:00:00.000	2006-09-11 00:00:00.000	37
2	10309	2006-09-19 00:00:00.000	2006-10-23 00:00:00.000	37
3	10335	2006-10-22 00:00:00.000	2006-10-24 00:00:00.000	37
4	10373	2006-12-05 00:00:00.000	2006-12-11 00:00:00.000	37
5	10380	2006-12-12 00:00:00.000	2007-01-16 00:00:00.000	37
6	10429	2007-01-29 00:00:00.000	2007-02-07 00:00:00.000	37
7	10503	2007-04-11 00:00:00.000	2007-04-16 00:00:00.000	37
8	10516	2007-04-24 00:00:00.000	2007-05-01 00:00:00.000	37
9	10567	2007-06-12 00:00:00.000	2007-06-17 00:00:00.000	37
10	10646	2007-08-27 00:00:00.000	2007-09-03 00:00:00.000	37
11	10661	2007-09-09 00:00:00.000	2007-09-15 00:00:00.000	37
12	10687	2007-09-30 00:00:00.000	2007-10-30 00:00:00.000	37
13	10701	2007-10-13 00:00:00.000	2007-10-15 00:00:00.000	37
14	10712	2007-10-21 00:00:00.000	2007-10-31 00:00:00.000	37
15	10736	2007-11-11 00:00:00.000	2007-11-21 00:00:00.000	37
16	10897	2008-02-19 00:00:00.000	2008-02-25 00:00:00.000	37
17	10912	2008-02-26 00:00:00.000	2008-03-18 00:00:00.000	37
18	10985	2008-03-30 00:00:00.000	2008-04-02 00:00:00.000	37
19	11063	2008-04-30 00:00:00.000	2008-05-06 00:00:00.000	37

Blocul BEGIN/END:

Incapsularea codului unei proceduri stocate poate fi realizata utilizand blocul BEGIN/END. Acest bloc nu este obligatoriu decat in cazul in care, corpul procedurii presupune mai multe statement-uri. Spre exemplu: un SELECT, un UPDATE, iarasi un SELECT etc.

Totusi, pentru claritatea codului este recomandata utilizarea blocului BEGIN/END.

```
IF OBJECT_ID('sp_getOrders', 'P') IS NOT NULL
DROP PROC sp_getOrders;
GO
create procedure sp_getOrders (@idclient int, @start_date date='1900-01-01', @end_date
date='2999-12-31')
as
BEGIN
SET NOCOUNT ON;
SELECT
         orderid
       , orderdate
       , shippeddate
       , custid
from Sales.Orders
where
       custid=@idclient
       and orderdate>=@start_date
       and orderdate<=@end_date
```

```
END
```

OUTPUT (OUT abreviere) parametri:

O procedura stocata poate avea si parametri de iesire, acestia fiind intotdeauna optionali.

```
IF OBJECT_ID('sp_getOrders', 'P') IS NOT NULL
DROP PROC sp_getOrders;
GO
create procedure sp_getOrders
       ( @idclient int
       , @start_date date='1900-01-01'
       , @end_date date='2999-12-31'
       , @no_rows int=0 OUT)
as
BEGIN
SET NOCOUNT ON;
SELECT
         orderid
       , orderdate
       , shippeddate
       , custid
from Sales.Orders
where
       custid=@idclient
       and orderdate>=@start_date
       and orderdate<=@end_date;
set @no_rows=@@ROWCOUNT
END
```

Observatii: @@ROWCOUNT este o variabila de sistem care stocheaza numarul de randuri dintr-o citire sau dintr-o actiune pe o tabela (update, delete, insert).

Executarea unei proceduri cu parametri de iesire (OUT/OUTPUT):

```
Use Training;
go
declare @numar_randuri as int
exec sp_getOrders
         @idclient=37
       , @start_date='2007-01-01'
       , @end_date='2007-12-31'
       , @no_rows=@numar_randuri OUTPUT
select @numar_randuri as Nr_Randuri
```

Results | Messages

	orderid	orderdate	shippeddate	custid
1	10429	2007-01-29 00:00:00.000	2007-02-07 00:00:00.000	37
2	10503	2007-04-11 00:00:00.000	2007-04-16 00:00:00.000	37
3	10516	2007-04-24 00:00:00.000	2007-05-01 00:00:00.000	37
4	10567	2007-06-12 00:00:00.000	2007-06-17 00:00:00.000	37
5	10646	2007-08-27 00:00:00.000	2007-09-03 00:00:00.000	37
6	10661	2007-09-09 00:00:00.000	2007-09-15 00:00:00.000	37
7	10687	2007-09-30 00:00:00.000	2007-10-30 00:00:00.000	37
8	10701	2007-10-13 00:00:00.000	2007-10-15 00:00:00.000	37
9	10712	2007-10-21 00:00:00.000	2007-10-31 00:00:00.000	37
10	10736	2007-11-11 00:00:00.000	2007-11-21 00:00:00.000	37

	Nr_Randuri
1	10

Exemple proceduri stocate:

1. Sa se creeze o procedura stocata care va determina pentru fiecare tara client, tara specificata la executarea procedurii:
    a. Numarul de comenzi
    b. Numarul de comenzi livrate
    c. Costul de transport
    d. Valoarea vanzarilor

```
use Training;
Go

create proc Sales.GetStatistics_Customer_Country
(@country_name varchar(100)=NULL)
as
begin
set nocount on;

IF OBJECT_ID('tempdb.dbo.#orders') IS NOT NULL
DROP table #orders;

IF OBJECT_ID('tempdbo..#sales') IS NOT NULL
DROP table #sales;

select
      c.country
    , count(o.orderid) Numar_Comenzi
    , count(o.shippeddate) Numar_Comenzi_livrate
     ,sum(o.freight) Cost_Transport
into #orders
from Sales.Customers c
inner join Sales.Orders o on c.custid=o.custid
where c.country=@country_name
group by
 c.country

select
      c.country
```

```
    , sum(od.qty*od.unitprice) Valoare_Vanzari
into #sales
from Sales.Customers c
inner join Sales.Orders o on c.custid=o.custid
inner join Sales.OrderDetails od on od.orderid=o.orderid
where c.country=@country_name
group by
    c.country

select
      o.country
    , o.Numar_Comenzi
    , o.Numar_Comenzi_livrate
    , o.Cost_Transport
    , s.Valoare_Vanzari
from #orders o
left join #sales s on o.country=s.country
end


exec Sales.GetStatistics_Customer_Country 'USA'
```

2. Sa se creeze o procedura stocata care va avea parametrii de perioada optionali. Ea va determina pentru fiecare angajat valoarea vanzarilor. Totodata, procedura stocata va returna si valoarea totala a vanzarilor pentru toti angajatii.

```
Use Training;
GO

create proc sales.GetSales_Employees
(@start_date date='1900-01-01', @end_date date='2999-12-31', @total_sales
decimal(18,2)=0 OUT)

as

begin
set nocount on;

select
    e.empid,
    e.firstname,
    e.lastname,
    sum(od.qty*od.unitprice) Sales
from hr.Employees e
inner join Sales.Orders o on e.empid=o.empid
inner join Sales.OrderDetails od on od.orderid=o.orderid
where o.orderdate>=@start_date and o.orderdate<=@end_date
group by
    e.empid,
    e.firstname,
    e.lastname

set @total_sales=(
                        select
                                    sum(od.qty*od.unitprice)
                        from Sales.Orders o
                        inner join Sales.OrderDetails od on od.orderid=o.orderid
                        where o.orderdate>=@start_date and o.orderdate<=@end_date
                        )
```

```
    end

    declare @ttl_sls as decimal(18,2)
    exec sales.GetSales_Employees '2007-01-01', '2007-12-31'
        , @total_sales=@ttl_sls OUTPUT
    select @ttl_sls as Total_Sales


declare @ttl_sls as decimal(18,2)
exec sales.GetSales_Employees
        @total_sales=@ttl_sls OUTPUT
select @ttl_sls as Total_Sales
```

3. Sa se creeze o procedura stocata ca va retura valoarea primei comenzi livrate a fiecarui client si valoarea totala a vanzarilor pentru fiecare client.

```
    Use Training;
    GO

    create procedure Get_Customer_First_TotalSales
    as
    begin

    IF OBJECT_ID('tempdbo..#min') IS NOT NULL
    DROP table #min;

    IF OBJECT_ID('tempdb..#first') IS NOT NULL
    DROP table #first;

    IF OBJECT_ID('tempdb..#total') IS NOT NULL

    select
        custid,
        min(orderid) Min_Order_Id
    into #min
    from Sales.Orders
    group by
        custid

    select
        m.custid,
        od.orderid,
        sum(od.qty*od.unitprice) First_Order_Value
    into #first
    from #min m
    inner join Sales.OrderDetails od on od.orderid=m.Min_Order_Id
    group by
        m.custid,
        od.orderid

    select
        o.custid,
        sum(od.qty*od.unitprice) Total_Sales
    into #total
    from Sales.Orders o
    inner join Sales.OrderDetails od on o.orderid=od.orderid
    group by
```

```
        o.custid

    select
        f.custid,
        c.companyname,
        c.contactname,
        f.First_Order_Value,
        t.Total_Sales
    from #first f
    inner join #total t on f.custid=t.custid
    inner join Sales.Customers c on f.custid=c.custid

    end

exec Get_Customer_First_TotalSales
```

4. Sa se creeze o procedura stocata care va crea tabela Raport_Vanzari_luna_anterioara de fiecare data cand va fi rulata. Tabela va avea coloanele:
    a. IdLuna de forma yyyymm
    b. Valoare_Vanzari

In tabela se va insera valoarea totala a vanzarilor din luna aterioara rularii, deci procedura va avea ca parametru luna curenta.

```
Use Training;
Go

create procedure Get_Raport_Vanzari_luna_anterioara
(@reporting_date date)
as
begin

IF OBJECT_ID('Raport_Vanzari_luna_anterioara', 'T') IS NOT NULL
DROP table Raport_Vanzari_luna_anterioara;

create table Raport_Vanzari_luna_anterioara
(IdLuna int not null,
Valoare_Vanzari decimal(18,2))

declare @IdMonth int
set @IdMonth=(select Year(dateadd(mm,-1,@reporting_date))*100+month(dateadd(mm,-
1,@reporting_date)))


declare @start_date datetime
set @start_date=dateadd(dd,1,eomonth(dateadd(mm,-2,@reporting_date)))

declare @end_date datetime
set @end_date=eomonth(dateadd(mm,-1,@reporting_date))

insert into Raport_Vanzari_luna_anterioara
(IdLuna, Valoare_Vanzari)
select
@IdMonth as IdLuna,
```

```
sum(od.qty*od.unitprice) Valoare_Vanzari
from Sales.Orders o
inner join Sales.OrderDetails od on o.orderid=od.orderid
where o.orderdate>=@start_date and o.orderdate<=@end_date

end


exec Get_Raport_Vanzari_luna_anterioara '2007-05-06'
select * from Raport_Vanzari_luna_anterioara
```

5. Sa se modifice procedura creata la punctul 4. astfel incat executarea procedurii sa insereze in tabela existenta Raport_Vanzari_luna_anterioara valoarea vanzarilor din luna anterioara rularii. In cazul in care o rulare s-a realizat de 2 ori, sa se stearga datele din tabela pentru prima rulare si sa se insereze din nou.

```
DROP table Raport_Vanzari_luna_anterioara

create table Raport_Vanzari_luna_anterioara
(IdLuna int not null,
Valoare_Vanzari decimal(18,2))



Use Training;
Go

alter procedure Get_Raport_Vanzari_luna_anterioara
(@reporting_date date)
as
begin

declare @IdMonth int
set @IdMonth=(select Year(dateadd(mm,-1,@reporting_date))*100+month(dateadd(mm,-
1,@reporting_date)))

declare @start_date datetime
set @start_date=dateadd(dd,1,eomonth(dateadd(mm,-2,@reporting_date)))

declare @end_date datetime
set @end_date=eomonth(dateadd(mm,-1,@reporting_date))

delete from Raport_Vanzari_luna_anterioara
where IdLuna=@IdMonth

insert into Raport_Vanzari_luna_anterioara
(IdLuna, Valoare_Vanzari)
select
@IdMonth as IdLuna,
sum(od.qty*od.unitprice) Valoare_Vanzari
from Sales.Orders o
inner join Sales.OrderDetails od on o.orderid=od.orderid
```

```
where o.orderdate>=@start_date and o.orderdate<=@end_date

end

exec Get_Raport_Vanzari_luna_anterioara '2007-05-06'
select * from Raport_Vanzari_luna_anterioara
```

Proceduri stocate imbricate

Exemplu:

- Sa se creeze o procedura stocata care va extrage un raport cu vanzarile din anul anterior rularii.
- Sa se creeze o procedura stocata care va determina cati client cumparatori au fost in anul anterior.
- Sa se creeze o procedura stocata master care va rula cele doua proceduri de mai sus.

```
use training;
go

create procedure getSales_Prev_Year
(@reporting_date date)
as

begin
select
sum(od.qty*od.unitprice) Sales_Prev_Year
from Sales.Orders o
inner join Sales.OrderDetails od on o.orderid=od.orderid
where year(o.orderdate)=year(@reporting_date)-1
end

use training;
go

create procedure getCust_Prev_Year
(@reporting_date date)
as

begin
select
count(distinct o.custid) Cust_Prev_Year
from Sales.Orders o
where year(o.orderdate)=year(@reporting_date)-1
end

use training;
go

alter procedure getMaster_Prev_Year
(@reporting_date date)
as
```

```
begin

exec getSales_Prev_Year @reporting_date
exec getCust_Prev_Year @reporting_date

end


exec getCust_Prev_Year '2008-05-07'
exec getSales_Prev_Year '2008-05-07'
exec getMaster_Prev_Year '2008-05-07'
```

PROIECT Proceduri stocate

Departamentul Sales de Business solicita departamentului de Data Warehouse un raport cu vanzarile detaliate la nivel de client (custid, companyname, country), categoria de produse (categoryid, categoryname) pentru o perioada mentionata (@startdate, @enddate). Rezultatul rularii raportului va fi stocata intr-o tabela creata one time, cu numele Sales_Report_by_Customer.

Mentiuni suplimentare:

- Pas 1: se va defini mai intai tabela Sales_report_by_Customer cu urmatoarele coloane:
    - Startdate datetime
    - Enddate datetime
    - Custid int
    - Companyname nvarchar(40)
    - Country nvarchar(15)
    - CategoryId int
    - Categoryname nvarchar(15)
    - Sales decimal(18,2)
- Pas 2: se va defini procedura stocata care:
    - In cazul in care a fost deja rulata, se vor sterge randurile din tabela (se verifica daca startdate din tabela si enddate din tabela coincide cu parametrii de rulare
    - Va insera datele in tabela Sales_report_by_customer

Rezolvare:

```
create table Sales.Sales_report_by_customer
(
```

```
Startdate date,
Enddate date,
Custid int,
Companyname nvarchar(40),
country nvarchar(15),
CategoryId int,
Categoryname nvarchar(15),
Sales decimal(18,2)
)

create procedure Sales.Get_Sales_report_by_customer
(@startdate datetime, @enddate datetime)
as
begin
             if
             (select count(*)
                from Sales.Sales_report_by_customer
                where startdate=@startdate and enddate=@enddate
             )>0
             begin
             delete  from Sales.Sales_report_by_customer
                     where startdate=@startdate and enddate=@enddate
             end
insert into sales.Sales_report_by_customer
(startdate, enddate, Custid, Companyname, country, CategoryId, Categoryname, sales)
select
       @startdate as startdate,
       @enddate as enddate,
       c.custid,
       c.companyname,
       c.country,
       cc.categoryid,
       cc.categoryname,
       sum(od.qty*od.unitprice) as sales
from sales.Customers c
inner join sales.Orders o on c.custid=o.custid
inner join sales.OrderDetails od on od.orderid=o.orderid
inner join Production.Products p on p.productid=od.productid
inner join Production.Categories cc on cc.categoryid=p.categoryid
where o.orderdate>=@startdate and o.orderdate<=@enddate
group by
       c.custid,
       c.companyname,
       c.country,
       cc.categoryid,
       cc.categoryname
end


exec Sales.Get_Sales_report_by_customer '2007-01-01','2007-12-31'
select * from Sales.Sales_report_by_customer
```

Results | Messages

	Startdate	Enddate	Custid	Companyname	country	CategoryId	Categoryname	Sales
1	2007-01-01 00:00:00.000	2007-12-31 00:00:00.000	1	Customer NRZBB	Germany	1	Beverages	648.00
2	2007-01-01 00:00:00.000	2007-12-31 00:00:00.000	2	Customer MLTDN	Mexico	1	Beverages	60.00
3	2007-01-01 00:00:00.000	2007-12-31 00:00:00.000	3	Customer KBUDE	Mexico	1	Beverages	1862.50
4	2007-01-01 00:00:00.000	2007-12-31 00:00:00.000	4	Customer HFBZG	UK	1	Beverages	1377.00
5	2007-01-01 00:00:00.000	2007-12-31 00:00:00.000	5	Customer HGVLZ	Sweden	1	Beverages	9100.05
6	2007-01-01 00:00:00.000	2007-12-31 00:00:00.000	6	Customer XHXJV	Germany	1	Beverages	342.00
7	2007-01-01 00:00:00.000	2007-12-31 00:00:00.000	7	Customer QXVLA	France	1	Beverages	3990.80
8	2007-01-01 00:00:00.000	2007-12-31 00:00:00.000	8	Customer QUHWH	Spain	1	Beverages	310.00
9	2007-01-01 00:00:00.000	2007-12-31 00:00:00.000	9	Customer RTXGC	France	1	Beverages	2365.00
10	2007-01-01 00:00:00.000	2007-12-31 00:00:00.000	10	Customer EEALV	Canada	1	Beverages	2066.50

IV. USER DEFINED FUNCTIONS

O functie definita de utilizator (User defined function sau udf) este o rutina incapsulata si reutilizabila care returneaza o valoare scalara sau un tabel.

