AUTOMATYKA • 2010 • Tom 14 • Zeszyt 1

Łukasz Mazur*

Modelowanie wielowymiarowych wyrażeń w języku MDX

1. Wprowadzenie

Język MDX (*Multidimensional Expressions*) jest językiem specjalnie utworzonym w celu definiowania danych wielowymiarowych, jak też operowania na nich. Jest także narzędziem służącym do określania sposobu prezentacji danych o dużej złożoności strukturalnej. Na istniejącym rynku literatury informatycznej znajduje się stosunkowo mało opracowań zagadnień dotyczących korzystania z języka MDX. Poruszana w artykule tematyka modelowania jest bardzo istotna podczas wykonywania skomplikowanej analizy dużej ilości danych, w środowiskach przetwarzania analitycznego. Z tego względu, autor stworzył opracowanie wybranych konstrukcji modelowych, wraz z przykładowymi schematami ich zastosowań. Opracowanie to zawiera zbiorcze zestawienie szeregu funkcji podzielonych na poszczególne kategorie, w których zostały zaproponowane przypadki ich użycia. Dodatkowo załączono przemyślenia autora wraz z wynikającymi z nich wnioskami. Przedstawiono także przykładowe modele konstrukcji analitycznych wykorzystywanych w środowiskach produkcyjnych firm czy przedsiębiorstw. Opisywana tematyka analiz głównie zorientowana jest na aspektach budżetowych, lecz z powodzeniem może być wykorzystywana także w innych sferach analiz dla podmiotu gospodarczego.

Bardzo istotną sprawą, która zasygnalizowała powstanie tego nowego języka, jest zorientowanie na przetwarzanie wielowymiarowe oraz zorientowanie na tematyczność danych. Jest to odmienną cechą od typowego języka zapytań bazodanowych SQL, w którym utworzenie podobnych konstrukcji jest bardzo kłopotliwe, a w wielu przypadkach wręcz niemożliwe. Obecnie MDX staje się odrębnym standardem rozwijającym się niezależnie od języka SQL.

2. Bazowe elementy składni

2.1. Zapytanie

Zapytanie w MDX, podobnie jak w SQL, zawiera frazę SELECT (definiującą dane wynikowe), frazę FROM (definiującą kostkę wejściową) oraz frazę WHERE (określającą

* Doktorant, Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie

filtrowanie danych). Dostępnych jest także wiele funkcji służących do wyszukiwania danych i do operowania danymi. Szczegółowo zostały one opisane w dalszej części pracy. Możliwe jest także rozszerzanie funkcjonalności języka przez definiowanie własnych funkcji. Podstawowe wyrażenie wybierające dane MDX ma składnię przedstawiającą się następująco:

Definicja 1

```
SELECT <SPECIFIKACJA_OSI>
FROM   <SPECIFIKACJA_KOSTKI>
WHERE  <SPECIFIKACJA_PLASTRA>
```

Fraza SELECT określa wynikowy zbiór komórek oraz sposób jego prezentacji. Prezentacja zbioru określona jest poprzez przyporządkowanie elementom osi(*axis*) punktów przestrzeni wielowymiarowej. Istnieje maksymalnie do 128 osi, którym mogą być przypisane numery, a ponadto pięć pierwszych osi ma przypisane dodatkowo nazwy [7]:

0 – oś X (COLUMNS),

1 – oś Y (ROWS),

2 – oś Z (PAGES),

3 – SECTIONS,

4 – CHAPTERS.

W każdym przypadku oś może być identyfikowana za pomocą wyrażenia **AXIS**(INDEKS), gdzie INDEKS jest liczbą całkowitą z przedziału [0, 127]. Zbiory punktów mogą być podawane jawnie lub specyfikowane z wykorzystaniem szeregu funkcji. Specyfikacja osi ma następującą składnię:

Definicja 2

```
<SPECIFIKACJA_OSI> ::= <ZBIÓR_PUNKTÓW> ON <NAZWA_OSI>
<NAZWA_OSI>::= COLUMNS | ROWS | PAGES | SECTIONS | CHAPTERS | AXIS(<INDEKS>)
```

2.2. Definicja modelu kostki

Za pomocą języka MDX można zarówno definiować strukturę kostki, wypełniać danymi oraz wykonywać określone obliczenia.

Definicja przykładowej struktury kostki analitycznej o nazwie [KOSTKA ANALITYCZNA] ma postać [7]:

Listing 1

```
CREATE CUBE [KOSTKA ANALITYCZNA]      //utworzenie kostki.
   (DIMENSION WYM_CZAS TYPE TIME,     //tworzenie poszczególnych wymiarów.
    HIERARCHY [FISKALNY],             //tworzenie hierarchii w wymiarze.
        LEVEL [FISKALNY ROK] TYPE YEAR,  //tworzenie poziomów hierarchii.
        LEVEL [FISKALNY KWARTAL] TYPE QUARTER,
        LEVEL [FISKALNY MIESIAC] TYPE MONTH,
    HIERARCHY [KALENDARZOWY],
        LEVEL [KALENDARZOWY ROK] TYPE YEAR,
        LEVEL [KALENDARZOWY MIESIAC] TYPE MONTH,
       .
       .
       .
 MEASURE [$ SPRZEDAZ]                 //tworzenie miar
    FUNCTION SUM
    FORMAT 'CURRENCY'
 MEASURE [LICZBASPRZEDANYCH]
    FUNCTION SUM)
```

Jednym ze sposobów wypełniania danymi modelu analitycznego kostki jest realizacja poniższego skryptu (procedury). Zbiór wynikowy z zapytania wybierającego dla kostki o nazwie `[KOSTKA ANALITYCZNA]` przypisywany jest instrukcją **INSERT INTO** kostce o nazwie `[KOSTKA SPRZEDAZOWA]`:

Listing 2

```
INSERT INTO [KOSTKA SPRZEDAZOWA]                    //wstawianie danych
   (WYM_CZAS.ROK,
   WYM_CZAS.KWARTAŁ,
   WYM_CZAS.MIESIAC,
   WYM_PRODUKT.[NAZWA],
   MEASURES.MR_ILOSC)
     SELECT
        [KOSTKA ANALITYCZNA].[WYM_CZAS:ROK],
          [KOSTKA ANALITYCZNA].[WYM_CZAS:KWARTAŁ],
          [KOSTKA ANALITYCZNA].[WYM_CZAS.MIESIAC],
          [KOSTKA ANALITYCZNA].[WYM_PRODUKT:NAZWA],
          [KOSTKA ANALITYCZNA].[MEASURES:MR_ILOSC]
     FROM [KOSTKA ANALITYCZNA]
```

Rezultat obliczeń wyrażenia MDX w kostce analitycznej jest zależny od bieżącego kontekstu kostki, czyli tak zwanej perspektywy, z której widzimy model. Jeśli za pomocą filtrowania wytniemy fragment kostki (np. wartość sprzedaży danego produktu w danym kraju dla danego sprzedawcy), wyrażenie MDX wyliczy wartości dla wybranej części modelu analitycznego kostki.

W pracy przyjęta została zaproponowana przez autora konwencja nazewnicza, która będzie w tekście dalej używana dla miar, wymiarów, zbiorów nazwanych oraz elementów wyliczanych. Konwencja ta została opracowana w sposób następujący:

MR_*nazwa* – określenie nazwy dla tzw. miary fizycznej.
MK_*nazwa* – określenie nazwy dla miary kalkulowanej(obliczeniowej).
WYM_*nazwa* – określenie nazwy dla wymiaru.
NZ_*nazwa* – określenie nazwy dla nazwanego zbioru.
EO_*nazwa* – określenie nazwy dla elementów wyliczonych.

