

Implementacja indeksów dla analizy wyników eksploracji danych w Oracle 10g

Witold Andrzejewski, Mariusz Masewicz

Instytut Informatyki, Politechnika Poznańska
e-mail: Witold.Andrzejewski@cs.put.poznan.pl, Mariusz.Masewicz@cs.put.poznan.pl

Abstrakt

Olbrzymie bazy danych dotyczących sprzedaży produktów w supermarketach są zbyt duże, by można na nich przeprowadzić „ręczną” analizę. Odpowiedzią na ten problem jest użycie algorytmów eksploracji danych, czyli tzw. "analizy koszyka sklepowego", do automatycznej analizy danych w celu otrzymania użytecznej wiedzy. W wyniku takiej analizy (w zależności od parametrów algorytmów) można otrzymać olbrzymią liczbę wzorców, takich jak: zbiory częste, reguły asocjacyjne albo wzorce sekwencji. Wśród tych wzorców znajduje się wiele zależności, które są przypadkowe, nieciekawe, bądź już znane. Zadaniem człowieka jest odnalezienie wśród wyników eksploracji danych wzorców ciekawych i o praktycznym zastosowaniu. Taka analiza wymaga wykonania wielu zapytań do danych o złożonych typach, które są słabo wspierane przez komercyjne systemy zarządzania bazą danych. Rozwiązanie tego problemu jest możliwe w Oracle 10g, dzięki mechanizmowi *Data Cartridges* pozwalającemu na rozszerzenie funkcjonalności serwera Oracle. Pozwala on na implementację własnych indeksów wspierających różne typy zapytań na danych o złożonych typach. W niniejszej publikacji przedstawiono rozwiązania pozwalające na indeksowanie złożonych typów danych charakterystycznych dla analizy koszyka sklepowego, oraz sposób ich implementacji przy wykorzystaniu mechanizmu *Data Cartridges*.

1. Wstęp

Upowszechnienie systemów baz danych, magazynów danych i innych repozytoriów spowodowało, że w ostatnim czasie, w systemach tych nagromadzono olbrzymie wolumeny danych. Szacuje się, że od 2000r. rozmiar przechowywanych danych wzrósł dwukrotnie. Każdego roku gromadzonych jest 5 milionów terabajtów nowych danych. Kluczowy staje się zatem problem analizy i dostępu do tych ogromnych wolumenów.

Ręczna analiza olbrzymich wolumenów danych przechowywanych w systemach baz danych i magazynach danych staje się coraz mniej realistyczna. Z drugiej strony, przechowywane dane mogą stanowić potencjalne źródło istotnej i użytecznej wiedzy z punktu widzenia zastosowań naukowych i biznesowych. Istnieje zatem ewidentna potrzeba opracowania nowoczesnej technologii umożliwiającej analizę danych w celu pozyskania użytecznej i nietrywialnej wiedzy. Odpowiedzią na tą potrzebę jest technologia eksploracji danych. Mianem eksploracji danych określa się odkrywanie użytecznej i nietrywialnej wiedzy, w dużych wolumenach danych, rozumianej tutaj jako zależności, trendy i wzorce w nich występujące. Ułatwia ona opracowywanie nowych planów i strategii biznesowych zwiększających zyski firmy, pomaga w analizie obserwowanych zjawisk i pozwala zaproponować nowe teorie poprawiające zrozumienie otaczającego nas świata.

Technologia eksploracji danych umożliwia analizę różnych typów danych takich jak: łańcuchy białkowe i DNA, dane z kas fiskalnych, ceny akcji spółek giełdowych czy historia dostępu do stron WWW. Jednym z najbardziej popularnych zastosowań eksploracji danych jest analiza danych z kas fiskalnych, czyli tzw. analiza koszyka sklepowego. W wyniku analizy koszyka sklepowego można uzyskać zbiory produktów, które są często kupowane razem, bądź tzw. reguły asocjacyjne. Reguły asocjacyjne to reguły postaci: poprzednik→następnik, gdzie „poprzednik” i „następnik” są zbiorami produktów. Reguły te mówią, że jeżeli klient kupi produkty ze zbioru „poprzednik”, to istnieje wysokie prawdopodobieństwo, że kupi również produkty ze zbioru „następnik”. Znajomość takich reguł pozwala na przygotowywanie lepszych planów marketingowych, reklam, promocji, rozmieszczenia produktów w sklepach itp.

Jak łatwo zauważyć, rozmiary baz danych z kas fiskalnych osiągają olbrzymie rozmiary, co wynika z olbrzymiej liczby transakcji wykonywanych dziennie (np. w sieci sprzedaży Wal-Mart jest to ok. 20 milionów transakcji). Liczba odnalezionych w tych danych wzorców może być również olbrzymia a nawet może przekraczać rozmiar oryginalnej bazy danych [BMMR02]. Dlatego też, wyniki eksploracji danych muszą być również składowane w bazie danych, w celu późniejszej analizy. Na taką analizę składa się pobieranie interesujących wzorców i porównywanie ich z transakcjami, które albo potwierdzają, bądź zaprzeczają odnalezionym regułom [MKP03]. Analityk odszukuje interesujące go wzorce specyfikując jakie elementy powinny zawierać poprzednik i/lub następnik reguły asocjacyjnej. Standard SQL 2003 przewiduje możliwość konstrukcji takich zapytań za pomocą nowych operatorów operujących na zbiorach. Operatory te zostały zaimplementowane w Oracle 10g [SQL03]. Niestety, zgodnie z wiedzą autora, zapytania dotyczące zbiorów nie są wydajnie wspierane przez żaden dostępny obecnie komercyjny system zarządzania bazą danych (w tym Oracle 10g). Problem ten jest możliwy do rozwiązania, jeżeli system zarządzania bazą danych daje użytkownikowi możliwość implementacji własnych indeksów dla specyficznych typów danych i specyficznych typów zapytań. Mechanizm ten jest dostępny m. in. w systemie zarządzania bazą danych Oracle pod nazwą *Data Cartridges* [DCDG].

W niniejszym artykule przedstawiono logiczną i fizyczną strukturę indeksu wspierającego zapytania polegające na odszukiwaniu wszystkich zbiorów zawierających zadany przez użytkownika zbiór, oraz przedstawiono jego przykładową implementację za pomocą mechanizmu *Data Cartridges*. Przedstawiona implementacja została przystosowana do indeksowania danych z kas fiskalnych (przeprowadzonych transakcji) oraz wyników ich eksploracji uzyskanych za pomocą

mechanizmów eksploracji danych zaimplementowanych w systemie zarządzania bazą danych Oracle (opcja Oracle Data Mining).

Rozdział 2 artykułu zawiera dokładny opis typu zapytań wspieranego przez prezentowany indeks oraz omawia operator zdefiniowany w standardzie SQL 2003, który pozwala na ich konstrukcję. Rozdział 3 zawiera opis logicznej i fizycznej struktury indeksu, oraz algorytmy realizacji zapytań z jego wykorzystaniem. W punkcie 4 przedstawiono sposób implementacji indeksu za pomocą mechanizmu *Data Cartridges*. Rozdział 5 podsumowuje artykuł.

2. Zapytania o nadzbiory

Jak wspomniano wcześniej, chcemy stworzyć indeks, który wspierałby zapytania polegające na odszukiwaniu wszystkich zbiorów zawierających zadany przez użytkownika zbiór. Takie zapytania nazywa się *zapytaniami o nadzbiory* a zadany przez użytkownika zbiór *zbiorem zapytania*. Rozważmy następujący przykład. Niech będzie dana tabela TRANSAKCJE, która przechowuje dane o kolejnych transakcjach przeprowadzonych w supermarkecie. Załóżmy, że tabela ta jest tworzona za pomocą poniższych poleceń.

```
CREATE TYPE SET_TYPE AS TABLE OF VARCHAR2(30);
/

CREATE TABLE TRANSAKCJE (
  ID NUMBER(10) PRIMARY KEY,
  TOWARY SET_TYPE
);
```

Przykładową zawartość tabeli TRANSAKCJE przedstawiono w tabeli 1. Jak łatwo zauważyć, w definicji tabeli wykorzystano typ użytkownika reprezentujący kolekcję łańcuchów. Jest to typ, który będzie wykorzystywany we wszystkich poleceniach w tym artykule, do reprezentacji zbiorów.