Functiile definite de utilizator accepta parametri, la fel ca si procedurile stocate.

Functiile definite de utilizator utilizeaza datele dintr-o baza de date, insa nu si DDL-urile dintr-o baza de date, lucru facut de procedurile stocate.

Tipuri de functii definite de utilizator:

- Scalar
- Tabelare

Functiile de tip scalar returneaza o valoare catre utilizator.

Functiile de tip tabelar returneaza un tabel. Functiile de tip tabelar se impart in alte doua subcategorii:

- Inline table-valued function (IF in apelarea functiei OBJECT_ID)
- Table-valued function (TF in apelarea functiei OBJECT_ID)

Functiile de tip scalar au tipul FN pentru apelarea functiei OBJECT_ID.

Functii definite de utilizator de tip scalar

Exemplu: Sa se construiasa o functie care va returna cantitate ori pret.

```
create function udf_get_value
(@unitprice as decimal(19,4),
 @qty as smallint)
returns decimal(19,4)
as
       begin

       return @unitprice*@qty

       end;
```

Functia poate fi utilizata pentru a determina unitprice * qty din tabela Sales.OrderDetails.

```
select orderid, unitprice, qty, productid, dbo.udf_get_value(unitprice,qty) as sales
from sales.orderdetails;
```

Results | Messages

	orderid	unitprice	qty	productid	sales
1	10248	14.00	12	11	168.00
2	10248	9.80	10	42	98.00
3	10248	34.80	5	72	174.00
4	10249	18.60	9	14	167.40
5	10249	42.40	40	51	1696.00
6	10250	7.70	10	41	77.00
7	10250	42.40	35	51	1484.00
8	10250	16.80	15	65	252.00

```
select orderid, sum(dbo.udf_get_value(unitprice,qty)) as sales
from sales.orderdetails
group by orderid;
```

100 %

Results | Messages

	orderid	sales
1	10248	440.00
2	10249	1863.40
3	10250	1813.00

Exemplu: Sa se construiasca o functie care va determina ziua peste 5 ani pornind de la o data mentionata.

```
create function udf_get_dataafter5years
(
@data_bonus datetime
)
returns datetime
as
	begin
		return dateadd(yy,5,@data_bonus)
	end;
```

Functia poate fi utilizata pentru a determina data bonus pentru fiecare angajat, stiind ca fiecare angajat va primi bonusul la 5 ani de la angajare.

```
select empid, firstname, lastname, hiredate, dbo.udf_get_dataafter5years(hiredate) as
Data_Bonus
from hr.Employees;
```

Results | Messages

	empid	firstname	lastname	hiredate	Data_Bonus
1	1	Sara	Davis	2002-05-01 00:00:00.000	2007-05-01 00:00:00.000
2	2	Don	Funk	2002-08-14 00:00:00.000	2007-08-14 00:00:00.000
3	3	Judy	Lew	2002-04-01 00:00:00.000	2007-04-01 00:00:00.000
4	4	Yael	Peled	2003-05-03 00:00:00.000	2008-05-03 00:00:00.000
5	5	Sven	Buck	2003-10-17 00:00:00.000	2008-10-17 00:00:00.000
6	6	Paul	Suurs	2003-10-17 00:00:00.000	2008-10-17 00:00:00.000
7	7	Russell	King	2004-01-02 00:00:00.000	2009-01-02 00:00:00.000
8	8	Maria	Cameron	2004-03-05 00:00:00.000	2009-03-05 00:00:00.000
9	9	Zoya	Dolgopyatova	2004-11-15 00:00:00.000	2009-11-15 00:00:00.000

Exercitii:

1. Sa se defineasca o functie scalara care va determina daca diferenta dintre doua date calendaristice este mai mare de 10 zile. Rezultatul va fi true sau false (bit). Sa se utilizeze functia pentru a calcula diferenta dintre orderdate si shippeddate din tabela Sales.Orders si sa se evalueze daca perioada de livrare este peste 10 zile sau sub 10 zile.

Rezolvare:

```
create function udf_get_period
(
@startdate as datetime,
@enddate as datetime
)
returns bit
as
	begin
	declare @check bit

	set @check = case when datediff(dd,@startdate,@enddate)>10 then 1 else 0 end
	return (@check)
	end;

select
	orderid,
	orderdate,
	shippeddate,
	datediff(dd,orderdate,shippeddate) as dif,
	dbo.udf_get_period(orderdate,shippeddate) as flag
from sales.Orders
```

Results | Messages

	orderid	orderdate	shippeddate	dif	flag
1	10248	2006-07-04 00:00:00.000	2006-07-16 00:00:00.000	12	1
2	10249	2006-07-05 00:00:00.000	2006-07-10 00:00:00.000	5	0
3	10250	2006-07-08 00:00:00.000	2006-07-12 00:00:00.000	4	0
4	10251	2006-07-08 00:00:00.000	2006-07-15 00:00:00.000	7	0
5	10252	2006-07-09 00:00:00.000	2006-07-11 00:00:00.000	2	0
6	10253	2006-07-10 00:00:00.000	2006-07-16 00:00:00.000	6	0
7	10254	2006-07-11 00:00:00.000	2006-07-23 00:00:00.000	12	1

Functii definite de utilizator de tip tabelar

O functie de tip tabelar returneaza ca rezultat un tabel. Ea poate fi utilizata in cadrul clauzei FROM.

Inline table-valued functions

Functiile de tip tabelar cu o singura linie de cod se numesc inline table-valued functions. Functiile de tip inline table-valued **sunt singurele functii care pot sa nu contina blocul BEGIN/END.**

Functiile de tip inline table-valued contin un singur select care returneaza un tabel.

Exemplu: Sa se defineasca o functie care determina produsele vandute pentru o anumita categorie de produse mentionata.

```
create function udf_get_sales_category
(
@category_name varchar(100)
)
returns table as return
(select
       c.categoryid,
       c.categoryname,
       p.productid,
       p.productname,
       sum(od.qty*od.unitprice) as Sales
from sales.OrderDetails od
inner join Production.Products p on od.productid=p.productid
inner join Production.Categories c on c.categoryid=p.categoryid
where c.categoryname=@category_name
group by
       c.categoryid,
       c.categoryname,
       p.productid,
       p.productname
)

select *
from udf_get_sales_category('Beverages');
```

Results | Messages

	categoryid	categoryname	productid	productname	Sales
1	1	Beverages	1	Product HHYDP	14277.60
2	1	Beverages	2	Product RECZE	18559.20
3	1	Beverages	24	Product QOGNU	4782.60
4	1	Beverages	34	Product SWNJY	6678.00
5	1	Beverages	35	Product NEVTJ	14536.80
6	1	Beverages	38	Product QDOMO	149984.20

Exercitii:

1. Sa se defineasca o functie tabelara care sa determine incasarile per fiecare client pentru o tara client mentionata.

Rezolvare:

```
create function udf_sales_cust_country
(@country_name as varchar(100))
```

```
returns table as return
(select
       c.country,
       c.custid,
       c.companyname,
       cast(sum(od.qty*od.unitprice) as decimal(19,2)) as Sales
from Sales.customers c
inner join sales.Orders o on c.custid=o.custid
inner join sales.OrderDetails od on od.orderid=o.orderid
where c.country=@country_name
group by
       c.country,
       c.custid,
       c.companyname
)

select *
from udf_sales_cust_country('France')
```

	country	custid	companyname	Sales
1	France	7	Customer QXVLA	19088.00
2	France	9	Customer RTXGC	23850.95
3	France	18	Customer BSVAR	1615.90
4	France	23	Customer WVFAF	11666.90
5	France	26	Customer USDBG	3172.16
6	France	40	Customer EFFTC	1992.05
7	France	41	Customer XIIWM	10272.35
8	France	74	Customer YSHXL	2423.35
9	France	84	Customer NRCSK	9937.10
10	France	85	Customer ENQZT	1480.00

Table-valued functions

Functiile de tip table-valued contin obligatoriu blocul BEGIN/END si au ca rezultat un tabel, iar tabelul trebuie declarant, spre deosebire de inline table-valued functions.

O functie de tip table-valued contine multiple linii de cod.

Exemplu: Sa se determine intr-un tabel tarile clientilor si tarile angajatilor mentionand in dreptul fiecarei tari daca provine de la angajat sau de la client. Functia va rula cu parametrul country_name si va returna daca o tara apartine clientilor si / sau angajatilor.

```
USE Training
GO

create FUNCTION UDF_getCountries
( @country_name as nvarchar(100))
RETURNS @Countries TABLE
(
       Country_name NVARCHAR (100),
       Type NVARCHAR (20)
)
AS
BEGIN
              INSERT INTO @Countries
```

```
                SELECT distinct Country, 'Employee' as Type
                FROM HR.Employees
                where country=@country_name


        INSERT INTO @Countries
                SELECT distinct Country, 'Customers' as Type
                FROM Sales.Customers
                where country=@country_name

        RETURN
END

select * from UDF_getCountries ('USA')
```

	Country_name	Type
1	USA	Employee
2	USA	Customers

Exemplu: Sa se modifice functia de mai sus astfel incat sa nu primeasca parametru si va avea ca rezultat toate tarile. In cazul in care se doreste sa se filtreze tabela, atunci conditia se va insera in clauza WHERE la rularea functiei.

```
USE Training
GO

alter FUNCTION UDF_getCountries ()
RETURNS @Countries TABLE
(
        Country_name NVARCHAR (100),
        Type NVARCHAR (20)
)
AS
BEGIN
        INSERT INTO @Countries
                SELECT distinct Country, 'Employee' as Type
                FROM HR.Employees


        INSERT INTO @Countries
                SELECT distinct Country, 'Customers' as Type
                FROM Sales.Customers


        RETURN
END

select * from UDF_getCountries ()
where Country_name='Italy'
```

Results | Messages

	Country_name	Type
1	Italy	Customers

Exercitii:

1. Sa se construiasca o functie care va returna pentru un client suma vanzarilor, stiindu-se clientul.
2. Sa se construiasca o functie care va returna suma totala a incasarilor pentru o perioada de timp definita de utilizator prin start_date si end_date
3. Sa se defineasca o functie care va determina pentru un anumit client : suma cantitatilor si suma incasarilor pentru o anumita perioada de timp definita de utilizator prin start_date si end_date.
4. Sa se defineasca o functie care va determina la nivel de o anumita tara, definite de utilizator, numarul de clienti distincti, numarul de comenzi, suma incasarilor pe o anumita perioada definita de utilizator prin start_date si end_date.
5. Sa se creeze o functie care va determina la nivel de oras, numarul de clienti distincti, numarul de facturi si suma cantitatilor pe o anumita perioada definita de utilizator prin start_date si end_date.
6. Sa se construiasca o functie care va determina pentru fiecare firma de livrare numarul de clienti distincti si cantitatile totale livrate pe o anumita perioada definite de utilizator prin start_date si end_date.

Rezolvare:

```
/*
	1.	Sa se construiasca o functie care va returna pentru un client suma
vanzarilor, stiindu-se clientul.
*/
-- functie scalara
create function dbo.udf_ValoareClient(@custid int)
returns decimal(18,4)
as
begin
return (
	select sum(od.qty * od.unitprice) Valoare
	from Sales.OrderDetails od
	join Sales.Orders o on od.orderid = o.orderid
	join Sales.Customers c on c.custid = o.custid
	where c.custid = @custid
	)
end

select dbo.udf_ValoareClient(37)

/*
	2.	Sa se construiasca o functie care va returna suma totala a  incasarilor
pentru o perioada de timp
	definita de utilizator prin start_date si end_date
*/
-- functie scalara
create function dbo.udf_ValoareTotala
```

```
(@start_date date, @end_date date)
returns decimal(18,4)
as
begin
	return(
		select sum(od.qty * od.unitprice) Valoare
		from Sales.OrderDetails od
		join Sales.Orders o on od.orderid = o.orderid
		where o.orderdate >= @start_date and o.orderdate <= @end_date
		)
end

select dbo.udf_ValoareTotala('2008-01-01','2008-04-01')

/*
	3.	Sa se defineasca o functie care va determina pentru un anumit client :
suma cantitatilor si suma incasarilor pentru o anumita
	perioada de timp definita de utilizator prin start_date si end_date.
*/
-- functie tabelara
create function dbo.udf_ValoareTotalaClient(@custid int, @start_date date, @end_date
date)
returns table
as
return (
	select c.custid
	,sum(od.qty) Cantitate
	,sum(od.qty * od.unitprice) Valoare
	from Sales.OrderDetails od
	join Sales.Orders o on od.orderid = o.orderid
	join Sales.Customers c on c.custid = o.custid
	where c.custid = @custid and o.orderdate >= @start_date and o.orderdate <=
@end_date
	group by c.custid
)

select *
from dbo.udf_ValoareTotalaClient(37,'2001-01-01','2010-01-01')

/*
	4.	Sa se defineasca o functie care va determina la nivel de o anumita tara,
definite de utilizator,
	numarul de clienti distincti, numarul de comenzi, suma incasarilor pe o anumita
perioada definita de utilizator
	prin start_date si end_date.
*/

create function dbo.udf_ValoareTotalaTara(@country varchar(40), @start_date date,
@end_date date)
returns table
as
return (
	select c.country
	,count(distinct c.custid) Nr_Clienti
	,count(distinct o.orderid) Nr_Comenzi
	,sum(od.qty) Cantitate
	from Sales.OrderDetails od
	join Sales.Orders o on od.orderid = o.orderid
	join Sales.Customers c on c.custid = o.custid
	where c.country = @country and o.orderdate >= @start_date and o.orderdate <=
@end_date
```

```
	group by c.country
)

select *
from dbo.udf_ValoareTotalaTara('USA','2001-01-01','2010-01-01')

/*
	5.	Sa se creeze o functie care va determina la nivel de oras, numarul de
clienti distincti,
	numarul de facturi si suma cantitatilor pe o anumita perioada definita de
utilizator prin start_date si end_date.
*/

create function dbo.udf_ValoareTotalaOras(@start_date date, @end_date date)
returns table
as
return (
	select c.city
	,count(distinct c.custid) Nr_Clienti
	,count(distinct o.orderid) Nr_Comenzi
	,sum(od.qty) Cantitate
	from Sales.OrderDetails od
	join Sales.Orders o on od.orderid = o.orderid
	join Sales.Customers c on c.custid = o.custid
	where o.orderdate >= @start_date and o.orderdate <= @end_date
	group by c.city
)

select *
from dbo.udf_ValoareTotalaOras('2001-01-01','2010-01-01')

/*
	6.	Sa se construiasca o functie care va determina pentru fiecare firma de
livrare numarul de clienti distincti si cantitatile
	totale livrate pe o anumita perioada definite de utilizator prin start_date si
end_date.
*/

create function dbo.udf_ValoareTotalaLivrare(@start_date date, @end_date date)
returns table
as
return (
	select s.companyname
	,count(distinct c.custid) Nr_Clienti
	,count(distinct o.orderid) Nr_Comenzi
	,sum(od.qty) Cantitate
	from Sales.OrderDetails od
	join Sales.Orders o on od.orderid = o.orderid
	join Sales.Customers c on c.custid = o.custid
	join Sales.Shippers s on s.shipperid = o.shipperid
	where o.orderdate >= @start_date and o.orderdate <= @end_date
	group by s.companyname
)

select *
from dbo.udf_ValoareTotalaLivrare('2001-01-01','2010-01-01')
```

V. CONTROL FLOW STATEMENTS: IF/ELSE, WHILE

T-SQL ofera cateva sintaxe pentru a controla executia unui cod. In general, acestea sunt utilizate in cadrul procedurilor stocate. In acest caz, codul inglobat in cadrul procedurii stocate poate fi gestionat de blocuri de control.