W podanych dalej przykładach nazwy będą składały się z wyszczególnionych przedrostków oraz nazwy właściwej. Przedrostki reprezentują właściwą funkcję w modelu analitycznym.

2.3. Ograniczanie zbioru wynikowego

Instrukcja SELECT określa wynikowy zbiór komórek oraz sposób jego prezentacji. Rozszerzeniem składni jest zastosowanie frazy WHERE, w której określane jest wyrażenie będące swego rodzaju filtrem ograniczającym zwracany zbiór komórek do pewnego plastra (*slicer*). Domyślnie przyjmuje się, że każdy wymiar, który nie został przyporządkowany do żadnej osi, reprezentowany jest we frazie WHERE poprzez swój człon główny o nazwie [ALL]. Człon ten zawiera wszystkie elementy potomne dla danego wymiaru. Jawne określenie innego członu powoduje, że tylko komórki budowane z wykorzystaniem tego członu należą do wynikowego zbioru komórek. Składnia frazy WHERE jest następująca [7]:

Definicja 3

```
[WHERE [<SPECIFIKACJA_PLASTRA>]]
     <SPECIFIKACJA_PLASTRA> ::= <PUNKT>
```

Przykład zastosowania filtrowania dla roku 2008 i miary MR_KOSZT:

Listing 3

```
WHERE ([WYM_CZAS].[ROK].&[2008], [MR_KOSZT])
```

Ograniczy to zbiór komórek do tych, które dotyczyły okresu w roku 2008. Umieszczenie miary MR_KOSZT we frazie WHERE oznacza, że tylko ta miara jest dla nas interesująca. Następujące wyrażenie zwraca zbiór komórek z odpowiednio zagregowanymi wartościami miara [LICZBAKLIENTOW] odnoszącymi się do roku 2007:

Listing 4

```
SELECT {[ALL SAMOCHOD], FIAT, FIAT.CHILDREN, FORD} ON COLUMNS,
       [WYM_REGION].[NAZWA].MEMBERS ON ROWS
FROM   [KOSTKA ANALITYCZNA]
WHERE  ([WYM_CZAS].[ROK].&[2007],[LICZBAKLIENTOW])
```

2.4. Operatory

Konstrukcje wyrażeń w MDX zawierają różnego typu operatory, które pomagają w tworzeniu konstrukcji. Poza typowymi operatorami matematycznymi istnieje kilka innych, które warto zaprezentować z względu na ich odmienny charakter. Ze względu na swoiste znaczenie podczas zapisu wyrażeń przedstawionych zostanie pięć operatorów:

NON EMPTY, „{}”, „**,**”, „**:**”, „&”.

W celu wyeliminowania niepowiązanych rekordów zastosować możemy operator **NON EMPTY**.

Listing 5

```
SELECT
       NON EMPTY [MEASURES].MEMBERS ON COLUMNS,
       NON EMPTY {[WYM_KATEGORIAPRODUKTU].[NAZWA KATEGORII].MEMBERS,
       [WYM_KATEGORIAPRODUKTU].[NAZWA TOWARU].MEMBERS} ON ROWS
FROM   [KOSTKA ANALITYCZNA]
```

Aby można było budować zbiory oraz manipulować nimi, używane są dodatkowe operatory, „**{ }**” – nawiasy, przechowujące elementy w tak zwanej krotce jako zbiór (Listing 6).

Listing 6

```
{
  [WYM_MIASTO].[NAZWA].&[KRAKÓW],
  [WYM_MIASTO].[NAZWA].&[POZNAŃ],
  [WYM_MIASTO].[NAZWA].&[GDAŃSK],
  [WYM_MIASTO].[NAZWA].&[KATOWICE],
  [WYM_MIASTO].[NAZWA].&[WROCŁAW]
}
```

Gdzie „ **,** „ – przecinki oddzielające elementy od siebie, a dwukropki „ **:** „ – oznaczają zakres elementów, czyli podzbiór elementów pomiędzy wartościami brzegowymi oznaczonymi elementami przed i po dwukropku.

Listing 7

```
{
  [WYM_MIASTO].[NAZWA].&[KRAKÓW]:[MIASTO].[NAZWA].&[WROCŁAW]
}
```

Symbol „ **&** ” – oznacza wyszczególnienie konkretnego elementu członkowskiego.

Listing 8

```
[WYM_MIASTO].[NAZWA].&[KRAKÓW]
```

3. Rozszerzone konstrukcje zapytań

MDX jest językiem zapytań w strukturach baz OLAP (*online analytical processing*), umożliwiającym zadawanie zapytań on-line w kostkach analitycznych. Pod pojęciem zapytań, podobnie jak w przypadku SQL, rozumiemy również możliwość modyfikowania danych, a więc w przypadku kostek analitycznych możliwość realizacji specyficznych obliczeń wykonywanych przed odpytaniem modelu. Obliczenia zrealizowane wstępnie, umieszczane są jako element składowy zapytania (np. miara kalkulowana) o określonej nazwie. Po przez te działania, włączane są do zbioru wynikowego w etapie następnym, czyli odpytania modelu. Obecnie język MDX staje się standardem przemysłowym, całkowicie niezależnym od języka SQL, posiada także swoiste cechy odróżniające go z znacznej mierze od poprzednika.

3.1. Klauzula WITH dla elementów

We frazie WITH można definiować miary obliczane oraz przypisywać nazwy zbiorom. Elementy obliczone mogą być traktowane jako funkcje tworzone na czas trwania zapytania. Rozszerzenie oraz uogólnienie przedstawionego szkieletu zapytania w części wstępnej artykułu będzie wyglądało jak przedstawiono poniżej:

Definicja 4

```
[WITH     <SPECIFIKACJA_FORMUŁY>  [,<SPECIFIKACJA_FORMUŁY>...]]
 SELECT   [<SPECIFIKACJA_OSI>  [,<SPECIFIKACJA_OSI>...]]
 FROM     [<SPECIFIKACJA_KOSTKI>]
[WHERE    [<SPECIFIKACJA_PLASTRA>]]
```

Zdefiniowany na początku przedstawionego zapytania element obliczony po frazie WITH, może być wykorzystywany w dalszej części jego definicji, poprzez wyszczególnienie odnoszącej się do niego nazwy. Unika się w ten sposób wielokrotnych obliczeń i zwiększa przejrzystość zapytania. Składnia tej frazy jest następująca:

Definicja 5

```
WITH <SPECIFIKACJA_FORMUŁY>[,<SPECIFIKACJA_FORMUŁY>  ...]
     <SPECIFIKACJA_FORMUŁY> ::= <SPECYFIKACJA_CZŁONU>|<SPECYFIKACJA_ZBIORU>
     <SPECYFIKACJA_CZŁONU>  ::= MEMBER<RODZIC_CZŁONU>.<NAZWA_CZŁONU>
      AS '<WYRAŻENIE>'
     <SPECYFIKACJA_ZBIORU> ::= SET<NAZWA_ZBIORU> AS '<ZBIÓR>'
```

Jako prezentację konstrukcji, poniższe wyrażenie definiuje miarę obliczaną `[MK_ZYSKPROCENTOWO]` i włącza go do zbioru wynikowego, który pokazuje jego rozkład na poszczególne regiony:

Listing 9

```
WITH
MEMBER   MEASURES.[MK_ZYSKPROCENTOWO] AS  // miara kalkulowana
       '([MEASURES].[MR_PRZYCHOD] - [MEASURES].[MR_KOSZT]
       )/[MEASURES].[MR_KOSZT] * 100'
SELECT {
       [MEASURES].[MR_PRZYCHOD],
       [MEASURES].[MR_KOSZT],
       [MK_ZYSKPROCENTOWO]} ON COLUMNS,  // elementy kolumny
       [WYM_LOKALIZACJA].[KRAJ].[REGION].MEMBERS ON ROWS // el. wierszy
FROM   [KOSTKA ANALITYCZNA]          // nazwa kostki
```

