Perspektywy (views) w systemach baz danych: aktualny stan technologii *

Robert Wrembel
Politechnika Poznańska, Instytut Informatyki
ul. Piotrowo 3A, 60-965 Poznań
e–mail: Robert.Wrembel@cs.put.poznan.pl

Abstrakt. Bardzo ważnym mechanizmem w systemach baz danych, zarówno relacyjnych, relacyjno–obiektowych, jak i obiektowych jest perspektywa. Intuicyjnie, perspektywa jest swego rodzaju „oknem", przez które użytkownik bazy danych widzi udostępniane przez nią dane. W tradycyjnych systemach relacyjnych baz danych, perspektywy są wykorzystywane m.in. do: (1) ograniczenia i autoryzacji dostępu do danych, (2) zapewnienia logicznej niezależności danych przez odseparowanie aplikacji od schematu konceptualnego bazy danych, (3) uproszczenia schematu bazy danych, (4) prezentowania tych samych danych w różny sposób. Inną, wykorzystywaną szeroko dziedziną zastosowań perspektyw są magazyny. Zadaniem perspektyw jest tutaj: (1) integracja danych pochodzących z różnych geograficznie rozproszonych źródeł, oraz (2) materializowanie wyników czasochłonnych zapytań typu OLAP. Do tego celu są stosowane tzw. *perspektywy materializowane*.

W artykule zostały przedstawione perspektywy w systemach relacyjnych i relacyjno–obiektowych, zilustrowane systemem *Oracle*, oraz koncepcje perspektyw obiektowych i semistrukturalnych. Omówione zostały również zagadnienia związane z wykorzystaniem materializowanych perspektyw w magazynach danych.

1. Wprowadzenie

W systemach relacyjnych i relacyjno–obiektowych perspektywa jest strukturą logiczną opartą o zapytanie do bazy danych, umożliwiającą dostęp do podzbioru atrybutów i rekordów jednej lub wielu tabel. Oznacza to, że perspektywa nie posiada własnych danych, lecz udostępnia tylko te dane, które są wynikiem zapytania ją definiującego. Inaczej jest w przypadku tzw. *perspektywy zmaterializowanej* (ang. materialized view). Perspektywa taka posiada własne trwałe dane – będące wynikiem zmaterializowania danych wyznaczonych przez zapytanie definiujące tę perspektywę.

Rozwijające się gałęzie przemysłu, handlu, medycyny, nauki wymagają składowania i przetwarzania ogromnych ilości danych. Dane są zwykle przechowywane w systemach informatycznych posiadających różne struktury i wykorzystujących różne modele danych (np. hierarchiczne, relacyjne, obiektowe), w dokumentach tekstowych, czy arkuszach kalkulacyjnych. Ponieważ przedsiębiorstwa, instytucje, organizacje (producenci danych) są często geograficznie rozproszone, więc same dane mają również charakter rozproszony. Heterogeniczność i rozproszenie informacji utrudnia do nich dostęp i ich analizę. Z tego powodu istnieje potrzeba integracji tych informacji.

W praktyce stosuje się dwie technologie integracji danych, tzw. *system sfederowanych baz danyc*h (ang. federated database system) i technologię *magazynu danych* (ang. data warehouse) [10]. W obu przypadkach perspektywy są wykorzystywane jako mechanizm zintegrowanego dostępu do danych.

W systemach magazynów danych dane są eksplorowane za pomocą złożonych zapytań zawierających od kilku do kilkudziesięciu operacji łączenia, filtrowania, grupowania i agregowania danych. Są to tzw. zapytania typu OLAP (ang. On Line Analytical Processing). Ze względu na złożoność zapytań OLAP oraz rozmiary adresowanych przez nie danych, czas wykonania tych

* Praca wspierana z grantu Komitetu Badań Naukowych, nr 8T11C01018

zapytań może trwać dziesiątki minut, a nawet godziny. Kluczowym problemem staje się więc efektywność systemu bazy danych. Jednym z rozwiązań zwiększających efektywność systemu jest materializowanie częściowych/pośrednich wyników najczęściej wykonywanych zapytań za pomocą perspektyw zmaterializowanych.

Struktura niniejszego artykułu jest następująca: w punktach drugim, trzecim i czwartym zostaną omówione odpowiednio koncepcje perspektywy relacyjnej, obiektowej i relacyjno–obiektowej. Punkt piąty jest poświęcony zagadnieniom integracji danych za pomocą perspektyw. Punkt szósty omawia perspektywy zmaterializowane w *Oracle8i*. Punkt siódmy prezentuje koncepcje perspektywy semistrukturalnej, a punkt ósmy podsumowuje artykuł.

2. Perspektywa relacyjna

2.1. Podstawowe własności perspektywy

Perspektywa stosowana w systemach relacyjnych posiada następujące własności (por. [18]):

1. Perspektywa bazuje na tabelach, na innych perspektywach lub jednocześnie na tabelach i perspektywach, nazywanych odpowiednio *tabelami* i *perspektywami bazowymi* (ang. base tables, base views).
2. Z użytkowego punktu widzenia, perspektywa ma strukturę taką samą jak tabela, a więc jest zbiorem rekordów o określonych atrybutach. Na poziomie fizycznym perspektywa różni się od tabeli tym, że nie posiada własnych danych. W konsekwencji, na atrybutach perspektywy nie można definiować indeksów.
3. W słowniku bazy danych perspektywa jest przechowywana w postaci zapytania ją definiującego. Zapytanie skierowane do perspektywy jest łączone z zapytaniem ją definiującym, a następnie tak przetworzone zapytanie jest kierowane do tabel bazowych.

2.2. Cele stosowania perspektywy

W tradycyjnych systemach relacyjnych baz danych podstawowe cele stosowania perspektyw są następujące:

1. Autoryzacja dostępu do danych

   Ponieważ perspektywy udostępniają tylko podzbiór rekordów i ich atrybutów zawartych w tabelach bazowych, ukrywają w ten sposób dane przed nieupoważnionym użytkownikiem. Użytkownicy otrzymują prawa dostępu do perspektyw, natomiast nie otrzymują praw dostępu do tabel bazowych.

   Przykładem stosowania perspektyw do ukrycia pewnych informacji jest słownik bazy danych. W ramach tego słownika wyróżnia się perspektywy przeznaczone dla użytkowników bazy danych (np. w systemie *Oracle*: *user_db_links*, *user_triggers*) oraz perspektywy przeznaczone dla administratorów bazy danych, udostępniające więcej informacji niż perspektywy dla użytkowników (np. w systemie *Oracle*: *dba_db_links*, *dba_triggers*).
2. Ułatwienie dostępu do danych

   W przypadku częstych odwołań do tych samych tabel przy pomocy tych samych lub nieznacznie zmodyfikowanych skomplikowanych zapytań, zaleca się użycie tych zapytań do zdefiniowania odpowiedniej perspektywy. Odwołania do tej perspektywy umożliwiają pobranie tych samych danych co poprzednio, lecz za pomocą znacznie prostszych zapytań.

   Przykładem stosowania perspektyw w celu ułatwienia dostępu do danych jest słownik bazy danych, np. w systemie *Oracle*: perspektywy *dba_tablespaces*, *dba_data_files*, *dba_rollback_segs*.
3. Możliwość prezentowania tych samych danych w różny sposób

W definicji perspektywy mogą wystąpić np. wyrażenia arytmetyczne operujące na atrybutach tabel bazowych i literałach. Umożliwia to wstępne przetworzenie danych z tabel bazowych i ich prezentację w postaci (formacie) preferowanej przez użytkownika.