Dintre acestea, amintim:

- IF/ELSE
- WHILE

Control flow-ul if/else

Exemplu:

- Sa se creeze o procedura stocata (sp_ReportCustomersYesterday) care va extrage la nivel de fiecare client:
    - Numarul de comenzi plasate ieri
    - Valoarea incasarilor pentru comenzile plasate ieri
- Daca numarul de clienti care au plasat comenzi ieri este diferit de NULL (adica sunt clienti care au plasat comenzi) atunci sa se execute procedura sp_ReportCustomersYesterday.

Rezolvare:

Creare procedura stocata: sp_ReportCustomersYesterday

```
use training;
go

create procedure sp_ReportCustomersYesterday
as
begin
select
c.custid as Customer_Number,
c.companyname as Customer_Name,
count(distinct o.orderid) as No_of_Orders,
sum(od.qty*od.unitprice) as Total_Amount
from Sales.Customers c
inner join Sales.Orders o on c.custid=o.custid
inner join Sales.OrderDetails od on o.orderid=od.orderid
where orderdate=dateadd(dd,-1,cast(cast(getdate() as date) as datetime))
group by
c.custid,
c.companyname
end
```

Verificarea numarului de clienti si executarea procedurii stocate sp_ReportCustomersYesterday:

```
Use Training;
go

if (select count(distinct custid)
       from sales.Orders
       where orderdate=dateadd(dd,-1,cast(cast(getdate() as date) as datetime))
       )>0
exec sp_ReportCustomersYesterday
else print 'No orders yesterday'
```

sau

```
if (select count(distinct custid)
       from sales.Orders
       where orderdate=dateadd(dd,-1,cast(cast(getdate() as date) as datetime))
       )>0
BEGIN
exec sp_ReportCustomersYesterday
END
else print 'No orders yesterday'
```

Nota: Blocul Begin/End nu este obligatoriu. Necesitatea lui apare atunci cand sunt mai multe statementuri de executat in cazul in care conditia este adevarata.

Daca blocul Begin/End este omis in cazul mai multor statementuri, atunci va fi executat doar primul statement.

Control flow-ul while

Exemplu:

- Se presupune ca business-ul a solicitat un raport lunar cu incasarile la nivel de tara client. Raportul este nou si se doreste a fi extras si din urma, pentru 2007. Raportul va fi incarcat intr-o tabela din baza de date pentru fiecare luna incheiata. Tabela are 3 coloane: country, month_id, sales.
- Se va creea o procedura stocata cu incasarile per luna (luna data de utilizator ca parametru, de tipul yyyymm)
- Procedura stocata va fi rulata pornind din ianuarie 2007 pana in decembrie 2007 utilizand control flow-ul while.

Rezolvare:

1) Se defineste tabela in care se vor incarca datele.

```
CREATE TABLE [Sales].[countries_sales](
       [country] [nvarchar](15) NOT NULL,
       [month_id] [int] NULL,
       [sales] [money] NULL
)
```

2) Se defineste procedura de incarcare date

```
CREATE procedure [Sales].[sp_MonthlySales]
(@reporting_month int)

--declare @reporting_month int --200701
--set @reporting_month=200801

as
insert into sales.countries_sales
select
       c.country,
       @reporting_month as month_id,
       sum(od.qty*od.unitprice) as sales
--into sales.countries_sales
from sales.customers as c
inner join sales.orders o on c.custid=o.custid
inner join sales.orderdetails od on od.orderid=o.orderid
where year(o.orderdate)*100+month(o.orderdate)=@reporting_month
group by
       c.country
```

3) Se executa procedura pentru o luna

```
EXEC [Sales].[sp_MonthlySales] 200801
```

4) Se executa procedura in interiorul control flow-ului while

```
Use Training;
Go


declare @reportingMonth_start int
set @reportingMonth_start=200701

declare @reportingMonth_end int
set @reportingMonth_end=200712

begin

       while @reportingMonth_start<=@reportingMonth_end
       begin
       exec sp_MonthlySales @reportingMonth_start
       set @reportingMonth_start=@reportingMonth_start+1
       end

end
```

Control flow-uri while imbricate

Exemplu:

- Se presupune ca business-ul a solicitat un raport lunar cu incasarile la nivel de tara client. Raportul este nou si se doreste a fi extras si din urma, pentru 2007 si 2008. Raportul va fi incarcat intr-o tabela din baza de date pentru fiecare luna incheiata.
- Se va creea o procedura stocata cu incasarile per luna (luna data de utilizator ca parametru)
- Procedura stocata va fi rulata pornind din ianuarie 2007 pana in decembrie 2008 utilizand control flow-ul while.

Rezolvare:

```
Use Training;
Go


-- truncate table sales.countries_sales


declare @reportingYear_start int
set @reportingYear_start=2007

declare @reportingMonth_start int
set @reportingMonth_start=1

declare @reportingYear_end int
set @reportingYear_end=2008

declare @reportingMonth_end int
set @reportingMonth_end=12

begin
while @reportingYear_start<=@reportingYear_end
begin

       while @reportingMonth_start<=@reportingMonth_end

       begin
       declare @reportingMonth int
       set @reportingMonth=@reportingYear_start*100+@reportingMonth_start

       exec Sales.sp_MonthlySales @reportingMonth
       set @reportingMonth_start=@reportingMonth_start+1
       end

set @reportingYear_start=@reportingYear_start+1
set @reportingMonth_start=1
end
end
```

VI. MERGE STATEMENT. OUTPUT

Merge simplu

Merge Statement este utilizata in multiple cazuri:

- Inserarea zilnica dintr-o tabela sursa intr-o tabela target a randurilor noi
- Updatarea zilnica a unei tabele target comparand cu tabela sursa
- Stergerea zilnica a randurilor din tabela target care nu se regasesc in tabela sursa

In general, utilizarea lui MERGE statement presupune un proces de actualizare a unei baze de date comparand cu importurile zilnice din diverse surse.

In acelasi timp, MERGE statement poate fi utilizata si pentru tabele target agregate avand ca surse tabele neagregate.

Exemplu:

Pas 1: tabelul sursa are date, tabelul target este gol

sursa

custid	name
1	test 1
2	test 2

target

custid	name

Pas 2: dupa merge se actualizeaza tabelul target

sursa

custid	name
1	test 1
2	test 2

target

custid	name
1	test 1
2	test 2

Pas 3: se modifica numele clientului 2 in tabelul sursa, devine test 22

sursa

custid	name
1	test 1
2	test 22

dupa merge:

sursa

custid	name

target

custid	name

1	test 1
2	test 22

1	test 1
2	test 22

Pas 4: se insereaza un client nou in sursa: 3/ test 3.

sursa

custid	name
1	test 1
2	test 22
3	test 3

target

custid	name
1	test 1
2	test 22

Dupa merge:

sursa

custid	name
1	test 1
2	test 22
3	test 3

target

custid	name
1	test 1
2	test 22
3	test 3

Pas 5: se sterge clientul 1 din tabelul sursa

sursa

custid	name
2	test 22
3	test 3

target

custid	name
1	test 1
2	test 22
3	test 3

Dupa merge:

sursa

custid	name
2	test 22
3	test 3

target

custid	name
2	test 22
3	test 3

Sintaxa MERGE:

MERGE INTO target_table as TGT

USING source_table as SRC

ON TGT.primary_key=SRC.primary_key (predicatul)

WHEN MATCHED (AND diferite conditii suplimentare optionale)

THEN (UPDATE and DELETE) – sunt modificari pe randurile existente

WHEN NOT MATCHED BY TARGET (AND diferite conditii suplimentare optionale)

THEN (INSERT) – sunt randuri noi in sursa, care nu sunt in target

WHEN NOT MATCHED BY SOURCE (AND diferite conditii suplimentare optionale)

THEN (UPDATE and DELETE) – sunt modificari pe randurile existente

Obs:

- Target_table este tabela care trebuie mentinuta actualizata
- Source_table poate fi un SELECT care returneaza un tabel, poate fi un CTE (common table expression), poate fi o tabela temporara construita anterior sau o tabela permanenta.

Exemplu: Se considera tabela Sales.Orders careia i se face o copie cu numele Sales.OrdersTGT fara a avea date si inca o copie cu numele Sales.OrdersSRC cu date. In fiecare zi, in tabela Sales.OrdersSRC se intampla modificari:

- Randuri noi cu comenzi noi
- Modificate comenzile anterioare
- Sterse comenzi

Tabela Sales.OrdersTGT trebuie mentinuta in mod identic cu tabela Sales.OrdersSRC.

Pas 1: creare copii: Sales.OrdersTGT si Sales.OrdersSRC

```
select orderid, empid, freight, shippeddate
into Sales.OrdersTGT
from Sales.Orders
where 1=0

-- copia Sales.OrdersTGT nu va avea si PK

alter table Sales.OrdersTGT
add constraint PK_OrdersTGT primary key (orderid)


select orderid, empid, freight, shippeddate
into Sales.OrdersSRC
from Sales.Orders


-- copia Sales.OrdersSRC nu va avea si PK

alter table Sales.OrdersSRC
add constraint PK_OrdersSRC primary key (orderid)
```

Pas 2: construirea MERGE statement astfel incat:

- Sa transfere toate randurile din Sales.OrdersSRC in Sales.OrdersTGT prima data
- Sa se actualizeze modificarile pe coloanele empid, shippeddate, freight.
- Sa se actualizeze comenzile noi inserate in Sales.OrdersSRC
- Sa se actualizeze comenzile sterse din tabela Sales.OrdersSRC

```
   SET IDENTITY_INSERT Sales.OrdersTGT on; -- pentru a se putea face inserturi in
   coloana OrderId care este IDENTITY(1,1)
   GO

   MERGE INTO Sales.OrdersTGT as TGT
   USING Sales.OrdersSRC as SRC
      ON TGT.orderid=SRC.orderid
   WHEN MATCHED and (TGT.empid<>SRC.empid or TGT.shippeddate<>SRC.shippeddate or
   TGT.freight<>SRC.freight)
   THEN update
      set TGT.empid=SRC.empid,
             TGT.freight=SRC.freight,
             TGT.shippeddate=SRC.shippeddate
   WHEN NOT MATCHED BY TARGET
   THEN insert (orderid, empid, freight, shippeddate)
   values (SRC.orderid, SRC.empid, SRC.freight, SRC.shippeddate)
   WHEN NOT MATCHED BY SOURCE
   THEN delete;
SET IDENTITY_INSERT Sales.OrdersTGT off;
```

Verificare date in tabela target:

```
select * from sales.OrdersTGT
```

830 rows

Pas 3:

- Randuri noi in tabela sursa Sales.OrdersSRC

```
   SET IDENTITY_INSERT Sales.OrdersSRC on; -- pentru a se putea face inserturi in
   coloana OrderId care este IDENTITY(1,1)
   GO
   insert into Sales.OrdersSRC (orderid, empid, freight, shippeddate)
    values (1000,5, 100, '2019-04-03');
   SET IDENTITY_INSERT Sales.OrdersSRC off; -- pentru a se putea face inserturi in
   coloana OrderId care este IDENTITY(1,1)
```

- Rulare MERGE pentru a insera in tabela target randul ordered=1000 din sursa

```
   GO
    SET IDENTITY_INSERT Sales.OrdersTGT ON; -- pentru a se putea face inserturi in
   coloana OrderId care este IDENTITY(1,1)
   GO
    MERGE INTO Sales.OrdersTGT as TGT
   USING Sales.OrdersSRC as SRC
      ON TGT.orderid=SRC.orderid
```

```
    WHEN MATCHED and (TGT.empid<>SRC.empid or TGT.shippeddate<>SRC.shippeddate or
    TGT.freight<>SRC.freight)
    THEN update
        set TGT.empid=SRC.empid,
               TGT.freight=SRC.freight,
               TGT.shippeddate=SRC.shippeddate
    WHEN NOT MATCHED BY TARGET
    THEN insert (orderid, empid, freight, shippeddate)
    values (SRC.orderid, SRC.empid, SRC.freight, SRC.shippeddate)
    WHEN NOT MATCHED BY SOURCE
    THEN delete;
SET IDENTITY_INSERT Sales.OrdersTGT OFF;

select * from Sales.OrdersTGT
```

orderid empid freight shippeddate

1000 5 100.00 2019-04-03 00:00:00.000

Pas 4:

- modificari pe randuri existente in tabela sursa

```
update Sales.OrdersSRC
set empid=6
where orderid=1000;
```

- rulare MERGE pt a actualize tabela target

```
    MERGE INTO Sales.OrdersTGT as TGT
    USING Sales.OrdersSRC as SRC
        ON TGT.orderid=SRC.orderid
    WHEN MATCHED and (TGT.empid<>SRC.empid or TGT.shippeddate<>SRC.shippeddate or
    TGT.freight<>SRC.freight)
    THEN update
        set TGT.empid=SRC.empid,
               TGT.freight=SRC.freight,
               TGT.shippeddate=SRC.shippeddate
    WHEN NOT MATCHED BY TARGET
    THEN insert (orderid, empid, freight, shippeddate)
    values (SRC.orderid, SRC.empid, SRC.freight, SRC.shippeddate)
    WHEN NOT MATCHED BY SOURCE
    THEN delete;


select * from Sales.OrdersTGT
```

orderid empid freight shippeddate

1000 6 100.00 2019-04-03 00:00:00.000

Pas 5:

- stergere randuri din tabela sursa

```
    delete from Sales.OrdersSRC
    where orderid=1000
```

- rulare MERGE statement pentru a actualize tabela target

```
MERGE INTO Sales.OrdersTGT as TGT
USING Sales.OrdersSRC as SRC
    ON TGT.orderid=SRC.orderid
WHEN MATCHED and (TGT.empid<>SRC.empid or TGT.shippeddate<>SRC.shippeddate or
TGT.freight<>SRC.freight)
THEN update
    set TGT.empid=SRC.empid,
          TGT.freight=SRC.freight,
          TGT.shippeddate=SRC.shippeddate
WHEN NOT MATCHED BY TARGET
THEN insert (orderid, empid, freight, shippeddate)
values (SRC.orderid, SRC.empid, SRC.freight, SRC.shippeddate)
WHEN NOT MATCHED BY SOURCE
THEN delete;
```

```
select count(*) from Sales.OrdersTGT;
```

830 randuri

Indicat este ca intreg merge-ul sa fie incapsulat intr-o procedura stocata.

```
create procedure sales.sp_merge_ordersTGT
as
SET IDENTITY_INSERT Sales.OrdersTGT ON;
MERGE INTO Sales.OrdersTGT as TGT
USING Sales.OrdersSRC as SRC
       ON TGT.orderid=SRC.orderid
WHEN MATCHED and (TGT.empid<>SRC.empid or TGT.shippeddate<>SRC.shippeddate or
TGT.freight<>SRC.freight)
THEN update
       set TGT.empid=SRC.empid,
              TGT.freight=SRC.freight,
              TGT.shippeddate=SRC.shippeddate
WHEN NOT MATCHED BY TARGET
THEN insert (orderid, empid, freight, shippeddate)
values (SRC.orderid, SRC.empid, SRC.freight, SRC.shippeddate)
WHEN NOT MATCHED BY SOURCE
THEN delete;
SET IDENTITY_INSERT Sales.OrdersTGT OFF;
```

Merge cu OUTPUT

Exemplu: Se considera tabela Sales.Orders careia i se face o copie cu numele Sales.OrdersTGTHIST (va avea inregistrarile istorizate) si inca o copie cu numele Sales.OrdersSRCINI (va avea inregistrarile asa cum vin din sursa zilnic, mai intai full si apoi zilnic delta = doar comenzile noi, modificarile pe comenzi vechi, nu se sterg comenzi)

In tabela Sales.OrdersTGTHIST se vor adauga 2 coloane suplimentare care asigura istorizarea datelor: Valid_From (data type = date) si Valid_To (data type = date).

