Data Cartridge

Krzysztof Jankiewicz
Instytut Informatyki
Politechniki Poznańskiej
Krzysztof.Jankiewicz@cs.put.poznan.pl
na podstawie pracy magisterskiej pana
Remigiusza Dworczaka

Tworzenie nowych typów indeksów domenowych

- Wymaga zdefiniowania następujących komponentów:
  - Typ indeksu – obiekt w schemacie użytkownika, który specyfikuje metody służące do zarządzania wszystkimi aspektami nowej struktury indeksowej
  - Indeks – używając typu indeksu, uprzednio zdefiniowana struktura indeksowa może zostać utworzona na atrybutach wybranej tabeli. Indeks jest zatem instancją typu indeksu
  - Zbiór operatorów – zawiera operatory zdefiniowane przez programistę, które mogą być użyte w zapytaniach SQL, a wartości wyrażeń zawierających je mogą zostać obliczone zarówno dzięki powiązaniu ich ze zdefiniowanymi dla nich funkcjami, jak i przez samą strukturę obiektową
  - Tabela o organizacji indeksu – służąca do składowania danych indeksu. Komponent ten jest opcjonalny, gdyż dane indeksu mogą być składowane w systemie plików poza systemem zarządzania bazą danych

Interfejs ODCIIndex

- Aby utworzyć w schemacie użytkownika nowy typ indeksowy, należy wcześniej zadeklarować i zaimplementować typ obiektowy, który zawiera metody pokrywające interfejs ODCIIndex
- Zbiór metod składających się na interfejs ODCIIndex można podzielić na następujące kategorie
  - Metody definiujące indeks
  - Metody służące do zarządzania strukturą indeksu
  - Metody służące do przeszukiwania struktury indeksu
- Oprócz tych metod implementacja typu obiektowego powinna zawierać jeszcze metodę ODCIGetInterfaces, która ustala listę nazw interfejsów implementowanych przez ten typ.

Metody definiujące indeks

- ODCIIndexCreate
  - Metoda wywoływana podczas wywoływania polecenia CREATE INDEX
  - Typowa akcja: utworzenie tabel lub plików do przechowywania danych indeksu. Kiedy tabela, na której założony został indeks nie jest pusta, metoda ta powinna zbudować indeks na istniejących danych.
- ODCIIndexDrop
  - Metoda wywoływana podczas wykonywania polecenia DROP INDEX
  - Typowa akcja: usunięcie tabeli bądź plików z danymi indeksu
- ODCIIndexAlter
  - Metoda wywoływana podczas polecenia ALTER INDEX
- ODCIIndexTruncate
  - Metoda wywoływana podczas wydfania polecenia TRUNCATE
  - Typowa akcja: obcięcie tabeli lub wyzerowanie wielkości pliku

Metody służące do zarządzania strukturą indeksu

- ODCIIndexInsert
  - Metoda wywoływana podczas wykonywania polecenia INSERT INTO
  - Typowa akcja: wstawienie nowych danych do struktury indeksu przechowywanej w pliku bądź tabeli
- ODCIIndexUpdate
  - Metoda wywoływana podczas wykonywania polecenia UPDATE
- ODCIndexDelete
  - Metoda wywoływana podczas wykonywania polecenia DELETE
  - Typowa akcja: usunięcie danych indeksu z pliku bądź tabeli odnośnie usuwanego wiersza

Metody służące do przeszukiwania struktury indeksu

- ODCIIndexStart
  - Zadaniem tej metody jest inicjalizacja przeszukiwania struktury indeksu.
  - Typowa akcja: parsowanie przekazanego jako parametr predykatu, przygotowanie oraz wykonanie zapytania
- ODCIIndexFetch
  - Zadanie tej metody sprowadza się do zapisania w zmiennych programu zwróconych przez zapytanie identyfikatorów krotek (iteracja po zbiorze wyników)
- ODCIIndexClose
  - Zadanie tej metody sprowadza się jedynie do zwolnienia pamięci zaalokowanej podczas wykonywania ODCIndexStart oraz zajmowanej przez niektóre struktury OCI pamięci