Spróbujmy utworzyć teraz zapytanie, które będzie podsumowaniem przychodu i kosztów w Krakowie i w Tarnowie. Efekt taki uzyskujemy tworząc postać modelu następująco:

Listing 10

```
WITH
  MEMBER [MEASURES].[MK_KRAKOW_I_TARNOW] AS
         'SUM({[KRAKOW], [TARNOW]})»
  SELECT {[PRZYCHOD], [KOSZT]} ON COLUMNS,
         {WYM_REGION.MEMBERS, [MK_KRAKOW_I_TARNOW]} ON ROWS
  FROM   [KOSTKA ANALITYCZNA]
```

W następnym schemacie użyjemy funkcji AGGREGATE, aby agregowanie poszczególnych miar było realizowane zgodnie z ich funkcjami agregującymi ustawionymi we właściwościach wymiaru:

Listing 11

```
WITH
  MEMBER [REGION].[MK_KRAKOW_I_TARNOW] AS
         'AGGREGATE({[KRAKOW],[TARNOW]})'
  SELECT {[PRZYCHOD],[DATAOSTSPRZ],[LICZBATRANS]} ON COLUMNS,
         {REGION.MEMBERS,[MK_KRAKOW_I_TARNOW]} ON ROWS
  FROM   [KOSTKA ANALITYCZNA]
```

Możliwe jest także utworzenie wyrażenia pozwalającego zbudować, za pomocą kilku po sobie następujących wyrażeń, wiele miar wyliczanych, do których później możemy w klauzuli SELECT odwołać się po podaniu jej nazwy. Taki przykładowy model miar wyliczanych jako średnie z różnych okresów został zobrazowany na następującym listingu:

Listing 12

```
WITH
  MEMBER [MEASURES].[MK_SR12] AS AVG(
    [WYM_CZAS].[MIESIAC].CURRENTMEMBER.LAG(11):[WYM_CZAS].[MIESIAC],
```

```
    [MEASURES].[MR_ILOSCSPRZEDAZY]
    )
  MEMBER [MEASURES].[MK_SR6] AS AVG(
    [WYM_CZAS].[MIESIAC].CURRENTMEMBER.LAG(5):[WYM_CZAS].[MIESIAC],
    [MEASURES].[MR_ILOSCSPRZEDAZY]
    ) .
      .
      // możliwe definicje dodatkowych miar kalkulowanych
      .
  MEMBER [MEASURES].[MK_SR3] AS AVG(
    [WYM_CZAS].[MIESIAC].CURRENTMEMBER.LAG(2):[WYM_CZAS].[MIESIAC],
    [MEASURES].[MR_ILOSCSPRZEDAZY]
    ) . . .  // dalsza część zapytania
```

Aby wymusić jawnie kolejność, trzeba w trakcie obliczania takich miar wykorzystać słowo kluczowe: `SOLVE_ORDER = WARTOŚĆ`, gdzie `WARTOSC` jest liczbą całkowitą określającą kolejność. Przykładowo, używając w zapytaniu obliczanych członów, możemy łatwo zdefiniować nowy wymiar czasu, służący do przedstawienia roku z rozbiciem na połowy:

Listing 13

```
WITH
MEMBER MEASURES.[MK_ZYSKPROCENTOWY] AS
     '([MEASURES].[MR_ZYSK]) /[MEASURES].[MR_WARTOSCZAKUPU]»,
       FORMAT_STRING = «#.##%‘,
       SOLVE_ORDER = 2
MEMBER [DATA FAKTURY].[MIESIAC].PIERWSZAPOLOWA AS // element wyliczony
      ' [MIESIAC].&[1]+[MIESIAC].&[2]+[MIESIAC].&[3]+
       [MIESIAC].&[4]+[MIESIAC].&[5]+[MIESIAC].&[6]’,
       SOLVE_ORDER = 0
MEMBER [DATA FAKTURY].[MIESIAC].DRUGAPOLOWA AS  // element wyliczony
      ' [MIESIAC].&[7]+[MIESIAC].&[8]+[MIESIAC].&[9]+
       [MIESIAC].&[10]+[MIESIAC].&[11]+[MIESIAC].&[12]’,
       SOLVE_ORDER = 1
SELECT
      { [PIERWSZAPOLOWA],
       [DRUGAPOLOWA],
       [DATA FAKTURY].MIESIAC.MEMBERS} ON COLUMNS,
       [KATEGORIATOWARUG].[NAZWA KATEGORII].MEMBERS ON ROWS
FROM   [KOSTKA ANALITYCZNA]
WHERE  (MEASURES.[MK_ZYSKPROCENTOWY])
```

3.2. Klauzula WITH dla zbiorów

Klauzula WITH może być również wykorzystana dla określenia pewnej zastępczej nazwy dla kilku wybranych przez nas elementów. Konstrukcję taką będziemy rozumieć jako nazwany zbiór, który może być traktowany jako podzapytanie. Nazwany zbiór tworzo-

ny jest na czas trwania zapytania głównego. Zbiór taki posiada nazwę, przez którą może on być wywoływany z zapytania głównego.

Definicja 6

```
WITH
 SET [NAZWA_ZBIORU1] AS '...'
 SET [NAZWA_ZBIORU2] AS '...'
  .
  .
  .
```

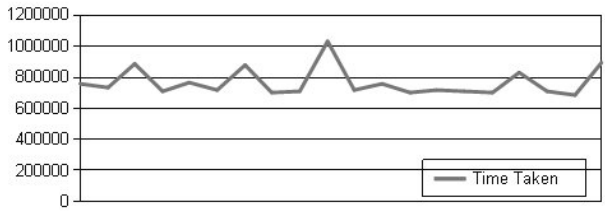
Zastosowanie frazy WITH do nadawania nazwy zbiorom ilustruje poniższy przykład, gdzie nazwany zbiór [NZ_SZYBKIESAMOCHODY] składa się z elementów będących modelami samochodów: porsche, lotus, corvette oraz audi:

Listing 14

```
WITH
 SET [NZ_SZYBKIESAMOCHODY] AS
     '{ [WYM_SAMOCHOD].[NAZWA].&[PORSCHE],
       [WYM_SAMOCHOD].[NAZWA].&[LOTUS],
       [WYM_SAMOCHOD].[NAZWA].&[CORVETTE],
       [WYM_SAMOCHOD].[NAZWA].&[AUDI]}'
SELECT
      { [MEASURES].[MR_ILOSC]} ON COLUMNS,
      { [NZ_SZYBKIESAMOCHODY],[WYM_SAMOCHOD].[NAZWA]} ON ROWS
FROM   [KOSTKA ANALITYCZNA]
```

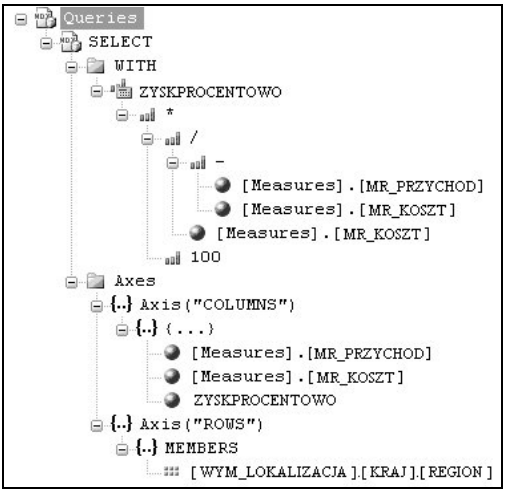
3.3. Badanie wydajności zapytań

Z powodów wydajnościowych bardzo istotną czynnością dla programisty oraz projektanta jest analiza kodu zapytań MDX. W bieżącym podrozdziale autor zaprezentował bardzo pomocne narzędzie MDX Script Performance Analyser, służące do analizy kodu skryptów. Badanie wydajności za pomocą dedykowanego do tego celu narzędzia jest istotne po to, aby umożliwić tworzenie ich w optymalnej postaci. Działania takie mają ogromny wpływ na czas wykonania zapytań (rys. 1), a przy dużej ilości użytkowników bazy OLAP – na poprawne jej działanie [3].