Din punct de vedere SCD (Slowly Changing Dimension), este SCD de tip 2: se cunoaste imaginea anterioara de la data x la data y si imaginea curenta de la data y la infinit.

Pas 1:

- creare tabele copie – tabela target va fi goala

```
select orderid, empid, freight, shippeddate
into Sales.OrdersTGTHIST
from Sales.Orders
where 1=0

alter table Sales.OrdersTGTHIST
add Valid_From date not null;

alter table Sales.OrdersTGTHIST
add Valid_To date not null;


alter table Sales.OrdersTGTHIST
add constraint PK_OrdersTGTHIST primary key (orderid, Valid_From)
```

Obs: PK-ul din tabela target va fi formata din orderid si Valid_from pt ca un orderid poate aparea de mai multe ori din cauza modificarilor de la o zi la alta.

Tabela Target va avea datele istorizate – se va sti valabilitatea fiecarui rand prin intervalul valid_from, valid_to.

Tabela sursa va fi delta: randuri noi sau randuri modificate in fiecare zi.

```
select orderid, empid, freight, shippeddate
into Sales.OrdersSRCINI
from Sales.Orders
```

Initial, tabela sursa va contine toate informatiile (primul full), iar din a doua zi, ea vine delta.

```
alter table Sales.OrdersSRCINI
add constraint PK_OrdersSRCINI primary key (orderid)
```

Pas 2:

- construire MERGE statement astfel incat:
  - update Valid_To pe randuri existente inchizand randul in cazul in care se modifica in tabela sursa.
  - insert randuri noi si Valid_From va fi data curenta, iar Valid_to va fi 9999-12-31, aceasta valoare aratand ca aceasta inregistrare este inca valida, are valabilitate infinit.

```
    --drop table #tbl
    create table #tbl
    (action varchar(100),
    orderid int,
    empid int,
    freight money,
    shippeddate datetime)

SET IDENTITY_INSERT Sales.OrdersSRCINI ON; -- pentru a se putea face inserturi in
coloana OrderId care este IDENTITY(1,1)
GO
SET IDENTITY_INSERT Sales.OrdersTGTHIST ON; -- pentru a se putea face inserturi in
coloana OrderId care este IDENTITY(1,1)
GO


declare @stop_valid_to date
set @stop_valid_to=dateadd(dd,0,cast(getdate() as date))

declare @new_valid_from date
set @new_valid_from=dateadd(dd,1,cast(getdate() as date))


set identity_insert tg2 on;

merge into Sales.OrdersTGTHIST as tg2
using Sales.OrdersSRCINI as src2 on tg2.orderid=src2.orderid and tg2.valid_to='9999-
12-31'

when matched and (tg2.empid<>src2.empid or tg2.freight<>src2.freight)
       then update
              set
                     tg2.valid_to=@stop_valid_to
when not matched by target
       then insert (orderid, empid, freight, valid_from, valid_to)
       values (src2.orderid, src2.empid, src2.freight, @new_valid_from, '9999-12-31')
output
       $action as action_type,
       deleted.orderid, deleted.empid, deleted.freight
       into #tbl;

insert into tg2 (orderid, empid, freight, valid_from, valid_to)
select src2.orderid, src2.empid, src2.freight, @new_valid_from as valid_from, '9999-
12-31' as valid_to
from src2
inner join #tbl t on src2.orderid=t.orderid
where t.action_type in ('update')

set identity_insert tg2 off;
```

Nota: $action poate lua una dintre cele 3 valori:

- INSERT (randuri noi)
- UPDATE (randuri modificate)
- DELETE (randuri sterese)

in functie de ceea ce se intampla in MERGE.

Nota:

- Inserted: reflecta randurile noi sau updatate
- Deleted: reflecta randurile sterse sau updatate

Obs: In aceasta maniera:

- au fost inchise randurile vechi
- au fost inserate randurile vechi cu valori noi

Exercitii:

1. Sa se creeze doua copii pentru tabela Sales.Customers avand coloanele: custid, companyname, country. Fiecare tabela copie nu va avea initial date. Cele doua tabele copie se vor numi: Sales.CustSRC si Sales.CustTGT. Tabela sursa va veni full in fiecare zi.

- Sa se insereze in tabela Sales.CustSRC (atentie la coloana custid care este identity) urmatorii clienti:
    - 8001, Test1, Romania
    - 8002, Test2, Romania
    - 8003, Test3, Romania
- Sa se construiasca MERGE astfel incat:
    - Orice client nou venit in tabela sursa sa fie inserat in tabela target
    - Orice modificare venita pe un client in tabela sursa se se propage in tabela target
    - ! Daca se vor sterge clienti din tabela sursa, acestia sa nu se stearga din tabela target (in cadrul lui MERGE, nu se va trata WHEN NOT MATCHED BY SOURCE).

Rezolvare:

Pas 1: se vor define cele doua tabele: target si sursa, ambele vor fi goale. Se va stabili PK pentru fiecare tabela.

```
Use Training;
GO

select
       custid,
       companyname,
       country
into Sales.CustSRC
```

```
from sales.Customers
where 1=0

alter table Sales.CustSRC
add constraint PK_custSRC primary key (custid)


select
       custid,
       companyname,
       country
into Sales.CustTGT
from sales.Customers
where 1=0

alter table Sales.CustTGT
add constraint PK_custTGT primary key (custid)
```

Pas 2: se vor insera cei 3 clienti in tabela sursa tratand identity_insert.

```
set identity_insert Sales.CustSRC on;
insert into Sales.CustSRC (custid, companyname, country) values
(8001,'Test1','Romania')
insert into Sales.CustSRC (custid, companyname, country) values
(8002,'Test2','Romania')
insert into Sales.CustSRC (custid, companyname, country) values
(8003,'Test3','Romania')
set identity_insert Sales.CustSRC off;
```

Pas 3: se va construi MERGE astfel incat:

- When matched – sa faca update pe randurile existente
- When not matched by target – sa faca insert in tabela target
- When not matched by source – nu se va trata

```
set identity_insert Sales.CustTGT on;
merge Sales.CustTGT tg
using Sales.CustSRC sr on tg.custid=sr.custid

when matched and (tg.companyname<>sr.companyname or tg.country<>sr.country)
then update
    set tg.companyname=sr.companyname,
           tg.country=sr.country
when not matched by target
then insert (custid, companyname, country)
     values (sr.custid, sr.companyname, sr.country);
set identity_insert Sales.CustTGT off;
```

Pentru a rula mai usor MERGE-ul, acesta va fi incapsulat intr-o procedura stocata.

```
create procedure Sales.sp_MERGE_Cust
as
set identity_insert Sales.CustTGT on;
```

```
merge Sales.CustTGT tg
using Sales.CustSRC sr on tg.custid=sr.custid

when matched and (tg.companyname<>sr.companyname or tg.country<>sr.country)
then update
       set tg.companyname=sr.companyname,
              tg.country=sr.country
when not matched by target
then insert (custid, companyname, country)
        values (sr.custid, sr.companyname, sr.country);
set identity_insert Sales.CustTGT off;
```

Sa se testeze MERGE-ul de mai sus luand in considerare toate scenariile mentionate.

Scenariul 1: vin modificari in sursa pe cei 3 clienti

```
update sales.CustSRC
       set companyname='Test 1 modificat'
where custid=8001;
```

```
select * from sales.CustSRC
```

	custid	companyname	country
1	8001	Test 1 modificat	Romania
2	8002	Test2	Romania
3	8003	Test3	Romania

Se va rula procedura Sales.sp_MERGE_cust pentru a testa daca modificarea pe clientul 8001 se va propaga in tabela target.

```
exec Sales.sp_MERGE_Cust
```

```
select * from sales.CustTGT
```

	custid	companyname	country
1	8001	Test 1 modificat	Romania
2	8002	Test2	Romania
3	8003	Test3	Romania

Scenariul 2: vin clienti noi in sursa

```
set identity_insert Sales.CustSRC on;
insert into Sales.CustSRC (custid, companyname, country) values
(8004,'Test4','Romania')
set identity_insert Sales.CustSRC off;
```

```
select * from sales.CustSRC
```

	custid	companyname	country
1	8001	Test 1 modificat	Romania
2	8002	Test2	Romania
3	8003	Test3	Romania
4	8004	Test4	Romania

Se va rula procedura Sales.sp_MERGE_cust pentru a verifica daca noul client din sursa va fi inserat in tabela target.

```
exec Sales.sp_MERGE_Cust

select * from sales.CustTGT
```

	custid	companyname	country
1	8001	Test 1 modificat	Romania
2	8002	Test2	Romania
3	8003	Test3	Romania
4	8004	Test4	Romania

Scenariul 3: se sterge un client din sursa

```
delete from sales.CustSRC
where custid=8002

select * from sales.CustSRC
```

	custid	companyname	country
1	8001	Test 1 modificat	Romania
2	8003	Test3	Romania
3	8004	Test4	Romania

Se va rula procedura pentru a testa daca randul sters din tabela sursa va fi sters si din tabela target.

```
exec Sales.sp_MERGE_Cust

(0 row(s) affected)
```

Mesajul de rulare al procedurii returneaza 0 randuri, cee ace inseamna ca nu a fost sters clientul 8002 din tabela target, cee ace este corect, conform cerintei.

```
select * from sales.CustTGT
```

	custid	companyname	country
1	8001	Test 1 modificat	Romania
2	8002	Test2	Romania
3	8003	Test3	Romania
4	8004	Test4	Romania

2. Sa se creeze doua copii pentru tabela Sales.Customers avand coloanele: cusid, companyname, country. Fiecare tabela copie nu va avea initial date. Cele doua tabele copie se vor numi: Sales.CustSRC2 si Sales.CustTGT2

- Sa se insereze in tabela Sales.CustSRC2 (atentie la coloana custid care este identity) urmatorii client:
    - 8001, Test1, Romania
    - 8002, Test2, Romania
    - 8003, Test3, Romania

Obs: Tabela Sales.CustSRC2 vine zilnic full. In cazul in care anumiti clienti nu mai vin in tabela Sales.CustSRC2, atunci:

- Sa se transfere initial toti clientii in tabela Sales.CustTGT2 (nu este necesar MERGE)
    - In acest moment ambele tabele sunt egale
- Sa se adauge o coloana noua la tabela Sales.CustTGT2 cu numele IsDeleted
- Coloana IsDeleted va avea 0 initial pentru toate inregistrarile (se va face un update pentru toate randurile cu valoarea 0 fara where)
- Sa se construiasca MERGE Statement astfel incat:
    - Daca sunt modificari pe un client in tabela sursa, atunci modificarile sa se propage si in tabela target, iar flag-ul IsDeleted ramane 0.
    - Daca vine un client nou in tabela sursa acesta sa fie inserat in tabela target cu IsDeleted = 0
    - Daca se sterge un client din tabela sursa sa se faca update pe coloana IsDeleted cu valoarea 1 pentru acest client

Rezolvare:

Pas 1:

- Se vor defini cele doua tabele sursa si target si se va stabili PK pentru fiecare tabela

```
Use Training;
GO

select
    custid,
```

```
    companyname,
    country
into Sales.CustSRC2
from sales.Customers
where 1=0

alter table Sales.CustSRC2
add constraint PK_custSRC2 primary key (custid)


select
    custid,
    companyname,
    country
into Sales.CustTGT2
from sales.Customers
where 1=0

alter table Sales.CustTGT2
add constraint PK_custTGT2 primary key (custid)
```

- Se vor face inserturile in tabela sursa

```
set identity_insert Sales.CustSRC2 on;
insert into Sales.CustSRC2 (custid, companyname, country) values
(8001,'Test1','Romania')
insert into Sales.CustSRC2 (custid, companyname, country) values
(8002,'Test2','Romania')
insert into Sales.CustSRC2 (custid, companyname, country) values
(8003,'Test3','Romania')
set identity_insert Sales.CustSRC2 off;
```

- Se va adauga coloana IsDeleted in tabela target

```
alter table Sales.CustTGT2
add IsDeleted bit;
```

```
select * from Sales.CustTGT2
```

custid	companyname	country	IsDeleted

Pas 2: Se va face insertul initial si update-ul pentru IsDeleted = 0

Obs: pentru a transfera din sursa in target pentru prima data datele, nu este necesara o procedura ce incapsuleaza un MERGE. Operatiunea de insert initial este one time, nu este necesara o automatizare a acesteia.

```
set identity_insert Sales.CustTGT2 on;
insert into Sales.CustTGT2 (custid, companyname, country, isDeleted)
select
       custid,
       companyname,
       country,
       0 as IsDeleted
from Sales.CustSRC2
set identity_insert Sales.CustTGT2 off;
```

Pas 3: se va define o procedura stocata care incapsuleaza MERGE-ul cu toate cazurile mentionate.

```
create procedure Sales.sp_MERGE_Cust2
as
set identity_insert Sales.CustTGT2 on;
merge Sales.CustTGT2 tg
using Sales.CustSRC2 sr on tg.custid=sr.custid

when matched and (tg.companyname<>sr.companyname or tg.country<>sr.country or
tg.Isdeleted=1)
then update
       set tg.companyname=sr.companyname,
              tg.country=sr.country,
              tg.Isdeleted=0
when not matched by target
then insert (custid, companyname, country, IsDeleted)
        values (sr.custid, sr.companyname, sr.country, 0 )
when not matched by source
then update
       set tg.IsDeleted=1;
set identity_insert Sales.CustTGT off;
```

Testare:

```
select * from sales.CustSRC2;
```

	custid	companyname	country
1	8001	Test1	Romania
2	8002	Test2	Romania
3	8003	Test3	Romania

```
select * from sales.CustTGT2;
```

	custid	companyname	country	IsDeleted
1	8001	Test1	Romania	0
2	8002	Test2	Romania	0
3	8003	Test3	Romania	0

Scenariul 1: rand modificat in tabela sursa

```
update sales.CustSRC2
```

```
        set companyname='Test 1 modificat'
where custid=8001;
```

```
select * from sales.CustSRC2
```

	custid	companyname	country
1	8001	Test 1 modificat	Romania
2	8002	Test2	Romania
3	8003	Test3	Romania

```
exec Sales.sp_MERGE_Cust2
```

```
select * from sales.CustTGT2
```

	custid	companyname	country	IsDeleted
1	8001	Test 1 modificat	Romania	0
2	8002	Test2	Romania	0
3	8003	Test3	Romania	0

Scenariul 2: rand nou in tabela sursa

```
set identity_insert Sales.CustTGT2 off;
set identity_insert Sales.CustSRC2 on;
insert into Sales.CustSRC2 (custid, companyname, country) values
(8004,'Test4','Romania')
set identity_insert Sales.CustSRC2 off;
```

```
select * from sales.CustSRC2
```

	custid	companyname	country
1	8001	Test 1 modificat	Romania
2	8002	Test2	Romania
3	8003	Test3	Romania
4	8004	Test4	Romania

```
exec Sales.sp_MERGE_Cust2
```

```
select * from sales.CustTGT2
```

	custid	companyname	country	IsDeleted
1	8001	Test 1 modificat	Romania	0
2	8002	Test2	Romania	0
3	8003	Test3	Romania	0
4	8004	Test4	Romania	0

Scenariul 2: rand sters in tabela sursa

```
delete from sales.CustSRC2
where custid=8002

select * from sales.CustSRC2
```

	custid	companyname	country
1	8001	Test 1 modificat	Romania
2	8003	Test3	Romania
3	8004	Test4	Romania

```
exec Sales.sp_MERGE_Cust2

select * from sales.CustTGT2
```

	custid	companyname	country	IsDeleted
1	8001	Test 1 modificat	Romania	0
2	8002	Test2	Romania	1
3	8003	Test3	Romania	0
4	8004	Test4	Romania	0

VII. CURSORS

Seturi de date versus cursori

Un set de date reprezinta un intreg format din mai multe randuri, toate indeplinind aceleasi conditii.

