Transakcia, Kurzory

1) TRANSAKCIA

1a) Databázová transakcia, Vlastnosti ACID
1b) Explicitná transakcia
1c) LOG protokol
1d) Príklad

2) Kurzory

2a) Deklarácia kurzora
2b) Vnorené kurzory

3) Pivot tabuľky

3a) Klauzula PIVOT
3b) Manualny PIVOT
3c) Poloautomatický PIVOT

1) TRANSAKCIA

1a) Databázová transakcia, Vlastnosti ACID
1b) Explicitná transakcia
1c) LOG protokol
1d) Príklad

1a) Databázová transakcia je samostatný celok príkazov na prevod dát. ____

Ak transakcia je

- úspešná, potom všetky operácie v rámci transakcie s dátami sú vykonané a uložené do DB.
- neúspešná, potom operácie v rámci transakcie sú anulované a zmeny v dátach sú odstránené

Transakcia spĺňa **vlastnosti** ACID:

- **A – Atomicity -** transakcia je atomická operácia: vykoná sa buď ako **celok** alebo vôbec nie
- **C – Consistency – dôslednosť -** transakciou sa nenaruší žiadne **integritné** obmedzenie v rámci DB
- **I – Isolation –** vrátenie transakcie **nevyvolá** ďalšiu transakciu
- **D – Durability – trvalosť -** úspešné transakcie sú **uložené** do databázy

1b) Explicitná transakcia:

Formy transakcie

- Autocommit transactions – každý individuálny príkaz je transakcia;
- Implicit transactions - nová transakcia je implicitne zahájená po dokončení predchádzajúcej transakcie pomocou COMMIT alebo ROLLBACK;
- Batch-scoped transactions – platí len pre multiple active result sets (MARS);
- **Explicit transactions** – každá transakcia je explicitne počatá s BEGIN TRANSACTION a explicitne ukončená s COMMIT alebo ROLLBACK príkazom
  - **BEGIN TRANSACTION**
  - **COMMIT or ROOLBACK**
    Commit označuje **koniec úspešnej** implicitnej alebo explicitnej transakcie.
    Ak **@@TRANCOUNT** je 1, COMMIT TRANSACTION zabezpečí, aby všetky zmeny dát vykonané od začiatku transakcie stali natrvalo súčasťou databázy, uvoľní prostriedky potrebné pre transakcie, a @@TRANCOUNT nastaví na 0. Ak @@TRANCOUNT je väčšie ako 1, zníži sa @@TRANCOUNT iba o 1 a transakcia zostane aktívna.
    ROLLBACK TRANSACTION sa využíva na vymazanie všetkých zmien dát vykonané od začiatku transakcie alebo uloženého bodu v prípade neúspešnej transakcie. Príkaz tiež uvoľní prostriedky držané transakciou.

1c) LOG protokol

Do transaction LOG protokolu sa zaznamenávajú všetky **transakcie** a **zmeny/úpravy** databázy vykonané každou transakciou. Ak dôjde k zlyhaniu systému a dôjde k strate kontaktu s klientom a *transakcia* sa nerealizuje úplne, databáza sa vráti do pôvodného konzistentného stavu pomocou

log súboru. Ak server zlyhá, databáza môže zostať v stave, že niektoré *úpravy* neboli nikdy zapísané z vyrovnávacej pamäte do dátových súborov.

Koncepčne log súbor je **reťazcom** (protokolových) log záznamov. **Fyzicky**, postupnosť log záznamov je uložená efektívne do fyzických **súborov**, ktoré implementujú transaction log.

Niektoré **typy obnovení** (recovery)

- Individuálne obnovenie transakcie.
- Obnova všetkých nedokončených transakcií.
- Rolovanie obnovenej databázy, súboru, skupiny súborov alebo stránky späť do bodu zlyhania.

Obnova DB servera pozostáva z troch fáz

1. fáza analýzy analyzuje protokol transakcií pomocou kontrolných bodov, a vytvorí tabuľku špinavých stránok a tabuľku aktívnych transakcií (ATT).
2. fáza Redo sa stará o neúplné úpravy zaznamenané v protokole, ktoré nie sú úplne zapísané do dátových súborov. Špinavé stránky sa tu zapisujú do dátových súborov.
3. fáza Undo roluje späť neúplné transakcie nájdené v ATT, aby sa zaistilo zachovanie integrity databázy. Po rollback sa databáza prepne do režimu online.