Przykład – Cel

```
SELECT last_name, salary FROM
   (SELECT last_name,
           DENSE_RANK() OVER
             (ORDER BY salary DESC) rank_val,
           salary
    FROM employees)
   WHERE rank_val BETWEEN 10 AND 20;
```

```
LAST_NAME                 SALARY
------------------------- ----------
Tucker                     10000
Baer                       10000
Bloom                      10000
King                       10000
Fox                         9600
Bernstein                   9500
Sully                       9500
Greene                      9500
Hunold                      9000
Hall                        9000
Faviet                      9000
McEwen                      9000
Hutton                      8800
Taylor                      8600
Livingston                  8400
Gietz                       8400
Chen                        8200
Fripp                       8200
Weiss                       8000
Smith                       8000
Olsen                       8000
Kaufling                    7900
Urman                       7800

23 rows selected.
```



```
CREATE OR REPLACE TYPE position_im AS OBJECT
(
  curnum  NUMBER,
  howmany NUMBER,
  lower_bound NUMBER,  --lower_bound and upper_bound are used for the
  upper_bound NUMBER,  --index-based functional implementation.
  STATIC FUNCTION ODCIGETINTERFACES(ifclist OUT SYS.ODCIOBJECTLIST) RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXCREATE
    (ia SYS.ODCIINDEXINFO, parms VARCHAR2, env SYS.ODCIEnv) RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXTRUNCATE (ia SYS.ODCIINDEXINFO,
                                     env SYS.ODCIEnv) RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXDROP(ia SYS.ODCIINDEXINFO,
                                env SYS.ODCIEnv) RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXINSERT(ia SYS.ODCIINDEXINFO, rid ROWID,
                                  newval NUMBER, env SYS.ODCIEnv) RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXDELETE(ia SYS.ODCIINDEXINFO, rid ROWID, oldval NUMBER,
                                  env SYS.ODCIEnv) RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXUPDATE(ia SYS.ODCIINDEXINFO, rid ROWID, oldval NUMBER,
                                  newval NUMBER, env SYS.ODCIEnv) RETURN NUMBER,
  STATIC FUNCTION ODCIINDEXSTART(SCTX IN OUT position_im, ia SYS.ODCIINDEXINFO,
                                 op SYS.ODCIPREDINFO, qi SYS.ODCIQUERYINFO,
                                 strt NUMBER, stop NUMBER, lower_pos NUMBER,
                                 upper_pos NUMBER, env SYS.ODCIEnv) RETURN NUMBER,
  MEMBER FUNCTION ODCIINDEXFETCH(SELF IN OUT position_im, nrows NUMBER,
                                 rids OUT SYS.ODCIRIDLIST, env SYS.ODCIEnv)
                                 RETURN NUMBER,
  MEMBER FUNCTION ODCIINDEXCLOSE(env SYS.ODCIEnv) RETURN NUMBER
);
/
```

Przykład – ODCIGetInterfaces

```
CREATE OR REPLACE TYPE BODY position_im IS
   STATIC FUNCTION ODCIGETINTERFACES (ifclist OUT SYS.ODCIOBJECTLIST)
       RETURN NUMBER IS
   BEGIN
     ifclist := SYS.ODCIOBJECTLIST(SYS.ODCIOBJECT('SYS','ODCIINDEX2'));
     RETURN ODCICONST.SUCCESS;
   END ODCIGETINTERFACES;

. . .
```