4. Logiczna niezależność danych

   Aplikacje korzystające z bazy danych za pomocą perspektyw nie wymagają modyfikacji w przypadku zmodyfikowania tabel bazowych. W przypadku zmiany schematu tabel bazowych należy zmodyfikować wyłącznie definicję odpowiednich perspektyw tak, aby ich schemat pozostał taki, jak poprzednio. Wówczas aplikacje nie będą wymagały żadnej modyfikacji.

5. Możliwość wprowadzenia dodatkowego poziomu ograniczeń integralnościowych

   Ograniczenia te są zawarte w definicji perspektywy i rozszerzają listę ograniczeń integralnościowych związanych z tabelami bazowymi (np. klauzula ***with check option*** w systemie *Oracle*).

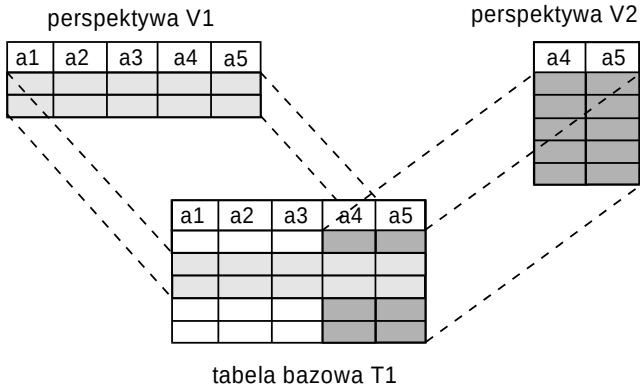
Perspektywy zmaterializowane pozwalają dodatkowo m.in. na:

1. Integrację danych pochodzących z rozproszonych homo– i heterogenicznych źródeł danych oraz uniezależnienie użytkowników od konieczności dostępu do rozproszonych danych źródłowych.
2. Materializowanie częściowych wyników najczęściej wykonywanych zapytań OLAP, przez co możliwe jest skrócenie czasu odpowiedzi na takie zapytania.

2.3. Rodzaje perspektyw

Ze względu na postać zapytania definiującego perspektywę, perspektywy można podzielić na *proste* i *złożone*. Perspektywa jest prostą jeżeli zapytanie ją definiujące:

- odwołuje się wyłącznie do jednej tabeli bazowej;
- nie wykorzystuje funkcji języka SQL w celu przetwarzania wartości atrybutów udostępnianych przez perspektywę;
- nie wykorzystuje wartości wyliczanych, np. *pensja*podatek*;
- nie wykorzystuje funkcji grupowych (np. suma, średnia);
- nie wykorzystuje klauzul ***connect by*** oraz ***start with*** (system *Oracle*);
- nie wykorzystuje operatora ***distinct*** i operatorów zbiorowych (np. suma, część wspólna).

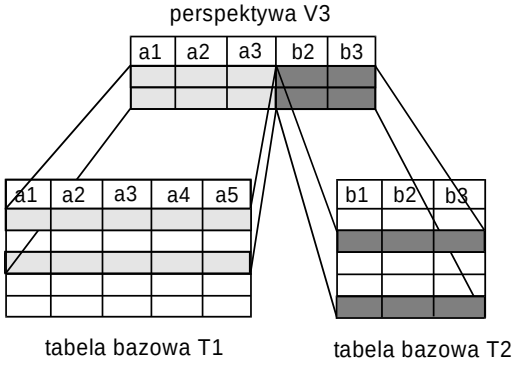


Rys.1. Perspektywy bazujące na pojedynczej tabeli

Na rysunku 1 przedstawiono dwie perspektywy typu prostego, bazujące na tabeli T_1. Pierwsza z nich, oznaczona jako V_1 udostępnia wszystkie atrybuty wybranych rekordów, natomiast perspektywa V_2 udostępnia wybrane atrybuty wszystkich rekordów tabeli bazowej.

Drugim rodzajem perspektywy jest perspektywa złożona. Dla tego rodzaju perspektywy zapytanie ją definiujące nie spełnia przynajmniej jednego warunku przedstawionego dla perspektywy prostej.

Przykład perspektywy opartej o dwie tabele bazowe przedstawiono na rysunku 2. Perspektywa taka może udostępniać podzbiór połączonych rekordów, wybrane atrybuty wszystkich połączonych rekordów lub podzbioru tych rekordów.



Rys.2. Perspektywa bazująca na dwóch tabelach

Rodzaj perspektywy decyduje o możliwościach uaktualniania danych przez nią udostępnianych (zob. punkt 1.5).

2.4. Przetwarzanie zapytań kierowanych do perspektywy

Do perspektywy można się odwoływać w zapytaniach w taki sam sposób jak do tabel. W przypadku skierowania zapytania do perspektywy, system zarządzania bazą danych łączy to zapytanie z zapytaniem definiującym perspektywę, m.in.:

- w klauzuli ***select*** pojawia się wspólna część zbioru atrybutów wyspecyfikowanych w zapytaniu użytkownika i zapytaniu definiującym perspektywę;
- w klauzuli ***where*** pojawiają się warunki wyboru pochodzące zarówno z zapytania użytkownika, jak i definicji perspektywy (łączone operatorem ***and***).

Następnie, tak przetransformowane zapytanie jest kierowane do tabel bazowych perspektywy.

Innym rodzajem perspektywy, jest tzw. perspektywa *in–line*, która jest tworzona dynamicznie w czasie wykonania zapytania. Perspektywa taka to zapytanie występujące w klauzuli ***from*** innego zapytania. Jako przykład wykorzystania tego rodzaju perspektywy rozważmy zapytanie wyznaczające pięć pierwszych sklepów, które sprzedały towary za największą kwotę. Jest to tzw. zapytanie typu top–N.

Dane są dwie tabele *sklepy* i *sprzedaż* o schematach przedstawionych poniżej:

```
SQL> desc sklepy
 Name          Null?      Type
 ------------  --------   ----
 SKLEP_ID      NOT NULL   NUMBER(3)
 NAZWA         NOT NULL   VARCHAR2(20)
 MIASTO        NOT NULL   VARCHAR2(15)
```

```
SQL> desc sprzedaz
 Name          Null?      Type
 ------------  --------   ----
 PRODUKT_ID               NUMBER(4)
 L_SZTUK                  NUMBER(3)
 CENA_JEDN                NUMBER(7,2)
 DATA                     DATE
 SKLEP_ID                 NUMBER(3)
```

Przykładowe zapytanie (w systemie *Oracle8i*) z perspektywą typu *in–line* zostało przedstawione poniżej:

```
select *
from (select nazwa, sum(l_sztuk*cena_jedn)
      from sprzedaz sp, sklepy sk
```

```
      where sp.sklep_id=sk.sklep_id
      group by nazwa
      order by sum(l_sztuk*cena_jedn) desc)
where rownum < 6;
```

2.5. Możliwości modyfikowania danych poprzez perspektywę

W zapytaniach perspektywa może być stosowana w sposób identyczny, jak tabela. Natomiast możliwości bezpośredniego modyfikowania danych udostępnianych przez perspektywę są ograniczone.

