Laboratorium 2 – MS SQL Server 2008

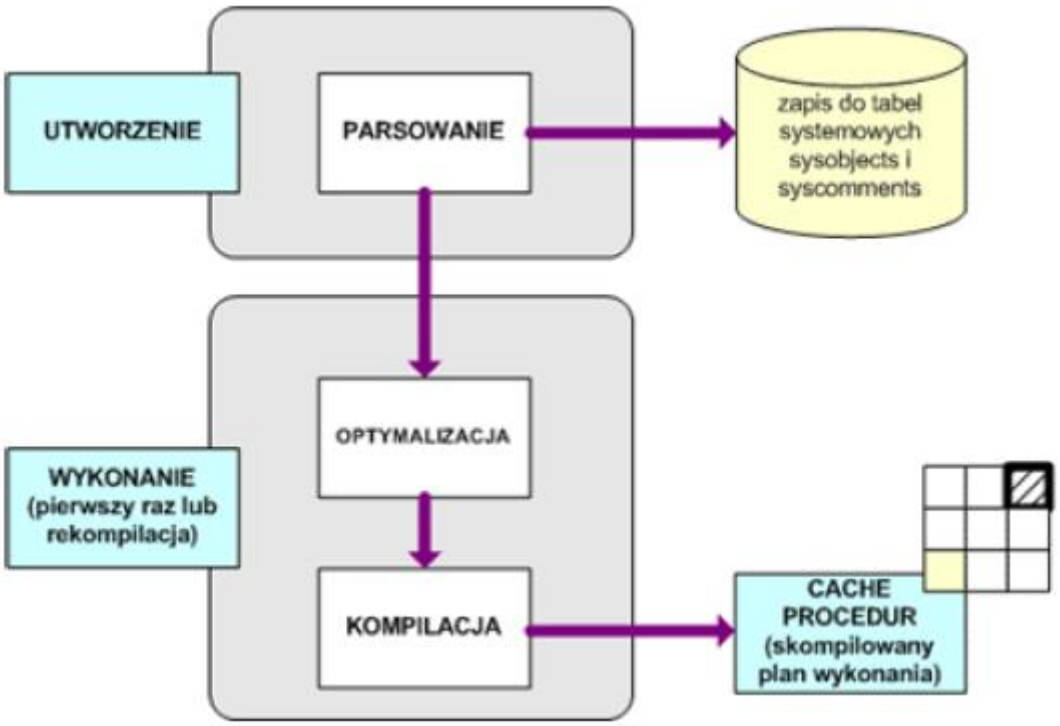
Temat: Język T-SQL

Opracowanie: A.Dydejczyk

Proces wykonywania zapytania przez SQL Server.

Proces wykonania pojedynczego zapytania w języku T-SQL w Microsoft SQL Server 2008:

1. Sprawdzenie i rozdzielenie kodu na fragmenty (często nazywane symbolami) interpretowane przez SZBD. Proces ten nazywamy analizą leksykalną.
2. Sprawdzenie kodu pod względem poprawności składni (kontrola poprawności semantycznej – czyli czy kod nie odwołuje się do nieistniejących obiektów lub używa nieistniejących poleceń oraz kontrola poprawności syntaktycznej – czy użyta składnia jest poprawna).
3. Standaryzacja wyodrębnionej części kodu, tzn. SZBD zapisuje go w jednoznacznej postaci (usuwając niepotrzebne znaczniki).
4. Optymalizacja – każde zapytanie może posiadać wiele przygotowanych tzw. *planów wykonania* (ang. *execution plan*). MS SQL Server posiada wewnętrzny proces zwany *Optymalizatorem Zapytań*, który wybiera optymalny sposób dostępu do danych, tzn. taki plan wykonania zapytania, w którym serwer będzie skanował (przeszukiwał) najmniejszą ilość stron danych. Na optymalizację szczególny wpływ mają struktura indeksów oraz sposób łączenia tabel.
5. Kompilacja zapytania wg optymalnego planu wykonania i wykonanie skompilowanego zapytania.
6. Zwrócenie wyniku działania zapytania do klienta.



Schemat dostępu do danych w SQL server 2008.

Proces wykonywania procedury składowanej.

