ORACLE

Oracle Database 12c
Programowanie w języku PL/SQL

Twój przewodnik po PL/SQL!

Michael McLaughlin

Helion

Oracle Press™

Tytuł oryginału: Oracle Database 12c PL/SQL Programming

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-246-9923-0

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/or12ps
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/or12ps.zip

Printed in Poland.

Spis treści

CZĘŚĆ II
Programowanie w języku PL/SQL

CZĘŚĆ III
Dodatki i słowniczek

ROZDZIAŁ

2

Nowe funkcje

W tym rozdziale opisane są nowe funkcje języków SQL i PL/SQL, które bezpośrednio wpływają na sposób pisania programów w języku Oracle PL/SQL i zarządzania nimi. W czasie gdy powstawała ta książka, możliwe było jeszcze pojawienie się dodatkowych funkcji w tym wydaniu bazy lub w jej następnej wersji. Informacje o najnowszych zmianach znajdziesz w przewodniku *Oracle Database New Features Guide*.

Omówienie nowych mechanizmów jest w tym rozdziale podzielone na dwie części:

- nowe funkcje SQL-a,
- nowe funkcje języka PL/SQL.

Niektóre funkcje zasługują na wielostronicowe opisy z przykładowym kodem, natomiast inne wymagają tylko krótkiego przedstawienia, ponieważ są omówione w innych miejscach książki. Wtedy wskazuję tylko rozdział z opisem danego mechanizmu. Jeśli dopiero poznajesz język PL/SQL, możesz przejść bezpośrednio do rozdziału 3., aby zapoznać się z podstawami tego języka. Następnie wróć do niniejszego rozdziału.

Nowe funkcje SQL-a

Oracle udostępnia w bazie Oracle Database 12*c* wiele nowych funkcji. Zmiany w składni SQL-a są stosunkowo liczne, dlatego tu opisano głównie te, które mają wpływ na działanie kodu.

- W Oracle Database 12*c* można korzystać z katalogów wirtualnych przy tworzeniu obiektów `LIBRARY` z zewnętrznymi procedurami.
- Technologia Flashback została usprawniona dzięki wprowadzeniu wymiarów rzeczywistego czasu (ang. *valid-time* — VT).
- W Oracle Database 12*c* rozbudowano składnię instrukcji `JOIN` ze standardu ANSI 92. Teraz można wywołać polecenie `LEFT OUTER JOIN` dla dwóch lub więcej tabel podanych po lewej stronie złączenia.
- Wartości domyślne kolumn mogą teraz zawierać referencje do pseudokolumn `.nextval` i `.currval`, co jest wygodnym rozwiązaniem.
- W bazie Oracle Database 12*c* wprowadzono kolumnę `IDENTITY` (kolumnę identyfikacyjną), która automatycznie generuje sekwencyjne wartości klucza sztucznego.
- W Oracle Database 12*c* pojawiła się klauzula `ON NULL` dla wartości domyślnych, co chroni przed bezpośrednim zastąpieniem takich wartości wartością `null`.
- W Oracle Database 12*c* zwiększono pojemność typów danych `VARCHAR2`, `NVARCHAR2` i `RAW` (wymaga to poprawnego ustawienia parametrów bazy).

- Oracle Database 12*c*, podobnie jak baza Microsoft SQL Server, umożliwia przekazywanie wyników zapytań bezpośrednio do programów zewnętrznych.
- Oracle Database 12*c* zapewnia natywną obsługę SQL-a w zakresie ograniczania liczby wyników zapytania i przedziałów zwracanych wyników.
- W Oracle Database 12*c* pojawił się nowy sterownik zastępujący bibliotekę kliencką MySQL 5.5.
- Do SQL-a dodano polecenia `CROSS APPLY`, `OUTER APPLY` i `LATERAL`, przydatne przy pracy z tabelami zagnieżdżonymi.
- Obecnie można tworzyć widoki za pomocą uprawnień jednostki definiującej (jest to dawny sposób, obecnie używany jako domyślny) lub przy użyciu uprawnień jednostki wywołującej. Jedyną różnicą jest składnia. W modelu z uprawnieniami jednostki definiującej używane jest polecenie `DEQUEATH DEFINER`, a dla jednostki wywołującej używa się instrukcji `DEQUEATH INVOKER`.

Tworzenie obiektów LIBRARY za pomocą katalogu wirtualnego

Obiekty `LIBRARY` to repozytoria z zewnętrznymi bibliotekami. Takie biblioteki są napisane w C lub w językach, które można wywoływać w C. Użycie takiej biblioteki wymaga umieszczenia pliku z nią w katalogu i wskazania nazwy tego katalogu w pliku *listener.ora* i instrukcji `CREATE LIBRARY`.