Tabela 1. Przykładowa tabela składująca zbiory

ID	TOWARY
1.	{'Piwo','Wino','Apap'}
2.	{'Chleb','Mąka','Mleko'}
3.	{'Chleb','Piwo','Mleko','Mąka'}

Aby skonstruować zapytanie o nadzbiory należy skorzystać z operatora SUBMULTISET OF, który sprawdza, czy jeden ze zbiorów zawiera się w drugim. Operator ten jest częścią standardu SQL 2003 i jest zaimplementowany w Oracle 10g. Przykładowe zapytanie o nadzbiory, wykorzystujące operator SUBMULTISET OF, które odszukuje wszystkie transakcje, w których zakupiono 'Piwo' i 'Apap', wygląda następująco:

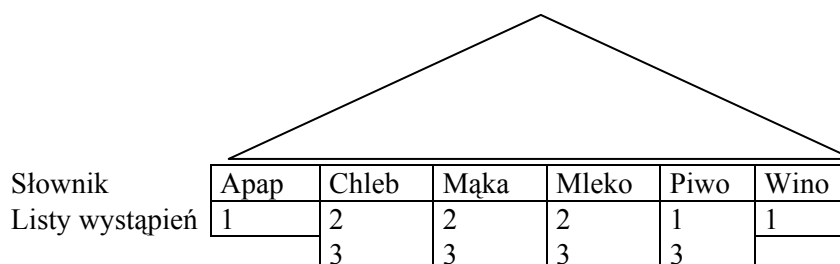
```
SELECT ID, TOWARY FROM TRANSAKCJE
WHERE SET_TYPE('Piwo', 'Apap') SUBMULTISET OF TRANSAKCJE;
```

Jak łatwo zauważyć, w wyniku zapytania zwrócona zostanie krotka opisująca pierwszą transakcję. Zapytania skonstruowane w ten sposób są wykonywane w Oracle 10g bardzo wolno, gdyż operator SUBMULTISET OF nie jest wspierany przez żaden indeks. Konieczne jest zatem zaimplementowanie indeksu wspierającego tego typu zapytania.

3. Indeks „plik odwrócony”

3.1. Struktura logiczna indeksu

Najlepszym indeksem dla zapytań o nadzbiory, jest indeks „plik odwrócony” [AKMM05]. Indeks ten znany jest przede wszystkim z jego zastosowań w dziedzinie Information Retrieval i jest wykorzystywany m. in. do wyszukiwania dokumentów tekstowych [BaRi99]. Plik odwrócony składa się z dwóch części: słownika i list wystąpień. Słownik stanowi listę wszystkich elementów zbiorów, które wystąpiły chociaż raz w bazie danych (każdy element pojawia się na liście tylko raz). Słownik z reguły ma organizację indeksu, pozwalającą na szybkie odszukiwanie elementów w nim zapisanych. Z każdym elementem w słowniku związana jest lista wystąpień, na której znajdują się identyfikatory zbiorów, w których ten element się znajduje. Rysunek 1 przedstawia strukturę pliku odwróconego dla przykładowej tabeli TRANSAKCJE (tabela 1).



Rysunek 1. Logiczna struktura indeksu „plik odwrócony”, dla przykładowej tabeli TRANSAKCJE

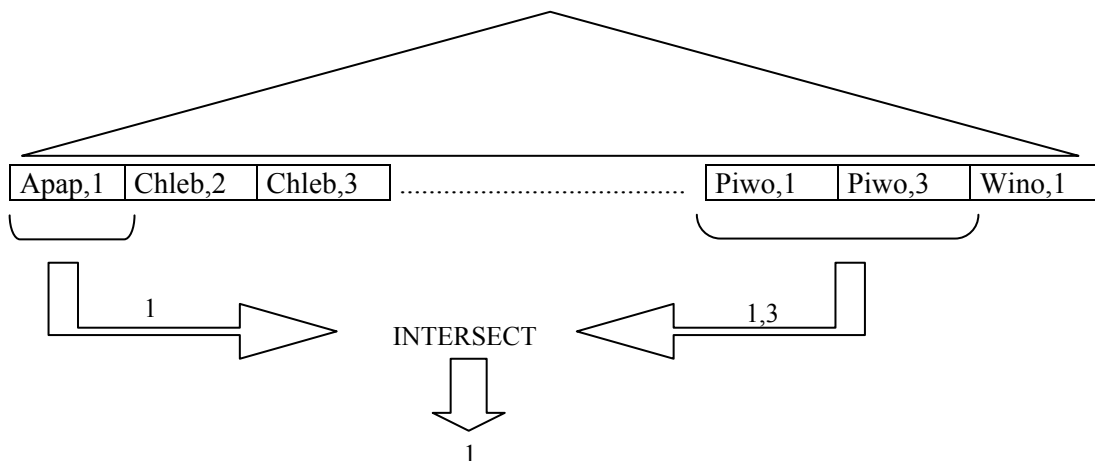
Realizacja zapytań, za pomocą pliku odwróconego, odbywa się poprzez odczyt i analizę odpowiednich list wystąpień. Aby wytłumaczyć algorytm realizacji zapytań, rozważmy najpierw zapytania, w których zbiór zapytania składa się z jednego elementu. Jak łatwo zauważyć, aby zrealizować takie zapytanie (odczytać identyfikatory nadzbiorów zbioru zapytania) wystarczy odczytać listę wystąpień związaną z elementem zawartym w zbiorze zapytania. Identyfikatory odczytane z tej listy stanowią wynik zapytania. W sytuacji, gdy zbiór zapytania zawiera więcej elementów, należy odczytać listy wystąpień związane ze wszystkimi elementami zbioru zapytania i znaleźć identyfikatory, które występują na wszystkich odczytanych listach (znaleźć część wspólną tych list). Rozważmy przykładowe zapytanie przedstawione w poprzednim rozdziale. W zapytaniu tym, szukamy wszystkich nadzbiorów zbioru {‘Piwo’, ‘Apap’}. W celu realizacji zapytania za pomocą indeksu, odczytujemy listy wystąpień odpowiadające elementom: ‘Piwo’ (1,3) i ‘Apap’ (1). Ponieważ jedynie identyfikator 1 występuje na obu tych listach, jedynie zbiór o numerze 1 z przykładowej tabeli stanowi wynik zapytania.

3.2. Struktura fizyczna indeksu

Indeks „plik odwrócony” można w prosty sposób zaimplementować jako tabelę zorganizowaną jako indeks (IOT). Rozważmy tabelę zorganizowaną jako indeks, złożoną z dwóch kolumn: SET_NUMBER i SET_ELEMENT, których konkatenacja, w kolejności (SET_ELEMENT, SET_NUMBER), tworzy klucz podstawowy. Niech każdy wiersz tej tabeli reprezentuje informację o tym, że w zbiorze SET_NUMBER znajduje się element SET_ELEMENT. Jak łatwo zauważyć, wszystkie wartości SET_NUMBER dla konkretnej wartości SET_ELEMENT tworzą listę wystąpień tego elementu. Ponieważ rozważana tabela ma organizację indeksu, a kolumna SET_ELEMENT stanowi prefiks klucza podstawowego, wiersze reprezentujące kolejne identyfikatory zbiorów, w których znajduje się dany element są zgrupowane w sąsiadujących blokach na dysku. Możliwe jest zatem wydajne odczytanie listy wystąpień dla zadanego elementu bez konieczności czytania całej tabeli.