Przykład – ODCIIndexCreate

```
. . .
  STATIC FUNCTION ODCIINDEXCREATE (ia SYS.ODCIINDEXINFO, parms VARCHAR2,
                                   env SYS.ODCIEnv) RETURN NUMBER IS
   stmt    VARCHAR2(2000);
   BEGIN
     -- konstrukcja polecenia tworzącego strukturę indeksową
     stmt := 'Create Table ' || ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
          '_STORAGE_TAB' || '(col_val, base_rowid, constraint pk PRIMARY KEY ' ||
          '(col_val, base_rowid)) ORGANIZATION INDEX AS SELECT ' ||
          ia.INDEXCOLS(1).COLNAME || ', ROWID FROM ' ||
          ia.INDEXCOLS(1).TABLESCHEMA || '.' || ia.INDEXCOLS(1).TABLENAME;
     EXECUTE IMMEDIATE stmt;
     RETURN ODCICONST.SUCCESS;
   END;
. . .
```

```
Create Table SCOTT.SALARY_INDEX_STORAGE_TAB
(col_val, base_rowid,
 constraint pk PRIMARY KEY (col_val, base_rowid))
ORGANIZATION INDEX AS
SELECT "SALARY", ROWID FROM SCOTT.EMPLOYEES
```

```
SQL> desc sys.ODCICOLINFOLIST
 sys.ODCICOLINFOLIST VARRAY(32) OF SYS.ODCICOLINFO
 Nazwa                Typ
 -------------------- ----------------
 TABLESCHEMA          VARCHAR2(30)
 TABLENAME            VARCHAR2(30)
 COLNAME              VARCHAR2(4000)
 COLTYPENAME          VARCHAR2(30)
 COLTYPESCHEMA        VARCHAR2(30)
 TABLEPARTITION       VARCHAR2(30)
```

```
SQL> desc SYS.ODCIINDEXINFO
 Nazwa                          Typ
 ------------------------------ ----------------
 INDEXSCHEMA                    VARCHAR2(30)
 INDEXNAME                      VARCHAR2(30)
 INDEXCOLS                      ODCICOLINFOLIST
 INDEXPARTITION                 VARCHAR2(30)
 INDEXINFOFLAGS                 NUMBER
 INDEXPARADEGREE                NUMBER
```


Przykład – ODCIIndexDrop, ODCIIndexTruncate

```
. . .
  STATIC FUNCTION ODCIINDEXDROP (ia SYS.ODCIINDEXINFO,
                                 env SYS.ODCIEnv) RETURN NUMBER IS
    stmt VARCHAR2(2000);
  BEGIN
    stmt := 'DROP TABLE ' || ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
            '_STORAGE_TAB';
    EXECUTE IMMEDIATE stmt;
    RETURN ODCICONST.SUCCESS;
  END;

  STATIC FUNCTION ODCIINDEXTRUNCATE (ia SYS.ODCIINDEXINFO,
                                     env SYS.ODCIEnv) RETURN NUMBER IS
    stmt VARCHAR2(2000);
  BEGIN
    stmt := 'TRUNCATE TABLE ' || ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
            '_STORAGE_TAB';
    EXECUTE IMMEDIATE stmt;
    RETURN ODCICONST.SUCCESS;
  END;
. . .
```

```
DROP TABLE SCOTT.SALARY_INDEX_STORAGE_TAB
```


Przykład – ODCIIndexInsert, ODCIndexDelete

```
. . .
  STATIC FUNCTION ODCIINDEXINSERT (ia SYS.ODCIINDEXINFO, rid ROWID,
                                   newval NUMBER, env SYS.ODCIEnv)
  RETURN NUMBER IS
    stmt VARCHAR2(2000);
  BEGIN
    stmt := 'INSERT INTO ' || ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
            '_STORAGE_TAB  VALUES (''' || newval || ''' , ''' || rid || ''' )';
    EXECUTE IMMEDIATE stmt;
    RETURN ODCICONST.SUCCESS;
  END;

  STATIC FUNCTION ODCIINDEXDELETE (ia SYS.ODCIINDEXINFO, rid ROWID,
                                   oldval NUMBER, env SYS.ODCIEnv)
  RETURN NUMBER IS
    stmt VARCHAR2(2000);
  BEGIN
    stmt := 'DELETE FROM ' || ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
            '_STORAGE_TAB WHERE col_val = ''' || oldval || ''' AND ' ||
            'base_rowid = ''' || rid || '''';
    EXECUTE IMMEDIATE stmt;
    RETURN ODCICONST.SUCCESS;
  END;
. . .
```

```
INSERT INTO SCOTT.SALARY_INDEX_STORAGE_TAB
VALUES ('8300' , 'AAAK8PAABAAAOQLAAJ' )
```

```
DELETE FROM SCOTT.SALARY_INDEX_STORAGE_TAB
WHERE col_val = '8300'
AND base_rowid = 'AAAK8PAABAAAOQLAAI'
```