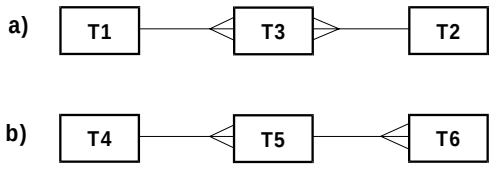
Jeżeli perspektywa jest prosta i dodatkowo udostępnia wszystkie atrybuty, których wartość jest obowiązkowa (***not null***), to za pomocą takiej perspektywy można wstawiać dane do jej tabeli bazowej, modyfikować dane widoczne przez tę perspektywę i je usuwać.

W systemie *Oracle* informacje na temat atrybutów perspektyw, które można uaktualniać są dostępne za pomocą perspektywy słownika danych *USER_UPDATABLE_COLUMNS*.

Jeżeli perspektywa prosta zawiera atrybuty wyliczane przez funkcje SQL lub wyrażenia arytmetyczne, to za jej pomocą można stawiać i modyfikować dane pod warunkiem że wartości wyliczane pozostają puste (przy wstawianiu) i nie są modyfikowane. Natomiast usuwanie rekordów jest zawsze możliwe.

Jeżeli natomiast perspektywa jest oparta o połączenie, to można przez nią uaktualniać dane pochodzące wyłącznie z tabeli zawierającej klucz obcy (***foreign key***). Usuwanie rekordów jest również możliwe, jednak warunki wyboru rekordów muszą zostać wyspecyfikowane z wykorzystaniem atrybutów tabeli zawierającej klucz obce i/lub atrybutów połączeniowych. Wstawianie rekordów jest zabronione.

Jeżeli perspektywa bazuje na łączeniu tabel w schemat gwiazdy (por. rysunek 3a), wówczas operacje modyfikowania i usuwania mogą dotyczyć wyłącznie tabeli T_3. Natomiast jeżeli perspektywa bazuje na tabelach łączonych w hierarchię (por. rysunek 3b), to operacje modyfikowania i usuwania danych mogą dotyczyć wyłącznie tabeli T_6.



Rys. 3. Łączenie tabel w schemat gwiazdy i hierarchię

Jeżeli perspektywa jest oparta o połączenie zewnętrzne (ang. outer join), wówczas jedyną możliwą operacją jest wyszukiwanie danych za jej pomocą.

Powyższe ograniczenia można zastąpić własnym sposobem modyfikowania zawartości tabel bazowych perspektywy. Do tego celu służy wyzwalacz ***instead-of*** definiowany dla perspektywy.

Jako przykład wykorzystania tego wyzwalacza rozważmy trzy następujące tabele *klienci_poznan*, *klienci_krakow*, *klienci_pozostali*, każda o identycznym schemacie, przedstawionym poniżej:

```
SQL> desc klienci_poznan
 Nazwa             NULL?    Typ
 ----------------- -------- ------------
 KLIENT_ID                  NUMBER(2)
 NAZWISKO                   VARCHAR2(10)
 MIASTO                     VARCHAR2(10)
```

Tabele te posłużyły do zdefiniowania perspektywy w następujący sposób:

```
create view v_klienci
as  select * from klienci_poznan
    union
    select * from klienci_krakow
    union
    select * from klienci_pozostali;
```

W celu zapewnienia możliwości wstawiania rekordów przez tę perspektywę należy zdefiniować dla niej wyzwalacz ***instead-of***. Przykładowy kod takiego wyzwalacza został przedstawiony poniżej:

```
create or replace trigger v_kl_modyfikacja
instead of insert on v_klienci
begin
  if :new.miasto='Poznań'
    then insert into klienci_poznan
                values (:new.klient_id, :new.nazwisko, :new.miasto);
  elsif :new.miasto='Kraków'
    then insert into klienci_krakow
                values (:new.klient_id, :new.nazwisko, :new.miasto);
  else
    insert into klienci_pozostali
                values (:new.klient_id, :new.nazwisko, :new.miasto);
  end if;
end;
```

Klauzula ***instead of*** umożliwia wskazanie zdarzenia, które spowoduje uruchomienie wyzwalacza. W klauzuli tej wykorzystuje się słowa kluczowe ***insert***, ***update***, ***delete*** umożliwiające zdefiniowanie wyzwalacza reagującego odpowiednio na wstawienie, uaktualnienie, usunięcie rekordów dostępnych za pomocą perspektywy.

3. Perspektywa obiektowa

Rozwijające się dziedziny zastosowań systemów informatycznych takie jak: komputerowe wspomaganie projektowania (CAD), komputerowe wspomaganie wytwarzania (CAM), systemy wspomagania inżynierii oprogramowania (CASE), czy geograficzne systemy informacyjne (GIS) wymagają modeli danych umożliwiających modelowanie obiektów o złożonych strukturach i skomplikowanych powiązaniach z innymi obiektami oraz modelowanie własności dynamicznych obiektów. Relacyjny model danych jest zbyt ubogi dla tego typu zastosowań. Z tego względu, coraz częściej wykorzystuje się obiektowe bazy danych, które łączą w sobie cechy technologii obiektowych z cechami systemów zarządzania bazami danych.

3.1. Koncepcje modelu obiektowego

Model obiektowy jest oparty o następujące koncepcje: obiekt, hermetyczność, klasę, dziedziczenie i polimorfizm.

Obiekt. Jedną z podstawowych koncepcji podejścia obiektowego jest *obiekt* (ang. object), reprezentujący w systemie komputerowym encję modelowanego świata rzeczywistego. Własności statyczne encji są modelowane za pomocą atrybutów obiektu, a jej własności dynamiczne – za pomocą operacji wykonywanych na obiekcie. Operacje te są zwane *metodami* (ang. methods). Zbiór wszystkich metod obiektu stanowi jego *interfejs*.

Zmiana stanu obiektu, tj. wartości jego atrybutów, może zostać zrealizowana tylko za pomocą odpowiedniej metody. Własność obiektu polegająca na tym, że jego struktura i implementacja metod nie jest dostępna dla programów nazywa się *hermetycznością* (ang. encapsulation). Dzięki hermetyczności, zmiana implementacji struktury obiektów i metod nie wpływa na poprawność

pracy innych programów odwołujących się do tych obiektów, jeśli tylko ich interfejs pozostaje niezmieniony.

Każdy obiekt posiada unikalny identyfikator, który jest nadawany przez system w momencie tworzenia obiektu.

Klasa. Wszystkie obiekty posiadające taką samą strukturę i metody są grupowane w *klasę* (ang. class). Klasa jednocześnie definiuje strukturę (za pomocą atrybutów) i zachowanie się (za pomocą metod) wszystkich swoich instancji. Dziedzinami atrybutów klasy mogą być wartości elementarne, takie jak liczby, łańcuchy znaków, lub wartości złożone, takie jak listy i zbiory. Dziedzinami atrybutów mogą być również inne klasy – w tym przypadku, klasy są połączone tzw. *związkiem kompozycji* (ang. composition relationships). Związki takie umożliwiają modelowanie struktur złożonych.

Dziedziczenie. Model obiektowy umożliwia wywodzenie nowych klas z klas już istniejących. Mechanizm ten nazywa się *dziedziczeniem* (ang. inheritance). Klasa wywiedziona nazywa się *podklasą* (ang. subclass), a klasa, z której wywiedziono inną – *nadklasą* bądź *superklasą* (ang. superclass). Atrybuty i metody nadklasy są dziedziczone przez wszystkie podklasy z niej wywiedzione. Podklasa może również zawierać: dodatkowe atrybuty i (lub) dodatkowe metody, atrybuty odziedziczone i (lub) metody odziedziczone, które zostały przedefiniowane lokalnie. Na skutek dziedziczenia, w wielu klasach istnieją takie same metody, ale operujące na różnych obiektach. W wyniku przedefiniowania w podklasie metod odziedziczonych, mogą istnieć metody o takich samych nazwach, lecz różniące się semantyką i implementacją. Ta własność nazywa się *polimorfizmem* (ang. polymorphism).