Jak wspomniano w podpunkcie 3.1, zapytania o nadzbiory są realizowane poprzez znalezienie części wspólnej list wystąpień poszczególnych elementów wchodzących w skład zbioru zapytania. Operację tą można łatwo wykonać za pomocą operatora INTERSECT. Ponieważ kolumna SET_NUMBER wchodzi w skład klucza podstawowego tabeli zorganizowanej jako indeks, wartości na odczytanych listach wystąpień są posortowane, co dodatkowo przyspiesza operację INTERSECT.

Koncepcję struktury fizycznej indeksu i implementacji algorytmu realizacji zapytań ilustruje rysunek 2.



Rysunek 2. Fizyczna struktura indeksu „plik odwrócony”, dla przykładowej tabeli TRANSAKCJE

4. Implementacja indeksu

4.1. Oracle Data Mining i struktura indeksowanych tabel

Algorytmy eksploracji danych zaimplementowane w Oracle 10g (opcja Oracle Data Mining) wymagają specyficznych struktur tabel przechowujących dane podlegające eksploracji. Zbiory w tych tabelach powinny być reprezentowane przez tabele zagnieżdżone typu DM_NESTED_NUMERICALS, lub DM_NESTED_CATEGORICALS. Oba te typy reprezentują zagnieżdżone tabele obiektów typu odpowiednio DM_NESTED_NUMERICAL i DM_NESTED_CATEGORICAL. Obydwa typy obiektowe posiadają atrybuty ATTRIBUTE_NAME i VALUE. Atrybut ATTRIBUTE_NAME jest typu VARCHAR2(30) i powinien przechowywać nazwę, bądź identyfikator elementu zbioru (np. zakupionego przez klienta towaru). Atrybut VALUE jest albo typu NUMBER (DM_NESTED_NUMERICAL), albo typu VARCHAR2 (DM_NESTED_CATEGORICAL). W przypadku analizy koszyka sklepowego atrybut ten, nie ma znaczenia i powinien przyjmować we wszystkich obiektach jedną wartość, np. 1. Tabele przechowujące dane o transakcjach powinny mieć dwie kolumny. Pierwsza kolumna powinna przechowywać identyfikatory transakcji, a druga powinna być jednego z dwóch typów opisanych powyżej i przechowywać zbiory zakupionych towarów. Opisaną wyżej strukturę tabeli przedstawia rysunek 3.

```

CREATE TYPE DM_NESTED_NUMERICAL AS
OBJECT (
  ATTRIBUTE_NAME VARCHAR2(30)
  .....
);

CREATE TYPE DM_NESTED_NUMERICALS AS
TABLE OF DM_NESTED_NUMERICAL;

CREATE TABLE MINED_DATA (
  SET_NUMBER NUMBER(10),
  SET_CONTENTS DM_NESTED_NUMERICALS
);

```

Predefiniowane w schemacie DMSYS

Definiowane przez użytkownika

Rysunek 3. Struktura przykładowej tabeli, która może podlegać eksploracji danych

Wynik eksploracji danych jest składowany w schemacie użytkownika, w wewnętrznej formie Oracle Data Mining. Użytkownik może odczytać te wyniki w postaci zrozumiałej poprzez zaimplementowaną w Oracle Data Mining funkcję tabelaryczną `DBMS_DATA_MINING.GET_ASSOCIATION_RULES`. Wśród kolumn tabeli zwracanej przez tę funkcję można wyróżnić:

- `RULE_ID` - identyfikator reguły asocjacyjnej (typ liczbowy),
- `ANTECEDENT` - tabela zagnieżdżona reprezentująca poprzednika reguły decyzyjnej,
- `CONSEQUENT` - tabela zagnieżdżona reprezentująca następnika reguły decyzyjnej.

Atrybuty `ANTECEDENT` i `CONSEQUENT` są typu `DM_PREDICATES`. Typ ten reprezentuje kolekcję obiektów typu `DM_PREDICATE`. Wśród atrybutów tego typu, z punktu widzenia indeksowania zbiorów, interesujący jest jedynie atrybut `ATTRIBUTE_NAME`, który reprezentuje nazwę, bądź identyfikator elementu zbioru wchodzącego w skład poprzednika, bądź następnika reguły decyzyjnej. Ponieważ wynik eksploracji danych dostępny jest poprzez funkcję tabelaryczną, nie istnieje bezpośrednia możliwość założenia na nim indeksu. Z tego powodu, przed założeniem indeksu, konieczna jest materializacja wyników zwracanych przez funkcję tabelaryczną w postaci zwykłej tabeli, za pomocą polecenia:

```

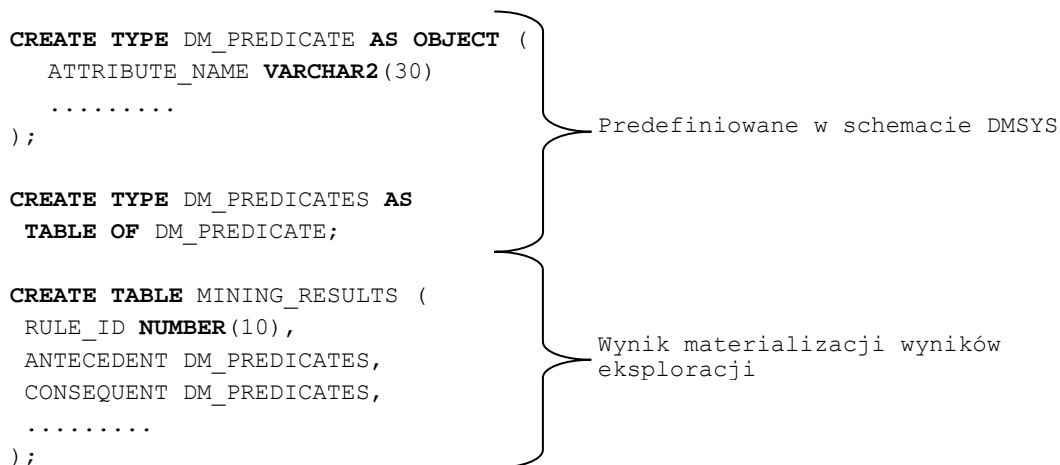
CREATE TABLE NAZWA_TABELI AS
SELECT * FROM TABLE(dbms_data_mining.get_association_rules('NAZWA_MODELU'))

```

gdzie:

- `NAZWA_TABELI`, to nazwa tworzonej tabeli,
- `NAZWA_MODELU`, to nazwa wygenerowanego zbioru wyników eksploracji danych

Strukturę tabeli ze zmaterializowanymi wynikami eksploracji przedstawia rysunek 4.



Rysunek 4. Struktura tabeli stanowiącej wynik materializacji wyników funkcji tabelarycznej zwracającej wyniki eksploracji danych

Konstrukcja zapytań o nadzbiory, z wykorzystaniem operatora `SUBMULTISET OF`, do tabel opisanych w niniejszym rozdziale jest nieco bardziej skomplikowana niż było to w przypadku zapytań do przykładowej tabeli `TRANSAKCJE`. Wynika to z faktu, że kolekcje reprezentujące zbiory składają tutaj obiekty, w których, prócz nazw elementów, składowane są dodatkowe dane, które należy zignorować przy porównywaniu zbiorów. Załóżmy, że tabela `MINED_DATA` przechowuje dane o tych samych transakcjach co tabela `TRANSAKCJE` przedstawiona w rozdziale 2. Zapytanie odszukujące w tej tabeli wszystkie nadzbiory zbioru {'Piwo', 'Apap'}, wygląda następująco:

```

SELECT * FROM MINED_DATA X
WHERE SET_TYPE('Piwo', 'Apap') SUBMULTISET OF CAST(MULTISET(
  SELECT VALUE(Y).ATTRIBUTE_NAME FROM TABLE(X.SET_CONTENT)
) AS SET_TYPE);

```

Jak łatwo zauważyć, w zapytaniu tym wykonujemy konwersję kolekcji obiektów typu `DM_NESTED_NUMERICAL` do kolekcji łańcuchów (`SET_TYPE`), pomijając zbędne atrybuty tych obiektów, a następnie testujemy warunek zawierania się zbiorów za pomocą operatora `SUBMULTISET OF`. Zapytania do tabeli z wynikami eksploracji, z wykorzystaniem operatora `SUBMULTISET OF`, buduje się w analogiczny sposób.