Przykład – ODCIIndexUpdate

```
. . .
  STATIC FUNCTION ODCIINDEXUPDATE (ia SYS.ODCIINDEXINFO, rid ROWID,
                                   oldval NUMBER, newval NUMBER,
                                   env SYS.ODCIEnv) RETURN NUMBER IS
    stmt VARCHAR2(2000);
  BEGIN
    stmt := 'UPDATE ' || ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
            '_STORAGE_TAB SET col_val = ''' || newval || ''' '||
            'WHERE base_rowid = '''|| rid ||'''';
    EXECUTE IMMEDIATE stmt;
    RETURN ODCICONST.SUCCESS;
  END;
. . .
```

```
UPDATE SCOTT.SALARY_INDEX_STORAGE_TAB
SET col_val = '8400'
WHERE base_rowid = 'AAAK8PAABAAAOQLAAJ'
```


Przykład – ODCIIndexStart (1)

```
STATIC FUNCTION ODCIINDEXSTART (SCTX IN OUT position_im, ia SYS.ODCIINDEXINFO,
                                op SYS.ODCIPREDINFO, qi SYS.ODCIQUERYINFO,
                                strt NUMBER, stop NUMBER, lower_pos NUMBER,
                                upper_pos NUMBER, env SYS.ODCIEnv) RETURN NUMBER IS
```

- SCTX – instancja definiowanego typu obiektowego
- op – instancja typu obiektowego SYS.ODCIPREDINFO. Przechowuje on niezbędne informacje na temat predykatu zawierającego uprzednio zdefiniowane operatory i wyspecyfikowanego w zapytaniu SQL. Dotyczą one nazwy schematu w którym został zdefiniowany operator i odpowiadająca mu funkcja, a także nazwy samego operatora i funkcji
- qi - instancja typu obiektowego SYS.ODCIQUERYINFO. Przechowuje on informacje na temat kontekstu zadanego przez użytkownika zapytania.
- strt i stop – przechowują odpowiednio dolne oraz górne ograniczenie wartości wyrażenia wyspecyfikowanego w zapytaniu przy operatorze

```
SQL> desc SYS.ODCIPREDINFO
 Name            Type
 --------------- ------------
 OBJECTSCHEMA    VARCHAR2(30)
 OBJECTNAME      VARCHAR2(30)
 METHODNAME      VARCHAR2(30)
 FLAGS           NUMBER
```

```
SQL> desc SYS.ODCIQUERYINFO
 Name       Type
 ---------- --------------
 FLAGS      NUMBER
 ANCOPS     ODCIOBJECTLIST
```


Przykład – ODCIIndexStart (2)

```
. . .
 STATIC FUNCTION ODCIINDEXSTART (SCTX IN OUT position_im, ia SYS.ODCIINDEXINFO,
                                 op SYS.ODCIPREDINFO, qi SYS.ODCIQUERYINFO,
                                 strt NUMBER, stop NUMBER, lower_pos NUMBER,
                                 upper_pos NUMBER, env SYS.ODCIEnv) RETURN NUMBER IS


  rid              VARCHAR2(5072);
  storage_tab_name VARCHAR2(65);
  lower_bound_stmt VARCHAR2(2000);
  upper_bound_stmt VARCHAR2(2000);
  range_query_stmt VARCHAR2(2000);
  lower_bound      NUMBER;
  upper_bound      NUMBER;
  cnum             INTEGER;
  nrows            INTEGER;

 BEGIN

   -- tu może pojawić się "Wykrycie nieprawidłowego użycia"

   storage_tab_name := ia.INDEXSCHEMA || '.' || ia.INDEXNAME ||
                       '_STORAGE_TAB';
   upper_bound_stmt := 'Select MIN(col_val) FROM (Select /*+ INDEX_DESC(' ||
                       storage_tab_name || ') */ DISTINCT ' ||
                       'col_val FROM ' || storage_tab_name || ' ORDER BY ' ||
                       'col_val DESC) WHERE rownum <= ' || lower_pos;
   EXECUTE IMMEDIATE upper_bound_stmt INTO upper_bound;
. . .
```

```
SELECT last_name, salary FROM employees
   WHERE position_between(salary, 10, 20)=1
   ORDER BY salary DESC;
```

```
Select MIN(col_val)
FROM (Select /*+ INDEX_DESC(SCOTT.SALARY_INDEX_STORAGE_TAB)*/
      DISTINCT col_val
      FROM SCOTT.SALARY_INDEX_STORAGE_TAB
      ORDER BY col_val DESC)
WHERE rownum <= 10;
```

```
MIN(COL_VAL)
------------
       10000
```