3.2. Koncepcje perspektywy obiektowej

Komercyjne systemy zarządzania obiektowymi bazami danych, np. *O2* (Ardent Software) i systemy składowania danych obiektowych, np. *GemStone* (Servio Corporation), *ObjectStore* (Object Design), *Objectivity/DB* (Objectivity), *Poet* (Poet Software), *VERSANT* (Versant Object Technology) nie wspierają mechanizmu perspektyw. Natomiast istnieje kilka systemów prototypowych, w których zaimplementowano ten mechanizm. Ponieważ nie istnieje jednak standard definiujący perspektywę obiektową trudno mówić o jednoznacznej definicji takiej perspektywy. W systemach prototypowych i opracowaniach naukowych można wyróżnić cztery podstawowe podejścia do definicji perspektywy obiektowej: zapytanie, funkcja, klasa wirtualna, schemat. Szersze omówienie koncepcji perspektyw obiektowych Czytelnik może znaleźć w [17, 20]. Ze względu na ograniczoną liczbę stron niniejszego artykułu, koncepcje perspektyw obiektowych zostaną przedstawione w skrócie.

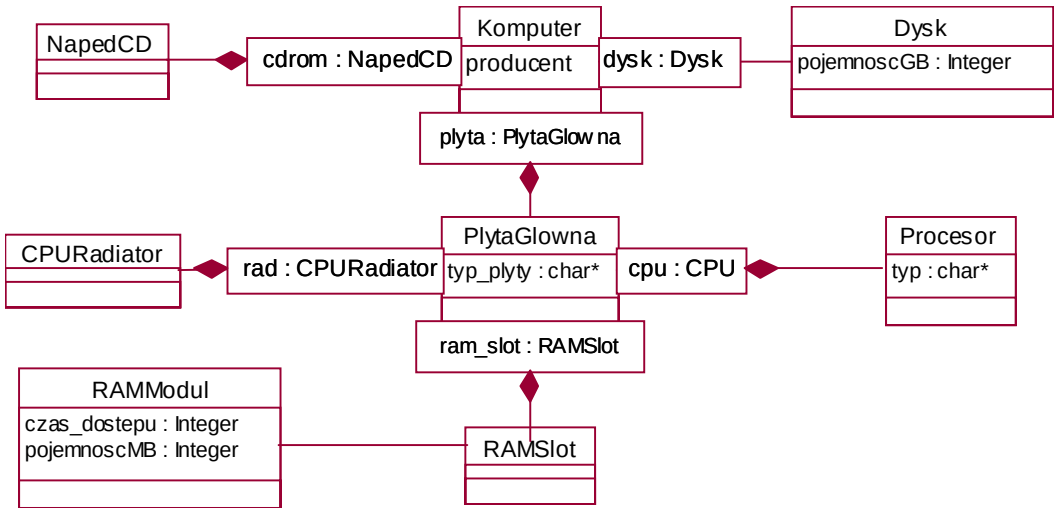
W podejściu pierwszym [7, 14] perspektywę definiuje się jako zapytanie (podobnie, jak perspektywę relacyjną) operujące na klasach. Takie rozwiązanie ma jednak tę wadę, że złożona struktura klas bazowych (związki kompozycji, dziedziczenia) zostaje "spłaszczona" w zapytaniu. W związku z powyższym koncepcja ta nie jest odpowiednia w zastosowaniach projektowych, gdzie ważne jest zachowanie złożonej struktury danych.

W podejściu drugim [15] perspektywę definiuje się jako funkcję składowaną, której wynikiem działania jest kolekcja obiektów wyznaczonych przez tę funkcję. Funkcja może zawierać warunki wyboru odpowiednich obiektów. Nie jest jednak możliwa restrukturalizacja danych źródłowych.

W koncepcji trzeciej (por. [20]) perspektywa jest definiowana jako klasa, tzw. *klasa wirtualna*, a w koncepcji czwartej (por. [20, 21]) – jako *schemat* złożony z wielu klas wirtualnych połączonych związkami dziedziczenia i kompozycji. W obu tych koncepcjach definicja perspektywy składa się z dwóch elementów: (1) definicji struktury klasy wirtualnej i (2) zapytania lub metody wyznaczającej obiekty dostępne za pomocą tej klasy, tj. perspektywy. Klasa wirtualna jest definiowana w oparciu o tzw. klasy bazowe. Zapytanie jest formułowane w obiektowym języku zapytań OQL, składnią i funkcjonalnością przypominającym język SQL, lecz posiadającym większą funkcjonalność. Zapytanie takie może wykorzystywać m.in. klasyczne operatory projekcji, selekcji, połączenia,

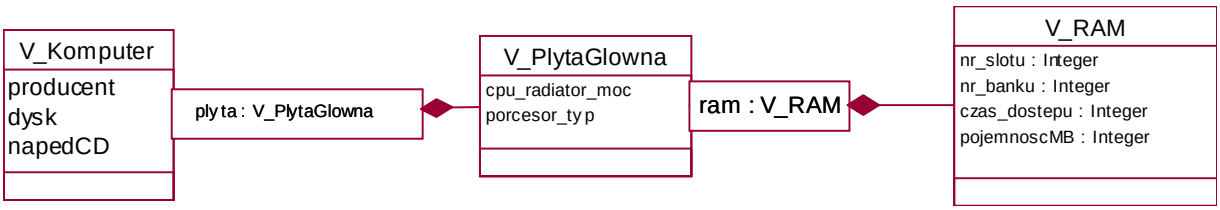
operatory zbiorowe, grupowanie, operacje na atrybutach oraz wywołania metod klas bazowych. Obiekty dostępne za pomocą perspektywy mogą być również wyznaczane w sposób proceduralny – za pomocą obiektowego języka 3GL wspieranego przez bazę danych.

W przypadku perspektywy obiektowej definiowanej jako pojedyncza klasa powstaje problem umieszczenia takiej klasy w hierarchii dziedziczenia klas bazowych. Na skutek utworzenie klasy wirtualnej, hierarchia dziedziczenia często wymaga przebudowy po to, aby nową klasę wirtualną umieścić we właściwym miejscu tej hierarchii. Jest to wadą tego rozwiązania.



Rys. 4. Przykładowy schemat bazowy

W celu zilustrowania koncepcji perspektywy definiowanej jako schemat, rozważmy schemat bazowy złożony z ośmiu następujących klas bazowych: *Komputer*, *NapędCD*, *Dysk*, *PłytaGłówna*, *CPURadiator*, *Procesor*, *RamSlot* i *RAMModuł*. Klasy te są połączone związkami kompozycji. Schemat ten, w notacji UML [5] został przedstawiony na rysunku 4.