4.2. Implementacja indeksu za pomocą mechanizmu *data cartridges*

4.2.1. Utworzenie operatora

Aby możliwe było wspieranie jakichkolwiek zapytań za pomocą indeksu implementowanego przez użytkownika, należy najpierw zaimplementować operator, którego funkcjonalność będzie wspierana przez indeks. W naszym przypadku należy zatem zaimplementować operator o funkcjonalności zbliżonej do operatora `SUBMULTISET OF` pokazanego wcześniej. Niestety nie jest to do końca możliwe. W przeciwieństwie do operatora `SUBMULTISET OF`, który posiada notację infiksową, operatory definiowane przez użytkowników mają notację prefiksową przypominającą notację wywołania funkcji (najpierw nazwa operatora, potem, w nawiasach, lista parametrów). Operator implementuje się poprzez: stworzenie składowanej funkcji implementującej funkcjonalność operatora (wykorzystywaną w sytuacji, gdy nie jest używany indeks) oraz utworzenie operatora za pomocą polecenia `CREATE OPERATOR`, z wykorzystaniem wcześniej utworzonej funkcji. W naszym przypadku, operator ten musiałby posiadać parametr typu `SET_TYPE` i drugi pa-

parametr typu DM_PREDICATES (albo DM_NESTED_NUMERICALS) i testować, czy jeden ze zbiorów, zawiera się w drugim. Niestety istnieje kilka ograniczeń dotyczących definiowanych przez użytkownika operatorów, które mają być wspomagane przez indeks. Pierwszym ograniczeniem jest to, że pierwszym parametrem operatora musi być parametr dotyczący indeksowanej kolumny. Drugie ograniczenie wynika z faktu, iż Oracle nie pozwala na założenie indeksu na kolumnie typu kolekcji, a zatem pierwszy parametr operatora nie może być typu kolekcji. Jest to niestety sprzeczne z tym, co pisano wcześniej o wymaganych dla naszych celów typach parametrów operatora. Problem ten można ominąć w następujący sposób. Tworzymy specjalną kolumnę (np. typu liczbowego), na której zakładamy indeks. Zadaniem tej kolumny jest reprezentowanie indeksowanego zbioru w zapytaniach. Indeks jednak zostanie zaimplementowany w taki sposób, że będzie indeksował dane z innej kolumny niż ta, na której został faktycznie założony. Niestety takie rozwiązanie uniemożliwia napisanie poprawnie działającej funkcji implementującej funkcjonalność operatora, gdyż pierwszy parametr tej funkcji nie będzie zbiorem. Ponieważ jednak funkcja ta nie będzie wykorzystywana (operator będzie zawsze wspierany przez indeks), nie stanowi to dużego problemu. Utworzenie funkcji jest konieczne jedynie po to by możliwe było utworzenie operatora. Rozważmy obecnie jaką funkcjonalność powinien posiadać nowy operator. Operator ten powinien testować, czy zbiór reprezentowany przez kolumnę przekazaną jako pierwszy parametr aktualny (czyli de facto zbiór z tabeli) jest nadzbiorem zbioru przekazanego jako drugi parametr aktualny. Jeżeli warunek ten byłby spełniony operator powinien zwracać wartość 1, a w przeciwnym wypadku wartość 0. Należy tutaj zwrócić uwagę na jedną rzecz. Otóż przy powyższych założeniach, jedynie zapytania testujące, czy wartość operatora jest równa 1 są zapytaniami o nadzbiory. Podsumowując powyższe rozważania, funkcja implementująca funkcjonalność operatora w sytuacji, gdy nie jest on wspomagany przez indeks, może wyglądać następująco:

```
CREATE FUNCTION FUNCIMPLSUPERSET (SET_A NUMBER, SET_B SET_TYPE) RETURN NUMBER
AS
BEGIN
RETURN NULL; --Bo i tak to się nigdy nie wykona.
END;
```

Kiedy funkcja jest już utworzona, nowy operator tworzy się za pomocą polecenia CREATE OPERATOR. W poleceniu tym podaje się nazwę operatora, jego parametry, typ zwracanej wartości oraz implementującą go funkcję. Operator, dla którego stworzymy indeks, można utworzyć za pomocą poniższego polecenia.

```
CREATE OPERATOR SUPERSET BINDING (NUMBER, SET_TYPE) RETURN NUMBER
USING FUNCIMPLSUPERSET;
```

Zakładając, że indeks został założony na kolumnie SET_NUMBER tabeli MINED_DATA (choć faktycznie indeksujemy kolumnę SET_CONTENTS), zapytanie o nadzbiory, wykorzystujące nowo utworzony operator może wyglądać następująco:

```
SELECT * FROM MINED_DATA
WHERE SUPERSET (SET_NUMBER, SET_TYPE ('Pivo', 'Apap'))=1;
```

4.2.2. Utworzenie typu obiektowego

Kolejnym krokiem przy tworzeniu indeksu jest stworzenie typu obiektowego implementującego metody interfejsu ODCIINDEX. W skład tego interfejsu wchodzi metody odpowiedzialne za wykonywanie różnych operacji dotyczących indeksu. Minimalny zestaw metod, które należy zaimplementować, składa się z:

- ODCIGETINTERFACES – zwraca listę implementowanych interfejsów i ich wersję (dla Oracle 9i i 10g powinno to być SYS.ODCIINDEX2),
- ODCIINDEXCREATE – metoda wykonywana, kiedy budowany jest nowy indeks (polecenie CREATE INDEX),

- ODCIINDEXDROP – metoda wykonywana, kiedy usuwany jest indeks (polecenie DROP INDEX),
- ODCIINDEXSTART – metoda wykonywana w momencie rozpoczęcia wykonywania zapytania,
- ODCIINDEXFETCH – metoda wykonywana w trakcie wykonywania zapytania, zwraca zadaną liczbę ROWID krotek spełniających warunek selekcji,
- ODCIINDEXCLOSE – metoda wykonywana po zakończeniu realizacji zapytania.

Powyższy zestaw metod wystarczy do zaimplementowania indeksu, który można utworzyć i usunąć, oraz wykorzystać w celu realizacji jakiegoś zapytania. Indeks ten nie będzie jednak mógł być automatycznie modyfikowany w celu odzwierciedlenia zmian zachodzących w indeksowanej tabeli. Aby to było możliwe, należy zaimplementować kolejne trzy metody interfejsu ODCIINDEX:

- ODCIINDEXINSERT – metoda wstawiająca nową wartość do indeksu,
- ODCIINDEXDELETE – metoda usuwająca wartość z indeksu,
- ODCIINDEXUPDATE – metoda aktualizująca wartość zapisaną w indeksie.

Pozostałe metody interfejsu ODCIINDEX dotyczą metadanych indeksu, oraz obsługi poleceń ALTER INDEX i TRUNCATE TABLE. Metody te, jak również wyżej wymienione trzy metody służące do aktualizacji indeksu, nie zostaną opisane bliżej w tym artykule z powodu braku miejsca.