Przykład – ODCIIndexStart (3)

```
. . .
   IF (lower_pos != upper_pos) THEN
     lower_bound_stmt := 'Select MIN(col_val) FROM (Select /*+ INDEX_DESC(' ||
                         storage_tab_name || ') */ DISTINCT ' ||
                         'col_val FROM ' || storage_tab_name ||
                         ' WHERE col_val < ' || upper_bound || ' ORDER BY ' ||
                         'col_val DESC) WHERE rownum <= ' ||
                         (upper_pos - lower_pos);
     EXECUTE IMMEDIATE lower_bound_stmt INTO lower_bound;
   ELSE
     lower_bound := upper_bound;
   END IF;

   IF (lower_bound IS NULL) THEN
     lower_bound := upper_bound;
   END IF;

   range_query_stmt := 'Select base_rowid FROM ' || storage_tab_name ||
                       ' WHERE col_val BETWEEN ' || lower_bound || ' AND ' ||
                       upper_bound;

   cnum := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(cnum, range_query_stmt, DBMS_SQL.NATIVE);

   -- set context as the cursor number
   SCTX := position_im(cnum, 0, 0, 0);

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

```
Select MIN(col_val)
FROM (Select /*+ INDEX_DESC(SCOTT.SALARY_..._TAB) */
      DISTINCT col_val
      FROM SCOTT.SALARY_INDEX_STORAGE_TAB
      WHERE col_val < 10000
      ORDER BY col_val DESC)
WHERE rownum <= 10;
```

```
MIN(COL_VAL)
------------
        7800
```

cnum – identyfikator kursora

```
Select base_rowid
FROM SCOTT.SALARY_INDEX_STORAGE_TAB
WHERE col_val BETWEEN 7800 AND 10000;
```

```
BASE_ROWID
------------------
AAAK8PAABAAAOQKAAd
AAAK8PAABAAAOQKAAn
AAAK8PAABAAAOQKAA1
AAAK8PAABAAAOQKABG
AAAK8PAABAAAOQKABM
AAAK8PAABAAAOQKAAb
AAAK8PAABAAAOQKAAm
AAAK8PAABAAAOQKABe
. . .
```

Wykrycie nieprawidłowego użycia

```
. . .
  -- Take care of some error cases.
  -- The only predicates in which position operator can appear are
  --    op() = 1      OR
  --    op() = 0      OR
  --    op() between 0 and 1
  IF (((strt != 1) AND (strt != 0)) OR
      ((stop != 1) AND (stop != 0)) OR
      ((strt = 1) AND (stop = 0))) THEN
    RAISE_APPLICATION_ERROR(-20101,
                            'incorrect predicate for position_between operator');
  END IF;

  IF (lower_pos > upper_pos) THEN
    RAISE_APPLICATION_ERROR(-20101, 'Upper Position must be greater than or
    equal to Lower Position');
  END IF;

  IF (lower_pos <= 0) THEN
    RAISE_APPLICATION_ERROR(-20101, 'Both Positions must be greater than zero');
  END IF;
. . .
```


Przykład – ODCIIndexFetch (1)

```
MEMBER FUNCTION ODCIINDEXFETCH (SELF IN OUT position_im, nrows NUMBER,
                                rids OUT SYS.ODCIRIDLIST, env SYS.ODCIEnv)
 RETURN NUMBER IS
```

- nrows – parametr wejściowy – liczba wierszy spełniających wyspecyfikowany w zapytaniu predykat. Wartość tego parametru jest określona gdy dane indeksu przechowywane są w tabeli relacyjnej. Wówczas w ramach poprzedniej funkcji ODCIIndexStart zostaje przygotowane i wykonane zapytanie SQL na tej tabeli.
- rids – parametr wyjściowy – przechowuje listę identyfikatorów wierszy spełniających predykat
- SELF – przechowuje instancję definiowanego typu obiektowego
  - howmany – ile wierszy zostało pobranych w wyniku wcześniejszych wywołań funkcji ODCIIndexFetch (wcześniejszych pobrań)
  - curnum – identyfikator kursora z rowid wierszy spełniających warunek

Przykład – ODCIIndexFetch (2)