Rys. 5. Przykładowa perspektywa obiektowa zdefiniowana jako schemat

Na podstawie klas tego schematu utworzono schemat perspektywy, złożony z trzech następujących klas: *V_Komputer*, *V_PłytaGłówna* i *V_RAM*, przedstawiony na rysunku 5. Klasa wirtualna *V_Komputer* została wywiedziona z klas bazowych *Komputer*, *NapędCD* i *Dysk* przez ich połączenie i projekcję wybranych atrybutów. Zbiór wystąpień klasy *V_Komputer* może zostać ograniczony do wybranych obiektów, spełniających odpowiednie warunki. Klasa *V_PłytaGłówna* została wywiedziona z klas bazowych *PłytaGłówna*, *CPURadiator* i *Procesor* przez ich połączenie i projekcję wybranych atrybutów. W podobny sposób wywiedziono klasę wirtualną *V_RAM* z klas bazowych *RAMModuł* i *RAMSlot*. Do tak zdefiniowanych klas schematu perspektywy można wydawać zapytania w języku OQL tak, jak do klas bazowych.

4. Perspektywa relacyjno–obiektowa w *Oracle8*

Perspektywa relacyjno–obiektowa jest nowym rodzajem perspektywy, wprowadzonym w relacyjno–obiektowej bazie danych *Oracle8*.

4.1. Rozszerzenia obiektowe w SZBD *Oracle8*

System Zarządzania Bazą Danych *Oracle8* został rozszerzony o następujące mechanizmy obiektowe (por. [9, 19]):

- możliwość definiowania własnych typów danych o złożonej strukturze;
- możliwość definiowania operacji (metod) na typach;
- możliwość wykorzystania predefiniowanych typów danych reprezentujących kolekcje obiektów oraz predefiniowanych metod operujących na tych kolekcjach;
- tożsamość obiektów, zrealizowaną za pomocą unikalnych identyfikatorów obiektów (ang. object identifiers) nadawanych przez system;
- przeciążanie nazw metod.

W *Oracle8* rozluźniono jednak własność hermetyczności obiektów. Oznacza to, że do atrybutów obiektu można się odwoływać zarówno za pomocą metod, jak również bezpośrednio – adresując wybrany atrybut. *Oracle8* nie wspiera mechanizmu dziedziczenia i późnego wiązania.

Trwałość wystąpień danego typu (tj. obiektów) realizuje się poprzez zapisanie ich w tabeli związanej z tym typem.

4.2. Definicja perspektywy relacyjno–obiektowej

Perspektywy relacyjno–obiektowe umożliwiają odwzorowanie relacyjnego modelu danych w model obiektowy i odwrotnie, odwzorowanie modelu obiektowego w relacyjny. Ta nowa własność umożliwia pracę starych aplikacji relacyjnych na strukturach danych modelu obiektowego. Perspektywy obiektowe umożliwiają również budowanie obiektowych aplikacji pracujących na relacyjnych bazach danych.

W celu zilustrowania sposobu definiowania nieskomplikowanej perspektywy relacyjno–obiektowej w *Oracle8* rozważmy poniższy przykład. Załóżmy, że w bazie istnieją dane relacyjne, przechowywane w dwóch tabelach *komputery* i *dyski*, o schematach przedstawionych poniżej:

```
SQL> desc komputery
 Nazwa          NULL?    Typ
 -------------- -------- ------------
 KOMP_ID        NOT NULL NUMBER(2)
 TYP_PLYTY      NOT NULL VARCHAR2(10)
 RAM            NOT NULL NUMBER(3)
 PROCESOR                VARCHAR2(15)
```

```
SQL> desc dyski
 Nazwa           NULL?    Typ
 --------------- -------- ------------
 DYSK_ID         NOT NULL NUMBER(4)
 KOMP_ID                  NUMBER(2)
 POJEMNOSCGB     NOT NULL NUMBER(5,1)
 PRODUCENT       NOT NULL VARCHAR2(20)
```

Dane te mają być prezentowane w formacie obiektowym. W tym celu zostaną zdefiniowane dwa typy obiektowe i perspektywa dokonująca odpowiedniej transformacji. Typy obiektowe *TypDysk* i *TypKomputer* będą reprezentowały struktury obiektowe dla danych pochodzących odpowiednio z tabel *dyski* i *komputery*. Definicje tych typów podano poniżej:

```
create or replace type TypDysk as object
(dysk_id number(4),
 pojemnoscGB number(5,1),
 producent varchar2(20));

create or replace type TypNestedDysk as table of TypDysk;

create or replace type TypKomputer as object
(komp_id number(4),
 typ_plyty varchar2(10),
 ram number(3),
 procesor varchar2(15),
 dysk TypNestedDysk);
```

Typ o nazwie *TypNestedDysk* posłuży do zaimplementowania kolekcji dysków zamontowanych w danym komputerze. Jest to kolekcja typu zagnieżdżona tabela (ang. nested table) (por. [19]).

Powyższe typy posłużą jako konstruktory obiektów udostępnianych przez relacyjno–obiektową perspektywę. Jej definicja została przedstawiona poniżej:

```
create or replace view ov_komputery
of TypKomputer
with object identifier (komp_id)
as
select komp_id, typ_plyty, ram, procesor,
  cast (multiset (select TypDysk(dysk_id, pojemnoscGB, producent)
                  from dyski
                  where komp_id=k.komp_id) as TypNestedDysk)
from komputery k;
```

Klauzula ***of*** powyższego polecenia, służy do wskazania typu obiektowego (w naszym przypadku *TypKomputer*) definiującego typ danych dostępny przez perspektywę *ov_komputery*. W momencie odwołania się w zapytaniu do tej perspektywy tworzone są jej wystąpienia, tj. dostępne przez nią obiekty. Obiekty te nie są trwałe i nie posiadają identyfikatorów. Własność tożsamości wymaga jednak, aby wszystkie obiekty były jednoznacznie identyfikowane. W celu wskazania zbioru atrybutów jednoznacznie identyfikujących obiekt jest wykorzystywana klauzula ***with object identifier***. W powyższym przykładzie atrybutem kluczowym jest *komp_id*.

Dla każdego rekordu przetwarzanego przez zapytanie definiujące perspektywę są wyznaczane wartości atrybutów *komp_id*, *typ_plyty*, *ram*, *procesor* i *dysk*. Atrybut *dysk* został zdefiniowany jako kolekcja dysków. W celu wyznaczenia wartości atrybutu *dysk*, jest uruchamiane podzapytanie skorelowane, które za pomocą konstruktora *TypDysk* tworzy obiekty typu *TypDysk*. Obiekty te są następnie transformowane za pomocą operatora ***multiset*** do zbioru wieloelementowego, a następnie, za pomocą operatora ***cast ... as***, do kolekcji typu *TypNesteDysk*.