Wykonywanie procedur składowanych odbywa się inaczej niż wykonywanie pojedynczych zapytań SQL. Poniżej prezentujemy schemat utworzenia i pierwszego wykonania procedury na przykładzie MS SQL Server:

1. Tworzenie definicji procedury składowanej – tzn. wykonanie polecenie CREATE PROCEDURE.
2. Sprawdzanie kodu procedury pod względem syntaktyki.
3. Zapisanie nazwy procedury i jej kodu (tzw. ciało) do odpowiednich widoków systemowych bazy danych (sysobjects oraz syscomments).
4. Wywołanie procedury składowanej przez użytkownika z odpowiednimi parametrami używając polecenia EXEC.
5. Właściwe wykonanie procedury – optymalizacja planu wykonania i kompilacja.
6. Skompilowany optymalny plan wykonania jest zapisywany w tzw. buforze (z ang. *cache'u)* procedur.

Problemy i zagadnienia omawiane w trakcie kolejnych zajęć laboratoryjnych związane z językiem T-SQL.

A. Rozszerzenia języka SQL w ramach bazy danych MS-SQL (lab02)
    a. Funkcje pozycjonujące („rankingu”): ROW_NUMBER, RANK, DENSE_RANK, NTILE
    b. Operator GROUPING SETS
    c. Operatory PIVOT i UNPIVOT
    d. Recursive Query Expressions
B. Funkcje definiowane przez użytkownika (User Define Function) (lab03)
C. Procedury składowane (lab04)
D. Uwierzytelnienie i autoryzacja (lab05)

Ćwiczenie A. ROW_NUMBER, RANK, DENSE_RANK, NTILE

Link do stron z opisem technologii:

- *Stairway to Advanced T-SQL Level 7: Ordering Your Data Using Ranking Functions* *http://www.sqlservercentral.com/articles/Stairway+Series/130313/*
- *Funkcje RANK() oraz DENSE_RANK()* *http://www.sqlpedia.pl/funkcja-rank-dense-rank/*

RANK

Zwraca pozycję dla każdego wiersza w określonej części zbioru wynikowego

DENSE_RANK

Zwraca następną pozycję dla każdego wiersza w określonej części zbioru wynikowego

ROW_NUMBER

Zwraca pozycję porządkową wiersza dla każdego wiersza w pogrupowanym zbiorze wynikowym

NTILE

Dzieli wiersze w każdej części zbioru wynikowego na określoną liczbę pozycji opierając się na wartościach.

Funkcje wymagają określenie według jakich parametrów mają działać. W tym celu dodajemy klauzulę OVER z kolumnami, po których chcemy, aby dane działanie było wykonywane. Użycie klauzul numerujących jest jak widać proste, trudniejszą sprawą może być wytłumaczenie, na jakiej zasadzie działają poszczególne funkcje pozycjonujące. Dla ułatwienia przykład.

W ramach tego punktu realizujemy zapytania w bazie danych : AdventureWorks.

```
USE AdventureWorks
GO
```

1. Przykład: Wykorzystanie funkcji pozycjonującej Row_Number()

```
--- Skrypt Lab2.01
SELECT Row_Number() OVER (ORDER BY [LastName]) AS RowNumber
      ,[ContactID]
      ,[FirstName]
      ,[LastName]
FROM [Person].[Contact]
```

2. Wykorzystanie funkcji rankingu Rank(), Dense_Rank() i NTile()

```
--- Skrypt Lab02.02
SELECT Row_Number() OVER (ORDER BY [LastName]) AS RowNumber
      ,Rank() OVER (ORDER BY [LastName]) AS Rank
      ,Dense_Rank() OVER (ORDER BY [LastName]) AS DenseRank
      ,NTile(3) OVER (ORDER BY [LastName]) AS NTile_3
```

```
        ,NTile(4) OVER (ORDER BY [LastName]) AS NTile_4
        ,[ContactID]
        ,[FirstName]
        ,[LastName]
FROM [Person].[Contact]
```

Ćwiczenie B. Wykorzystanie operatora GROUPING SETS w MS SQL 2008

Link do stron z opisem technologii:

- *Stairway to T-SQL DML Level 8: Using the ROLLUP, CUBE and GROUPING SET operator in a GROUP BY Clause*
  http://www.sqlservercentral.com/articles/Stairway+Series/87629/
- *Użycie klauzuli GROUP BY z operatorami ROLLUP, CUBE i klauzulą GROUPING SETS*
  https://technet.microsoft.com/pl-pl/library/bb522495(v=sql.105).aspx