Rys. 1. Przebieg czasu wykonania w trakcie realizacji poszczególnych kroków zapytania

Za pomocą parsera MDX Studio, można wyświetlić drzewko obrazujące strukturę aktualnie wykonywanego zapytania. Pomocne jest to podczas analizy bardziej złożonych konstrukcyjnie zapytań, w których dopuszczalne jest wielopoziomowe zagnieżdżanie podzapytań [4] (rys. 2).



Rys. 2. Przykładowa struktura kodu zapytania MDX w postaci drzewa

4. Przegląd funkcji występujących w MDX

Bardzo istotnym aspektem modelowania różnego typu konstrukcji analitycznych jest operowanie na wartościach. Wartości te, przetwarzamy za pomocą specjalnie utworzonego aparatu matematycznego. Szerokie spektrum wykorzystania analiz dużych zbiorów danych, wymusiło ogromne zapotrzebowanie na funkcje, operujące na takich zbiorach, jak również przetwarzające wartości liczbowe. Wraz z potrzebą tego typu przeliczeń, w języku MDX, zaimplementowano wiele rodzajów potrzebnych funkcji, w zależności od przeznaczenia. W bieżącym rozdziale przedstawiono szereg funkcji, wraz z przykładem ich zastosowania. Wybrane przykłady są potężnym narzędziem dla analityka, podczas wykonywania, wymagających z punktu widzenia analizy danych, zapytań.

4.1. Określanie porządku

Funkcja **ORDER**(ZBIÓR, WYRAŻENIE | LICZBA [, BASC | BDESC]) – porządkuje elementy zbioru ZBIÓR, zgodnie z regułami określonymi przez wyrażenie. Wartość flagi jako BASC porządkuje elementy rosnąco, natomiast wartość flagi jako BDESC malejąco. Następujące zapytanie zwraca zbiór komórek uporządkowany według wartości przychodu. Uporządkowanie odbywa się rosnąco oddzielnie w obrębie każdego członu wymia-

ru MARKA. Sterowane jest to funkcją ORDER, a do zbioru wynikowego należą tylko komórki niepuste, co jest efektem użycia operatora NON EMPTY [2]:

Listing 15

```
SELECT  { [MR_PRZYCHOD]} ON COLUMNS,
          NON EMPTY ORDER(CROSSJOIN(
        { [WYM_SAMOCHOD].[FIAT].CHILDREN,
          [WYM_SAMOCHOD].[FORD].CHILDREN},
          [WYM_CZAS].[ROK].MEMBERS),
          [MR_PRZYCHOD], DESC) ON ROWS
FROM    [KOSTKA ANALITYCZNA]
```

Aby uporządkowanie było globalne, a nie tylko ograniczone do poszczególnych członów wymiaru Marka, należy w funkcji Order użyć parametru BDESC zamiast DESC. Następujące wyrażenie wypisuje 5 pierwszych komórek o najwyższej wartości miary obliczanej [MK_ZYSKPROC]. Sterowane jest to funkcją TOPCOUNT, dla której schemat zastosowania został przedstawiony na przykładzie:

Listing 16

```
WITH MEMBER MEASURES.[MK_ZYSKPROC] AS
       '  ([MR_PRZYCHOD]-[MR_KOSZT])/[MR_KOSZT]*100'

SELECT  { [MR_PRZYCHOD],[MK_ZYSKPROC]} ON COLUMNS,
          TOPCOUNT(CROSSJOIN({[WYM_SAMOCHOD].[FIAT].CHILDREN,
            [WYM_SAMOCHOD].[FORD].CHILDREN},
            [WYM_CZAS].[ROK].MEMBERS),5, [MK_ZYSKPROC]) ON ROWS
FROM      [KOSTKA ANALITYCZNA]
```

4.2. Funkcje logiczne (*logical functions*)

W języku MDX występuje szereg funkcji logicznych, które można używać podczas tworzenia wyrażenia warunkowego. Wyrażenie takie można uwrażliwić na fakt sprawdzenia czy dany element z naszego zbioru, spełnia(lub nie) określoną funkcję w hierarchii wymiaru.

IIF(WARUNEK, PRAWDA, FAŁSZ) – funkcja weryfikująca prawdziwość warunku WARUNEK i zwracająca wartość PRAWDA, jeśli warunek jest prawdziwy, lub FAŁSZ w przeciwnym przypadku. PRAWDA i FAŁSZ mogą być zbiorami elementów.

Poniżej zaprezentowano krótką charakterystykę kilku tego typu funkcji:

ISANCESTOR(ELEMENT1, ELEMENT2) – funkcja przyjmująca wartość TRUE, jeśli element ELEMENT1 jest przodkiem elementu ELEMENT2.

ISCHILD(ELEMENT1, ELEMENT2) – funkcja przyjmująca wartość TRUE, jeśli element ELEMENT1 jest potomkiem elementu ELEMENT2.

ISEMPTY(ELEMENT) – funkcja przyjmująca wartość TRUE w przypadku, gdy element jest pusty (#EMPTY).

ISGENERATION(ELEMENT, INDEKS) – funkcja przyjmująca wartość TRUE, jeśli element ELEMENT należy do pokolenia wskazanego przez indeks.

ISLEAF(ELEMENT) – funkcja przyjmująca wartość TRUE, jeśli element ELEMENT znajduje się na najniższym poziomie hierarchii wymiarów (jest liściem drzewa wymiarów).

ISLEVEL(ELEMENT, POZIOM) – funkcja przyjmująca wartość TRUE, jeśli element ELEMENT znajduje się w hierarchii wymiarów na poziomie określonym przez POZIOM.

ISSIBLING(ELEMENT1, ELEMENT2) – funkcja przyjmująca wartość TRUE, jeśli element ELEMENT1 znajduje się na tym samym poziomie w hierarchii wymiarów i ma ten sam element nadrzędny co element ELEMENT2 [5].

4.3. Funkcje na zbiorach (*set functions*)

Funkcje na zbiorach operują elementami członkowskimi tych zbiorów i w zależności od potrzeby ich zastosowania mamy możliwość dopasowania danych wejściowych do konkretnego zbioru wynikowego, który chcemy uzyskać.

Do filtrowania bardziej szczegółowego, język MDX udostępnia funkcję FILTER. Zwraca ona zbiór, który jest wynikiem filtrowania na podstawie określonego warunku.