Un cursor reprezinta o solutie iterativa, care evalueaza rand cu rand un input de date.

Recomandarea este sa se foloseasca seturi de date versus cursori, exceptand cazurile in care rezolvarea unui task impune o solutie iterativa, cursor.

Exemplu:

Sa se determine pentru fiecare angajat vechimea pana astazi. Presupunem cazul in care, fiecare angajat solicita o zi in plus de concediu, iar evaluarea se va face pentru fiecare angajat si in acelasi timp trebuie tiparit mesajul: “Vechimea angajatului Popescu Maria este de x ani”

Nota: Pentru exemplul de mai sus, se poate defini o procedura stocata care are ca parametru id-ul de angajat si se poate rula procedura pentru fiecare angajat. Problema apare atunci cand se doreste rularea procedurii pentru toti angajatii.

Pasi de lucru cu un cursor:

- Se declara cursorul bazat pe o interogare care returneaza toti angajatii (DECLARE)
- Se deschide cursorul (OPEN)
- Se preia primul angajat = prima iteratie (FETCH NEXT) intr-o variabila
- Se utilizeaza o bucla pentru a parcurge toti angajatii (WHILE loop) pana cand functia @@FETCH_STATUS ia valoarea 0
- Dupa rularea tuturor iteratiilor, se inchide cursorul (CLOSE)
- Si este dezalocat (DEALLOCATE)

Valorile posibile pentru @@FETCH_STATUS pot fi:

- 0 – cand aducerea iteratiei anterioara a fost cu success
- -1 – cand randul este in afara setului de date
- -2 – cand randul adus lipseste

```
DECLARE @empid AS INT;
DECLARE @TEXT NVARCHAR(100);

DECLARE emp_cursor CURSOR FOR
          SELECT empid
          FROM HR.Employees;
```

```
    OPEN emp_cursor;
    FETCH NEXT FROM emp_cursor INTO @empid;
    WHILE @@FETCH_STATUS=0
    BEGIN
       SET @TEXT=(SELECT 'Vechimea angajatului '+firstname+' '+lastname+' este
    '+cast(DATEDIFF(yy,hiredate,getdate()) as varchar(10))+' ani'
                           from Hr.Employees
                           where empid=@empid)
       print (@text)
    FETCH NEXT FROM emp_cursor INTO @empid;
    END;
    CLOSE emp_cursor;
    DEALLOCATE emp_cursor;
```

```
Vechimea angajatului Sara Davis este 17 ani
Vechimea angajatului Don Funk este 17 ani
Vechimea angajatului Judy Lew este 17 ani
Vechimea angajatului Yael Peled este 16 ani
Vechimea angajatului Sven Buck este 16 ani
Vechimea angajatului Paul Suurs este 16 ani
Vechimea angajatului Russell King este 15 ani
Vechimea angajatului Maria Cameron este 15 ani
Vechimea angajatului Zoya Dolgopyatova este 15 ani
```

Exemplu:

Sa se determine pentru fiecare client valoarea totala a vanzarilor. Sa se incapsuleze codul intr-o procedura stocata care va avea ca parametru @custid.

Apoi sa se ruleze procedura stocata pentru toti clientii si sa se afiseze mesajul “Valoarea vanzarilor clientului X cu numele Y este de Z”.

```
create procedure sales.get_sales_cust (@custid int)
as
select
'Valoarea vanzarilor clientului '+cast(c.custid as varchar(50))+' cu numele de
'+c.companyname+' este de '+ cast(sum(od.qty*od.unitprice) as varchar(50))
from sales.Customers c
inner join sales.orders o on c.custid=o.custid
inner join sales.OrderDetails od on od.orderid=o.orderid
where c.custid=@custid
group by
       c.custid,c.companyname


exec sales.get_sales_cust 37
```

Valoarea vanzarilor clientului 37 cu numele de Customer FRXZL este de 57317.39

```
DECLARE @custid AS INT;

DECLARE cust_cursor CURSOR FOR
              SELECT custid
              FROM Sales.Customers;
```

```
OPEN cust_cursor;
FETCH NEXT FROM cust_cursor INTO @custid;
WHILE @@FETCH_STATUS=0
BEGIN
       exec sales.get_sales_cust @custid
FETCH NEXT FROM cust_cursor INTO @custid;
END;
CLOSE cust_cursor;
DEALLOCATE cust_cursor;
```

Exercitii:

1. Sa se defineasca o procedura stocata care va determina vanzarea totala pentru o tara mentionata.

   Apoi, sa se ruleze procedura stocata pentru toate tarile in tabela Customers utilizand un cursor.

VIII. SQL DYNAMIC. Pivotare tabele. Depivotare tabele. Dynamic Pivot.

SQL Dynamic este utilizat pentru a genera cod T-SQL si a fi executat.

Tabela INFORMATION_SCHEMA.TABLES

Exemplu: Sa se extraga numarul de comenzi sau numarul de produse sau numarul de categorii de produse sau numarul de clienti. Pentru aceasta cerinta sa se creeze un script care va avea ca filtru pe numele tabelului. Apoi, sa se transforme scriptul in procedura stocata cu parametru numele tabelului. Rezultatul scriptului sau al procedurii stocate va fi un count de nr total de randuri si numele tabelului. Rezultatul scriptului va fi inserat intr-o tabela permanenta cu urmatoarele coloane:

Reporting_Date = getdate()

Table_Name = provenit din parametru

No_Rows = rezultatul count-ului.

Pasul 1: se defineste selectul care extrage din tabela mentionata (o tabela aleasa).

```
select
getdate() as Reporting_Date,
'Production.Products' as Table_name,
count(*) as No_Rows
from Production.Products;
```

Pasul 2: Pornind de la scriptul de mai sus, se genereaza codul care returneaza selectul de mai sus.

```
declare @Reporting_date varchar(10)
set @Reporting_date=cast(cast(getdate() as date) as varchar(10))


declare @table_name varchar(50)
set @table_name='Production.Products'


select
'select '''+@Reporting_date+''' as Reporting_Date,'''
 +@table_name+''' as Table_Name,
count(*) as No_Rows
from '+
@table_name
from information_schema.TABLES
where TABLE_SCHEMA+'.'+TABLE_NAME=@table_name
```

Pasul 3: Se defineste tabela permanenta in care se doreste insertul.

```
create table Sales.Audit
(Reporting_date datetime,
Table_name varchar(50),
No_Rows int)
```

Pasul 4: Se creeaza variabila @sql care memoreaza codul sql generat

```
declare @sql varchar(200)
set @sql=
              (select
                     'select '''+@Reporting_date+''' as Reporting_Date,'''
                     +@table_name+''' as Table_Name,
                     count(*) as No_Rows
                     from '+ @table_name
              from information_schema.TABLES
              where TABLE_SCHEMA+'.'+TABLE_NAME=@table_name
              )
```

Pasul 5: se testeaza ce returneaza @sql

```
declare @sql varchar(200)
set @sql=
              (select
                     'select '''+@Reporting_date+''' as Reporting_Date,'''
                     +@table_name+''' as Table_Name,
                     count(*) as No_Rows
                     from '+ @table_name
              from information_schema.TABLES
              where TABLE_SCHEMA+'.'+TABLE_NAME=@table_name
              )
select @sql
```

```
select '2019-05-24' as Reporting_Date,'Production.Products' as Table_Name,

                     count(*) as No_Rows

                     from Production.Products
```

	Reporting_Date	Table_Name	No_Rows
1	2019-05-24	Production.Products	77

Pasul 6: se adauga insertul in scriptul dynamic, insert in tabela create mai sus. Se adauga si delete-ul in scriptul dynamic.

```
declare @insert varchar(100)
set @insert =' insert into Sales.Audit (Reporting_Date, Table_Name, No_Rows)
              '
declare @delete varchar(100)
set @delete=' delete from Sales.Audit where table_name='''+@table_name+''' and
Reporting_Date='''+@Reporting_date+''''
```

Pasul 6: Se concateneaza cele 3 variabile @sql, @insert, @delete astfel incat sa rezulte scriptul final.

```
declare @final_sql varchar(400)
set @final_sql = @delete + @insert +@sql

select @final_sql
```

Rezultat select @final_sql:

```
delete from Sales.Audit where table_name='Production.Products' and
Reporting_Date='2019-05-24'
insert into Sales.Audit (Reporting_Date, Table_Name, No_Rows)
select '2019-05-24' as Reporting_Date,'Production.Products' as Table_Name,
                    count(*) as No_Rows
                    from Production.Products
```

Rulare rezultat select @final_sql:

```
select * from sales.Audit
```

	Reporting_date	Table_name	No_Rows
1	2019-05-01 00:00:00.000	Production.Categories	8

Comanda EXECUTE sau EXEC:

```
declare @Reporting_date varchar(10)
set @Reporting_date=cast(cast(getdate() as date) as varchar(10))


declare @table_name varchar(50)
set @table_name='Production.Products'


declare @sql varchar(200)
set @sql=
             (select
                    'select '''+@Reporting_date+''' as Reporting_Date,'''
                    +@table_name+''' as Table_Name,
                    count(*) as No_Rows
                    from '+ @table_name
             from information_schema.TABLES
             where TABLE_SCHEMA+'.'+TABLE_NAME=@table_name
             )

declare @insert varchar(100)
set @insert =' insert into Sales.Audit (Reporting_Date, Table_Name, No_Rows)
                '
declare @delete varchar(100)
set @delete=' delete from Sales.Audit where table_name='''+@table_name+''' and
Reporting_Date='''+@Reporting_date+''''

declare @final_sql varchar(400)
set @final_sql = @delete + @insert +@sql
```

```
exec (@final_sql)
```

Pas 7: Incapsulare cod sql in procedura stocata cu parametru @table_name si @reporting_date:

```
create procedure sales.sp_get_statistics (@table_name varchar(50))
as

declare @Reporting_date varchar(10)
set @Reporting_date=cast(cast(getdate() as date) as varchar(10))


declare @sql varchar(200)
set @sql=
              (select
                     'select '''+@Reporting_date+''' as Reporting_Date,'''
                     +@table_name+''' as Table_Name,
                     count(*) as No_Rows
                     from '+ @table_name
              from information_schema.TABLES
              where TABLE_SCHEMA+'.'+TABLE_NAME=@table_name
              )

declare @insert varchar(100)
set @insert =' insert into Sales.Audit (Reporting_Date, Table_Name, No_Rows)
                     '
declare @delete varchar(100)
set @delete=' delete from Sales.Audit where table_name='''+@table_name+''' and
Reporting_Date='''+@Reporting_date+''''

declare @final_sql varchar(400)
set @final_sql = @delete + @insert +@sql


exec (@final_sql)

exec sales.sp_get_statistics 'Production.Products'


select * from sales.Audit
```

	Reporting_date	Table_name	No_Rows
1	2019-05-01 00:00:00.000	Production.Categories	8
2	2019-05-01 00:00:00.000	Production.Products	77

Rulare procedura sales.sp_get_statistics pentru toate tabelele din baza de date Training:

Acest lucru se va face cu un cursor care sa parcurga rand cu rand fiecare tabela

```
DECLARE @table_name AS varchar(50);
```

```
DECLARE tbl_cursor CURSOR FOR
          SELECT
          TABLE_SCHEMA+'.'+TABLE_NAME as Table_name
          FROM  INFORMATION_SCHEMA.TABLES
          where TABLE_NAME<>'Audit';
OPEN tbl_cursor;
FETCH NEXT FROM tbl_cursor INTO @table_name;
WHILE @@FETCH_STATUS=0
BEGIN
   exec sales.sp_get_statistics @table_name
FETCH NEXT FROM tbl_cursor INTO @table_name;
END;
CLOSE tbl_cursor;
DEALLOCATE tbl_cursor;
```

Sau cu control flow-ul while:

```
declare @tbl as table
(id int identity(1,1),
table_name varchar(50))

insert into @tbl
SELECT
          TABLE_SCHEMA+'.'+TABLE_NAME as Table_name
          FROM  INFORMATION_SCHEMA.TABLES
          where TABLE_NAME<>'Audit'

declare @id int
set @id = 1

declare @max_id int
set @max_id = (select max(id) from @tbl)


while @id<=@max_id
begin

          declare @tbl_nm varchar(50)
          set @tbl_nm = (select table_name from @tbl where id=@id)

          exec sales.sp_get_statistics @tbl_nm

          set @id=@id+1

end
```

PIVOT si UNPIVOT tabele

Tabelele pivot presupun transpunerea randurilor pe post de coloane si agregarea informatiilor.

Exemplu: Sa se determine per tara client si an order date suma vanzarilor.

```
select
c.country,
year(o.shippeddate) An,
sum(od.qty*od.unitprice) as Vanzare
from Sales.Customers c
```

```
inner join Sales.Orders o on o.custid=c.custid
inner join Sales.OrderDetails od on od.orderid=o.orderid
where o.shippeddate is not null
group by
c.country,
year(o.shippeddate)
```

	country	An	Vanzare
1	Mexico	2006	4687.90
2	Belgium	2007	12087.10
3	Italy	2007	8430.55
4	Switzerland	2006	4289.70
5	Germany	2006	35900.80
6	Spain	2008	8278.44
7	Ireland	2008	22796.34
8	Venezuela	2006	8098.50

Nota: Anul este pe coloana An, pe verticala. Se doreste un rezultat de forma:

Country / Vanzari_2006 / Vanzari_2007 / Vanzari_2008

Rezolvare utilizand CASE:

```
select
c.country,
sum(case when year(o.shippeddate)=2006 then od.qty*od.unitprice end) as  Vanzari_2006,
sum(case when year(o.shippeddate)=2007 then od.qty*od.unitprice end) as  Vanzari_2007,
sum(case when year(o.shippeddate)=2008 then od.qty*od.unitprice end) as  Vanzari_2008
from Sales.Customers c
inner join Sales.Orders o on o.custid=c.custid
inner join Sales.OrderDetails od on od.orderid=o.orderid
where o.shippeddate is not null
group by
c.country
```

	country	Vanzari_2006	Vanzari_2007	Vanzari_2008
1	Finland	3210.80	14280.65	2287.00
2	USA	35427.20	124743.30	101311.88
3	Italy	1004.20	8430.55	6762.40
4	Brazil	23849.30	42621.26	44275.12
5	Germany	35900.80	120038.07	85970.26
6	Switzerland	4289.70	18702.50	9341.30

Rezolvare utilizand PIVOT:

```
select
country,
[Vanzare_2006],
[Vanzare_2007],
[Vanzare_2008]
from
(select
c.country,
'Vanzare_'+cast(year(o.shippeddate) as varchar(4)) An,
sum(od.qty*od.unitprice) Vanzare
from Sales.Customers c
inner join Sales.Orders o on o.custid=c.custid
```

```
inner join Sales.OrderDetails od on od.orderid=o.orderid
where o.shippeddate is not null
group by
c.country,
year(o.shippeddate)) as SourceTable
PIVOT
(Sum(Vanzare)
for An in ([Vanzare_2006],[Vanzare_2007],[Vanzare_2008])
) as PivotTable
```

	country	Vanzare_2006	Vanzare_2007	Vanzare_2008
1	Argentina	NULL	1816.60	5921.50
2	Austria	27912.00	61748.73	39714.40
3	Belgium	6438.80	12087.10	16609.08
4	Brazil	23849.30	42621.26	44275.12
5	Canada	7949.60	34970.10	11104.90
6	Denmark	1246.20	25034.50	8257.25
7	Finland	3210.80	14280.65	2287.00

