

KBD2 – Komercyjne bazy danych

dr inż. Tomasz Traczyk

4. Cechy relacyjno-objektowe bazy danych Oracle

Listopad 2006

Copyright © Tomasz Traczyk
Instytut Automatyki i Informatyki Stosowanej Politechniki Warszawskiej
Materiały dydaktyczne przeznaczone są wyłącznie do indywidualnego użytku studiujących.
Rozpowszechnianie kopii bez pisemnej zgody autora jest zabronione.

Streszczenie

Wykład prezentuje wybrane bazy Oracle związane z obiektowością.

Cechy relacyjno-objektowe Oracle

- Idea relacyjno-objektowej b.d.
 - Struktury relacyjne są nadal podstawą budowy b.d.
 - W tabelach relacyjnych można umieszczać złożone obiekty
 - Te obiekty mają cechy właściwe dla OODBMS
 - SQL jest rozszerzony o możliwość działania na takich obiektach (np. SQL3)
- Zalety relacyjno-objektowej b.d.
 - Pełna kompatybilność z rozwiązaniami relacyjnymi
 - Możliwość wykorzystania części zalet obiektowości w połączeniu z zaletami bazy relacyjnej
 - Ewolucyjne przejście do obiektowości
- Co zrealizowano w Oracle?
 - Możliwość definiowania złożonych typów danych
 - Konstrukcje do budowania złożonych typów danych
 - Wewnętrzna niezmienna identyfikacja obiektów (tożsamość)
 - Możliwość wyposażenia obiektów w metody
 - *Object views* – interfejs między światem relacyjnym a obiektowym
 - Przeciążanie operatorów
 - Dziedziczenie
- Czego nie zrealizowano w Oracle?
 - Hermetyzacja

Typy obiektowe

- Tworzenie typu:
`CREATE OR REPLACE TYPE nazwa AS OBJECT(atrybuty [, nagłówki_metod])`
- Specyfikacja „niekompletnego typu” – używana w wypadku odwołań rekursywnych:
`CREATE OR REPLACE TYPE nazwa`
- Specyfikacja ciał metod:
`CREATE OR REPLACE TYPE BODY nazwa AS metody END;`
- Konstruktor
 - ma nazwę taką jak nazwa typu
 - ma parametry odpowiadające atrybutom obiektu
 - służy do kreowania obiektów np. w PL/SQL

Metody

- Metody
 - MEMBER FUNCTION|PROCEDURE – metody dotyczące obiektu (wystąpienia typu), mają pseudoparametr SELF odwołujący się do obiektu na którym działają
 - STATIC FUNCTION|PROCEDURE – metody dotyczące typu (nie ma SELF)
 - dla metod należy używać pragmy RESTRICT_REFERENCES tak jak dla podprogramów w pakietach
- Wywołania metod w PL/SQL
 - dla metod typu MEMBER: obiekt.metoda([parametry])
 - dla metod typu STATIC: nazwa_typu.metoda([parametry])
 - wewnątrz metody dostęp do atrybutów obiektu jest taki, jak do zmiennych
- Metody porównujące
 - MAP MEMBER – funkcja przekształcająca zawartość obiektu na wartość skalarną, używana automatycznie do sortowania i porównań
 - ORDER MEMBER – funkcja porównująca SELF z obiektem przekazanym jako parametr, wynik liczbowy: ujemny, zerowy lub dodatni; służy do sortowania i porównań
 - dla danego typu można zdefiniować albo funkcję MAP, albo ORDER – i tylko jedną (MAP szybciej sortuje, ORDER zwykle łatwiej napisać)

Dziedziczenie

- Zapis: CREATE TYPE nazwa_typu UNDER nazwa_typu_bazowego...
- Sterowanie dziedziczeniem
 - klauzula FINAL nie pozwala tworzyć typów potomnych
 - klauzula NOT INSTANTIABLE tworzy typ abstrakcyjny
 - klauzule te mogą dotyczyć także poszczególnych metod
- Przykrywanie metod
 - metody typu MEMBER, przykrywające (*override*) metody klasy rodzicielskiej, muszą być deklarowane z klauzulą OVERRIDING; używane jest dynamiczne (późne) wiązanie
 - metody typu STATIC, przykrywające (*hide*) metody klasy rodzicielskiej, wiązane są w czasie kompilacji
 - metody klasy potomnej mogą także przeciążać metody klasy rodzicielskiej