```
MEMBER FUNCTION ODCIINDEXFETCH (SELF IN OUT position_im, nrows NUMBER,
                                rids OUT SYS.ODCIRIDLIST, env SYS.ODCIEnv)
 RETURN NUMBER IS
  cnum     INTEGER;
  rid_tab DBMS_SQL.Varchar2_table;
  rlist    SYS.ODCIRIDLIST := SYS.ODCIRIDLIST();
  i        INTEGER;
  d        INTEGER;
 BEGIN
  cnum := SELF.curnum;

  IF self.howmany = 0 THEN
    dbms_sql.define_array(cnum, 1, rid_tab, nrows, 1);
    d := DBMS_SQL.EXECUTE(cnum);
  END IF;

  d := DBMS_SQL.FETCH_ROWS(cnum);

  IF d = nrows THEN
    rlist.extend(d);
  ELSE
    rlist.extend(d+1);
  END IF;

  DBMS_SQL.COLUMN_VALUE(cnum, 1, rid_tab);
. . .
```

```
SQL> desc SYS.ODCIRIDLIST
 SYS.ODCIRIDLIST VARRAY(32767) OF VARCHAR2(5072)
```

```
DBMS_SQL.COLUMN_VALUE(
c IN INTEGER,
position IN INTEGER,
<table_variable> OUT <datatype>);
```

```
. . .
  for i in 1..d loop
    rlist(i) := rid_tab(i+SELF.howmany);
  end loop;

  SELF.howmany := SELF.howmany + d;
  rids := rlist;

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


Przykład – ODCIIndexClose

```
  MEMBER FUNCTION ODCIINDEXCLOSE (env SYS.ODCIEnv)
    RETURN NUMBER IS
    cnum INTEGER;
  BEGIN
    cnum := SELF.curnum;
    DBMS_SQL.CLOSE_CURSOR(cnum);
    RETURN ODCICONST.SUCCESS;
  END;

END;  -- koniec definicji ciała typu
/
```

Funkcja, operator, typ indeksu, indeks, zapytanie

```
CREATE OR REPLACE FUNCTION function_for_position_between
                          (col NUMBER, lower_pos NUMBER, upper_pos NUMBER)
RETURN NUMBER AS
begin
  return -1;
end;
/
```

```
CREATE OR REPLACE OPERATOR position_between
   BINDING (NUMBER, NUMBER, NUMBER) RETURN NUMBER
USING function_for_position_between;
```

```
CREATE INDEXTYPE position_indextype
   FOR position_between(NUMBER, NUMBER, NUMBER)
   USING position_im;

CREATE INDEX salary_index ON employees(salary)
   INDEXTYPE IS position_indextype;

SELECT last_name, salary FROM employees
   WHERE position_between(salary, 10, 20)=1
   ORDER BY salary DESC;
```

```
CREATE INDEXTYPE psbtree
FOR
  eq(VARCHAR2, VARCHAR2),
  lt(VARCHAR2, VARCHAR2),
  gt(VARCHAR2, VARCHAR2)
USING psbtree_im
```


Implementacja funkcyjna oparta na indeksie

- Poprzednie rozwiązanie jest oczywiście wystarczające dla obsługi operatora. Funkcja pełni rolę jedynie elementu implementacyjnego.
- Wykorzystanie funkcji w tym przypadku jest bezcelowe
- W większości przypadków, jednakże stworzenie nowego typu indeksu może być wykorzystane nie tylko przez operator ale i funkcję.
- Funkcja ta będzie wykorzystana nie tylko podczas bezpośredniego wywołania, ale również w przypadkach kiedy operator zostanie użyty w funkcyjny sposób – nie do selekcji wierszy a do zwracania wyniku określonego wyrażenia np.:

```
SELECT last_name, salary, position_between(salary, 10, 20)
FROM employees
ORDER BY salary DESC;
```