W Oracle Database 12*c* można zastąpić katalog fizyczny katalogiem wirtualnym. Poniższa składnia pokazuje, jak utworzyć bibliotekę za pomocą katalogu fizycznego:

```
SQL> CREATE OR REPLACE LIBRARY demo IS
  2  '<katalog_główny_oracle>/<niestandardowa_biblioteka>/<nazwa_pliku>.<rozszerzenie>';
  3  /
```

Za pomocą nowej składni obiekt `LIBRARY` można utworzyć na podstawie katalogu wirtualnego:

```
SQL> CREATE OR REPLACE LIBRARY demo IS '<nazwa_biblioteki.so>' IN
  2  nazwa_katalogu_wirtualnego;
```

Drugi argument to katalog wirtualny. Tworzenie takich katalogów opisano w podrozdziale „Katalogi wirtualne" w dodatku B. W Oracle Database 12*c* proces tworzenia katalogów tego rodzaju się nie zmienił.

Definiowanie tabel z obsługą wymiaru VT

W Oracle Database 12*c* dostępna jest obecnie obsługa wymiaru VT (ang. *valid time*). Ten wymiar różni się od wymiaru czasu transakcji (ang. *transaction time* — TT). VT odpowiada dacie zdarzenia biznesowego, na przykład zatrudnienia, promocji lub zwolnienia pracownika. TT odpowiada fizycznemu momentowi wstawienia lub zaktualizowania wiersza.

W Oracle Database 11*g* pojawiły się archiwa Flashback Data Archive, w których używano czasu TT. Technologia Flashback pozwala przyjrzeć się danym archiwalnym w celu przeanalizowania trendów w zapytaniach, różnic w raportach lub historii transakcji. Są to wymiary retrospektywne, ponieważ dane są w nich dzielone na podstawie czasu operacji.

W Oracle Database 12*c* wprowadzono obsługę wymiaru VT w wyniku sformalizowania w definicjach tabel dwóch podejść. Jedno polega na definiowaniu okresów przez jawne przypisanie ich do kolumn. Drugie definiuje okresy w kolumnach niejawnie. Nowa składnia SQL-a związana z wymiarem VT to `PERIOD FOR`. Jej zastosowanie zobaczysz w przykładowych instrukcjach `CREATE TABLE` w dalszych punktach.

Warto zauważyć, że to wymiar VT, a nie TT steruje operacjami retrospektywnymi. Za pomocą wymiaru VT można zarządzać procesem ILM (ang. *Information Lifecycle Management*).

Tabele z jawnie dodanymi kolumnami VT

Przyjrzyj się teraz przykładowej tabeli `rental`. Obejmuje ona kolumny `check_out_date` i `return_date`. W wersjach starszych niż Oracle Database 12*c* za zarządzanie tymi kolumnami odpowiadał interfejs API aplikacji. Zawierają one ważną logikę biznesową opisującą, jak wypożyczalnia filmów wideo, na przykład Redbox, nalicza płatności. Wymiar VT pozwala teraz jawnie zidentyfikować takie kolumny:

```
SQL> CREATE TABLE rental
  2  ( rental_id       NUMBER GENERATED ALWAYS AS IDENTITY
  3  , customer_id     NUMBER CONSTRAINT nn_rental_01 NOT NULL
  4  , check_out_date  DATE   CONSTRAINT nn_rental_02 NOT NULL
  5  , return_date     DATE
  6  , created_by      NUMBER  CONSTRAINT nn_rental_03 NOT NULL
  7  , creation_date   DATE    CONSTRAINT nn_rental_04 NOT NULL
  8  , last_updated_by NUMBER  CONSTRAINT nn_rental_05 NOT NULL
  9  , last_update_date DATE   CONSTRAINT nn_rental_06 NOT NULL
 10  , PERIOD FOR rental_event (check_out_date, return_date)
 11  , CONSTRAINT pk_rental    PRIMARY KEY(rental_id)
 12  , CONSTRAINT fk_rental_01 FOREIGN KEY(customer_id)
 13    REFERENCES contact(contact_id)
 14  , CONSTRAINT fk_rental_02 FOREIGN KEY(created_by)
 15    REFERENCES system_user(system_user_id)
 16  , CONSTRAINT fk_rental_03 FOREIGN KEY(last_updated_by)
 17    REFERENCES system_user(system_user_id));
```

Wiersze 4. i 5. to kolumny VT z logiką biznesową. W wierszu 10. do okresu związanego z regułą biznesową jawnie przypisywany jest identyfikator. Pozwala to na wywoływanie retrospektywnych zapytań dotyczących danego okresu.

Oto przykładowe zapytanie z wykorzystaniem wymiaru VT:

```
SQL> SELECT *
  2  rental AS OF PERIOD FOR rental_event
  3              TO_TIMESTAMP('04-AUG-2013 12:00:00 AM');
```

Ponadto jeśli używasz pakietu `dbms_flashback_archive`, możesz zastosować człon `AS OF` do przedziałów VT. Zachęcam do korzystania z podejścia z jawnym wskazywaniem kolumn VT.