Typ obiektowy składa się z dwóch części: z definicji typu, w której znajduje się deklaracja atrybutów i metod typu, oraz z ciała typu, w którym umieszcza się implementacje metod zadeklarowanych w definicji typu. Poniżej przedstawiono definicję typu implementującego metody interfejsu ODCIINDEX, który zostanie następnie wykorzystany do stworzenia nowego typu indeksu. Znaczenie parametrów poszczególnych metod, oraz atrybutów typu zostaną omówione przy okazji omawiania ciała typu.

```
CREATE TYPE SETINDEXMETHODS AS OBJECT (
  querycursor INTEGER,
  STATIC FUNCTION ODCIGETINTERFACES(ifclist OUT SYS.ODCIOBJECTLIST)
  RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXCREATE(ia sys.ODCIINDEXINFO, parms VARCHAR2,
  env SYS.ODCIENV) RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXDROP(ia sys.ODCIINDEXINFO, env sys.ODCIENV)
  RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXSTART(sctx IN OUT SETINDEXMETHODS,
  ia SYS.ODCIINDEXINFO, op SYS.ODCIPREDINFO, qi SYS.ODCIQUERYINFO,
  strt NUMBER, stop NUMBER, set_b SET_TYPE, env SYS.ODCIENV) RETURN NUMBER,
  MEMBER FUNCTION ODCIINDEXFETCH(nrows NUMBER, rids OUT SYS.ODCIRIDLIST,
  env SYS.ODCIENV) RETURN NUMBER,
  MEMBER FUNCTION ODCIINDEXCLOSE(env SYS.ODCIENV) RETURN NUMBER
);
```

Ciało typu tworzone jest za pomocą polecenia CREATE TYPE BODY. „Szkielet” polecenia tworzącego ciało typu SETINDEXMETHODS, przedstawiono poniżej. Implementacje poszczególnych metod zostaną omówione w kolejnych podrozdziałach tego rozdziału.

```
CREATE TYPE BODY SETINDEXMETHODS IS
  ...--Implementacje metod
END;
```

4.2.3. Metoda ODCIGETINTERFACES

Metoda ODCIGETINTERFACES zwraca listę interfejsów implementowanych przez typ obiektowy i powinna być implementowana przez każdy typ, który implementuje interfejsy *Data Cartridges*. W przypadku typu implementującego metody indeksu, metoda ta powinna zwracać, w zależności od wersji Oracle'a, SYS.ODCIINDEX (Oracle 8i) lub SYS.ODCIINDEX2 (Oracle 9i i późniejsze). Przykładowa implementacja tej metody wygląda następująco:

```

STATIC FUNCTION ODCIGETINTERFACES (ifclist OUT SYS.ODCIOBJECTIST)
RETURN NUMBER IS
BEGIN
    ifclist:=SYS.ODCIOBJECTLIST (SYS.ODCIOBJECT ('SYS', 'ODCIINDEX2'));
    RETURN ODCICONST.SUCCESS;
END;

```

Jedynym parametrem tej metody jest parametr wyjściowy ifclist typu SYS.ODCIOBJECTLIST, który reprezentuje kolekcję obiektów typu SYS.ODCIOBJECT. Obiekty tego typu służą do opisywania obiektów bazy danych. Posiadają one dwa atrybuty OBJECT-SCHEMA i OBJECTNAME, które przechowują odpowiednio: nazwę schematu obiektu i nazwę obiektu. Wstawiany do wynikowej kolekcji obiekt nie odpowiada jednak rzeczywistemu obiektowi znajdującemu się w bazie danych i przechowuje jedynie nazwę i wersję interfejsu (tutaj ODCIINDEX2). Funkcja zwraca kolekcję z nazwami implementowanych przez typ interfejsów poprzez wyjściowy parametr ifclist. Prócz tego metoda, powinna zwrócić wartość liczbową opisaną przez stałą ODCICONST.SUCCESS oznaczającą, że wykonanie metody zakończyło się sukcesem.

4.2.4. Metoda ODCIINDEXCREATE

Metoda ODCIINDEXCREATE jest wywoływana, kiedy użytkownik wykona polecenie CREATE INDEX, zakładające nowy indeks. Metoda ta powinna utworzyć wszystkie struktury danych składające się na indeks i odpowiednio wypełnić je, w zależności od danych, które znajdują się już w indeksowanej tabeli. Metoda ODCIINDEXCREATE ma zawsze 3 parametry. Pierwszy parametr, typu SYS.ODCIINDEXINFO, przekazuje do metody informacje o schemacie, w którym tworzony jest indeks, nazwie indeksu, kolumnach tabeli na których tworzony jest indeks oraz dodatkowe parametry dotyczące m. in. partycjonowania indeksu. Kolejny parametr jest typu VARCHAR2 i przekazuje w postaci łańcucha dodatkowe parametry podane przez użytkownika w poleceniu CREATE INDEX. Tutaj, powinna to być nazwa kolumny z indeksowanymi zbiorami. Ostatnim parametrem metody jest parametr typu SYS.ODCIENV, który wykorzystywany jest m. in. przy tworzeniu indeksów partycjonowanych. W naszym prostym przykładzie indeksu, nie będzie on wykorzystywany. Dokładne znaczenie wymienionych typów, oraz opis ich atrybutów, można znaleźć w [DCDG]. Poniżej przedstawiono implementację metody ODCIINDEXCREATE tworzącą omawiany w niniejszym artykule indeks.

```

STATIC FUNCTION ODCIINDEXCREATE (ia SYS.ODCIINDEXINFO,
    parms VARCHAR2, env SYS.ODCIENV) RETURN NUMBER IS
    stmt VARCHAR2 (32767);
BEGIN
    stmt:='CREATE TABLE '||ia.IndexSchema||'.SETIDXTAB1$'||ia.IndexName||
        ' (SET_NUMBER, SET_ELEMENT, PRIMARY KEY (SET_ELEMENT, SET_NUMBER))'||
        ' ORGANIZATION INDEX COMPRESS 1 AS'||
        ' SELECT Y.ROWID AS SET_NUMBER, VALUE(X).ATTRIBUTE_NAME AS SET_ELEMENT'||
        ' FROM '||ia.indexcols(1).tableschema||'.'||ia.indexcols(1).tablename||
        ' Y CROSS JOIN TABLE(Y.'||parms||') X';
    EXECUTE IMMEDIATE stmt;
    stmt:='ANALYZE TABLE '||ia.IndexSchema||'.SETIDXTAB1$'||ia.IndexName||
        ' COMPUTE STATISTICS';
    EXECUTE IMMEDIATE stmt;

```

```
RETURN ODCICONST.SUCCESS;
END;
```

Przedstawiona metoda ODCIINDEXCREATE tworzy tabelę zorganizowaną jako indeks omawianą w rozdziale 3.2, a następnie zbiera dla niej statystyki. Tabela ta tworzona jest zawsze według następującego wzoru:

```
CREATE TABLE NAZWA_TABELI-INDEKSU (
  SET_NUMBER, SET_ELEMENT, PRIMARY KEY (SET_ELEMENT, SET_NUMBER))
ORGANIZATION INDEX COMPRESS 1 AS
SELECT Y.ROWID AS SET_NUMBER, VALUE(X).ATTRIBUTE_NAME AS SET_ELEMENT
FROM NAZWA_INDEKSOWANEJ_TABELI Y CROSS JOIN TABLE(Y.NAZWA_KOLUMNY) X;
```

gdzie:

- NAZWA_TABELI-INDEKSU, to nazwa tabeli utworzona przez konkatencję łańcucha „SETIDXTAB1\$” i nazwy tworzonego indeksu. Dodatkowo, do tej nazwy doklepany jest prefiks mówiący o tym, iż tabela ma zostać utworzona w schemacie, w którym ma znaleźć się indeks. Dane potrzebne do utworzenia tych nazw są wyciągane z parametru ia typu SYS.ODCIINDEXINFO, a konkretnie, z atrybutów tego typu o nazwach INDEXSCHEMA i INDEXNAME, oznaczających odpowiednio: nazwę schematu, w którym ma zostać utworzony indeks i nazwę nowego indeksu.
- NAZWA_INDEKSOWANEJ_TABELI, to nazwa tabeli na której zakładany jest indeks. Nazwę tą można uzyskać z parametru ia typu SYS.ODCIINDEXINFO. Jednym z atrybutów tego typu jest kolekcja INDEXCOLS obiektów typu SYS.ODCICOLINFO, które opisują indeksowane kolumny. Typ SYS.ODCICOLINFO posiada m. in. takie atrybuty jak TABLESCHEMA, TABLENAME i COLNAME, które oznaczają odpowiednio: schemat indeksowanej tabeli, nazwę indeksowanej tabeli i nazwę indeksowanej kolumny. W tym przypadku wykorzystano jedynie dwa pierwsze z tych atrybutów.
- NAZWA_KOLUMNY, to nazwa indeksowanej kolumny. Normalnie, aby uzyskać nazwę indeksowanej kolumny wykorzystuje się dane zapisane w kolekcji INDEXCOLS (patrz poprzedni punkt). Ponieważ jednak stosujemy zabieg, w którym indeksowaniu podlega inna kolumna niż ta, na której zakładany jest indeks, jako nazwę kolumny przyjmujemy nazwę podaną przez użytkownika jako parametr indeksu.