Tabele obiektów (*object tables*)

- Tworzenie: `CREATE TABLE nazwa_tabeli OF nazwa_typu`
- Obiekty w takiej tabeli są równoważne wierszom (*row objects*)
- Obiekty mogą być także przechowywane w kolumnach tabeli relacyjnej (*column objects*)
- Atrybuty obiektów w tabeli obiektów są dostępne tak jak kolumny – przez zwykłe zapytania relacyjne
- Zapytania obiektowe:
 - alias tabeli pełni rolę zmiennej korelacyjnej
 - funkcja `VALUE` pobiera wartość całego obiektu:
`SELECT VALUE(o) FROM tabela_obiektowa o`

Identyfikatory i referencje

- Obiekty mają wbudowane wewnętrzne niezmiennie identyfikatory (OIDs)
 - generowane przez system (`OBJECT IDENTIFIER IS SYSTEM GENERATED`)
 - zbudowane z użyciem klucza głównego (`OBJECT IDENTIFIER IS PRIMARY KEY`)
- OIDs można indeksować
- Atrybutem obiektu może być referencja do innego obiektu: `REF typ_obiektowy`
- Referencja może wskazywać na nieistniejący obiekt (*dangling ref*)
 - dereferencja daje wówczas `NULL`
 - można to sprawdzić operatorem `IS DANGLING`
- Referencja może być ograniczona
 - klauzulą `SCOPE IS` do obiektów w określonej tabeli obiektowej (może „zwisnąć”)
 - ograniczeniami typu `REFERENCES` podobnymi do kluczy obcych (nie może „zwisnąć”, ale taka referencja nie może wskazywać do *nested tables*)

Perspektywy obiektowe (*object views*)

- Prezentują dane relacyjne w sposób relacyjno-obiektowy
- Powody stosowania
 - umożliwienie stosowania obiektów w językach programowania i zapytaniach (np. dereferencje zamiast złączeń)
 - zmniejszenie ruchu na sieci (obiekty są „montowane” na serwerze i przesyłane w całości)
- Definiowanie:
`CREATE VIEW nazwa OF typ_obiektowy [WITH OBJECT IDENTIFIER (kolumny)] AS SELECT...`

Kolekcje

- VARRAYs
 - Uporządkowane tablice o określonej maksymalnej liczbie elementów tego samego typu
 - Tworzenie:
`CREATE TYPE nazwa AS VARRAY(max_wymiar) OF typ_elementu`
 - Przechowywanie w tabelach obiektów
 - * trzymane razem z wierszem albo jako BLOB
 - * pobierane z bazy w całości
- Nested tables
 - Nieuporządkowane tabele elementów tego samego typu
 - Tworzenie:
`CREATE TYPE nazwa AS NESTED TABLE OF typ_elementu`
 - Przechowywanie w tabelach obiektów
 - * w osobnych tabelach relacyjnych
 - * konieczne jest określenie tabel przechowujących *nested tables*:
`NESTED TABLE kolumnaa STORE AS tabela`

^aKolumna lub atrybut typu *nested table*

- Te kolekcje mogą być tworzone jako obiekty PL/SQL oraz zapisywane w bazie danych w kolumnach
- Mogą być zagnieżdżane (w każdej konfiguracji)

Tablice asocjacyjne

`TYPE nazwa_tablicy IS TABLE OF typ_elementów INDEX BY ...`

- Są obiektami PL/SQL tworzonymi w pamięci, nie mogą być zapisane w bazie danych
- Używane jako bufor do obliczeń, niewielkie tymczasowe słowniki itp.
- Nie mają z góry założonej wielkości
 - nowe elementy wprowadza się przez podstawienie
 - elementy można usuwać metodą DELETE