FILTER(ZBIÓR, WARUNEK) – funkcja ta zwraca krotki należące do zbioru elementów ZBIÓR, spełniające kryteria selekcji określone przez WARUNEK. Model zapytania dla omawianej funkcji jest następujący:

Listing 17

```
SELECT
       [KATEGORIATOWARU].[NAZWA KATEGORII].MEMBERS ON COLUMNS,
        FILTER({[LOKALIZACJAKLIENTA].[MIASTO].MEMBERS},
       ([MEASURES].[ZYSK], [DATAFAKTURY].[ALL]) > 500) ON ROWS
FROM   [KOSTKA ANALITYCZNA]
WHERE  ([MEASURES].[ZYSK])
```

CROSSJOIN(ZBIÓR1, ZBIÓR2) – zwraca iloczyn kartezjański zbioru ZBIÓR1 i zbioru ZBIÓR2. ZBIÓR2 nie może zawierać żadnych wymiarów wykorzystywanych w zbiorze ZBIÓR1. CROSSJOIN nie może być zastosowany więcej niż raz występującego tego samego wymiaru. Zastosowanie złączenia krzyżowego na wymiarach w celu wyświetlenia hierarchii, będzie miało następującą modelową postać [2]:

Listing 18

```
SELECT
   ADDCALCULATEDMEMBERS(MEASURES.MEMBERS) ON COLUMNS,
    {
     NONEMPTY (CROSSJOIN(
     FILTER(
     [KATEGORIATOWARU].[NAZWA KATEGORII].MEMBERS,
     [KATEGORIATOWARU].[NAZWAKATEGORII]<>
     [KATEGORIATOWARU].[NAZWA KATEGORII].[ALL]),
     [KATEGORIAPRODUKTU].[NAZWA TOWARU].MEMBERS))}ON ROWS
FROM [KOSTKA ANALITYCZNA]
```

GENERATE(ZBIÓR1, ZBIÓR2 [,ALL]) – funkcja zwracająca zbiór powstały w wyniku wykonania operacji: „dla każdego elementu ze zbioru ZBIÓR1 zwróć ZBIÓR2". Klauzula [ALL] pozwala na pozostawienie duplikujących się elementów w zbiorze wynikowym. Możemy więc za pomocą tej funkcji wyświetlić zysk dla każdej kategorii towaru w każdym pierwszym kwartale (pierwszy potomek hierarchii) dla każdego roku:

Listing 19

```
WITH SET [NZ_KWARTAL] AS
   GENERATE  ([DATA].[DATAFAKTURY].[ROK].MEMBERS,
             {[DATA].[DATAFAKTURY].CURRENTMEMBER.FIRSTCHILD})
SELECT  [NZ_KWARTAL] ON COLUMNS,
        [TOWAR].[NAZWA KATEGORII].MEMBERS ON ROWS
FROM    [KOSTKA ANALITYCZNA]
WHERE   ([MEASURES].[MR_ZYSK])
```

Omawiając wyżej wymienioną funkcję, możemy również utworzyć model wyświetlający nam zysk dla każdej kategorii towaru, w każdym pierwszym kwartale (pierwszy potomek hierarchii – numeracja od 0) dla każdego roku:

Listing 20

```
WITH SET [NZ_KWARTALPIERWSZY] AS
   GENERATE  ([DATA].[DATAFAKTURY].[ROK].MEMBERS,
             {[DATA].[DATAFAKTURY].CURRENTMEMBER.CHILDREN(0)})
SELECT
          [NZ_KWARTALPIERWSZY] ON COLUMNS,
          [TOWAR].[NAZWA KATEGORII].MEMBERS ON ROWS
FROM      [KOSTKA ANALITYCZNA]
WHERE     ([MEASURES].[MR_ZYSK])
```

PARENT(ELEMENT) – funkcja zwracająca element nadrzędny w hierarchii wymiarów w stosunku do elementu ELEMENT. PARENT może być używane wielokrotnie. W naszym przypadku przeniesie nas już na poziom [ALL].

Listing 21

```
WITH
MEMBER MEASURES.[MK_ZYSKKATEGORII] AS
    ( [KATEGORIAPRODUKTU].CURRENTMEMBER.PARENT.PARENT,
      [MEASURES].[MR_ZYSK])
MEMBER MEASURES.[MK_ZYSKPROCENTOWY] AS
    ( [KATEGORIAPRODUKTU].CURRENTMEMBER,  [MEASURES].[MR_ZYSK])/
    ( [KATEGORIAPRODUKTU].CURRENTMEMBER.PARENT.PARENT,
      [MEASURES].[MR_ZYSK]),
      FORMAT_STRING = «0.00%»
SELECT
    { [MEASURES].[MK_ZYSKKATEGORII],
      [MEASURES].[MR_ZYSK],
      [MEASURES].[MK_ZYSKPROCENTOWY]} ON COLUMNS,
      [KATEGORIAPRODUKTU].[NAZWA TOWARU] ON ROWS
FROM [KOSTKA ANALITYCZNA]
```

EXCEPT(ZBIÓR1, ZBIÓR2 [,ALL]) – funkcja zwracająca różnice zbiorów ZBIÓR1 i ZBIÓR2. Flaga `[ALL]` pozwala na pozostawienie duplikatów. Użycie nazwanych zbiorów i funkcji EXCEPT (odnajduje różnice pomiędzy dwoma zbiorami), umożliwia nam skonstruowanie wyrażenia, które pokaże procentową sprzedaż dla każdej grupy towarów w porównaniu do całości pomniejszonej o sprzedaż opon [2]:

Listing 22

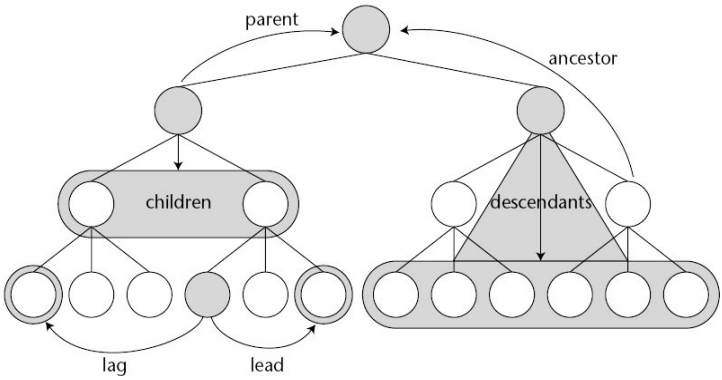
```
WITH SET [NZ_OPROCZOPON] AS
     EXCEPT([KATEGORIAPRODUKTU].[NAZWA  KATEGORII].MEMBERS,
     [TOWAR].[KATEGORIAPRODUKTU].[NAZWA  KATEGORII].[OPONY])
MEMBER MEASURES.[MK_PROCENTSPRZEDAZY] AS
    ( [KATEGORIAPRODUKTU].CURRENTMEMBER,  [MEASURES].[MR_ZYSK])/
     SUM([NZ_OPROCZOPON],[MEASURES].[MR_ZYSK]),
     FORMAT_STRING = «#.00%»
SELECT
    { [MEASURES].[ZYSK],
     [MEASURES].[MK_PROCENTSPRZEDAZY]} ON COLUMNS,
     [NZ_OPROCZOPON] ON ROWS
FROM [KOSTKA ANALITYCZNA]
```

4.4. Funkcje elementów członkowskich (*member functions*)

Funkcje elementów członkowskich operują na zależnościach pomiędzy elementami w hierarchii wymiaru. Funkcja MEMBERS zwraca elementy wskazanego wymiaru lub poziomu wymiarów. Funkcja CHILDREN zwraca elementy ze zbioru „dzieci" dla określonego elementu członkowskiego wymiaru (rys. 3). Obie funkcje są używane przy formułowaniu wyrażeń, ale nie zapewniają możliwości rozwijania do niższych poziomów hierarchii. Aby uzyskać miary dla członów składających się na te kategorie, zapytalibyśmy o to ich dzieci, czyli – `CHILDREN`:

Listing 23

```
SELECT MEASURES.MEMBERS ON COLUMNS,
      {[WYM_KATEGORIAPRODUKTU].[NAZWA  KATEGORII].[OPONY].CHILDREN,
       [WYM_KATEGORIAPRODUKTU].[NAZWA KATEGORII].[RADIA].CHILDREN} ON ROWS
FROM   [KOSTKA ANALITYCZNA]
```