Powyższe polecenie tworzące tabelę implementującą plik odwrócony, można podzielić logicznie na dwie części. Pierwsza część tworzy tabelę zorganizowaną jako indeks, o dwóch kolumnach: SET_ELEMENT i SET_NUMBER, w której kluczem głównym jest konkatencja kolumn (SET_ELEMENT, SET_NUMBER). Jest to zgodne z koncepcją fizycznej struktury indeksu opisanej w rozdziale 3.2. Wyjaśnienia wymaga jedynie klauzula COMPRESS 1. Użycie jej wynika z faktu, iż przedstawiamy listy wystąpień jako zbiory wierszy składających się z kolumny określającej, którego elementu dotyczy wpis z listy wystąpień, oraz z kolumny określającej faktyczną wartość z listy wystąpień. Taka organizacja powoduje znaczne powiększenie rozmiaru indeksu, poprzez wielokrotne powielenie identyfikatora elementu zbioru. Klauzula COMPRESS powoduje zmniejszenie indeksu poprzez usunięcie powtarzających się wartości prefiksów klucza indeksu o zadanej długości. Uzyskana w ten sposób struktura bardzo przypomina strukturę logiczną opisaną w rozdziale 3.1. Drugą część polecenia tworzy zapytanie wypełniające utworzoną strukturę indeksową odpowiednimi danymi. Zapytanie to uzyskuje odpowiedni zbiór danych poprzez połączenie indeksowanej tabeli z zagnieżdżonymi w każdym wierszu kolekcjami (zbiorami) obiektów i odczytanie z tych obiektów nazw elementów (atrybut ATTRIBUTE_NAME). Jak łatwo zauważyć, dzięki temu, że obiekty typów DM_PREDICATE i DM_NESTED_NUMERICAL posiadają atrybut ATTRIBUTE_NAME, zapytanie wykorzystane w tym poleceniu wypełni poprawnie indeks niezależnie od tego, czy indeksowana jest tabela z danymi o sprzedaży, czy też tabela z wy-

nikami eksploracji danych. Jako numer zbioru zapytanie zwraca ROWID krotki, w której zapisano indeksowany zbiór.

Po utworzeniu struktury indeksu i wypełnieniu jej danymi, metoda zbiera statystyki tej tabeli za pomocą polecenia `ANALYZE TABLE... COMPUTE STATISTICS`. Ostatecznie, po wykonaniu wszystkich operacji, metoda zwraca wartość `ODCICONST.SUCCESS` oznaczającą, że wszystkie operacje wykonano poprawnie.

4.2.5. Metoda `ODCIINDEXDROP`

Metoda `ODCIINDEXDROP` wykonywana jest w sytuacji, gdy użytkownik wyda polecenie `DROP INDEX`. Zadaniem tej metody jest usunięcie wszystkich struktur utworzonych wcześniej za pomocą metody `ODCIINDEXCREATE`. Metoda posiada zawsze dwa parametry, odpowiednio typów `SYS.ODCIINDEXINFO` i `SYS.ODCIENV`. Znaczenie tych parametrów jest identyczne, jak w przypadku metody `ODCIINDEXCREATE`, przy czym parametr typu `SYS.ODCIINDEXINFO` opisuje teraz indeks, który należy usunąć, a nie, jak poprzednio, utworzyć. Poniżej przedstawiono implementację metody `ODCIINDEXDROP` usuwającą omawiany w niniejszym artykule indeks.

```
STATIC FUNCTION ODCIINDEXDROP(ia SYS.ODCIINDEXINFO, env SYS.ODCIENV)
RETURN NUMBER IS
  stmt VARCHAR2(32767);
BEGIN
  stmt:='DROP TABLE '||ia.IndexSchema||'.SETIDXTAB1$'||ia.IndexName;
  EXECUTE IMMEDIATE stmt;
  RETURN ODCICONST.SUCCESS;
END;
```

Przedstawiona metoda `ODCIINDEXDROP` usuwa tabelę zorganizowaną jako indeks utworzoną przez metodę `ODCIINDEXCREATE`. Robi to wykonując polecenie SQL skonstruowane według wzoru:

```
DROP TABLE NAZWA_TABELI-INDEKSU;
```

gdzie:

- `NAZWA_TABELI-INDEKSU`, to nazwa tabeli utworzonej przez metodę `ODCIINDEXCREATE`. Nazwa ta, podobnie jak poprzednio, konstruowana jest za pomocą danych zapisanych w parametrze `ia` typu `SYS.ODCIINDEXINFO`.

Po usunięciu tabeli, metoda zwraca wartość `ODCICONST.SUCCESS` oznaczającą, że wszystkie operacje wykonano poprawnie.

4.2.6. Metoda `ODCIINDEXSTART`

Metoda `ODCIINDEXSTART` wykonywana jest zawsze, kiedy użytkownik wykona zapytanie z użyciem operatora wspomaganego przez indeks. Zadaniem tej metody jest zainicjowanie realizacji zapytania, i zwrócenie instancji typu implementującego interfejs `ODCIINDEX` jako kontekstu wykonywanego zapytania. Zwrócona instancja powinna posiadać atrybuty, w których powinny być zapisane wszystkie dane konieczne do wykonywania zapytania i do późniejszego zwolnienia zajętych przez metodę `ODCIINDEXSTART` zasobów. Liczba i typy parametrów metody `ODCIINDEXSTART` zależą od parametrów i typu zwracanej wartości przez wspierany operator. W sytuacji, gdy indeks wspiera więcej niż jeden operator, może być konieczne utworzenie więcej niż jednej metody `ODCIINDEXSTART`. Pierwszym parametrem metody jest parametr typu obiektowego implementującego interfejs `ODCIINDEX`, w którym metoda ta jest zaimplementowana. Jest to parametr, poprzez który zostanie zwrócony kontekst wykonywania zapytania opisany wcześniej. Kolejnym parametrem, jest parametr znany z poprzednio omawianych metod, jest to parametr typu `ODCIINDEXINFO` opisujący wykorzystywany indeks. Kolejne dwa parametry

odpowiednio typów: SYS.ODCIPREDINFO i SYS.ODCIQUERYINFO, przekazują informacje o nazwie i sposobie wykorzystania operatora w zapytaniu (SYS.ODCIPREDINFO), oraz o wskaźnikach dla optymalizatora i pomocniczych operatorach użytych w zapytaniu (SYS.ODCIQUERYINFO). Typ kolejnych dwóch parametrów zależy od typu zwracanej przez wspierany operator wartości. Parametry te oznaczają kolejno najmniejszą i największą wartość jaką może przyjąć operator (określane w zapytaniu za pomocą operatorów <, >, <=, >=, itp.). Kolejne parametry metody ODCIINDEXSTART powinny być takie same, jak drugi, trzeci i kolejne parametry wspieranego operatora (pierwszy parametr pochodzi z indeksowanej kolumny, a zatem kolejne wartości muszą być odczytywane przez indeks). W tym przypadku mamy do czynienia tylko z jednym parametrem, który stanowi zbiór zapytania. Ostatnim parametrem jest parametr typu SYS.ODCIENV, który jest wymagany przez interfejs ODCIINDEX, ale w obecnej wersji Oracle nie jest wykorzystywany. Poniżej przedstawiono implementację metody ODCIINDEXSTART rozpoczynającą realizację zapytań z wykorzystaniem omawianego w niniejszym artykule indeksu.

```

STATIC FUNCTION ODCIINDEXSTART(sctx IN OUT SETINDEXMETHODS,
  ia SYS.ODCIINDEXINFO, op SYS.ODCIPREDINFO, qi SYS.ODCIQUERYINFO,
  strt NUMBER, stop NUMBER, set_b SET_TYPE, env SYS.ODCIENV) RETURN NUMBER IS
  stmt VARCHAR2(32767);
  tempcursor INTEGER;
  rid ROWID;
  i NUMBER;
  num NUMBER;
BEGIN
  --Część 1 Sprawdzenie poprawności wartości operatora
  IF NOT (strt=1 AND stop=1 AND BITAND(1,op.flags)=1) THEN
    RAISE_APPLICATION_ERROR(-20202, 'Bledna wartosc operatora.');
```