Depivotare tabele:

```
select
country,
[Vanzare_2006],
[Vanzare_2007],
[Vanzare_2008]
into #tbl
from
(select
c.country,
'Vanzare_'+cast(year(o.shippeddate) as varchar(4)) An,
sum(od.qty*od.unitprice) Vanzare
from Sales.Customers c
inner join Sales.Orders o on o.custid=c.custid
inner join Sales.OrderDetails od on od.orderid=o.orderid
where o.shippeddate is not null
group by
c.country,
year(o.shippeddate)) as SourceTable
PIVOT
(Sum(Vanzare)
for An in ([Vanzare_2006],[Vanzare_2007],[Vanzare_2008])
) as PivotTable


select * from #tbl
```

	country	Vanzare_2006	Vanzare_2007	Vanzare_2008
1	Argentina	NULL	1816.60	5921.50
2	Austria	27912.00	61748.73	39714.40
3	Belgium	6438.80	12087.10	16609.08
4	Brazil	23849.30	42621.26	44275.12
5	Canada	7949.60	34970.10	11104.90
6	Denmark	1246.20	25034.50	8257.25
7	Finland	3210.80	14280.65	2287.00

```
SELECT country, substring(An,9,4) As An, Vanzare
```

```
FROM
   (SELECT Country, Vanzare_2006, Vanzare_2007, Vanzare_2008
   FROM #tbl) p
UNPIVOT
   (Vanzare FOR An IN
      (Vanzare_2006, Vanzare_2007, Vanzare_2008)
)AS unpvt
```

	country	An	Vanzare
1	Argentina	2007	1816.60
2	Argentina	2008	5921.50
3	Austria	2006	27912.00
4	Austria	2007	61748.73
5	Austria	2008	39714.40
6	Belgium	2006	6438.80
7	Belgium	2007	12087.10
8	Belgium	2008	16609.08

Dynamic Pivot

```
select
c.country,
year(o.shippeddate) An,
sum(od.qty*od.unitprice) as Vanzare
into Sales.SRC
from Sales.Customers c
inner join Sales.Orders o on o.custid=c.custid
inner join Sales.OrderDetails od on od.orderid=o.orderid
where o.shippeddate is not null
group by
c.country,
year(o.shippeddate)


DECLARE @cols AS NVARCHAR(MAX),
    @query  AS NVARCHAR(MAX);

SET @cols = STUFF(
              (SELECT distinct ','+ QUOTENAME('Vanzare_'+cast(c.An as varchar(4)))
            FROM Sales.SRC c
            FOR XML PATH(''))
              ,1,1,'')
--select @cols
set @query = 'SELECT country, ' + @cols + ' from
            (
                select country
                    , vanzare
                    , ''Vanzare_''+cast(An as varchar(4)) As An
                from SRC
           ) x
            pivot
            (
                 sum(vanzare)
                for An in (' + @cols + ')
            ) p '
select @query
```

Rezultat:

```
SELECT country, [Vanzare_2006],[Vanzare_2007],[Vanzare_2008] from
            (
                select country
                    , vanzare
                    , 'Vanzare_'+cast(An as varchar(4)) As An
                from Sales.SRC
           ) x
            pivot
            (
                 sum(vanzare)
                for An in ([Vanzare_2006],[Vanzare_2007],[Vanzare_2008])
            ) p
```

Cu EXECUTE:

```
DECLARE @cols AS NVARCHAR(MAX),
```

```
    @query  AS NVARCHAR(MAX);

SET @cols = STUFF(
                (SELECT distinct ','+ QUOTENAME('Vanzare_'+cast(c.An as varchar(4)))
              FROM Sales.SRC c
              FOR XML PATH(''), TYPE
              ).value('.', 'NVARCHAR(MAX)')
        ,1,1,'')
--select @cols
set @query = 'SELECT country, ' + @cols + ' from
              (
                  select country
                      , vanzare
                      , ''Vanzare_''+cast(An as varchar(4)) As An
                  from Sales.SRC
            ) x
             pivot
             (
                  sum(vanzare)
                 for An in (' + @cols + ')
             ) p '
--select @query
execute(@query)
```

	country	Vanzare_2006	Vanzare_2007	Vanzare_2008
1	Argentina	NULL	1816.60	5921.50
2	Austria	27912.00	61748.73	39714.40
3	Belgium	6438.80	12087.10	16609.08
4	Brazil	23849.30	42621.26	44275.12
5	Canada	7949.60	34970.10	11104.90
6	Denmark	1246.20	25034.50	8257.25
7	Finland	3210.80	14280.65	2287.00

Functia STUFF: inlocuieste un text cu un alt text incepand de la pozitia mentionata.

```
SELECT STUFF('Filip Marcela',7,7,'Elena')
```

	(No column name)
1	Filip Elena

Functia QUOTENAME: adauga paranteze drepte unui text.

```
select quotename('Filip')
```

	(No column name)
1	[Filip]

Sintaxa:

```
SELECT distinct ',' + cast(an as varchar(4))
              FROM Sales.SRC
              FOR XML PATH ('')
```

- Concateneaza cu firgula fiecare valoare din coloana an din tabela Sales.SRC si pune valorile pe acelasi rand, cu mentiunea ca in fata primului an (2006) apare virgula.

	XML_F52E2B61-18A1-11d1-B105-00805F49916B
1	,2006,2007,2008

Sintaxa:

```
SELECT
    cols = STUFF(
                (SELECT distinct ',' + quotename(cast(an as varchar(4))) FROM
Sales.SRC FOR XML PATH (('')), 1, 1, ''
               )
```

- Elimina prima virgula.

PROIECT I - SQL Dynamic – Sales

Exemplu: Departamentul Sales de Business solicita departamentului de DWH un raport cu vanzarile la nivel de luna si an, rapoart actualizat in fiecare zi. Datele vor fi stocate intr-o tabela la care are acces departamentul de Sales.

Tabela va avea:

- Prima coloana numita Luna – varchar(20)
- Urmatoarele coloane vor fi anii sub forma: An_yyyy

Coloanele An_yyyy se vor defini dinamic, la trecerea de la un an la altul se va adauga dinamic inca o coloana numita An_yyyy(anul urmator).

Actualizarea tabelului se va face cu delete tot continutul si insert noul continut.

```
SELECT DATENAME(month, GETDATE()) AS 'Month Name'

create procedure Sales.sp_department_sales
as

IF OBJECT_ID('tempdb.dbo.#srcpv', 'U') IS NOT NULL
DROP table #srcpv;

IF OBJECT_ID('Training.Sales.Department_Sales', 'U') IS NOT NULL
DROP table Sales.Department_Sales;

select
       datename(month,o.orderdate) Luna,
       year(o.orderdate) An,
       sum(od.qty*od.unitprice) as Vanzare
into #srcpv
from Sales.Orders o
inner join Sales.OrderDetails od on od.orderid=o.orderid
where o.shippeddate is not null
group by
       datename(month,o.orderdate) ,
       year(o.orderdate)


DECLARE @cols AS NVARCHAR(MAX),
```

```
              @query  AS NVARCHAR(MAX);

SET @cols = STUFF(
              (SELECT distinct ','+ 'An_'+cast(An as varchar(4))
            from #srcpv
                      FOR XML PATH(''), TYPE
            ).value('.', 'NVARCHAR(MAX)')
       ,1,1,'')
--select @cols

set @query = 'SELECT Luna,' + @cols + '
                     into Sales.Department_Sales
                     from
            (
                select Luna
                    , vanzare
                    , ''An_''+cast(An as varchar(4)) As An
                from #srcpv
           ) x
            pivot
            (
                 sum(vanzare)
                for An in (' + @cols + ')
            ) p '
--select @query

execute (@query)

exec Sales.sp_department_sales

select * from sales.Department_Sales
```

	Luna	An_2006	An_2007	An_2008
1	April	NULL	55699.39	120987.56
2	August	26609.40	49981.69	NULL
3	December	50953.40	77476.26	NULL
4	February	NULL	41207.20	104561.95
5	January	NULL	66692.80	100854.72
6	July	29752.10	55464.93	NULL
7	June	NULL	39088.00	NULL
8	March	NULL	39979.90	109825.45
9	May	NULL	56823.70	6097.90
10	November	49704.00	45913.36	NULL
11	October	41203.60	70328.50	NULL
12	September	27636.00	59733.02	NULL

Modificari in tabela Orders astfel incat sa avem si anul 2009.

```
update sales.Orders
set orderdate='2009-05-01'
where orderid=10248;


exec Sales.sp_department_sales
```

```
select * from sales.Department_Sales
```

	Luna	An_2006	An_2007	An_2008	An_2009
1	April	NULL	55699.39	120987.56	NULL
2	August	26609.40	49981.69	NULL	NULL
3	December	50953.40	77476.26	NULL	NULL
4	February	NULL	41207.20	104561.95	NULL
5	January	NULL	66692.80	100854.72	NULL
6	July	29752.10	55464.93	NULL	NULL
7	June	NULL	39088.00	NULL	NULL
8	March	NULL	39979.90	109825.45	NULL
9	May	NULL	56823.70	6097.90	440.00
10	November	49704.00	45913.36	NULL	NULL
11	October	41203.60	70328.50	NULL	NULL
12	September	27636.00	59733.02	NULL	NULL

sau

Varianta: tabelul sales.Department_Sales nu se sterge de fiecare data, ci se adauga coloane noi, trecand de la un an la altul:

- Se defineste tabelul initial, cu coloanele stiute:

```
create table Sales.Raport_monthly
(Luna varchar(50),
An_2006 decimal(18,2),
An_2007 decimal(18,2),
An_2008 decimal(18,2)
constraint PK_luna primary key (Luna)
)
```

```
if OBJECT_ID('tempdb..#src') is not null
drop table #src

if OBJECT_ID('tempdb..##raport') is not null
drop table ##raport

select
       datename(month, o.orderdate) as Luna,
       year(o.orderdate) as An,
       sum(od.qty*od.unitprice) as Vanzari
into #src
from sales.Orders as o
inner join sales.OrderDetails as od on o.orderid=od.orderid
group by
       datename(month, o.orderdate),
       year(o.orderdate)

declare @cols nvarchar(max)
set @cols = stuff((
                    select distinct ','+quotename('An_'+cast(An as varchar(4)))
                    from #src
                    for xml path('')),1,1,'')

declare @qq nvarchar(max)
```

```
set @qq ='select Luna, '+@cols +'
              into ##raport
              from
                     (select
                            Luna,
                            ''An_''+cast(An as varchar(4)) as An,
                            Vanzari
                     from #src) as a
                     pivot
                     (sum(vanzari) for An in ('+@cols+')
                     ) p'
exec ( @qq)

declare @add_cols nvarchar(100)
set @add_cols =
stuff((
select ','+ column_name-- +' decimal(18,2)' -- comment aici decimal(18,2)
from
(Select
COLUMN_NAME
from tempdb.INFORMATION_SCHEMA.COLUMNS
where TABLE_NAME='##raport'
except
Select
COLUMN_NAME
from Training.INFORMATION_SCHEMA.COLUMNS
where TABLE_NAME='Raport_monthly' and TABLE_SCHEMA='Sales') as t
for xml path('')),1,1,'')

--select @add_cols

declare @add_cols_tp nvarchar(max) -- adaugat acelasi lucru ca mai sus, cu
decimal(18,2)
set @add_cols_tp =
stuff((
select ','+ column_name+' decimal(18,2)'
from
(Select
COLUMN_NAME
from tempdb.INFORMATION_SCHEMA.COLUMNS
where TABLE_NAME='##raport'
except
Select
COLUMN_NAME
from Training.INFORMATION_SCHEMA.COLUMNS
where TABLE_NAME='Raport_monthly' and TABLE_SCHEMA='Sales') as t
for xml path('')),1,1,'')


declare @alter nvarchar(max)
set @alter='alter table sales.raport_monthly
                     add '+@add_cols_tp

exec (@alter)


declare @update nvarchar(max)
set @update='update r
       set r.'+@add_cols+'=ra.'+@add_cols+'
from sales.raport_monthly as r
inner join ##raport as ra on r.luna=ra.luna'
```

```
exec (@update)
```

PROIECT II - SQL Dynamic – Orders

1. Sa se creeze o tabela cu numele Sales.Ord_bkp cu urmatoarele coloane: Orderid, Custid, Orderdate, Shipcountry. Tabela Sales.Ord_bkp nu contine date si va avea primary key pe coloana orderid.
2. Sa se construiasca dinamic statementul de merge care transfera date din tabela sursa in tabela target, cele doua putand fi stabilite de utilizator. Ambele tabele sunt pe aceeasi schema, Sales.
3. Sa se ruleze procedura pentru Sales.Ord_bkp si Sales.Orders

Rezolvare:

```
Create  table Sales.Ord_bkp
(orderid int not null primary key,
Custid int not null,
orderdate date not null,
shipcountry nvarchar(50)
)

GO

create procedure sp_dynamic_merge
(@TableName_src varchar(100), @TableName_tg varchar(100),
@SchemaName varchar(100))
as

--declare @TableName_src varchar(100)
--set @TableName_src = 'Ord_bkp'

--declare @TableName_tg varchar(100)
--set @TableName_tg = 'Orders'

--declare @SchemaName varchar(100)
--set @Schemaname='Sales'

declare @cols_src nvarchar(max)
SET @cols_src = STUFF(
              (select ','+column_name
                       from information_schema.columns
                       where table_name=@TableName_src and Table_Schema=@SchemaName
                     FOR XML PATH(''))
       ,1,1,'')

--select @cols_src

declare @cols_tg nvarchar(max)
SET @cols_tg = STUFF(
              (select ',t.'+column_name
                       from information_schema.columns
                       where table_name=@TableName_src and Table_Schema=@SchemaName
```

```
                    FOR XML PATH(''))
       ,1,1,'')

declare @pk nvarchar(100)
set @pk = (SELECT cu.column_name
            FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS tc
            INNER JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS cu
                ON tc.CONSTRAINT_TYPE = 'PRIMARY KEY'
                AND tc.CONSTRAINT_NAME = cu.CONSTRAINT_NAME
and tc.TABLE_NAME=@TableName_src and tc.table_schema=@SchemaName
)

--select @pk

declare @sql varchar(MAX)
SET @sql = 'MERGE ' +  @SchemaName+'.'+@TableName_src + ' AS s
USING ' +  @SchemaName+'.'+@TableName_tg + ' AS t
ON (s.'+@pk+' = t.'+@pk+' )
WHEN NOT MATCHED BY TARGET
    THEN INSERT('+@cols_src+')
       VALUES('+@cols_tg+');'

exec (@sql)
```

IX. SQL INJECTION

Functia QUOTENAME

Sql Injection reprezinta o tehnica de a insera un cod cu scop de distrugere. Este una dintre cele mai comune metode de hacking.

Exemplu: Sa se scrie un cod sql dynamic care face select * dintr-o tabela mentionata ca parametru, de pe schema dbo.

```
declare @tbl varchar(max)
set @tbl='sr1;'
declare @text varchar(max)
set @text = 'select * from dbo.'+@tbl

select @text;
select * from dbo.sr1;
```

- Inserarea codului drop table

```
declare @tbl varchar(max)
set @tbl='sr1; drop table sr1'
declare @text varchar(max)
set @text = 'select * from dbo.'+@tbl

select @text;


select * from dbo.sr1; drop table sr1;
```

In acest fel a fost stearsa tabela sr1 din baza de date Training, schema dbo.

Un prim pas pentru a evita distrugerea bazei de date prin inserarea de coduri cu scop distriuctiv este folosirea functiei QUOTENAME care adauga paranteze drepte unui parametru, atunci cand se foloseste numele unei tabele sau numele unei coloane.

```
declare @tbl varchar(max)
set @tbl='sr2'

declare @text varchar(max)
set @text = 'select * from dbo.'+quotename(@tbl)

select @text;
```

select * from dbo.[sr2]

Procedura sp_executesql

Pentru a evita sql injection, in loc sa fie folosita comanda execute, este recomandat sa se foloseasca procedura sp_executesql. Aceasta utilizeaza parametri de intrare, precum si parametri de output.