Tablice indeksowane liczbami

`INDEX BY BINARY_INTEGER | PLS_INTEGERa`

- Indeks jest liczbą całkowitą ze znakiem (wartości ujemne dozwolone)
- Wartości indeksów mogą być nieciągłe

^aSynonimy

Tablice indeksowane napisami

`INDEX BY VARCHAR2(długość)`

- Typowe tablice asocjacyjne

Operacje na kolekcjach

- Mogą być podstawiane instrukcją podstawienia oraz przekazywane do/z podprogramów
- VARRAYs i *nested tables* mogą być zapisywane do tabel instrukcjami DML
- Nie mogą być używane w porównaniach oraz klauzulach DISTINCT, GROUP BY i ORDER BY
 - wyjątek: *nested tables* z tym samym typem elementów mogą być porównywane operatorem równości
- Mogą być równe NULL i może być badane, czy tak jest
- W zapytaniach można do kolekcji odwoływać się jak do tabel, stosując operator TABLE
`SELECT ... FROM TABLE(kolekcja) ...`
gdzie *kolekcja* jest wyrażeniem lub podzapytaniem zwracającym kolekcję
- W ten sposób można też budować perspektywy relacyjne udostępniające dane z kolekcji tak jak dane w pełni relacyjne
- Błędne operowanie kolekcjami może prowadzić do wyjątków COLLECTION_IS_NULL, NO_DATA_FOUND, SUBSCRIPT_BEYOND_COUNT, SUBSCRIPT_OUTSIDE_LIMIT, VALUE_ERROR

Metody operujące na kolekcjach

EXTEND, TRIM	Dodaje puste elementy, usuwa końcowe (nie do tablic asocjacyjnych)
DELETE	Usuwanie pozycji
EXISTS	Istnienie pozycji pod indeksem
COUNT, LIMIT	Wielkość kolekcji
FIRST, LAST	Zakres indeksów
PRIOR, NEXT	„Nawigacja” po indeksach

Operacje masowe

- Dane między kolekcjami a bazą danych mogą być przekazywane przez operacje masowe (*bulk bind*)
- Operacje te są znacznie wydajniejsze od wykonywanych w pętli pojedynczych odczytów/zapisów

Instrukcja FORALL

- Służy do wykonywania instrukcji DML na podstawie zawartości kolekcji (oprócz tablic asocjacyjnych indeksowanych tekstem)
- Zapisuje wszystkie dane w jednej operacji, np.
`FORALL j IN tablica.FIRST..tablica.LAST INSERT INTO tabela(kolumna) VALUES tablica(j)`
- Może być użyta do warunków
`FORALL j IN INDICES OF tablica DELETE FROM tabela WHERE kolumna=tablica(j)`
- INDICES OF pomija nieistniejące pozycje tablicy

Klauzula BULK COLLECT

- Służy do odczytu wyniku zapytania do kolekcji w jednej operacji
- Może być użyta w klauzuli SELECT
`SELECT ... BULK COLLECT INTO lista tablic`
- Może być stosowana w operacjach na kursorach
`FETCH kursor BULK COLLECT INTO ...`
- Może być także użyta w klauzuli RETURNING zdań DML
- Nie można jej zagnieździć w pętli FORALL

Java w Oracle

- Maszyna wirtualna Javy wbudowana w bazę danych
- Możliwości:
 - Java 2
 - możliwość pisania procedur składowanych i wyzwalaczy w Javie
 - możliwość wykonywania w serwerze b.d. serwletów i JSP
 - wbudowany ORB (CORBA 2.0)
 - możliwość użycia EJB
- Ładowanie klas Javy
 - CREATE JAVA CLASS USING BFILE...
 - program loadjava
- Dostęp do Javy z PL/SQL: „publikowanie” klas, np.:

```
CREATE OR REPLACE FUNCTION nazwa_plsql([parametry_plsql]) RETURN typ_plsql AS LANGUAGE  
JAVA NAME 'nazwa ([parametry]) return java.lang.typ'
```
- Dostęp do danych z Javy
 - JDBC
 - SQLJ