Przykładowe zapytanie do perspektyw *ov_komputery* i jego wynik przedstawiono poniżej:

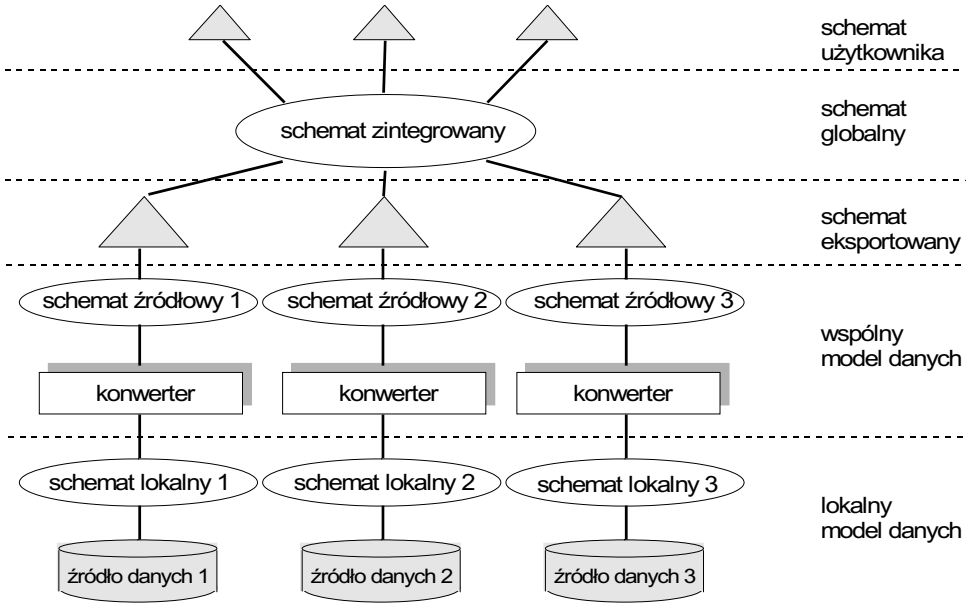
```
SQL> select typ_plyty plyta, procesor, dysk from ov_komputery;

PLYTA  PROCESOR    Dysk(DYSK_ID, POJEMNOSCGB, PRODUCENT)
------ ----------  -----------------------------------------
ATX    PentiumII   TypNestedDysk(TypDysk(1, 7,1, 'Seagate'),
                                  TypDysk(2, 10,2, 'Western Digital'))
ATX2   PentiumIII  TypNestedDysk(TypDysk(3, 20,5, 'Fujitsu'),
                                  TypDysk(4, 12, 'Maxtor'),
                                  TypDysk(5, 10, 'Maxtor'))
```

5. Integracja danych za pomocą perspektyw

Rozwijające się gałęzie przemysłu, handlu, medycyny, nauki wymagają składowania i przetwarzania dużych ilości danych. Dane te są zwykle przechowywane w systemach informatycznych posiadających różne struktury i wykorzystujących różne modele danych (np. hierarchiczne, relacyjne, obiektowe), w dokumentach tekstowych, czy arkuszach kalkulacyjnych. Ponieważ przedsiębiorstwa, instytucje, organizacje (producenci danych) są często geograficznie rozproszone, więc same dane mają również charakter rozproszony. Heterogeniczność i rozproszenie informacji utrudnia do nich dostęp i ich analizę. Z tego powodu niezbędne staje się wprowadzenie mechanizmów umożliwiających integrację rozproszonych i heterogenicznych danych. Jak wspomniano, integrację danych można zrealizować w systemie sfederowanych baz danych lub w magazynie danych (ang. data warehouse) [10].

W podejściu pierwszym, użytkownik operuje na globalnym, zintegrowanym schemacie bazy danych. Zapytanie użytkownika jest dekomponowane na zapytania kierowane do lokalnych baz

danych. Dekompozycja ta jest realizowana przez moduł programowy tzw. *mediator*. Do każdej z lokalnych baz danych jest kierowane zapytanie w języku „rozumianym" przez tę bazę. Następnie, wyniki każdego z zapytań są transformowane do modelu, w którym zaimplementowano schemat globalny, filtrowane i łączone w jeden zbiór danych wynikowych. Architekturę takiego systemu przedstawiono na rysunku 6.



Rys. 6. Architektura systemu z sfederowanych baz danych

Na rysunku tym integrowane są trzy źródłowe bazy danych (oznaczone jako *źródło danych 1, 2* i *3*), z których każda posiada swój własny model danych i schemat (oznaczone jako *schemat lokalny 1, 2* i *3*).

Zadaniem modułu *konwertera* jest transformowanie danych z modelu (formatu) wykorzystywanego w źródle, do jednolitego modelu integracyjnego, np. relacyjnego, relacyjno–obiektowego, obiektowego, semistrukturalnego. Schematy przetransformowane do wspólnego modelu oznaczono jako *schemat źródłowy 1, 2* i *3*. Dla każdego modelu danych źródłowych konieczne jest zastosowanie specyficznego modułu *konwertera*. Nie cały schemat źródłowy musi być udostępniany w systemie sfederowanym. Wybór części schematu źródłowego podlegającego integracji realizuje się za pomocą perspektyw, na rysunku reprezentowanych przez *schemat eksportowany*. Dopiero schematy eksportowane podlegają integracji i wchodzą w skład *schematu zintegrowanego*.

Na schemacie zintegrowanym operują użytkownicy, realizując za jego pośrednictwem dostęp do heterogenicznych i rozproszonych źródeł danych. Na schemacie zintegrowanym mogą również być zdefiniowane perspektywy, stanowiące tzw. *schemat użytkownika*, ograniczające dostęp tylko do wybranych informacji.

W podejściu drugim, tj. magazynu danych, rozproszone i heterogeniczne dane są transformowane do modelu magazynu, filtrowane i łączone, a następnie zapisane w jednym miejscu – magazynie danych. Zapytania użytkowników są kierowane do magazynu. Architektura integracyjna systemu magazynu danych jest podobna do tej z rysunku 6.

6. Perspektywa zmaterializowana

Kolejnym rodzajem perspektywy jest tzw. *perspektywa zmaterializowana* (ang. materialized view). Jednym z zadań perspektywy zmaterializowanej jest skrócenie czasu wykonywania skomplikowanych, czasochłonnych zapytań, zawierających połączenia i grupowania. Materializowanie danych ma w tym przypadku sens jeżeli w systemie często pojawiają się

zapytania identyczne lub podobne do tego, które występuje w definicji perspektywy. Dodatkowo, dane w tabelach bazowych takiej perspektywy nie powinny ulegać częstemu modyfikowaniu. Jeżeli w systemie pojawi się zapytanie, które może zostać wykonane z wykorzystaniem zmaterializowanych perspektyw, zamiast korzystania z tabel bazowych, wówczas optymalizator zapytań skonstruuje odpowiednie zapytanie do tych perspektyw. Jest to tzw. *zamiana zapytania* (ang. query rewriting). Proces ten jest niewidoczny dla użytkownika.

Perspektywa zmaterializowana fizycznie przechowuje rekordy będące wynikiem zapytania ją definiującego. Perspektywa zmaterializowana może być indeksowana i partycjonowana, podobnie jak tabela. Można również dla niej zdefiniować parametry składowania i przestrzeń tabel.

W przypadku zmaterializowanej perspektywy, podobnie jak w przypadku migawki, powstaje problem uaktualniania jej zawartości, w przypadku modyfikowania zawartości tabel bazowych tej perspektywy. System *Oracle* umożliwia odświeżanie perspektywy w jednym z dwóch trybów (por. [9, 13]): pełnym (***complete***) lub przyrostowym (***fast***). Tryb pełny polega na ponownym wykonaniu zapytania definiującego perspektywę i wypełnieniu jej aktualnymi danymi. Tryb przyrostowy polega na przesłaniu do perspektywy tylko tych rekordów, które się zmieniły od czasu ostatniego jej odświeżenia. W tym przypadku niezbędny jest dziennik, w którym transakcja będzie zapisywała zmiany dokonane na danych w tabelach bazowych perspektywy. W trybie przyrostowym mogą być odświeżane tylko perspektywy proste (por. punkt 1.3).

Odświeżanie perspektywy może zostać zrealizowane w trybie synchronicznym lub asynchronicznym. Tryb synchroniczny polega na odświeżeniu perspektywy natychmiast po zakończeniu transakcji modyfikującej dane w jej tabelach bazowych. Tryb asynchroniczny polega na okresowym odświeżaniu perspektywy, z określonym interwałem, lub manualnie, tj. na żądanie użytkownika.