Pages do pamati - WHERE, TRANS, LOCK, Zmeny v pamati, Zmeny do LOG, Commit

1d) Príklad

V rámci transakcie s využitím uložených procedúr budeme vykonať dve operácie

- vkladanie údajov (insert)
- vymazanie údajov (delete)

Pri vytvorení databázy TransakciaDB ukážeme ako je možné nastaviť niektoré parametre, ako napr. jej počiatočnú veľkosť a uvedieme dve úspešné transakcie a tri neúspešné.

A) Najprv vytvorme DB TransakciaDB, v nej dve tabuľky Osoba a OsobaUdaje a vložíme do nich dva, dvariadky

```
(1,'Bobo Bibi', 'IBM ABC'),
(2,'Tata Tutu', 'Micro XYZ')

(1, 'Roznava'),
(2, 'Kosice')
```

Skontroluj

- obsah adresára – Search DATA z C:\Program Files\Microsoft SQL Server
  C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA
- Adresar ma byt existujuci: C:\Csaba_____Vyuka

```
USE Master
GO
IF DB_ID ('TransakciaDB') IS NOT NULL DROP DATABASE TransakciaDB
GO

CREATE DATABASE TransakciaDB
---- Ak pristup pre subory .mdf a .ldf nie je zakazany, inak zakomentuj:
```

```
ON PRIMARY(
NAME = N'TransakciaData',
FILENAME = N'C:\Csaba\_____Vyuka\TransakciaDB.mdf',
-- SIZE = 3MB, -- = 5MB
MAXSIZE = 10MB,
FILEGROWTH = 20%
)

LOG ON(
NAME = N'TransakciaLog',
FILENAME = N'C:\Csaba\_____Vyuka\TransakciaDB.ldf',
SIZE = 1MB,
MAXSIZE = 5MB,
FILEGROWTH = 1MB
)
GO

USE TransakciaDB
GO
create table Osoba(
      idOsoba int primary key not null,
      meno nvarchar(10) not null,
      firma nvarchar(15)
)
create table OsobaUdaje(
      idOsoba int foreign key references dbo.Osoba(idOsoba),
      adresa nvarchar(30)
)
Insert into Osoba values
(1,'Bobo Bibi', 'IBM ABC'),
(2,'Tata Tutu', 'Micro XYZ')
Insert into OsobaUdaje values
(1, 'Roznava'),
(2, 'Kosice')
GO
```

B)Vytvorme uloženú procedúru sp_InsertDelete, ktorá pomocou **begin transaction ...** vloží a deletuje Osobu s kontrolou chýb a **commit** či **rollback.**

```
IF OBJECT_ID ( 'sp_InsertDelete', 'P' ) IS NOT NULL
    DROP PROCEDURE sp_InsertDelete;
GO
create procedure sp_InsertDelete
      @idOs int,
      @meno nvarchar(10),
      @firma nvarchar(15),
      @idOsOld int
as
      declare @erIns int
      declare @erDel int
```

```
      declare @er int

     -- 1)
      begin transaction
      -- a) Pridaj osobu
      insert into Osoba (idOsoba, meno, firma) values(@idOs, @meno, @firma)

      -- Mozna chyba po Insert
      set @erIns = @@error
      if @erIns > 0 set  @er  = @erIns

      -- b) Maz osobu
      delete from Osoba where idOsoba = @idOsOld

      -- Mozna chyba po Delete
      set @erDel = @@error
      if @erDel > 0 set  @er  = @erDel

      -- 2)
      -- If error, roll back
      if @er <> 0
        begin
          rollback
          RAISERROR('Csaba - Transaction  rolled  back', 11,1)
        end
      else
        begin
          commit
          print  'Csaba - Transaction  committed'
        end
      print 'Csaba - INSERT error number:' + cast(@erIns as nvarchar(8))
      print 'Csaba - DELETE error number:' + cast(@erDel as nvarchar(8))
return @er
GO
```

C) Teraz vložíme 2 osoby s novými id a vymažeme osobu s id 33. Operácie s danými hodnotami nenarúšajú integritu dát, preto systém ich vykoná a nehlási chybu.

```
--In 2008 DELETE returns error number 0 even though it has not deleted any
rows
select * from osoba
select * from OsobaUdaje

-- Pozri Messages:
exec sp_InsertDelete 3, 'Fero', null, 33
exec sp_InsertDelete 4, 'Jano', null, 33

select * from osoba
select * from OsobaUdaje
```

Ani jeden z nasledujúcich troch príkazov sa nevykoná a systém hlási rôzne chyby.