END IF;

```

  --Część 2 Konstrukcja zapytania
  stmt:='';
  i:=set_b.first;
  WHILE (i IS NOT NULL) LOOP
    stmt:=stmt||'SELECT SET_NUMBER FROM'||
      ' ia.IndexSchema||'.SETIDXTAB1$'||ia.IndexName||
      ' WHERE SET_ELEMENT=''||set_b(i)||''';
    i:=set_b.next(i);
    IF (i IS NOT NULL) THEN
      stmt:=stmt||' INTERSECT ';
```

END IF;

```

END LOOP;
  --Część 3 Utworzenie kursora dla skonstruowanego wcześniej zapytania
  tempcursor:=dbms_sql.open_cursor;
  dbms_sql.parse(tempcursor, stmt, dbms_sql.native);
  dbms_sql.define_column_rowid(tempcursor, 1, rid);
  num:=dbms_sql.execute(tempcursor);
  --Część 4 Utworzenie kontekstu i zakończenie metody
  sctx:=SETINDEXMETHODS(tempcursor);
  RETURN ODCICONST.SUCCESS;
END;
```

Funkcjonalność przedstawionej metody można podzielić na 4 części. Część 1 sprawdza, czy operator został w zapytaniu użyty zgodnie z przeznaczeniem. Zgodnie z tym co pisano wcześniej, operator ten powinien być wykorzystywany tylko w sytuacji, gdy porównywany jest z wartością jeden. Dlatego też sprawdzamy, czy najmniejsza i największa wartość jaką może przyjąć operator wynosi jeden, oraz, czy najmłodszy bit atrybutu FLAGS parametru metody ODCIINDEXSTART

typu SYS.ODCIPREDINFO jest ustawiony na jeden. Ten ostatni warunek oznacza, że wykorzystano w klauzuli WHERE zapytania operator =. Znaczenie poszczególnych bitów tego atrybutu można znaleźć w [DCDG]. Jeżeli którykolwiek z wyżej wymienionych warunków nie jest spełniony, zgłaszany jest wyjątek.

Druga część metody konstruuje zapytanie do tabeli indeksu, które zwraca ROWID krotek indeksowanej tabeli, które zawierają zbiór, stanowiący nadzbiór zbioru zapytania. Niech będzie dany zbiór zapytania $Q=\{q_1, q_2, \dots, q_n\}$. Zapytanie do indeksu jest zawsze konstruowane według następującego wzoru:

```
SELECT SET_NUMBER FROM NAZWA_TABELI-INDEKSU WHERE SET_ELEMENT=q1
INTERSECT
SELECT SET_NUMBER FROM NAZWA_TABELI-INDEKSU WHERE SET_ELEMENT=q2
INTERSECT
.....
INTERSECT
SELECT SET_NUMBER FROM NAZWA_TABELI-INDEKSU WHERE SET_ELEMENT=qn
```

gdzie:

- NAZWA_TABELI-INDEKSU, to nazwa tabeli utworzonej przez metodę ODCIINDEXCREATE. Nazwa ta, podobnie jak poprzednio, konstruowana jest za pomocą danych zapisanych w parametrze ia typu SYS.IDCIINDEXINFO.
- q_1, q_2, \dots, q_n , to elementy zbioru zapytania.

Jak łatwo zauważyć, przedstawione powyżej zapytanie zbudowane jest z zapytań które odczytują listy wystąpień dla kolejnych elementów zbioru zapytania. Zapytania te połączone są za pomocą operatora INTERSECT, dzięki któremu znajdowana jest część wspólna tych list.

Trzecia część metody otwiera nowy kursor, parsuje zbudowane w poprzedniej części metody zapytanie, definiuje typ zwracanych przez zapytanie wartości i, ostatecznie, wykonuje to zapytanie. tak przygotowany kursor można następnie wykorzystać do odczytania kolejnych ROWID krotek stanowiących wynik zapytania. Aby to było możliwe, konieczne jest przekazanie uchwytu na ten kursor to metody ODCIINDEXFETCH (opisanej dokładnie w następnym podrozdziale), której zadaniem jest przekazanie wyników operacji na indeksie do systemu zarządzania bazą danych. Aby przekazać ten uchwyt należy utworzyć kontekst wykonania zapytania. Kontekstem jest tutaj instancja typu implementującego metody indeksu, którego atrybuty mogą przechowywać (między innymi) uchwyt na kursor z wynikiem zapytania. Dlatego też typ SETINDEXMETHODS posiada atrybut quercursor typu NUMBER, którego zadaniem jest przechowywanie uchwytu kursora. Część 4 metody tworzy kontekst i zwraca go poprzez parametr sctx typu SETINDEXMETHODS, a następnie kończy wykonanie metody zwracając wartość ODCI.CONST.SUCCESS oznaczającą, że wszystkie operacje wykonano poprawnie.

4.2.7. Metoda ODCIINDEXFETCH

Metoda ODCIINDEXFETCH jest wykonywana wielokrotnie w trakcie realizacji zapytania. Zadaniem tej metody jest zwrócenie „paczki” ROWID krotek stanowiących wynik zapytania o określonym przez system zarządzania bazą danych rozmiarze. Metoda jest wywoływana tak długo, jak zwracane są nowe wyniki. Metoda ODCIINDEXFETCH posiada trzy parametry. Pierwszy z nich jest typu liczbowego i określa maksymalną liczbę ROWID, jakie powinny być zwrócone przez tą metodę. Kolejny parametr jest parametrem wyjściowym typu SYS.ODCIRIDLIST. Typ ten reprezentuje kolekcję ROWID. Ostatnim parametrem jest parametr typu ODCIENV, który nie jest przez tą metodę wykorzystywany. Poniżej przedstawiono implementację metody ODCIINDEXFETCH, którą można wykorzystać do pobrania wyników zapytania z indeksu omawianego w niniejszym artykule.

```

MEMBER FUNCTION ODCIINDEXFETCH(nrows NUMBER, rids OUT SYS.ODCIRIDLIST,
env SYS.ODCIENV) RETURN NUMBER IS
    tempcursor INTEGER;
    done BOOLEAN:=FALSE;
    idx INTEGER:=1;
    rlist SYS.ODCIRIDLIST := SYS.ODCIRIDLIST();
    row_ ROWID;
BEGIN
    tempcursor:=SELF.querycursor;
    WHILE NOT done LOOP
        IF idx > nrows THEN
            done := TRUE;
        ELSE
            rlist.EXTEND;
            IF dbms_sql.fetch_rows(tempcursor) > 0 THEN
                dbms_sql.column_value_rowid(tempcursor, 1, rlist(idx));
                idx := idx + 1;
            ELSE
                rlist(idx) := NULL;
                done := TRUE;
            END IF;
        END IF;
    END LOOP;
    rids:=rlist;
    RETURN ODCICONST.SUCCESS;
END;

```

W przeciwieństwie do poprzednio omawianych metod, metoda ODCIINDEXFETCH nie jest metodą statyczną. Jest ona aktywowana na rzecz obiektu utworzonego jako kontekst zapytania. Dzięki temu ma ona dostęp do wartości atrybutu querycursor przechowującego uchwyt na kursor, dzięki któremu można odczytywać kolejne wartości ROWID. Metoda zawiera prostą pętlę, która odczytuje ROWID z kursora i dodaje do wynikowej kolekcji tak długo, aż nie zostanie przekroczony limit, albo nie skończą się wyniki w kursorze. Po zakończeniu pętli metoda zwraca przez parametr wyjściowy kolejną paczkę (kolekcję) ROWID, a następnie zwraca wartość ODCICONST.SUCCESS oznaczającą, że wszystkie operacje wykonano poprawnie.