Do manualnego odświeżania zmaterializowanej perspektywy należy wykorzystać pakiet systemowy *DBMS_MVIEW* (w rzeczywistości synonim do pakietu *DBMS_SNAPSHOT*), w skład którego wchodzą procedury umożliwiające odświeżanie perspektywy.

Poniżej przedstawiono przykładowe polecenie tworzące zmaterializowaną perspektywę w bazie danych *Oracle8i*.

```
create materialized view suma_sprzedazy
build immediate
refresh complete
enable query rewrite
as
select nazwa, sum(l_sztuk*cena_jedn) suma
      from sprzedaz sp, sklepy sk
      where sp.sklep_id=sk.sklep_id
      group by nazwa;
```

Klauzula ***build immediate*** powoduje zmaterializowanie perspektywy natychmiast po jej utworzeniu. Perspektywa jest odświeżana w sposób pełny (klauzula ***refresh complete***). Klauzula ***enable query rewrite*** umożliwia wykorzystanie perspektywy do zamiany zapytania. Perspektywa utworzona z klauzulą ***disable query rewrite*** nie będzie wykorzystywana do zamiany zapytania do momentu jej uaktywnienia poleceniem:

```
alter materialized view nazwa_perspektywy
enable query rewrite;
```

O tym, czy zmaterializowana perspektywa zostanie wykorzystana do zamiany zapytania decydują dodatkowo trzy parametry konfiguracyjne instancji: *OPTIMIZER_MODE*, *QUERY_REWRITE_ENABLED* i *QUERY_REWRITE_INTEGRITY*. Zamiana zapytania jest możliwa tylko wówczas, gdy:

1. System wykorzystuje optymalizator kosztowy, tj. (1) parametr konfiguracyjny instancji *OPTIMIZER_MODE* przyjmuje wartość *choose* i (2) dla tabel bazowych zebrano statystyki.

2. Parametr konfiguracyjny *QUERY_REWRITE_ENABLED* przyjmie wartość *true*.

Powyższe parametry można również modyfikować dla pojedynczej sesji użytkownika. Użytkownik wydający zapytanie musi posiadać uprawnienie systemowe *Query Rewrite* lub *Global Query Rewrite*.

Po utworzeniu perspektywy *suma_sprzedazy* w wyżej opisany sposób, zostanie ona wykorzystana do wyznaczenia wyników poniższego zapytania:

```
select nazwa, sum(l_sztuk*cena_jedn) suma
from sprzedaz sp, sklepy sk
where sp.sklep_id=sk.sklep_id
having sum(l_sztuk*cena_jedn) > 30000
group by nazwa;
```

Plan wykonania tego zapytania jest następujący:

```
Execution Plan
----------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (FULL) OF 'SUMA_SPRZEDAZY'
```

W przypadku poniższego zapytania nie zostanie jednak wykorzystana perspektywa *suma_sprzedaży*, a jego wynik zostanie wyznaczony przy pomocy tabel bazowych.

```
select nazwa, sum(l_sztuk*cena_jedn) suma
from sprzedaz sp, sklepy sk
where sp.sklep_id=sk.sklep_id
and sk.miasto='Poznań'
group by nazwa;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (GROUP BY)
   2    1     NESTED LOOPS
   3    2       TABLE ACCESS (FULL) OF 'SKLEPY'
   4    2       TABLE ACCESS (FULL) OF 'SPRZEDAZ'
```

7. Perspektywa semistrukturalna

Internet i technologia Web zrewolucjonizowały sposób prezentowania danych i sposób dostępu do nich. Dotychczas, dane były publikowane najczęściej w formacie HTML. Obecnie, coraz częściej wykorzystuje się do tego celu język XML (zob. [16]), dobrze nadający się do opisu tzw. *danych semistrukturalnych* (ang. semistructured data) występujących w Internecie.

Dane semistrukturalne charakteryzują się tym, że nie mają ściśle określonej struktury. W przypadku tabeli relacyjnej, wszystkie jej rekordy posiadają atrybuty określone schematem tej tabeli. Przykładowo, wszyscy klienci są opisani numerem NIP, imieniem, nazwiskiem, adresem, numerem faksu i telefonu. Inaczej jest w przypadku danych semistrukturalnych. Część danych może posiadać pewne atrybuty, a innych nie posiadać. Przykładowo, część klientów może być opisana tylko numerem NIP, nazwiskiem i telefonem, a inna część klientów – tylko nazwiskiem i adresem.

W przypadku udostępniania danych w formacie XML bardzo ważnym mechanizmem stają się perspektywy. Ich zadaniem jest w tym przypadku:

- integracja dostępnych w Internecie informacji o różnej strukturze,
- umożliwienie nadania ściśle określonej struktury danym semistrukturalnym.

W przypadku perspektyw XML kluczową kwestią jest język zapytań, który umożliwiłby wyszukanie odpowiednich danych semistrukturalnych i prezentowanie ich w formacie perspektywy XML. Mimo, że prowadzone są intensywne prace badawcze [2, 8], język taki, który byłby standardem, jeszcze nie powstał.

Jako przykład potrzeby perspektyw XML rozważmy dwie bazy danych z zawartością udostępnianą w Internecie. Pierwsza z nich jest przewodnikiem turystycznym po Poznaniu i udostępnia informacje na temat ulic i znajdujących się na nich obiektów do zwiedzania. Druga baza jest przewodnikiem gastronomicznym po Poznaniu i zawiera informacje na temat ulic i znajdujących się na nich restauracji. Każda restauracja posiada menu. Turysta odwiedzający Poznań chciałby zaplanować wycieczkę w taki sposób, aby na jej trasie znajdowały się również dobre restauracje, tzn. chciałby on uzyskać zintegrowane informacje pochodzące z obu baz danych, np. w następującej postaci (nawias klamrowy oznacza zbiór wystąpień):

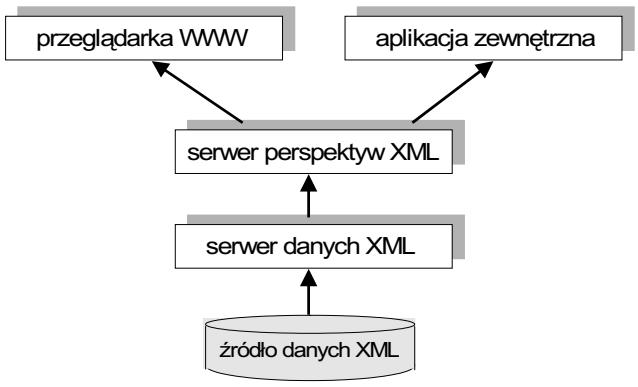
```
{dzielnica, {ulica, {obiekt do zwiedzania}, {restauracja, {menu} } } }
```

Innymi słowy, turysta chciałby wydać zapytanie: „*pokaż wszystkie ulice Starego Miasta wraz z obiektami do zwiedzania i restauracjami, a dla każdej restauracji pokaż jej menu*”.

7.1. Koncepcje

Jeżeli chodzi o perspektywy semistrukturalne, to istnieją trzy prototypowe systemy *Active View* [1, 3] i *Ozone* [3, 12] oraz *MIRO Web* [6]. Architektura systemu *Active View* została przedstawiona na rysunku 7.