- Tu chceme vkladať osobu s id, ktorá už existuje a vymazať osobu s existujúcim id, ale neexistujúcim cudzím kľúčom.

```
exec sp_InsertDelete 4, 'Stevo', null, 3 -- Err Primary Key
```

- Tu by sme chceli vložiť osobu s novým id a mazať inú osobu s existujúcim primárnym a cudzím kľúčom.

```
exec sp_InsertDelete 5, 'Stevo', null, 2 -- Err Delete Reference error
```

- Tu ide o snahu vložiť osobu s id, ktoré už existuje a mazať inú osobu s existujúcim primárnym a cudzím kľúčom.

```
exec sp_InsertDelete 4, 'Stevo', null, 2 -- Err PK + Err Delete
```

2) Kurzory

2a) Deklarácia kurzora
2b) Vnorené kurzory

Kurzor je DB prostriedok na získanie prístupu k jednotlivým **riadkom** tabuľky DB. Efektívne spracovanie tabuliek po riadkoch pomocou kurzora môže byť problematické, lebo DB-vé optimalizačné systémy samostatne ťažko nájdu optimálnu verziu užívateľského kódu s kurzorom. Napriek tomu, DB systémy podporujú kurzory (MS SQL, Oracle viac) na základe štandardu SQL-92.

SQL ako vysokoúrovňový deklaratívny jazyk vs. nízkoúrovňový cyklus.

Syntax.

1) Deklarácia
DECLARE kur1 CURSOR FOR SELECT ...
2) Otvorenie + použitie: získanie
3) Zatvorenie

Podrobnejsie: https://msdn.microsoft.com/en-us/library/ms180169.aspx https://msdn.microsoft.com/en-us/library/ms181441.aspx https://msdn.microsoft.com/en-us/library/ms191179.aspx

```
ISO Syntax
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR
     FOR select_statement
     [ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
[;]

Transact-SQL Extended Syntax
DECLARE cursor_name CURSOR [ LOCAL | GLOBAL ]
     [ FORWARD_ONLY | SCROLL ]
     [ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
     [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
     [ TYPE_WARNING ]
     FOR select_statement
     [ FOR UPDATE [ OF column_name [ ,...n ] ] ]
[;]
```

Štandardne: FORWARD_ONLY => FETCH NEXT FROM kur
SCROLL => FETCH FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE

```
FETCH
          [ [ NEXT | PRIOR | FIRST | LAST
                   | ABSOLUTE { n | @nvar }
                   | RELATIVE { n | @nvar }
               ]
               FROM
          ]
{ { [ GLOBAL ] cursor_name } | @cursor_variable_name }
[ INTO @variable_name [ ,...n ] ]
```

Skalárne funkcie pre kurzory

@@CURSOR_ROWS – konštanta vráti počet riadkov v kurzore

- 0 – a) kurzor nebol otvorený alebo bol zatvorený resp. dealokovaný
  b) ani jeden ridok. Vyskúšajte!
- -1 – kurzor je dynamický
- -n – počet asynchronne vrátených riadkov
- n – kurzor je úplne naplnený

@@FETCH_STATUS – používa sa v cykle na zistenie, či kurzor ešte obsahuje riadok

- 0 – FETCH príkaz bol úspešný
- -1 – FETCH príkaz bol úspešný alebo riadok bol za množinou výsledkov
- -2 – riadok chýba

Príklad 1: Vráťte prvú DB zo zoznamu DB pomocou sysdatabases a **kurzora.**

```
-- 1/3) Deklaracia
DECLARE db_cursor CURSOR  FOR
SELECT name FROM master.dbo.sysdatabases
      WHERE name NOT IN ('master','model','msdb')
-- 2/3) Otvorenie a pouzitie, ziskanie
OPEN db_cursor
-- Vysledok sa bud automaticky vypise alebo ulozi do premennej
-- DECLARE @jaj VARCHAR(50)
FETCH NEXT FROM db_cursor -- INTO @jaj; print @jaj
-- 3/3) Zatvorenie a uvolnenie
CLOSE db_cursor
DEALLOCATE db_cursor
```