1. Wykonujemy poniższe zapytanie w bazie danych **AdventureWorks**

```
--- Skrypt Lab02.04
USE AdventureWorks
GO
        SELECT MONTH(OrderDate),TerritoryID ,CustomerID,SUM(TotalDue)
        FROM Sales.SalesOrderHeader
        WHERE SalesPersonID=288 AND YEAR(OrderDate)=2004
        GROUP BY MONTH(OrderDate),TerritoryID,CustomerID
UNION ALL
        SELECT MONTH(OrderDate),TerritoryID,NULL,SUM(TotalDue)
        FROM Sales.SalesOrderHeader
        WHERE SalesPersonID=288 AND YEAR(OrderDate)=2004
        GROUP BY MONTH(OrderDate),TerritoryID
UNION ALL
        SELECT MONTH(OrderDate),NULL,CustomerID,SUM(TotalDue)
        FROM Sales.SalesOrderHeader
        WHERE SalesPersonID=288 AND YEAR(OrderDate)=2004
        GROUP BY MONTH(OrderDate),CustomerID
UNION ALL
        SELECT NULL,NULL,NULL,SUM(TotalDue)
        FROM Sales.SalesOrderHeader
        WHERE SalesPersonID=288 AND YEAR(OrderDate)=2004
```

2. Wykonujemy zapytanie z wykorzystaniem operatora GROUPING SETS

```
--- Skrypt Lab02.05
SELECT MONTH(OrderDate),TerritoryID ,CustomerID,SUM(TotalDue)
FROM Sales.SalesOrderHeader
WHERE SalesPersonID=288 AND YEAR(OrderDate)=2004
GROUP BY GROUPING SETS (
            (MONTH(OrderDate), TerritoryID,CustomerID),
            (MONTH(OrderDate), TerritoryID),
            (MONTH(OrderDate), CustomerID),
                                        () );
```

3. Analiza wykonanych zapytań w bazie – wykorzystujemy analizator planów zapytań.

Ćwiczenie C. Wykorzystanie operatorów PIVOT i UNPIVOT

Link do stron z opisem technologii:

- *Stairway to Advanced T-SQL Level 5: Turning Data On Its Side Using PIVOT Operator* http://www.sqlservercentral.com/articles/Stairway+Series/125187/
- *Stairway to Advanced T-SQL Level 6: Creating Rows Of Data Using The UNPIVOT Operator* http://www.sqlservercentral.com/articles/Stairway+Series/125504/
- *Używanie operatorów PIVOT i UNPIVOT* https://technet.microsoft.com/pl-pl/library/ms177410(v=sql.105).aspx

Ogólna postać zastosowania operatora **PIVOT**

```
SELECT
    [non-pivoted column], -- optional
    [additional non-pivoted columns], -- optional
    [first pivoted column],
    [additional pivoted columns]
FROM (
    SELECT query producing sql data for pivot
    -- select pivot columns as dimensions and
    -- value columns as measures from sql tables
) AS TableAlias
PIVOT
(
  <aggregation function>(column for aggregation or measure column)
  -- MIN,MAX,SUM,etc
  FOR [ ]
  IN (
    [first pivoted column], ..., [last pivoted column]
  )
) AS PivotTableAlias
ORDER BY clause – optional
```

Operatory PIVOT i UNPIVOT dają możliwość tworzenia tabel przestawnych w serwerach począwszy od wersji 2005. Wcześniej należało użyć wyrażenia CASE.

Zadania w tym punkcie realizujemy z wykorzystaniem bazy danych AdventureWorks.

1. Tabela przestawna dla sprzedaży w poszczególnych miesiącach.

```
--- Skrypt Lab02.08
CREATE TABLE SalesOrderTotalsMonthly
(
    CustomerID int NOT NULL,
    OrderMonth int NOT NULL,
    SubTotal money NOT NULL
)
```

```
GO
INSERT SalesOrderTotalsMonthly
        SELECT CustomerID, DATEPART(m, OrderDate), SubTotal
        FROM Sales.SalesOrderHeader
        WHERE CustomerID IN (1,2,4,6)
GO
SELECT * FROM SalesOrderTotalsMonthly
PIVOT (SUM(SubTotal) FOR OrderMonth IN
([1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12])) AS a
```