Exemplu: Sa se determine detaliile unei comenzi specificata de utilizator.

```
declare @orderid int
set @orderid = 10250

declare @sql nvarchar(max)
set @sql='select *
                from sales.orderdetails
                where orderid='+cast(@orderid as varchar(100))

--exec (@sql)
exec sp_executesql
       @statement=@sql,
       @params=N'@orderid  nvarchar(100)',
       @orderid=@orderid
```

Nota: Executia prin sp_executesql utilizand parametri, este mai sigura din punct de vedere sql injection. Executia utilizeaza componente: query-ul, parametri.

X. TRIGGERS

Triggerul poate fi comparat cu o procedura stocata care se executa automat in momentul in care se intampla o actiune pe o tabela sau pe un view.

Trigerii pot fi:

- DML Triggers (care actioneaza la INSERT, UPDATE, DELETE)
- DDL Triggers (care actioneaza la CREATE, ALTER, DROP)

DML Triggers

Un trigger DML este un T-SQL statement asociat unei tabele sau view, precum INSERT, UPDATE, DELETE sau o combinatie de acestea.

Triggerii DML pot fi de urmatoarele tipuri:

- AFTER (sau FOR)– se declanseaza dupa ce actiunea s-a terminat pe tabel, iar acesta poate fi doar tabel permanent
- INSTEAD OF – se declanseaza in loc de actiunea de pe tabel sau view, tabela putand fi doar permanenta.

In general, un trigger este utilizat in momentul in care se doreste ca o tabela sau view sa fie supusa auditarii, nu tine loc de development pentru tabele cu rezultate de business.

Exemplu: Sa se creeze o copie a tabelei Production.Products cu numele Production.ProductsTG1. Aceasta tabela stocheaza produsele si preturile aferente. In cazul in care, are loc o modificare de pret, atunci sa se stocheze intr-o tabela noua (Production.ProductsAudit): produsul, pretul vechi, pretul nou, data modificarii pretului.

Rezolvare:

Pasul 1: creare tabela Production.ProductsTG1

```
select *
into Production.ProductsTG1
from Production.Products


(77 row(s) affected)
```

Pasul 2: creare tabela de auditare a modificarii pretului:

```
create table Production.ProductsAudit
(productid int,
old_unitprice money,
```

```
new_unitprice money,
changedate datetime
)
```

Pasul 3: definire trigger care sa se declanseze in momentul in care are loc o modificare de pret

```
create trigger tg_productprice on Production.ProductsTG1
for Update
as
	insert into Production.ProductsAudit (productid, old_unitprice, new_unitprice,
changedate)
	select
		p1.productid,
		d.unitprice as old_unitprice,
		p1.unitprice as new_unitprice,
		getdate() as changedate
	from Production.ProductsTG1 p1
	inner join deleted d on p1.productid=d.productid
```

Nota: tabela inserted pastreaza randurile inserate (preturile noi)

Nota: tabela deleted pastreaza randurile sterse (preturile vechi)

Pasul 4: testare trigger:

- Modificare pret pentru un produs existent

```
update Production.ProductsTG1
set unitprice=100
where productid=1;
```

- Verificare preturi

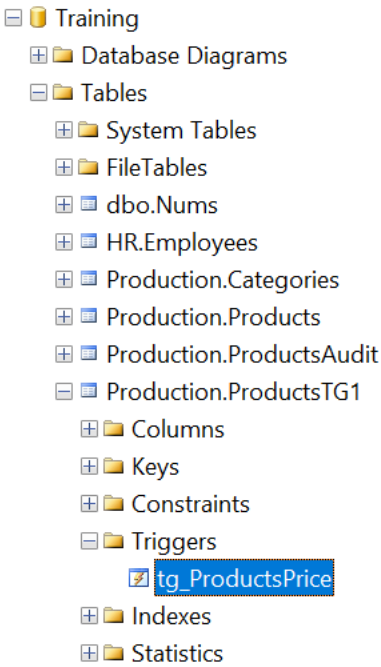
```
select * from Production.ProductsTG1
```

	productid	productname	supplierid	categoryid	unitprice	discontinued
1	1	Product HHYDP	1	1	100.00	0

```
select * from Production.ProductsAudit
```

	productid	old_price	new_price	changedate
1	1	18.00	100.00	2019-04-03

Triggerul poate fi vizualizat:



Exemplu: Sa se construiasca o tabela copie pentru tabela sales.orders cu urmatoarele coloane: orderid, orderdate, freight. Tabela se va actualiza zilnic doar prin insert. Pentru auditarea tabelei, se solicita un trigger care sa pastreze zilnic numarul de inserturi in aceasta tabela.

Rezolvare:

Pasul 1: creare tabela copie

```
select
       orderid,
       orderdate,
       freight
into Sales.Ord
from Sales.Orders
```

Pasul 2: creare tabela necesara auditarii inserturilor

```
create table Sales.ord_audit
(Reporting_Date date, No_Rows int)
```

Pasul 3: creare trigger pentru inserturi in tabela sales.ord

```
create trigger tg_ord on Sales.Ord
for insert
as
       insert into Sales.ord_audit (Reporting_Date, No_Rows)
       select
              cast(getdate() as date) as Reporting_Date,
              count(*) as No_Rows
```

```
from Sales.ord o
inner join inserted i on o.orderid=i.orderid
```

Pasul 4: inserate rand de test in tabela Sales.Ord

```
set identity_insert Sales.ord on;
insert into Sales.Ord (orderid, orderdate, freight)
select 1000 as orderid,
              '2019-05-25' as orderdate,
              100 as freight
set identity_insert Sales.ord on;
```

Pasul 5: selectare info din tabela de auditare

```
select * from Sales.ord_audit;
```

	Reporting_Date	No_Rows
1	2019-05-25	1

PROIECT: Import informatii din sursa, actualizare tabela target si auditarea actiunilor pe tabela target.

Avem urmatorul scenariu de development:

- In fiecare noapte se transfera date de la surse catre datawarehouse
- Echipa de datawarehouse preia datele de la sursa si le propaga in tabelele pentru raportare din baza de date, astfel incat, este necesar sa se stie:
    - Care sunt clientii modificati
    - Ce s-a modificat
    - Cand s-a modificat

Solutie:

- Analistul Tehnic trebuie sa stie cum vin datele din sursa: full sau delta, care este coloana sau coloanele din cheia primara, care sunt coloanele care se modifica si care nu se modifica, daca din sursa se pot sterge date sau nu. In functie de aceste raspunsuri se demareaza implementarea tehnica de catre developer.
- Developerul:
    - va creea o procedura stocata care va rula in fiecare noapte, dupa ce sursele trimit date in dwh. Procedura va incarca datele in tabelele target.
    - Procedura va face MERGE pentru a actualiza datele in dwh si pentru a pastra istoricul: Valid_From, Valid_To
    - Mai este necesara o procedura care sa auditeze load-ul in dwh:
        - Cate randuri au fost inserate si cate modificate.

Pentru exemplificare:

- Sa se creeze 2 copii (ambele goale) pentru tabela Sales.Customers cu numele Sales.CustTest1 si Sales.CustTest2 cu urmatoarele coloane: custid, companyname, country. Tabela Sales.CustTest2 va contine in plus si urmatoarele coloane: Valid_From, Valid_To. PK pentru Sales.CustTest1 va fi custid, iar PK pentru Sales.CustTest2 va fi Custid si Valid_from.
- In tabela Sales.CustTest1 (este tabela sursa si vine **delta**: clientii noi sau modifcati clientii vechi) se vor insera urmatorii clienti:
    - 1001, Test1, Romania
    - 1001, Test2, Romania
- Se va creea o procedura stocata care va gestiona MERGE Statement pentru transfer de date de la Sales.CustTest1 catre Sales.CustTest2 astfel:
    - Inserturi noi
    - Modificari cu inchidere randuri vechi si insert randuri vechi modificate
- Se vor define 2 tabele suplimentare utilizate pentru auditare: cu inca o procedura stocata se vor pastra modificarile in tabela de modificari, cu un trigger se gestioneaza inserturile:
    - Tabela pentru inserturi in Sales.CustTest2, numita Sales.CustTest2Audit
    - Tabela pentru update-uri pe Sales.CustTest2
- Se va creea o procedura de auditare care va extrage informatiile din tabelele create suplimentar:
    - Tabela pentru inserturi in Sales.CustTest2, numita Sales.CustTest2Audit
    - Tabela pentru update-uri pe Sales.CustTest2
- Procedura va extrage un raport cu nr de randuri updata-te pe tabela Sales.CustTest2 si nr de randuri inserate pe tabea Sales.CustTest2.

Rezolvare:

Pasul 1: se vor define cele doua tabele: sursa si target:

- Sales.CustTest1 (stim ca vine delta, este tabela sursa)
- Sales.CustTest2 (va fi tabela target istorizata cu Valid_from, Valid_to)

```
-
-- definesc tabela sursa = delta

create table Sales.CustTest1
(custid int not null,
companyname nvarchar(100) not null,
country nvarchar(100) not null
constraint pk_t1 primary key (custid)
```

```
)

-- definesc tabela target = cu versionare

create table Sales.CustTest2
(custid int not null,
companyname nvarchar(100) not null,
country nvarchar(100) not null,
valid_from date not null,
valid_to date not null
constraint pk_t2 primary key (custid, valid_from)
)

insert into Sales.CustTest1 values (1001,'Test1','Romania'),
                                                    (1002,'Test2','Romania')

-- procedura de merge cu output

create procedure sp_merge_custtest2 (@import_date date)
as

if OBJECT_ID('tempdb..#tbl') is not null
drop table #tbl

create table #tbl
(action_type varchar(100),
custid int)

declare @valid_from date
set @valid_from=dateadd(dd,0,@import_date)

declare @valid_to date
set @valid_to=dateadd(dd,-1,@import_date)

merge Sales.CustTest2 as t
using Sales.CustTest1 as s on t.custid=s.custid and t.valid_to='9999-12-31'

when matched and (t.companyname<>s.companyname or t.country<>s.country)
then
    update
    set t.valid_to=@valid_to
when not matched by target
then
    insert (custid, companyname, country, valid_from, valid_to)
    values (custid, companyname, country, @valid_from, '9999-12-31')
output
    $action as action_type,
    deleted.custid
    into #tbl;

insert into Sales.CustTest2 (custid, companyname, country, valid_from, valid_to)
select
    a.custid,
    a.companyname,
    a.country,
    @valid_from,
    '9999-12-31' as valid_to
from Sales.CustTest1 as a
inner join #tbl as tb on a.custid=tb.custid
where action_type='update'
```

```
-- tabela pentru modificari - auditare
create table Sales.CustTest2Audit_1
(custid int,
old_companyname varchar(40),
new_companyname varchar(40),
old_country varchar(15),
new_country varchar(15),
changedate datetime
)

-- tabela pentru inserturi - auditare
create table Sales.CustTest2Audit_2
(custid int,
companyname varchar(40),
country varchar(15),
changedate datetime)

-- procedura pentru captarea modificarilor
create procedure  sp_tg_audit_custtest2_update (@update_date date)
as

begin
   insert into Sales.CustTest2Audit_1 (
                       custid
                       , old_companyname
                       , new_companyname
                       , old_country
                       , new_country
                       , changedate)
   select
          t2.custid,
          t2.companyname as old_companyname,
          tt2.companyname as new_companyname,
          t2.country as old_country,
          tt2.country as new_country,
          tt2.valid_from as changedate
   from Sales.CustTest2 as t2
   inner join Sales.CustTest2 as tt2 on t2.custid=tt2.custid
   where tt2.valid_to='9999-12-31'
   and t2.valid_to=dateadd(dd,-1,@update_date)
end


-- triggerul pentru captarea inserturilor
create trigger sales.tg_audit_custtest2_insert on Sales.CustTest2
for insert
as
   insert into Sales.CustTest2Audit_2 (custid, companyname, country, changedate)
   select
          t2.custid,
          i.companyname,
          i.country,
          t2.valid_from as changedate
   from Sales.CustTest2 as t2
   inner join inserted as i on t2.custid=i.custid
   and i.custid not in (select custid from Sales.CustTest2 where valid_to<>'9999-
12-31')

create procedure sp_audit_CustTest2 (@data date)
as
   select
```

```
        changedate as Reporting_Date,
        'update' as Action_Type,
        count(custid) as No_Rows
from Sales.CustTest2Audit_1
where changedate=@data
group by changedate

union all

select
        changedate as Reporting_Date,
        'insert' as Action_Type,
        count(custid) as No_Rows
from Sales.CustTest2Audit_2
where changedate=@data
group by changedate

-- test 1 – prima rulare

exec sp_merge_custtest2 '2020-07-07'
exec sp_tg_audit_custtest2_update '2020-07-07'
select * from Sales.CustTest2Audit_1
select * from Sales.CustTest2Audit_2
select * from Sales.CustTest2
exec sp_audit_CustTest2 '2020-07-07'

-- test 2: vine o modificare si un client nou

truncate table Sales.CustTest1
insert into Sales.CustTest1 values (1001,'Test1','Italy'),
                                   (1003,'Test3','Romania')
exec sp_merge_custtest2 '2020-07-08'
exec sp_tg_audit_custtest2_update '2020-07-08'
select * from Sales.CustTest2Audit_1
select * from Sales.CustTest2Audit_2
select * from Sales.CustTest2
exec sp_audit_CustTest2 '2020-07-08'
```

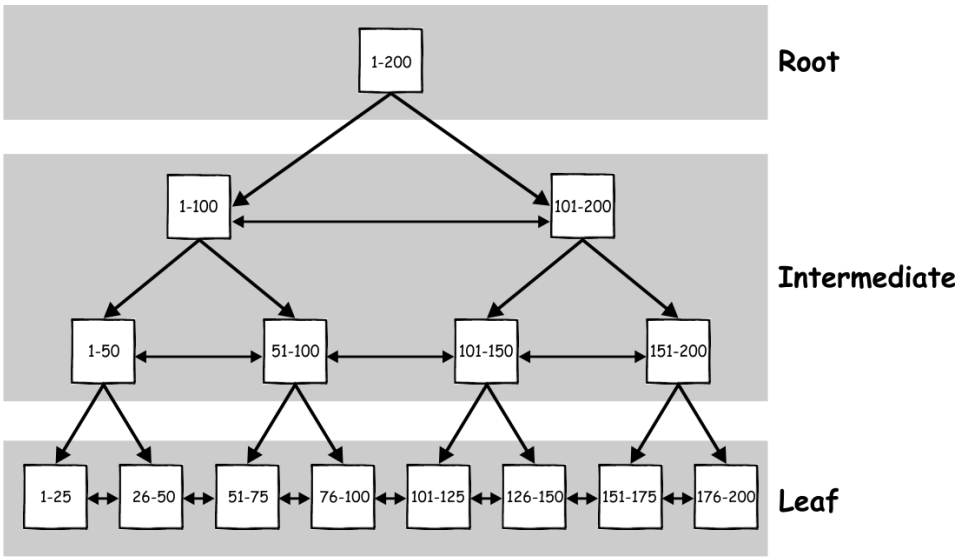
XI. INDEXES si EXECUTION PLAN

Un **index** este o structura pe disk asociata unei tabele sau unui view. Rolul indecsilor este de a creste performanta interogarii bazelor de date, de a extrage date mai rapid.