Príklad 2: Pokračovanie – vráťme zoznam všetkých (okrem niektorých systémových) DB pomocou kurzora - **cyklus**.

```
-- 1/3)
DECLARE db_cursor CURSOR
    FORWARD_ONLY
FOR
SELECT name FROM master.dbo.sysdatabases
                  WHERE name NOT IN ('master','model','msdb')
-- 2/3)
OPEN db_cursor
DECLARE @dbName VARCHAR(50)
FETCH NEXT FROM db_cursor INTO @dbName
print cast(@@CURSOR_ROWS as char(10))
WHILE @@FETCH_STATUS = 0
BEGIN
       print @dbName
       FETCH NEXT FROM db_cursor INTO @dbName
END
```

```
-- 3/3)
CLOSE db_cursor
DEALLOCATE db_cursor
```

Príklad 3: Vráťte názvy všetkých (okrem niektorých) databáz s prislušnými tabuľkami pomocou **vnorených cyklov**.

```
-- Pomocou dvoch kurzorov, pritom jeden je retazcovy prikaz.
-- Vysledok do MESSAGES!!!
DECLARE @db VARCHAR(255)
DECLARE @tab VARCHAR(255)
DECLARE @cmd NVARCHAR(500)

DECLARE kurDB CURSOR FOR
SELECT name FROM master.dbo.sysdatabases
      WHERE name NOT IN ('master', 'msdb', 'model', 'ReportServer',
                  'ReportServerTempDB', 'tempdb') and name !=
                 'AdventureWorks2014'
      ORDER BY name

OPEN kurDB

FETCH NEXT FROM kurDB INTO @db
WHILE @@FETCH_STATUS = 0
BEGIN
   print ''
   print @db + ':'
   SET @cmd = 'DECLARE kurTab CURSOR FOR SELECT table_name FROM ['
                                                    + @db +
                                              '].INFORMATION_SCHEMA.TABLES'
   EXEC (@cmd)  -- vytvor kurTab!
   OPEN kurTab

   FETCH NEXT FROM kurTab INTO @tab
   WHILE @@FETCH_STATUS = 0
   BEGIN
       print   '  ' + @tab
       FETCH NEXT FROM kurTab INTO @tab
   END

   CLOSE kurTab   DEALLOCATE kurTab

   FETCH NEXT FROM kurDB INTO @db
END
CLOSE kurDB   DEALLOCATE kurDB
```

Riešenie bez kurzora

Každej DB zodpoveajúce tabuľky (nedokumentovaná SP).

```
sp_msforeachdb 'select "?" AS db, * from [?].sys.tables'
```

Príklady na cvičení:

- Vkladať riadky do tabuľky pomocou kurzoru:
- Vytvoriť tabuľku t1 s tromi riadkami a tabuľku t2 do ktorej prenesiete riadky pomocou kurzora
- Stoličky

3) Kontingenčné/Pivot tabuľky

3a) Klauzula PIVOT
3b) Manualny PIVOT
3c) Dynamický - poloautomatický Pivot

PT je **dvojrozmerná agregačná** (frekvenčná, sumárna, spriemerná, ...) tabuľka, zovšeobecňujúca GROUP BY podľa dvoch stĺpcov, atribútov. PT je dôležitý nástroj pre získavanie informácií v DB a aplikáciach na dolovanie dát.

T-SQL podporuje PT pomocou operátorov PIVOT a UNPIVOT (opačná oprácia – vráti štíhlejší výsledok).

Pivot tabuľky môžeme vytvoriť
a) manuálne PIVOT ... IN ...
b) Automatický PIVOT ... IN pomocou reťazcových príkazov `QUOTENAME, XML PATH, STUFF`
c) bez Pivot pomocou (pozri ZS) `SUM(CASE WHEN ...`
d) Pivot tabuľky a Excel, R

a) Pivot tabuľky manuálne PIVOT ... IN ...
0) Tabuľka #T1
1) Pivot:
(Na cvičení:)
2a) Dodajme Vcelku:
2b) Dodajme Vcelku s odstránením NULL Vcelku:
2c) Dodajme Vcelku s úplným odstránením NULL (+ názvy stĺpcov):
3a) Bez pivot:
3b) Bez pivot: SUM(CASE ...