Rys. 3. Zależności występujące w hierarchii dla wymiaru [8]

Miary obliczane nie są dołączane, gdy pobierane są człony wymiarów. Ich uwzględnienie musi być jawnie narzucone, przez użycie funkcji ADDCALCULATEDMEMBERS. Przykład takiej sytuacji może wyglądać następująco:

Listing 24

```
SELECT
   ADDCALCULATEDMEMBERS(MEASURES.MEMBERS) ON COLUMNS,
   {[KATEGORIAPRODUKTU].[NAZWA KATEGORII].[TV],
   DESCENDANTS(
   [KATEGORIAPRODUKTU].[NAZWA KATEGORII].[TV],[NAZWA TOWARU])} ON ROWS
FROM [KOSTKA ANALITYCZNA]
```

W języku MDX możliwe jest wyznaczenie członów wyliczanych ad hoc, przy zastosowaniu klauzuli WITH i odczytaniu ich przez funkcję ADDCALCULATEDMEMBERS:

Listing 25

```
WITH MEMBER MEASURES.[MK_ZYSKPROCENTOWY] AS
     '( [MEASURES].[MR_ZYSK]) /  [MEASURES].[MR_WARTOSCZAKUPU]',
        FORMAT_STRING = «#.00%'
SELECT
        ADDCALCULATEDMEMBERS(MEASURES.MEMBERS) ON COLUMNS,
        [LOKALIZACJAKLIENTA].[WOJEWÓDZTWO].MEMBERS ON ROWS
FROM [KOSTKA ANALITYCZNA]
```

DESCENDANTS(ELEMENT [{ WARSTWA | INDEKS}[, FLAGA]]) – zwraca elementy podrzędne znajdujące się w warstwie WARSTWA, lub na głębokości INDEKS w hierarchii wymiarów elementu ELEMENT. FLAGA wskazuje, które elementy mają być zwracane / pomijane.

Wartości, jakie może przyjmować element FLAGA to [2]:

- *SELF* – zwraca tylko elementy z warstwy WARSTWA oraz element ELEMENT, jeśli należy on do warstwy WARSTWA.
- *AFTER* – zwraca elementy leżące poniżej warstwy WARSTWA.
- *BEFORE* – zawraca element ELEMENT oraz elementy znajdujące się w hierarchii wymiarów powyżej warstwy WARSTWA.
- *BEFORE_AND_AFTER* – zwraca element ELEMENT oraz wszystkie elementy znajdujące się poniżej niego w hierarchii wymiarów, ale nie należące do warstwy WARSTWA.
- *SELF_AND_AFTER* – zwraca wszystkie elementy leżące w warstwie WARSTWA, oraz w warstwach leżących poniżej niej w hierarchii wymiarów.
- *SELF_BEFORE_AFTER* – zwraca element ELEMENT oraz wszystkie elementy leżące poniżej niego w hierarchii wymiarów.
- *LEAVES* – zwraca elementy podrzędne elementu ELEMENT, znajdujące na najniższym poziomie hierarchii wymiarów (liście drzewa wymiarów).

Za pomocą użycia funkcji DESCENDANTS możliwe się staje odpytanie kostki o informacje na poziomie nazwy towaru dla wybranej jego kategorii:

Listing 26

```
SELECT MEASURES.MEMBERS ON COLUMNS,
      { [KATEGORIAPRODUKTU].[NAZWA  KATEGORII].[ZEGARY],
  DESCENDANTS(
       [KATEGORIAPRODUKTUH].[NAZWA  KATEGORII].[ZEGARY],
       [NAZWA TOWARU])} ON ROWS
FROM  [KOSTKA ANALITYCZNA]
```

CURRENTMEMBER(WYMIAR) – zwraca aktualnie przetwarzany element dla wskazanego wymiaru WYMIAR. Funkcja wykorzystywana m.in. w funkcjach iteracyjnych. Wyznaczmy przykładowo sprzedaż danej marki produktu jako udział procentowy w sprzedaży jego kategorii – czyli atrybutu, w odniesieniu do jego rodzica. Wyrażenie tworzące taką miarę obliczaną, może zostać uzyskane przy użyciu właściwości CURRENTMEMBER oraz PARENT.

Listing 27

```
WITH
MEMBER MEASURES.[MK_ZYSKKATEGORII] AS
    ([KATEGORIAPRODUKTU].CURRENTMEMBER.PARENT,
     [MEASURES].[MR_ZYSK])
MEMBER MEASURES.[MK_ZYSKPROCENTOWY] AS
    ([KATEGORIAPRODUKTU].CURRENTMEMBER,[MEASURES].[MR_ZYSK])/
    ([KATEGORIAPRODUKTU].CURRENTMEMBER.PARENT,[MEASURES].[MR_ZYSK]),
     FORMAT_STRING = '0.00%'
```

```
SELECT
    { [MEASURES].[MK_ZYSKKATEGORII],
      [MEASURES].[MR_ZYSK],
      [MEASURES].[MK_ZYSKPROCENTOWY]} ON COLUMNS,
      [KATEGORIAPRODUKTU].[NAZWA TOWARU] ON ROWS
FROM [KOSTKA ANALITYCZNA]
```

Wielokrotne użycie funkcji PARENT może być zastąpione przez obliczenie odpowiednich przodków członu CURRENTMEMBER przez funkcje ANCESTOR, zwracającą przodka na odpowiednim poziomie dla podanego członu. Konieczne jest wtedy użycie nazwy hierarchii [5]:

Listing 28

```
WITH
MEMBER MEASURES.[MK_ZYSKKATEGORII] AS
      (ANCESTOR([KATEGORIAPRODUKTU].CURRENTMEMBER,
      [KATEGORIAPRODUKTU].[NAZWA  KATEGORII]),[MEASURES].[MR_ZYSK])

MEMBER MEASURES.[MK_ZYSKPROCENTOWY] AS
    ([KATEGORIAPRODUKTU].CURRENTMEMBER,[MEASURES].[MR_ZYSK])/
      (ANCESTOR([KATEGORIAPRODUKTU].CURRENTMEMBER,
      [KATEGORIAPRODUKTU].[NAZWA KATEGORII]), [MEASURES].[MR_ZYSK]),
      FORMAT_STRING = «0.00%»
SELECT
      { [MEASURES].[MK_ZYSKKATEGORII],
        [MEASURES].[MR_ZYSK],
        [MEASURES].[MK_ZYSKPROCENTOWY]} ON COLUMNS,
      [KATEGORIAPRODUKTU].[NAZWA TOWARU] ON ROWS
FROM [KOSTKA ANALITYCZNA]
```

PREVMEMBER(ELEMENT [, GENERATION | LEVEL]) – funkcja zwracająca poprzedni element (zależnie od kolejności definiowania) w stosunku do ELEMENT, w ramach pokolenia (GENERATION) lub poziomu (LEVEL). Pokazanie wzrostu w okresie czasu umożliwia funkcja PREVMEMBER. Jeżeli mielibyśmy wyświetlić zysk ze sprzedaży i przyrostową zmianę od poprzedniego członu czasu na poziomie kolejnych miesięcy, to można to zrealizować następująco:

Listing 29

```
WITH
MEMBER MEASURES.[MK_WZROSTZYSKU] AS
      ( [MEASURES].[MR_ZYSK]) -
      (  [MEASURES].[MR_ZYSK],[DATAFAKTURY].PREVMEMBER),
        FORMAT_STRING = «###,###.00 ZŁ»
SELECT
    { [MEASURES].[MR_ZYSK],
      [MEASURES].[MK_WZROSTZYSKU]} ON COLUMNS,
      {DESCENDANTS([DATAFAKTURY],[DATAFAKTURY].[MIESIAC])} ON ROWS
FROM [KOSTKA ANALITYCZNA]
```