Implementacja funkcyjna oparta na indeksie

```
CREATE FUNCTION TextContains (Text IN VARCHAR2, Key IN VARCHAR2,
indexctx IN ODCIIndexCtx, scanctx IN OUT TextIndexMethods, scanflg IN NUMBER)
```

- Aby osiągnąć powyżej omówiony efekt funkcyjna implementacja powinna posiadać dodatkowo trzy parametry
  - Index context – zawierający informację o indeksie domenowym i identyfikator wiersza który podlega ewaluacji przez operator
  - Scan context – wartość kontekstowa współdzielona przez kolejne wywołania operatora – zgodna z typem indeksu wykorzystywanym przez operator
  - Scan flag – wskazująca na to czy jest ostatnie wywołanie operator i czy w związku z tym nie należy wykonać operacji kończących korzystanie z indeksu i operatora

```
SQL> desc sys.ODCIIndexCtx
 Nazwa          Typ
 -------------- --------------
 INDEXINFO      ODCIINDEXINFO
 RID            VARCHAR2(5072)
```


Implementacja funkcyjna oparta na indeksie

- Oprócz zmian parametrów funkcji należy również zmodyfikować sposób tworzenia operatora
- Podczas tworzenia operatora dodatkowo wykorzystywane są dwie klauzule:
  - WITH INDEX CONTEXT – określa, że funkcjonalna implementacja może wykorzystywać indeks domenowy
  - SCAN CONTEXT – definiuje typ wykorzystywany przez argument kontekstowy funkcji. Musi być to ten sam typ, który odpowiada typowi indeksu wspomagającego operator.

```
CREATE OPERATOR Contains
BINDING (VARCHAR2, VARCHAR2) RETURN NUMBER
WITH INDEX CONTEXT, SCAN CONTEXT TextIndexMethods
USING TextContains;
```

Implementacja funkcyjna oparta na indeksie

- Podczas wykorzystania implementacji funkcyjnej opeartej na indeksie RDBMS ustawia argumenty wywołań następująco:
  - Początkowy zbiór argmentów jest taki jaki został wyspecyfikowany przez użytkownika operatora.
  - Jeśli pierwszy argument nie jest kolumną atrybuty ODCIIndexCtx są ustawiane na NULL.
  - Jeśli pierwszy argument kolumną jest wówczas ODCIIndexCtx ustawiany jest następująco:
    - Jeśli nie ma odpowiedniego indeksu domenowego wówczas ODCIIndexInfo jest ustawiany na NULL, w przeciwnym przypadku jest ustawiany zgodnie z informacją i indeksie
    - Atrybut rowid przechowuje identyfikator wiersza na którym operuje
  - Podczas pierwszego wywołania scan context jest ustawiany na NULL. Jako że jest to argument IN/OUT wartość zwrócona przez pierwsze wywołanie jest przesyłana do kolejnych wywołań operatora
  - Argument scan flag jest ustawiany na wartość RegularCall dla zwykłego wywołania operatora. Po ostatnim wywołaniu implementacja funkcyjna wywoływana jest jeszcze raz w celu wykonania operacji końcowych. Podczas tego ostatniego wywołania scan flag ustawiony jest na CleanupCall a pozostałe argumenty poza scan context przyjmują wartość NULL

Przykład implementacji funkcyjnej – funkcja

```
CREATE OR REPLACE FUNCTION function_for_position_between
                          (col NUMBER, lower_pos NUMBER, upper_pos NUMBER,
                           indexctx IN SYS.ODCIIndexCtx,
                           scanctx IN OUT position_im,
                           scanflg IN NUMBER)
RETURN NUMBER AS
  rid              ROWID;
  storage_tab_name VARCHAR2(65);
  lower_bound_stmt VARCHAR2(2000);
  upper_bound_stmt VARCHAR2(2000);
  col_val_stmt     VARCHAR2(2000);
  lower_bound      NUMBER;
  upper_bound      NUMBER;
  column_value     NUMBER;
BEGIN
  IF (indexctx.IndexInfo IS NOT NULL) THEN
    storage_tab_name := indexctx.IndexInfo.INDEXSCHEMA || '.' ||
                        indexctx.IndexInfo.INDEXNAME || '_STORAGE_TAB';
    IF (scanctx IS NULL) THEN
      --Oznacza to, że jest to pierwsze wywołanie otwieramy zatem kursor
      --Małe sprawdzenia błędnego wywołania
      IF (lower_pos > upper_pos) THEN
        RAISE_APPLICATION_ERROR(-20101,
          'Upper Position must be greater than or equal to Lower Position');
      END IF;
      IF (lower_pos <= 0) THEN
        RAISE_APPLICATION_ERROR(-20101,
          'Both Positions must be greater than zero');
      END IF;
. . .
```