Príklad. Vytvorme tabuľku `##T1 na pivotovanie.`

```
USE tempdb;
if OBJECT_ID('##T1', 'U') IS NOT NULL DROP TABLE ##T1
CREATE TABLE ##T1(Oddel Char, Rok SMALLINT, Kvartal TINYINT, Obrat
DECIMAL(2,1))
GO
 INSERT INTO ##T1 (Oddel, Rok, Kvartal, Obrat)
 SELECT 'A', 2006, 1, 0.6 UNION ALL
 SELECT 'B', 2006, 1, 0.7 UNION ALL
 SELECT 'A', 2006, 3, 0.9 UNION ALL
 SELECT 'B', 2006, 3, 0.7 UNION ALL
 SELECT 'A', 2006, 4, 0.8 UNION ALL
 SELECT 'B', 2006, 4, 0.8 UNION ALL
 SELECT 'A', 2007, 1, 0.7 UNION ALL
 SELECT 'A', 2007, 2, 0.9 UNION ALL
 SELECT 'A', 2007, 2, 0.9 UNION ALL
 SELECT 'A', 2007, 3, 0.8 UNION ALL
 SELECT 'A', 2007, 3, 0.6 UNION ALL
 SELECT 'A', 2007, 4, 0.9 UNION ALL
 SELECT 'B', 2007, 4, 0.7;
```

Syntax-MS: npc ⇔ non-pivoted column, pc ⇔ pivoted column, cn ⇔ column name.

```
SELECT <npc>, [first pc] AS <cn>, [second pc] AS <cn>,...,[last pc] AS <cn>
  FROM
    (<SELECT query that produces the data>) AS <alias for the source query>
      PIVOT(<aggregation function>(<column being aggregated>)
      FOR
      [<column that contains the values that will become column headers>]
      IN ( [first pc], [second pc],...,[last pc])
      ) AS <alias for the pivot table>
  <optional ORDER BY clause>;
```

Sumárny obrat v jednotlivých kvartáloch v každom roku:

	Rok	kvart_1	2	3	4
1	2006	1.3	NULL	1.6	1.6
2	2007	0.7	1.8	1.4	1.6

```
---- a) Pivot manuálne – vo výsledku riadok, stĺpec:
SELECT Rok, [1] kvart_1,[2],[3],[4] FROM
   (SELECT Rok, Kvartal, Obrat FROM ##T1) pom
  ---- V hre budu/su tie stlpce, ktore vyssie vo vnorenom dopyte selektujeme.
  PIVOT (SUM(Obrat) FOR Kvartal IN ([1],[2],[3],[4])) piv
  ---- Z hodnot stlpca Kvartal tvorime maximalne 4 nove stlpce.
  -- ORDER BY Rok DESC
```

Sumárny obrat jednotlivých oddelení v každom roku:

```
SELECT Rok, [A],[B]
  FROM (SELECT Rok, Oddel, Obrat FROM ##T1)  pom
  PIVOT (SUM(Obrat) FOR Oddel IN ([A],[B])) piv
  ORDER BY Rok
```

	Rok	A	B
1	2006	2.3	2.2
2	2007	4.8	0.7

Dodajme stĺpec TOTAL (plus odstránenie prípadných NULL hodnôt)

```
SELECT Rok, ISNULL([A],0) A,ISNULL([B],0) B, [A]+[B] Total . . .
```

	Rok	A	B	Total
1	2006	2.3	2.2	4.5
2	2007	4.8	0.7	5.5

b) Automatický PIVOT ... IN pomocou reťazcových príkazov QUOTENAME, XML PATH, STUFF

Pri poloautomatickom riešení prvého príkladu budeme využívať nasledujúce funkcie (a tabuľku #T1): QUOTENAME, STUFF - maže a potom vkladá, FOR XML PATH a COALESCE.

```
SELECT QUOTENAME('Podme-domov')      -- [Podme-domov]
-- ⇔
SELECT QUOTENAME('Podme-domov',']') -- [Podme-domov]
```

FOR XML PATH - získanie výsledku SQL dotazu vo formáte XML (reťazec). ____
XML bude venovaná jedna prednáška.

```
print STUFF('Bxxxlava', 2, 3, 'ratis'); -- Bratislava
```

Postup:
1a) Zoznam všetkých hodnôt Kvartálu: -- 1 // 2 // 3 // 4
1b) Zoznam všetkých hodnôt Kvartálu v [ ] QUOTENAME : -- ,[1] // ,[2] // ,[3] // ,[4]
1c) Všetky hodnoty do jedného riadku FOR XML PATH('') -- ,[1],[2],[3],[4]
1d) Všetky hodnoty s odstránením prvej čiarky STUFF (…) -- [1],[2],[3],[4]
2c) Odstránenie NULL: COALESCE.