2. Tabele przestawne dla sprzedaży w ciągu kilku lat.

```
--- Skrypt Lab02.09
CREATE TABLE SalesOrderTotalsYearly
(
        CustomerID int NOT NULL,
        OrderYear int NOT NULL,
        SubTotal money NOT NULL
)
GO

INSERT SalesOrderTotalsYearly
        SELECT CustomerID, YEAR(OrderDate), SubTotal
        FROM Sales.SalesOrderHeader
        WHERE CustomerID IN (1,2,4,6,35)
GO
SELECT * FROM SalesOrderTotalsYearly
PIVOT (SUM(SubTotal) FOR OrderYear IN ([2002], [2003], [2004])) AS a
GO

SELECT * FROM SalesOrderTotalsYearly
PIVOT (SUM(SubTotal) FOR CustomerID in ([1], [2], [4], [6])) AS a
```

3. Wykorzystane operatora UNPIVOT do prezentacji danych w postaci tabeli transakcji.

```
--- Skrypt Lab02.10
CREATE TABLE YearlySalesPivot
(
        OrderYear int NOT NULL,
        [1] money NULL,
        [2] money NULL,
        [4] money NULL,
        [6] money NULL
)
GO

INSERT YearlySalesPivot
        SELECT * FROM SalesOrderTotalsYearly
        PIVOT (SUM(SubTotal) FOR CustomerID IN ([1], [2], [4], [6])) AS a
GO

SELECT * FROM YearlySalesPivot
UNPIVOT (SubTotal FOR CustomerID IN ([1], [2], [4], [6])) AS a
ORDER BY CustomerID
GO
```

```
SELECT * FROM YearlySalesPivot
```

4. Usuwamy utworzone w czasie ćwiczenia tabele.

```
DROP TABLE YearlySalesPivot
DROP TABLE SalesOrderTotalsYearly
DROP TABLE SalesOrderTotalsMonthly
```

Ćwiczenie D. Wyrażenia tabelaryczne (CTE) i zapytania rekurencyjne

Link do stron z opisem technologii:

- *WITH common_table_expression (Transact-SQL)*
  https://msdn.microsoft.com/en-us/library/ms175972.aspx
- *SQL Server CTE Basics*
  https://www.simple-talk.com/sql/t-sql-programming/sql-server-cte-basics/
- *Stairway to Advanced T-SQL Level 3: Understanding Common Table Expressions (CTEs)*
  http://www.sqlservercentral.com/articles/Stairway+Series/122606/
- *Arsenał programisty T-SQL – Common Table Expression (CTE)*
  https://technet.microsoft.com/pl-pl/library/arsenal-programisty-t-sql--common-table-expression-cte.aspx
- *CTE – Common Table Expressions*
  http://www.sqlpedia.pl/cte-common-table-expressions/

Ogólna postać wyrażenia tabelarycznego została przedstawiona poniżej.

```
WITH nazwa_wirtualnej_tabeli
AS
(
        zapytanie
)
```

Wyrażenia CTE służą jako wirtualne tabele lub perspektywy, z zakresem ważności danej sesji, bez odniesień do **tempdb** (tabel temporary). W ramach ćwiczenia wszystkie punkty zostaną wykonane w bazie danych AdventureWorks.

```
USE AdventureWorks
GO
```

1. Tworzenie wyrażenia tabelarycznego CTE (Common Table Expressions).

```
--- Skrypt Lab02.11
WITH SalesCTE(ProductID, SalesOrderID)
AS
(
        SELECT ProductID, COUNT(SalesOrderID)
        FROM Sales.SalesOrderDetail
        GROUP BY ProductID
)
SELECT * FROM SalesCTE
```

```
-- All Products and the number of times that they were ordered
```

2. Wykorzystanie klauzuli WHERE w wyrażeniu tabelarycznym.

```
--- Skrypt Lab02.12
WITH SalesCTE(ProductID, SalesOrderID)
AS
(
        SELECT ProductID, COUNT(SalesOrderID)
         FROM Sales.SalesOrderDetail
        GROUP BY ProductID
)
SELECT * FROM SalesCTE
WHERE SalesOrderID > 50
-- All Products that were ordered more than 50 times
```

3. Wykorzystanie funkcji agregujących w wyrażeniu tabelarycznym.

```
--- Skrypt Lab02.13
WITH SalesCTE(ProductID, SalesOrderID)
AS
(
        SELECT ProductID, COUNT(SalesOrderID)
        FROM Sales.SalesOrderDetail
        GROUP BY ProductID
)
SELECT AVG(SalesOrderID)
FROM SalesCTE
WHERE SalesOrderID > 50
-- Average number of times a Product was ordered
-- for all Products that appeared on an order
-- more than 50 times
```