Przykład implementacji funkcyjnej – funkcja (cd)

```
. . .
    --Uzyskaj dolną i górną wartość, która nas interesuje
    upper_bound_stmt := 'Select MIN(col_val) FROM (Select /*+ INDEX_DESC(' ||
                        storage_tab_name || ') */ DISTINCT ' ||
                        'col_val FROM ' || storage_tab_name || ' ORDER BY ' ||
                        'col_val DESC) WHERE rownum <= ' || lower_pos;
    EXECUTE IMMEDIATE upper_bound_stmt INTO upper_bound;
    IF (lower_pos != upper_pos) THEN
      lower_bound_stmt := 'Select MIN(col_val) FROM (Select /*+ INDEX_DESC(' ||
                          storage_tab_name || ') */ DISTINCT ' ||
                          'col_val FROM ' || storage_tab_name ||
                          ' WHERE col_val < ' || upper_bound || ' ORDER BY ' ||
                          'col_val DESC) WHERE rownum <= ' ||
                          (upper_pos - lower_pos);
      EXECUTE IMMEDIATE lower_bound_stmt INTO lower_bound;
    ELSE
      lower_bound := upper_bound;
    END IF;
    IF (lower_bound IS NULL) THEN
      lower_bound := upper_bound;
    END IF;
    --Przechowaj dolną i górną wartość dla późniejszy wywołań.
    scanctx := position_im(0, 0, lower_bound, upper_bound);
  END IF; -- koniec operacji dla pierwszego wywołania
. . .
```


Przykład implementacji funkcyjnej – funkcja (cd)

```
    --Pobierz wartość kolumny odpowiadającej rowid, i sprawdź czy mieści się
    --w wyliczonym przy pierwszym wymiarze zakresie.
    col_val_stmt := 'Select col_val FROM ' || storage_tab_name ||
                    ' WHERE base_rowid = ''' || indexctx.Rid || '''';
    EXECUTE IMMEDIATE col_val_stmt INTO column_value;
    IF (column_value <= scanctx.upper_bound AND
        column_value >= scanctx.lower_bound AND
        scanflg = ODCICONST.RegularCall) THEN
      RETURN 1;
    ELSE
      RETURN 0;
    END IF;
  ELSE -- indexctx.IndexInfo jest pusty zatem:
    RAISE_APPLICATION_ERROR(-20101, 'A column that has a domain index of' ||
                            'Position indextype must be the first argument');
  END IF;
END;
/
```

Przykład implementacji funkcyjnej – operator

```
CREATE OR REPLACE OPERATOR position_between
   BINDING (NUMBER, NUMBER, NUMBER) RETURN NUMBER
   WITH INDEX CONTEXT, SCAN CONTEXT position_im
   USING function_for_position_between;
```

```
CREATE OPERATOR Ordsys.Contains
BINDING
  (VARCHAR2, VARCHAR2) RETURN NUMBER
USING text.contains,
  (spatial.geo, spatial.geo) RETURN NUMBER
USING spatial.contains;
```

```
CREATE INDEXTYPE position_indextype
   FOR position_between(NUMBER, NUMBER, NUMBER)
   USING position_im;

CREATE INDEX salary_index ON employees(salary)
   INDEXTYPE IS
position_indextype;
```

```
SELECT last_name, salary FROM employees
   WHERE position_between(salary, 10, 20)=1
   ORDER BY salary DESC;

SELECT last_name, salary, position_between(salary, 10, 20)
FROM employees
ORDER BY salary DESC;
```