LEAD(ELEMENT, INDEKS) – funkcja zwracająca element, który znajduje się INDEKS kroków wprzód od elementu ELEMENT w ramach tej samej warstwy w hierarchii wymiarów. Funkcji LEAD, która zwraca człon oddalony w wymiarze o określoną liczbę pozycji od wskazanego członu, może być użyta w sposób przedstawiony na poniższym przykładowym modelu:

Listing 30

```
WITH
MEMBER MEASURES.[MK_WZROSTZYSKU] AS
      ( [MEASURES].[MR_ZYSK])-
      ( [MEASURES].[MR_ZYSK],[DATAFAKTURY].LEAD(-2)),
       FORMAT_STRING = «###,###.00 ZŁ»
SELECT
      { [MEASURES].[MR_ZYSK], MEASURES.[MK_WZROSTZYSKU]} ON COLUMNS,
      { DESCENDANTS([DATAFAKTURY],[DATAFAKTURY].[MIESIAC])} ON ROWS
FROM   [KOSTKA ANALITYCZNA]
```

NEXTMEMBER(ELEMENT [, GENERATION | LEVEL]) – funkcja zwracająca kolejny element (zależnie od kolejności definiowania) w stosunku do ELEMENT, w ramach pokolenia (GENERATION) lub poziomu (LEVEL). Użycie NEXTMEMBER w tym wyrażeniu pokazałoby sprzedaż dla każdego miesiąca zestawioną ze sprzedażami z poprzednich miesięcy [5].

Listing 31

```
WITH
MEMBER MEASURES.[MK_WZROSTZYSKU] AS
      ( [MEASURES].[MR_ZYSK]) -
      ( [MEASURES].[MR_ZYSK],
       [DATAFAKTURY].NEXTMEMBER),
       FORMAT_STRING = '###,###.00 ZŁ'
SELECT
      { [MEASURES].[MR_ZYSK], MEASURES.[MK_WZROSTZYSKU]} ON COLUMNS,
      { DESCENDANTS([DATA FAKTURY],[DATAFAKTURY].[MIESIAC])} ON ROWS
FROM   [KOSTKA ANALITYCZNA]
```

PARALLELPERIOD([WARSTWA [, INDEKS [, ELEMENT]]]) – funkcja zwraca element z wcześniejszego okresu czasu dla podanych parametrów. PARALLELPERIOD pozwala na łatwe porównanie wzrostu ze wzrostem z tego samego przedziału czasu w poprzednim kwartale:

Listing 32

```
WITH
MEMBER MEASURES.[MK_WZROSTZYSKU] AS
      ( [MEASURES].[MR_ZYSK]) - (MEASURES.[MR_ZYSK],
       PARALLELPERIOD([DATAFAKTURY].[KWARTAL])),
       FORMAT_STRING = '###,###.00 ZŁ'
```

```
SELECT
      { [MEASURES].[MR_ZYSK],
        [MEASURES].[MK_WZROSTZYSKU]} ON COLUMNS,
      { DESCENDANTS([DATAFAKTURY],[DATAFAKTURY].[MIESIAC])} ON ROWS
FROM    [KOSTKA ANALITYCZNA]
```

Pojawiają się błędy, gdy nie zdefiniowano „równoległego" miesiąca. Miesiące w obrębie kwartału są wykrywane jako kolejne wystąpienia na liście, niezgodnie z kalendarzem. Można sprawdzić, o ile wzrosła sprzedaż po pierwszym miesiącu sezonu. Używając kwartału do przedstawienia sezonu, można mierzyć różnicę w sprzedaży jednostek dla każdego miesiąca, porównując z miesiącem otwierającym kwartał:

Listing 33

```
WITH
MEMBER MEASURES.[MK_WZROSTZYSKU] AS
      ( [MEASURES].[ZYSK]) - (MEASURES.[ZYSK],
        OPENINGPERIOD([DATAFAKTURY].MIESIAC,
        [DATAFAKTURY].CURRENTMEMBER.PARENT)),
        FORMAT_STRING = '###,###.00 ZŁ'
SELECT
      { [MEASURES].[MR_ZYSK],
        [MEASURES].[MK_WZROSTZYSKU]} ON COLUMNS,
      { DESCENDANTS([DATAFAKTURY],[DATAFAKTURY].[MIESIAC])} ON ROWS
FROM    [KOSTKA ANALITYCZNA]
```

4.5. Funkcje agregujące

Specjalnym rodzajem funkcji w wyrażeniach MDX są funkcje agregujące, które pozwalają z danych szczegółowych dotyczących miar wyodrębnić wartości wyróżniającą się pod podanym kryterium, bądź pokazać całkowitą ich wartość agregując po danej hierarchii wymiaru przecinającego je.

SUM(ZBIÓR [, LICZBA]) – funkcja zwracająca sumę elementów wchodzących w skład zbioru ZBIÓR. Zapytanie MDX, które wyświetla 10 miast od góry, w oparciu o ilość transakcji sprzedaży, oraz jak dużo sprzedały łącznie pozostałe miasta. Takie wyrażenie pokaże także inne zastosowanie funkcji SUM, nawiązujące do nazwanych zbiorów i obliczanych członów:

Listing 34

```
WITH
SET  [NZ_GORNEDZIESIEC] AS
       TOPCOUNT([LOKALIZACJAKLIENTA].[MIASTO].MEMBERS,10,
       [MEASURES].[MR_ILOSC])
MEMBER [LOKALIZACJAKLIENTA].[EO_INNEMIASTA] AS
      ( [LOKALIZACJAKLIENTA].[ALL], MEASURES.CURRENTMEMBER) -
       SUM([NZ_GORNEDZIESIEC], MEASURES.CURRENTMEMBER)
SELECT [MEASURES].MEMBERS ON COLUMNS,
       {[NZ_GORNEDZIESIEC],[EO_INNEMIASTA]} ON ROWS
FROM    [KOSTKA ANALITYCZNA]
```

TOPCOUNT(ZBIÓR, INDEKS [, LICZBA]) – funkcja zwracająca indeks elementów zbioru ZBIÓR uporządkowanych od największego do najmniejszego.

TOPPERCENT(ZBIÓR, PROCENTOWOŚĆ, LICZBA) – zwraca najmniejszy istniejący podzbiór zbioru ZBIÓR, dla którego całkowity wynik przeliczeń jest nie mniejszy niż zadeklarowana procentowość przeliczeń całego zbioru ZBIÓR. Procentowość jest liczba z zakresu [0; 100]. LICZBA wskazuje kryteria selekcji. Elementy zwróconego podzbioru uporządkowane są od największego do najmniejszego.

TOPSUM(ZBIÓR, SUMA, LICZBA) – zwraca najmniejszy istniejący podzbiór zbioru ZBIÓR, dla którego całkowity wynik przeliczeń jest co najmniej równy podanemu wyrażeniu SUMA. Elementy zwróconego podzbioru uporządkowane są od największego do najmniejszego.