4. Wyświetlenie relacji pomiędzy pracownikiem i jego kierownikiem w bazie danych AdvantureWork.

```
--- Skrypt Lab02.14
SELECT FirstName, LastName, EmployeeID, ManagerID
FROM HumanResources.Employee
JOIN Person.Contact
ON HumanResources.Employee.ContactID= Person.Contact.ContactID
```

5. Prezentacja danych pracownika i jego kierownika.

```
--- Skrypt Lab02.15
WITH EmployeeManager AS
(
        SELECT FirstName, LastName, EmployeeID, ManagerID
        FROM HumanResources.Employee
        JOIN Person.Contact
        ON HumanResources.Employee.ContactID= Person.Contact.ContactID
)
SELECT emp.FirstName+'  '+emp.LastName as Employee ,
    Man.FirstName+' '+man.LastName as Manager
FROM EmployeeManager emp
LEFT OUTER JOIN EmployeeManager man
ON emp.ManagerID=man.EmployeeID
```

6. Rekursywne przetwarzanie zapytań w bazie danych. Część nierekursywna – informacja o pracowniku, który nie ma szefa.

```
--- Skrypt Lab02.16
WITH EmployeeManager AS
(
        SELECT FirstName+'  '+LastName as Employee,
                FirstName+'  '+LastName as Manager,
                ManagerID, EmployeeID, 0 AS EmployeeLevel
        FROM HumanResources.Employee
        JOIN Person.Contact
        ON HumanResources.Employee.ContactID= Person.Contact.ContactID
        WHERE ManagerID IS NULL
)
SELECT  Employee , Manager
FROM EmployeeManager
```

7. Wyrażenie tabelaryczne zwracające pracowników na wszystkich stopniach hierarchii. Uwaga, wyrażenie może się zapętlić.

```
--- Skrypt Lab02.17
WITH EmployeeManager AS
(
        SELECT FirstName+'  '+LastName as Employee,
                FirstName+'  '+LastName as Manager,
                ManagerID, EmployeeID, 0 AS EmployeeLevel
        FROM HumanResources.Employee
        JOIN Person.Contact
        ON HumanResources.Employee.ContactID= Person.Contact.ContactID
        WHERE ManagerID IS NULL
        UNION ALL
        SELECT FirstName+'  '+LastName as Employee,
                man.Employee as Manager,
                emp.ManagerID, emp.EmployeeID,
                man.EmployeeLevel+1  AS EmployeeLevel
        FROM HumanResources.Employee emp
        JOIN Person.Contact
        ON emp.ContactID=Person.Contact.ContactID
           JOIN EmployeeManager man
        ON emp.ManagerID=man.EmployeeID
)
SELECT  Employee , Manager
FROM EmployeeManager
```

Obrona przed zapętleniem, ograniczenie liczby zwracanych poziomów poprzez podanie poziomu rekurencji.

```
OPTION (MAXRECURSION 2)
```

Zadania

Zadanie Z1 (funkcje rankingowe):

a) *Wyjaśnić działanie operatora WHERE (RowNumber > 51) AND (RowNumber < 100)*
b) *Zrealizować („stronicowanie”)*
   i. *Użyć tabeli temporary, wykorzystać INSERT from SELECT i odpytać tabelę temporary.*
   ii. *Wykorzystać wyrażenie CTE*

Zadanie Z2 (funkcje grupujące):

Utwórz raport podający ilość dostawców o adresie głównej siedziby podzielony wg stanu i miasta. Dane pochodzą z tabel Purchasing.Vendor, Purchasing.VendorAddress, Person.Address, PersonStateProvince.

Zadanie Z3 (funkcja PIVOT):

a) *Opracuj zapytanie dla przykładów z punktu 1 lub 2 z wykorzystaniem operatora CASE.*

b) *W bazie tempdb (powinna być, jeżeli nie utworzyć) utworzyć tabelę z kolumnami dla godziny i minuty pomiaru (dwie osobne kolumny) oraz dwie następne kolumny dla wartości mierzonych zawartość CO2 i ilość przejeżdżających pojazdów. Wypełnić tabelę danymi. Wykorzystać PIVOT dla pokazania agregatów MIN, MAX, SUM w kolejnych godzinach (kolejne godziny powinny być w nagłówkach) dla obu mierzonych wartości.*