4.2.8. Metoda ODCIINDEXCLOSE

Metoda ODCIINDEXCLOSE jest wywoływana na końcu wykonywania zapytania w celu zwolnienia zasobów (np. kursorów) zarezerwowanych przez metodę ODCIINDEXSTART. Jej jedynym parametrem, jest parametr typu en SYS.ODCIENV, który nie jest jednak wykorzystywany. Poniżej przedstawiono implementację metody ODCIINDEXCLOSE, którą zamyka kursor zapisany w atrybucie querycursor.

```

MEMBER FUNCTION ODCIINDEXCLOSE (env SYS.ODCIENV) RETURN NUMBER IS
    tempcursor INTEGER;
BEGIN
    tempcursor:=SELF.querycursor;
    dbms_sql.close_cursor(tempcursor);
    RETURN ODCICONST.SUCCESS;
END;

```

Podobnie jak metoda ODCIINDEXFETCH, metoda ODCIINDEXCLOSE nie jest metodą statyczną i jest aktywowana na rzecz obiektu utworzonego przez metodę ODCIINDEXSTART jako kontekst zapytania. Dzięki temu metoda ma dostęp do pola querycursor, z którego może odczytać uchwyt kursora, a następnie go zwolnić. Podobnie jak wszystkie inne omawiane powyżej metody,

po zakończeniu wszystkich czynności metoda zwraca wartość ODCICONST.SUCCESS oznaczającą, że wszystkie operacje wykonano poprawnie.

4.2.9. Utworzenie typu indeksu

Ostatnim krokiem implementacji indeksu za pomocą mechanizmu *Data Cartridges* jest utworzenie typu indeksu. Wykonuje się to za pomocą polecenia CREATE INDEXTYPE. W poleceniu tym podaje się nazwę nowego typu indeksu, listę wspomaganych operatorów oraz nazwę typu obiektowego implementującego interfejs ODCIINDEX. W przypadku omawianego w niniejszym artykule indeksu, polecenie CREATE INDEXTYPE wygląda następująco:

```
CREATE INDEXTYPE SETINDEX FOR SUPERSET (NUMBER, SET_TYPE)
USING SETINDEXMETHODS;
```

4.2.10. Wykorzystanie indeksu

Aby utworzyć zaimplementowany już indeks należy użyć polecenia CREATE INDEX z klauzulami INDEXTYPE IS i PARAMETERS. Klauzula INDEXTYPE IS pozwala określić który typ indeksu ma być użyty (patrz podrozdział 4.2.9), klauzula PARAMETERS pozwala na przekazanie do procedury ODCIINDEXCREATE jednego łańcucha jako parametru. Składnia polecenia CREATE INDEX, z wykorzystaniem tych klauzul wygląda następująco:

```
CREATE INDEX ON TABELA(KOLUMNA) INDEXTYPE IS NAZWA_TYPU_INDEKSU
PARAMETERS ('PARAMETRY INDEKSU');
```

Rozważmy przykładową tabelę MINED_DATA z danymi o sprzedaży, która została przedstawiona w podpunkcie 4.1. Aby założyć na kolumnie SET_CONTENTS indeks dla zbiorów, należy najpierw wybrać jedną z pozostałych kolumn indeksowanej tabeli, która będzie „reprezentowała” kolumnę SET_CONTENTS (w razie konieczności można taką kolumnę utworzyć za pomocą polecenia ALTER TABLE). Jak wspomniano wcześniej, takie rozwiązanie jest konieczne, gdyż Oracle nie pozwala założyć indeksu na kolumnach typu kolekcji. Następnie, należy założyć indeks na wybranej kolumnie, a nazwę faktycznej kolumny składującej zbiory należy podać za pomocą klauzuli PARAMETERS. Polecenie zakładające indeks na przykładowej tabeli MINED_DATA wygląda następująco:

```
CREATE INDEX ON MINED_DATA (SET_NUMBER) INDEXTYPE IS SETINDEX
PARAMETERS ('SET_CONTENTS');
```

Zapytania wykorzystujące indeks należy konstruować z użyciem operatora SUPERSET, który został opisany w podrozdziale 4.2.1. Przykładowo, zapytanie odszukujące wszystkie zbiory zawierające elementy „Piwo” i „Apap” wygląda następująco:

```
SELECT * FROM MINED_DATA
WHERE SUPERSET (SET_NUMBER, SET_TYPE ('Piwo', 'Apap'))=1;
```

Aby założyć indeksy na kolumnach ANTECEDENT i CONSEQUENT tabeli stanowiącej wynik materializacji wyniku eksploracji danych, należy postępować w analogiczny sposób jak w przypadku tabeli MINED_DATA, tzn. należy wybrać (lub utworzyć) kolumny reprezentujące faktycznie indeksowane kolumny i założyć na nich indeks, podając jako parametr indeksu nazwę faktycznie indeksowanej kolumny.

5. Podsumowanie

W artykule przedstawiono logiczną i fizyczną strukturę indeksu wspierającego zapytania o nadzbiory. Przedstawiono sposób implementacji tego indeksu w systemie zarządzania bazą danych Oracle i omówiono metody ominięcia pewnych problemów związanych z indeksowaniem kolekcji. Przedstawiona implementacja została przystosowana do indeksowania wyników eksploracji

danych o sprzedaży uzyskiwanych dzięki funkcjonalności opcji Oracle Data Mining, oraz do indeksowania danych oryginalnych.

Artykuł porusza jedynie podstawy implementacji indeksów w systemie zarządzania bazą danych Oracle. Pominęto tematykę związaną z partycjonowaniem indeksu, jego aktualizacją, przetwarzaniem równoległym, zbieraniem statystyk i uwzględnianiu ich w optymalizatorze zapytań. Zainteresowanych dalszą lekturą na ten temat odsyłamy do [DCDG].

Bibliografia

- [AKMM05] Andrzejewski W., Królikowski Z., Morzy M., Morzy T.: Ocena efektywności hierarchicznego indeksu bitmapowego wspierającego wykonywanie zapytań w bazach danych z atrybutami zawierającymi zbiory, *Archiwum Informatyki Teoretycznej i Stosowanej*, 2005, tom 17, zeszyt 4, pp. 273-288
- [BaRi99] Baeza-Yates R., Ribeiro-Neto B.: *Modern Information Retrieval*, Addison Wesley Longman Publishing Co. Inc, 1999
- [BMMR02] Brzeziński J., Morzy T., Morzy M., Rutkowski Ł.: Algorytm optymalizacji zapytań eksploracyjnych z wykorzystaniem materializowanych perspektyw eksploracyjnych, *Raport IIPP*, RB-006/02, 2002
- [DCDG] Oracle Database Data Cartridge Developer's Guide, 10g Release 2 (10.2), http://download-uk.oracle.com/docs/cd/B19306_01/appdev.102/b14289.pdf
- [MKP03] Morzy M., Królikowski Z., Perek J.: Set-Oriented Indexes For Data Mining Queries, *Proceedings of the 5th International Conference on Enterprise Information Systems - ICEIS'2003*, April 2003, Angers, Francja, 2003
- [SQL03] SQL 2003 Standard Support in Oracle Database 10g. An Oracle White Paper, November 2003, http://www.oracle.com/technology/products/database/application_development/pdf/SQL_2003_TWP.pdf