Oczywiście istnieje także seria funkcji BOTTOM(odpowiedniki TOP*xxx*), jak i również różne funkcje statystyczne AVG, MEDIAN, MAX, MIN, VAR, STDDEV. Jednak ze względu na ograniczony charakter pracy nie możliwe jest przez autora omówienie wszystkich dostępnych w języku funkcji. Format zapisu dla wyżej wymienionych funkcji jest jednakowy [5]:

Listing 35

```
WITH
MEMBER [MK_SREDNIASPRZEDAZ] AS
       AVG(DESCENDANTS([DATAFAKTURY].[ROK],
                       [DATAFAKTURY].[MIESIAC]))
MEMBER [MK_SREDNIAILOSC] AS
       COUNT(DESCENDANTS([DATAFAKTURY].[ROK],
                       [DATAFAKTURY].[MIESIAC]),EXCLUDEEMPTY)
SELECT {[DATAFAKTURY].[ROK],
        [MK_SREDNIASPRZEDAZ],
        [MK_SREDNIAILOSC]} ON COLUMNS,
        [KATEGORIATOWARU].[NAZWA KATEGORII] ON ROWS
FROM    [KOSTKA ANALITYCZNA]
WHERE   ([MEASURES].[MR_ILOSC])
```

4.6. Formatowanie warunkowe

W klauzuli WITH, po definicji elementu wyliczanego możliwe jest także odpowiednie jego formatowanie. Warunkowe formatowanie może odbywać się poprzez użycie funkcji IIF(), w której określimy jakiego ono ma być rodzaju. Jest to bardzo wygodne konstrukcyjnie złożenie, udostępniające możliwość wyróżnienia w wyniku pól, które są istotnym elementem naszego raportu tzn. mają jakąś wyższą wartość informacyjną. Przykładowy model wyszczególniający(formatujący czcionkę) produkty o wartości większej od 100 może się tak przedstawiać:

Listing 36

```
WITH
SET  [NZ_TOWARY] AS
     [KATEGORIAPRODUKTU].[NAZWA  KATEGORII].MEMBERS
MEMBER [MEASURES].[MK_ZAKUP] AS [MEASURES].[MR_WARTOSC],
     FORMAT_STRING = IIF(MK_ZAKUP>800,'#.00','{#.00)'),
     FORE_COLOR = IIF(MK_ZAKUP>100,RGB(255,0,0), RGB(255,255,0)),
     BACK_COLOR = IIF(MK_ZAKUP>100,RGB(0,255,255), RGB(0,155,0)),
     FONT_FLAGS = IIF(MK_ZAKUP>100,MDFF_BOLD OR MDFF_UNDERLINE ,MDFF_ITALIC
     OR MDFF_STRIKEOUT),
     FONT_NAME = IIF(MK_ZAKUP>100,ARIAL,TIMES),
     FONT_SIZE = IIF(MK_ZAKUP>100,16,10)
SELECT
       [NZ_TOWARY] ON COLUMNS,
       [LOKALIZACJA].[WOJEWÓDZTWO].MEMBERS ON ROWS
FROM   [KOSTKA ANALITYCZNA]
WHERE     [MEASURES].[MK_ZAKUP]
CELL PROPERTIES VALUE, FORMATTED_VALUE, CELL_ORDINAL,
FORMAT_STRING, FORE_COLOR, BACK_COLOR, FONT_FLAGS, FONT_NAME,
FONT_SIZE
```

5. Wnioski

Mechanizmy obliczeń wykorzystywane w kostkach analitycznych zakładają realizacje agregacji dla poszczególnych poziomów w hierarchii wymiarów. W większości zastosowań, realizacja tylko mechanizmów sumowania na poziomie hierarchii, jest niewystarczająca. Tu zachodzi potrzeba wzbogacenia analizy o dodatkowe operatory oraz funkcje matematyczne. W procesach analitycznych pojawia się konieczność obliczeń związanych z porównywaniem między sobą wartości. Wartości te mogą pochodzić z dwóch równoległych okresów (np. lat lub miesięcy). Pomocą dla obsługi tego typu operacji, a nawet operacji bardziej skomplikowanych jest język MDX.

W pracy przedstawione zostały zaproponowane przez autora schematy użycia dostępnych funkcji, jak również opracowane podejście do konstruowania podobnych modeli. Opisano przypadki użycia języka MDX, w których wyraźnie jest dominującym narzędziem nad innymi. Zamieszczono wiele przykładów ilustrujących przetwarzanie danych wielowymiarowych, wspomagając się dedykowanymi do tego celu narzędziami.

Podsumowując przemyślenia, można stwierdzić, że język MDX, podobnie jak SQL, jest językiem zapytań jak i definiowania danych. SQL jest potężnym narzędziem wykorzystywanym w środowisku baz relacyjnych, jednak nie sprawdza się on w środowiskach wielowymiarowych struktur bazodanowych architektury OLAP, w której to rolę lidera przejmuje MDX.

Język MDX został zaimplementowany w wielu produktach Microsoftu. Dostępny jest poprzez warstwę pośredniczącą OLE DB lub ADO. Może on skutecznie wspomagać zintegrowaną analizę danych, gromadzonych w bazach i hurtowniach danych.

Mając do dyspozycji tego typu rozbudowane mechanizmy, które z łatwością umożliwiają tworzenie konstrukcji wspomagających osiągnięcie zamierzonego celu, jesteśmy w stanie wydobyć większą ilość istotnych informacji z danych. Bardziej wartościowe dla nas dane, są więc uzyskiwane mniejszym nakładem pracy. Nakład ten głównie koncentruje się na kosztownych czasowo obliczeniach podczas zwracania wyniku zapytania. Istotnymi aspektami są także trudne w opracowaniu modele analityczne, które tworzone za pomocą mniej wydajnych sposobów(np. konstrukcji relacyjnych w SQL), a nie za pomocą przeznaczonego do tego celu narzędzia, będą dodatkowymi obciążeniami czasowymi dla: wykonania zapytania oraz projektowania modelu.

Literatura

[1] Bielski W., *Praktycznie o hurtowniach danych, SQL Server i Business Intelligence*. http://www.bielski.biz/ – serwis tematyczny dotyczący zagadnień hurtowni danych.
[2] Kędziora P., Lewandowski M., *Funkcjonalność języka MDX w implementacji HYPERION.* Artykuł porównawczy, Politechnika Poznańska.
[3] *MDX Script Performance Analyser*: http://www.codeplex.com/mdxscriptperf. Narzędzie analizy wydajności zapytań MDX.
[4] *MDX Studio*: http://www.mosha.com/msolap/mdxstudio.htm – narzędzie przetwarzania zapytań MDX.
[5] Pankowski T., *Język MDX MultiDimensional Expressions.* www.put.poznan.pl/~pankowsk/ – materiały dydaktyczne.
[6] Pearson W.E., *MDX Essentials Series*. http://www.databasejournal.com – serwis wiedzy informatycznej, poświęcony zagadnieniom baz danych.
[7] Skurniak T., *Język MDX i obliczenia w kostkach analitycznych*. http://www.microsoft.com/poland/technet/article/art0064_01.mspx.
[8] Spofford G., Harinath S., Webb Ch., Huang D., Civardi F., *MDX Solutions with Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase*. Microsoft Corporation Published: March 2006.
[9] Sturm J., *Hurtownie danych.* Microsoft SQL Server 7.0. Przewodnik Techniczny, Microsoft Press/ APN PROMISE, Warszawa, 2000.
[10] Tkachuk R., *Introduction to MDX Scripting in Microsoft SQL Server 2005*. Microsoft Corporation Published: March 2005.
[11] Vassiliadis P., Sellis T., *A Survey of Logical Models for OLAP Databases*. ACM SIGMOD Record 28(4), 1999.
[12] Wrembel R., Królikowski Z., Morzy M., *Magazyny danych – stan obecny i kierunki rozwoju.* ProDialog 10, Wydawnictwo NAKOM, Poznań, 2000, 75–93.
[13] Youness S., *Professional Data Warehousing with SQL Server 7.0 and OLAP Services.* Wrox Press, Arden House, 2000.