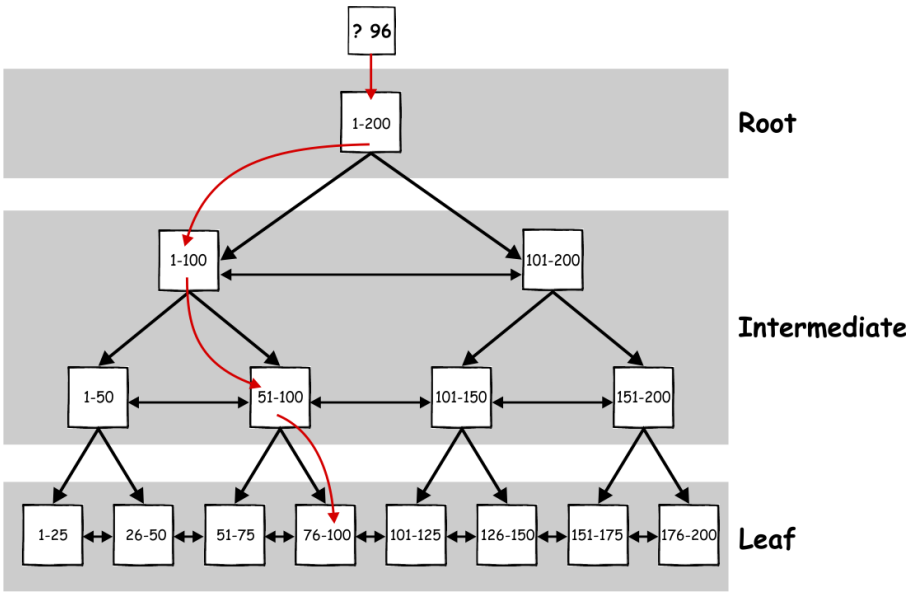
Un **index** este format din chei construite din una sau mai multe coloane dintr-o tabela sau dintr-un view.

Cheile sunt pastrate in structura B-tree ceea ce ofera lui SQL Server performanta mai mare de a extrage randul sau randurile asociate cu aceste chei.

Reprezentare grafica:



Se cauta valoare 96:



Tipuri de indecsi:

- Clustered
- Nonclustered

Index Clustered:

- Un index clustered **sorteaza** si **pastreaza** randurile in tabela sau view in functie de valorile din chei (incluse in definitia indexului), **key value**. Pe o tabela sau view poate exista **un singur index clustered** (intr-o tabela, randurile pot fi pastrate intr-o singura ordine), de obicei, acesta este PK.
- Singura data cand randurile sunt pastrate intr-o tabela intr-o anumita ordine este atunci cand tabela are un index clustered. In acest caz, tabela sau view-ul se numeste **clustered table.**
- Atunci cand exista un index clustered pe o tabela, datele fiind sortate (din punct de vedere al bazelor de date, informatiile sunt pastrate in pagini, cee ace inseamna, ca atunci cand se face un insert pe o tabela clustered, actiunea este costisitoare pentru ca se resorteaza datele in pagini).
- Avantajul indexului clustered este la citirea datelor din tabele, fiind sortate, query-ul de citire este performant. De asemenea, o tabela cu PK (cee ace inseamna si un index clustered) asigura integritatea datelor, deci nu putem renunta la index clustered pentru performanta statementurilor de DML (insert, update, delete).
- In cazul in care o tabela nu are un index clustered, randurile din tabela nu sunt stocate intr-o anumita ordine, iar tabela se numeste **heap.**
- Informatiile pastrate intr-o tabela heap nu sunt intr-o ordine anume, cee ace inseamna ca nu mai este necesara actiunea de sortare date in pagini. O tabela heap nu este performanta la citire, ci este performanta la scriere (DML statements).

Index Nonclustered:

- un index nonclustered are o structura separata de randurile dintr-o tabela sau view. Un index nonclustered este format din chei care **puncteaza** (are un pointer) catre randurile din tabela sau view. Pointer-ul se numeste **row locator.** Acest row locator puncteaza catre key value dintr-un index clustered, daca acesta exista.

  Tipuri de indecsi nonclustered:

  - **unique** – asigura unicitatea datelor, la fel cum se intampla cu PK, doar ca pe o tabela nu pot exista doua PK si atunci se poate proceda astfel:

- se creaza constrangerea de UNIQUE, ceea ce inseamna crearea automata a unui index nonclustered sau
- se creeaza un index unique nonclustered

Nota:

- crearea unei constrangeri de unicitate inseamna automat crearea unui index nonclustered unique
- crearea unui index nonclustered unique nu duce la crearea constrangerii de unicitate
- ambele asigura unicitatea datelor

Sintaxa:

```
create unique nonclustered index id_100 on test_mar (cnp)
```

- **filtered** – presupune crearea unui index nonclustered cu filtru luand on considerare faptul ca multe dintre interogari se realizeaza pe un anumit subset de date. Exemplu: O companie are 80% dintre clienti din USA, iar rapoartele de interes sunt pentru USA. Atunci se poate define un index nonclustered cu filtru pe USA.
  Sau, toate rapoartele se extrag pe o perioada de timp (orderdate) si de cele mai multe ori se doresc informatii doar despre comenzile livrate.

```
CREATE NONCLUSTERED INDEX id_test
    ON Sales.Orders (Orderdate)
    WHERE Shippeddate IS NOT NULL ;
```

- **included columns** – luand in considerare faptul ca in rapoarte, de interes este aproape tot timpul coloana freight pentru comenzile plasate, iar rapoartele se extrag punand de fiecare data conditia pe orderdate, atunci se poate define un index nonclustered pe orderdate cu coloanal inclusa freight.

```
CREATE NONCLUSTERED INDEX Id_ord
ON Sales.Orders (Orderdate)
INCLUDE (Freight);
```

Obs:

Indecsii sunt automat creati pe o tabela in momentul in care se defineste PK sau UNIQUE constraint.

Daca o tabela continue PK, atunci SQL Server creaza automat un index clustered.

Daca pe o tabela se defineste UNIQUE constraint, atunci SQL Server creaza automat un index nonclustered.

In general, crearea indecsilor se face in functie de o analiza a query-urilor rulate pe o baza de date:

- care sunt filtrele frecvente
- ce joinuri se fac
- care sunt coloanele cele mai utilizate in selecturi

Rolul indecsilor este a adduce performanta la citirea dintr-o baza de date.

Atunci cand o baza de date este pregatita pentru DML frecvente (baza de date OLTP), atunci, recomandarea este sa nu se adauge indecsi, pastrand doar PK.

Exemplu:

1) Sa se creeze 3 copii pentru tabelele: Sales.Customers, Sales.Orders, Sales.OrderDetails. Cele 3 copii se vor numi: Sales.Customers_Test, Sales.Orders_Test, Sales.OrderDetails_Test.

Nota: crearea celor 3 tabele inseamna definirea coloanelor, inserarea datelor, insa constrangerile nu se transfera de pe tabelele sursa.

```
Use Training;
GO

select *
into Sales.Customers_Test
from Sales.Customers
GO
select *
into Sales.Orders_Test
from Sales.Orders
GO
select *
into Sales.OrderDetails_Test
from Sales.OrderDetails
```

```
(91 row(s) affected)

(830 row(s) affected)

(2155 row(s) affected)
```

Fiecare din cele 3 tabele copie nu are PK sau alta constrangere.

2) Sa se determine lista clientilor din Italy folosind atat tabela Sales.Customers, cat si tabela Sales.Customers_Test.

Tabela Sales.Customers:

- Are pk pe Custid => clustered index (datele sortate in pagini).

- Planul estimate de executie indica faptul ca interogarea a mers pe indexul clustered, insa, a facut scan (verificat toata tabela) pe conditia de country, nefiind index nonclustered pe country.

```
14 select
15     custid,
16     companyname,
17     contactname
18 from Sales.Customers
19 where country='Italy'
20
```

19 %

Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
select custid, companyname, contactname from Sales.Customers where country='Italy'
```

SELECT Cost: 0 % — Clustered Index Scan (C… [Customers].[PK_Custome… Cost: 100 %

Tabela Sales.Customers_Test:

- Nu are PK, nu are index nonclustred pe coloanal country.
- Panul estimate de executie indica faptul ca s-a facut full scan, a scanat toata tabela pentru a extrage informatiile.

```
21 select
22     custid,
23     companyname,
24     contactname
25 from Sales.Customers_Test
26 where country='Italy'
```

119 %

Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
select custid, companyname, contactname from Sales.Customers_Test where country='Italy'
```

SELECT Cost: 0 % — Table Scan [Customers_Test] Cost: 100 %

Actiuni pe tabela Sales.Customers_Test:

- Creare PK

```
alter table Sales.Customers_Test
add constrainint PK_cust_test primary key (custid)
```

Verificam din nou planul estimate de executie:

```
select
    custid,
    companyname,
    contactname
from Sales.Customers_Test
where country='Italy'
```

119 %

Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
select custid, companyname, contactname from Sales.Customers_Test where country='Italy'
```

select custid, companyname, contactname

SELECT
Cost: 0 %

Clustered Index Scan (C…
[Customers_Test].[PK_cu…
Cost: 100 %

- Creare index nonclustered pe coloana country

```
create nonclustered index id_country on Sales.Customers_test (country)
```

Fiind o tabela mica, nu se poate observa o imbunatatire a interogarii.

Daca se ruleaza query-ul de mai jos, cautarea a mers strict pe indexul clustered.

```
select
    custid,
    companyname,
    contactname
from Sales.Customers_Test
where custid=89
```

119 %

Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
select custid, companyname, contactname from Sales.Customers_Test where custid=89
```

SELECT
Cost: 0 %

Clustered Index Seek (C…
[Customers_Test].[PK_cu…
Cost: 100 %

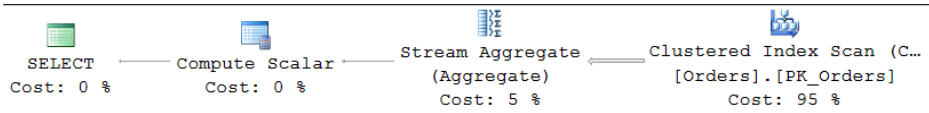
3) Sa se determine nr de comenzi si suma costului de transport din tabela Orders si din tabela Orders_Test pentru shipcountry Italy.

Pentru fiecare din query-urile de mai jos, se va afisa execution plan (costul de executiei al interogarii).

```
select
count(orderid),
sum(freight)
from Sales.Orders
where shipcountry<>'Italy'
```

Results | Messages | Execution plan

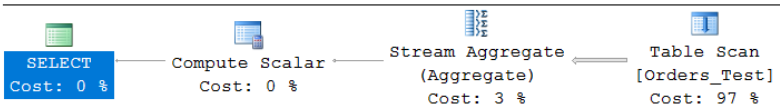
```
Query 1: Query cost (relative to the batch): 100%
SELECT COUNT([orderid]),SUM([freight]) FROM [Sales].[Orders] WHERE [shipcountry]<>@1
```




```
select
count(orderid),
sum(freight)
from Sales.Orders_Test
where shipcountry<>'Italy'
```

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT COUNT([orderid]),SUM([freight]) FROM [Sales].[Orders_Test] WHERE [shipcountry]<>@1
```



Obs:
Table Scan:

- Pentru tabela Sales.Orders este de 95%
- Pentru tabela Sales.Orders_Test este de 97%

Tabela Sales.Orders are PK => ca are index clustered.

Tabela Sales.Orders_Test nu are PK.

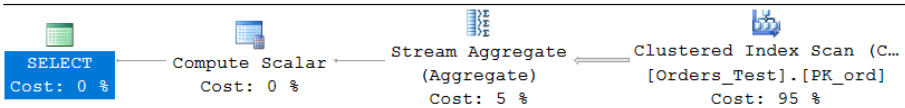
4) Creare PK pe tabela Sales.Orders_Test

```
alter table Sales.Orders_Test
add constraint PK_ord primary key (orderid)
```

5) Rerulare query pentru tabela Sales.Orders_Test

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT COUNT([orderid]),SUM([freight]) FROM [Sales].[Orders_Test] WHERE [shipcountry]<>@1
```



XII. PROIECTE

1. Sa se construiasca o noua baza de date cu numele Proiect.
2. Sa se creeze urmatoarele 3 tabele copie: Customers, Orders, OrderDetails in baza date date Proiect, cu informatiile din baza de date Training.
3. Sa se defineasca PK pentru fiecare din cele 3 tabele.
4. Sa se construiasca o procedura stocata care returneaza vanzarile pentru un anumit client mentionat (custid).
5. Sa se creeze un cursor care va rula procedura stocata pentru fiecare client din baza de date.
6. Sa se construiasca in baza de date Proiect o noua tabela copie pentru tabela Customers (Customers_bkp) cu urmatoarele coloane: custid, companyname, country, city si care va continue toate informatiile din tabela Customers.
7. Sa se defineasca o noua procedura stocata care sa suprascrie modificarile in tabela Customers_bkp in cazul in care acestea exista in tabela Customers sau sa insereze clientii noi. Nu se va trata delete-ul (in cazul in care un client din tabela Customers este sters, nu se doreste stergerea lui si din tabela Customers_bkp).
8. Sa se modifice companyname pentru clientul 1, astfel incat sa se numeasca Test, in tabela Customers.
9. Sa se ruleze procedura create la punctul 7. Sa se verifice daca in tabela Customers_bkp, clientul 1 are companyname Test.
10. Sa se genereze un script care sa determine numarul de randuri din fiecare tabela din baza de date Proiect.

```
CREATE DATABASE PROIECT;

SELECT *
INTO PROIECT.DBO.CUSTOMERS
FROM TRAINING.SALES.CUSTOMERS


SELECT *
INTO PROIECT.DBO.ORDERS
FROM TRAINING.SALES.ORDERS


SELECT *
INTO PROIECT.DBO.ORDERDETAILS
FROM TRAINING.SALES.ORDERDETAILS

ALTER TABLE PROIECT.DBO.CUSTOMERS
ADD CONSTRAINT PK_CUST PRIMARY KEY (CUSTID)

ALTER TABLE PROIECT.DBO.ORDERS
ADD CONSTRAINT PK_ORD PRIMARY KEY (ORDERID)

ALTER TABLE PROIECT.DBO.ORDERDETAILS
ADD CONSTRAINT PK_OD PRIMARY KEY (ORDERID, PRODUCTID)

CREATE PROCEDURE SP_GET_CUSTOMER_SALES
(@CUSTID INT)
AS
SELECT
       C.custid,
       C.companyname,
       SUM(OD.QTY*OD.UNITPRICE) AS SALES
FROM CUSTOMERS C
INNER JOIN ORDERS O ON C.CUSTID=O.CUSTID
INNER JOIN ORDERDETAILS OD ON O.ORDERID=OD.ORDERID
WHERE C.custid=@CUSTID
GROUP BY
       C.custid,
       C.companyname


EXEC SP_GET_CUSTOMER_SALES 87
```

	custid	companyname	SALES
1	87	Customer ZHYOS	16617.10

```
DECLARE @CUSTID INT
DECLARE @SALES TABLE (CUSTID INT, COMPANYNAME VARCHAR(100), SALES DECIMAL(18,2))
DECLARE CUST CURSOR FOR
       SELECT CUSTID FROM CUSTOMERS;
OPEN CUST;
FETCH NEXT FROM CUST INTO @CUSTID
WHILE @@FETCH_STATUS=0
BEGIN
       INSERT INTO @SALES
       EXEC SP_GET_CUSTOMER_SALES @CUSTID
       SELECT * FROM @SALES WHERE CUSTID=@CUSTID
FETCH NEXT FROM CUST INTO @CUSTID;
```

```
END;
CLOSE CUST;
DEALLOCATE CUST;



SELECT CUSTID, companyname, COUNTRY, CITY
INTO CUSTOMERS_BKP
FROM CUSTOMERS;


CREATE PROCEDURE SP_MERGE_CUST
AS

MERGE INTO CUSTOMERS_BKP AS TG
USING CUSTOMERS AS SR ON TG.CUSTID=SR.CUSTID

WHEN MATCHED AND (TG.COMPANYNAME<>SR.COMPANYNAME OR TG.COUNTRY<>SR.COUNTRY OR
TG.CITY<>SR.CITY)
THEN UPDATE
       SET TG.COMPANYNAME=SR.COMPANYNAME,
              TG.COUNTRY=SR.COUNTRY,
              TG.CITY=SR.CITY
WHEN NOT MATCHED BY TARGET
THEN INSERT (CUSTID, COMPANYNAME, COUNTRY, CITY)
VALUES (SR.CUSTID, SR.COMPANYNAME, SR.COUNTRY, SR.CITY);


update customers
set companyname='Test'
where custid=1;

set identity_insert Customers_bkp on;
exec SP_MERGE_CUST
set identity_insert Customers_bkp off;

select * from CUSTOMERS_BKP where custid=1


select
       'select '''+TABLE_NAME+''' as table_name,
       count(*) as no_rows
       from '+TABLE_NAME
from INFORMATION_SCHEMA.TABLES
```