Serwerem danych XML może być baza danych lub dowolne inne repozytorium posiadające możliwość prezentowania danych w formacie XML. W opisywanej prototypowej implementacji serwerem danych jest obiektowa baza danych *O2*, przechowująca dane XML w obiektowych strukturach danych. Serwerem perspektyw XML jest aplikacja napisana w języku Java. Aplikacja ta jest odpowiedzialna za restrukturalizację danych zgodnie z definicją perspektywy, autoryzację dostępu i integrację z innymi źródłami danych. Dane przekształcone przez tę aplikację są dostępne za pomocą przeglądarki WWW lub innej aplikacji klienta (np. programu w Java).



Rys. 7. Architektura systemu *Active View*

Podobne rozwiązanie do omówionego wyżej, zastosowano w systemie *MIRO Web*. Polega ono na przetransformowaniu danych semistrukturalnych do struktur relacyjno–obiektowych i przechowywanie tak przetworzonych danych w bazie relacyjno–obiektowej (*Oracle8*). Konwersja danych jest realizowana przez odpowiedni moduł programowy. Następnie, korzystając z mechanizmów bazy danych można zdefiniować perspektywy relacyjno–obiektowe na tych danych lub wykorzystać oprogramowanie w Java symulujące perspektywy.

W systemie *Ozone* dane semistrukturalne są modelowane jako graf, a implementowane jako kolekcje par <*etykieta, wartość*>, gdzie wartością może być ciąg znaków, liczba, data, itp., lub wskazanie do innego węzła grafu. Perspektywa jest reprezentowana przez obiekt, tzw. *proxy*, umożliwiający transformację danych semistrukturalnych do strukturalnych i odwrotnie.

7.2. Otwarte problemy badawcze

Jak wspomniano, jednym z zadań perspektywy semistrukturalnej jest zapewnienie dostępu do wielu źródeł danych w sieci Internet. W celu skrócenia dostępu do danych i uniezależnienia się od

czasowej niedostępności węzłów sieci, bardzo pożądanym jest możliwość zmaterializowania danych udostępnianych przez taką perspektywę.

Jednak w przypadku perspektyw XML problem uaktualniania zmaterializowanych perspektyw jest znacznie trudniejszy niż w przypadku klasycznych baz danych. Wynika to z następujących czynników:

1. Źródła danych, które wykorzystano do zmaterializowania perspektywy mogą być czasowo niedostępne, co uniemożliwia odświeżenie perspektywy.
2. Trudno oszacować koszt wykonania zapytań odświeżających taką perspektywę, ponieważ jej definicja XML zwykle odwołuje się do wielu rozproszonych i heterogenicznych źródeł danych (niekoniecznie baz danych). Koszt taki musi również uwzględniać czasy dostępu do każdego z węzłów i przepustowość sieci. Występują więc problemy optymalizacji zapytań.
3. Odświeżanie przyrostowe w wielu przypadkach nie będzie możliwe ze względu na brak dziennika operacji na danych źródłowych. Pociąga to za sobą konieczność odświeżania pełnego, co może być operacją czasochłonną.

8. Podsumowanie

Perspektywa jest mechanizmem niezbędnym w systemach baz danych. Cele jej stosowania są wielorakie, m.in.: ograniczenie i autoryzacja dostępu do danych, zapewnienie logicznej niezależności danych, uproszczenie schematu bazy danych, integracja i transformacja danych, materializowanie często wykorzystywanych, czasochłonnych zapytań. Mechanizm ten jest wspierany przez wszystkie relacyjne i relacyjno–obiektowe systemy zarządzania bazami danych, ale nie został zaimplementowany w żadnym z komercyjnych systemów obiektowych baz danych.

Rozwój systemów internetowych wymusza wprowadzenie mechanizmu perspektyw również do technologii Web. Obiecującym jest w tym przypadku język XML. Perspektywy definiowane w tym języku umożliwią integrację danych w sieci Internet. Obecnie wiele ośrodków naukowych (m.in. INRIA, Uniwersytety Stanford i Washington), prowadzi prace badawcze nad integracją Web'u za pomocą języka XML.

Bibliografia

1. Abiteboul S., Amann B., Cluet S., Milo T., Vianu V.: Active views for electronic commerce. Materiały konferencyjne: Conférences sur les Bases de Donées, 1998
2. Abiteboul S., Buneman P., Suciu D.: Data on the Web: from relations to semistructured data and XML. Morgan Kaufmann Publishers, 2000, ISBN 1-55860-622-X
3. Abiteboul S.: On Views and XML. SIGMOD Record, Vol. 28, No. 4, grudzień, 1999
4. Ault M.: Oracle8i Administration and Management. Wiley Computer Publishing, 2000, ISBN 0-471-35453-8
5. Booch G., Rumbaugh J., Jacobson I.: The Unified Modeling Language User Guide. Addison–Wesley, 1999, ISBN 0-201-57168-4
6. Bouganim L., Chan–Sine–Ying T., Tuyet–Tram Dang–Ngoc, Darroux J.L., Gardarin G., Sha F.: MIRO Web: Integrating Multiple Data Sources through Semistructured Data Types. Materiały konferencyjne: Very Large DataBases, 1999
7. Cattell R., G., G., Barry D., Berler M., Eastman J., Jordan D., Russel C., Shadow O., Stanienda T., Velez F.: Object Database Standard: ODMG 3.0, Morgan Kaufmann Publishers, 2000
8. Ceri S., Fraternali P., Paraboschi S.: XML: Current Developments and Future Challenges for the Database Community. Materiały konferencyjne: Advances in Database Technology – EDBT'2000, LNCS 1777
9. Dokumentacja RDBMS Oracle8i, rel. 8.1.6

10. Elmargamid A., Rusinkiewicz M., Sheth A.: Management of Heterogeneous and Autonomous Database Systems. Morgan Kaufmann Publishers, Inc. 1999, ISBN 1-55860-216-X
11. Gupta A., Mumick I.S.: Materialised Views: Techniques, Implementations, and Applications. MIT Press, 1999, ISBN 0-262-57122-6
12. Lahiri T., Abiteboul S., Widom J.: Ozone: Integrating structured and semistructured data. www.db.stanford.edu/tlahiri/ozone.pdf
13. Materiały szkoleniowe Oracle Polska: Oracle8i: New Features for Administrators
14. S. I. Yoo, H. J. Chang, *An Object–Oriented Query Model Supporting Views*, in Object Technologies for Advanced Software, First JSSST Int. Symposium, Japan, 1993, vol. 742 of LNCS, 1993
15. Subieta K., Płodzień J.: Object Views and Query Modification. Materiały konferencyjne: 4th IEEE International Baltic Workshop on Databases & Information Systems, Lithuania, 2000
16. Traczyk T.: Wprowadzenie do języka XML. Informatyka, 12/1999
17. Wieczerzycki W., Wrembel R.: Perspektywy w obiektowych bazach danych. Informatyka 10/95
18. Wieczerzycki W., Wrembel R.: Perspektywy w relacyjnych bazach danych. Informatyka 8/95
19. Wrembel R. Jezierski J., Zakrzewicz M.: System Zarządzania Bazą Danych Oracle7 i Oracle8. Wydawnictwo NAKOM, 1999, ISBN 83-86969-34-2
20. Wrembel R.: Object-Oriented Views: Virtues and Limitations. Materiały konferencyjne: 13th International Symposium on Computer and Information Sciences – ISCIS'98, 1998