2c) ISNULL nahradí NULL zadanou náhradnou hodnotou.

```
---- Pivot kódovo, poloautomaticky:
DECLARE @zoznam VARCHAR(100)
-- Vrati zoznam: [1],[2],[3],[4]
--  1a) SELECT DIST., 1b) SELECT QUOTEN., 1c) FOR XML PATH, id) STUFF
SET @zoznam =
STUFF(
(
      SELECT ',' + QUOTENAME(x)
        FROM (
                SELECT DISTINCT(Kvartal) x
                              FROM( SELECT * FROM ##T1)
                                    AS pom
            ) pom
            ORDER BY x
      FOR XML PATH('') -- ,[1],[2],[3],[4]
),1, 1, N''); -- maze z predu ciarku - nahradi s prazdnym znakom

print @zoznam          -- ⇔  [1],[2],[3],[4]

DECLARE @sql AS NVARCHAR(MAX)
SET @sql = N'
SELECT Rok, ' + @zoznam + '
  FROM (SELECT Rok, Kvartal, Obrat FROM ##T1)pom
  PIVOT ( SUM(Obrat) FOR Kvartal IN (' + @zoznam + ') ) piv
  ORDER BY Rok'

print @sql

EXEC sp_executesql @sql;
```

	Rok	1	2	3	4
1	2006	1.3	NULL	1.6	1.6
2	2007	0.7	1.8	1.4	1.6

Nahraďme NULL s 0 pomocou COALESCE, ktorý vráti prvú nie null hodnotu.
Napr.

```
SELECT N FROM ( VALUES(-1),(NULL),(-5) ) xxxTab(N)
SELECT COALESCE(N,0) FROM ( VALUES(-1),(NULL),(-5) ) xxxTab(N)
SELECT COALESCE(N,null,null,55) FROM ( VALUES(-1),(NULL),(-5) ) xxxTab(N)

SET @sql = N'
SELECT Rok,
 COALESCE([1],0)[1], COALESCE([2],0)[2],COALESCE([3],0)[3],COALESCE([4],0)[4],
 COALESCE([1],0)+COALESCE([2],0)+ COALESCE([3],0)+ COALESCE([4],0)Vcelku
 FROM (SELECT Rok, Kvartal, Obrat FROM ##T1)pom
  PIVOT ( SUM(Obrat) FOR Kvartal IN (' + @zoznam + ') ) piv
  ORDER BY Rok'
```

	Rok	1	2	3	4	Vcelku
1	2006	1.3	0.0	1.6	1.6	4.5
2	2007	0.7	1.8	1.4	1.6	5.5

c) Pivot tabul'ky bez Pivot

Pomocou `SUM(CASE WHEN Kvartal = 1 THEN ...`

```
SELECT Rok, SUM(Obrat) AS Vcelku
  FROM ##T1
    GROUP BY Rok
  go
```

	Rok	Vcelku
1	2006	4.5
2	2007	5.5

```
SELECT Rok, Kvartal, SUM(Obrat) AS Vcelku
  FROM ##T1
    GROUP BY Rok, Kvartal
       Order by rok,Kvartal
  Go
```

	Rok	Kvartal	Vcelku
1	2006	1	1.3
2	2006	3	1.6
3	2006	4	1.6
4	2007	1	0.7
5	2007	2	1.8
6	2007	3	1.4
7	2007	4	1.6

```
SELECT Rok,
        SUM(CASE WHEN Kvartal = 1 THEN Obrat ELSE 0 END) [Kv 1],
        SUM(CASE Kvartal WHEN   2 THEN Obrat ELSE 0 END) [Kv 2],
        SUM(CASE WHEN Kvartal = 3 THEN Obrat ELSE 0 END) [Kv 3],
        SUM(CASE WHEN Kvartal = 4 THEN Obrat ELSE 0 END) [Kv 4],
        SUM(Obrat) AS Vcelku
   FROM ##T1
     GROUP BY Rok
```

	Rok	Kv 1	Kv 2	Kv 3	Kv 4	Vcelku
1	2006	1.3	0.0	1.6	1.6	4.5
2	2007	0.7	1.8	1.4	1.6	5.5

d) Pivot tabuľky a Excel

- Power pivot – milióny riadkov
- Power view