Tabele z niejawnie definiowanymi kolumnami z czasem VT

Gdy Oracle wprowadza nowe funkcje, zawsze udostępnia różne możliwości. Dotyczy to także kolumn z czasem VT. Możesz utworzyć tabelę z niejawnie definiowanymi kolumnami z czasem VT. W tym celu usuń referencje do tych kolumn. W kodzie przykładowej tabeli `rental` oznacza to zmianę wiersza 10. w następujący sposób:

```
10 , PERIOD FOR rental_event
```

W tym wierszu pominięto kolumny z instrukcji `CREATE TABLE`.

Wzbogacona składnia instrukcji LEFT OUTER JOIN w bazach Oracle

Oracle Database 12*c* obsługuje obecnie polecenie `LEFT OUTER JOIN` umożliwiające podanie po lewej stronie złączenia dwóch i więcej tabel. Wcześniej można było w tym miejscu zastosować tylko jedną tabelę. W bazie Oracle Database 11*g* próby podania dwóch tabel powodowały błąd `ORA-01417`.

Oto zalety tej nowej funkcji:

- Podawanie tabel po lewej stronie umożliwia zmianę ich kolejności, co z kolei pozwala na generowanie lepszych planów wykonania instrukcji.
- Obsługa wielu widoków upraszcza pracę programistów piszących operacje `OUTER JOIN`.

Wadą rozbudowanej instrukcji `LEFT OUTER JOIN` z baz Oracle jest brak przenośności. Korzystne jest natomiast to, że można tworzyć wydajniejsze operacje `OUTER JOIN`.

Domyślne wartości kolumn oparte na sekwencjach

Oracle Database 12*c* umożliwia jawne wiązanie sekwencji z tabelami. Dostępne są dwie techniki. Jedna polega na utworzeniu sekwencji i jawnym powiązaniu jej z kolumną tabeli. Druga wymaga wykorzystania kolumn identyfikacyjnych (z modyfikatorem `IDENTITY`; jest to następna nowa funkcja). W pierwszym z tych podejść można wykorzystać domyślne wartości z niezależnych sekwencji.

Przyjrzyj się przykładowej tabeli customer. Dla uproszczenia ma ona tylko dwie kolumny. Pierwsza to kolumna z kluczem sztucznym zawierająca kolejne wartości. Te wartości nie są powiązane z danymi z żadnej tabeli i powinny być powiązane w sposób „jeden do jednego" z kluczem naturalnym tabeli. Klucz naturalny to jedna lub — zazwyczaj — więcej kolumn, które niepowtarzalnie identyfikują każdy wiersz tabeli. W tym przykładzie kluczem naturalnym jest kolumna customer_name. W praktyce jest mało prawdopodobne, by taka kolumna była dobrym kluczem naturalnym, jednak takie założenie pozwala uprościć przykład i skoncentrować się na domyślnych wartościach kolumny.

Przed utworzeniem tabeli należy przygotować sekwencję. Jest to odejście od techniki stosowanej w przeszłości w bazach Oracle, jednak tu omawiane są rozwiązania z nowego wspaniałego świata baz Oracle Database 12*c*. Utwórz więc sekwencję rozpoczynającą się od wartości 1:

```
SQL> CREATE SEQUENCE customer_s;
```

Tę sekwencję należy utworzyć na początku, ponieważ jest ona używana w czasie tworzenia tabeli o domyślnych wartościach kolumny.

```
SQL> CREATE TABLE customer
  2  ( customer_id   NUMBER DEFAULT customer_s.nextval
  3  , customer_name VARCHAR2(20));
```

Ponieważ ten przykład ma ilustrować zarządzanie wartościami klucza głównego i klucza obcego w ramach transakcji, trzeba też utworzyć następną sekwencję i tabelę. Tu będą to sekwencja preference_s i tabela preference.

Zamiast rozdzielać obie instrukcje, poniżej przedstawiono je jedna pod drugą:

```
SQL> CREATE SEQUENCE preference_s;
SQL> CREATE TABLE preference
  2  ( preference_id   NUMBER DEFAULT preference_s.nextval
  3  , customer_id     NUMBER DEFAULT customer_s.currval
  4  , preference_name VARCHAR2(20));
```

Wartości sekwencji podanej po słowie DEFAULT pozwalają pominąć wyzwalacze ON INSERT. Dzięki nim można też zrezygnować z bezpośredniego stosowania pseudokolumn .nextval i .currval w instrukcjach INSERT związanych z sekwencją. Trzeba jednak pamiętać, że zależność między instrukcjami .nextval i .currval się nie zmieniła. *Trzeba wywołać .nextval dla sekwencji przed wywołaniem dla niej .currval w danej sesji.*


Uwaga

Z powodu zależności między dwoma wspomnianymi pseudokolumnami sekwencji zalecam ostrożność przy stosowaniu tej techniki.

Teraz można wstawiać wiersze do obu tabel za pomocą zmodyfikowanej sygnatury (ang. *override signature*). Takie sygnatury to listy wszystkich wymaganych i używanych opcjonalnych kolumn, których wartości programista chce wstawić do tabeli. Przy wstawianiu danych do dwóch przykładowych tabel trzeba się upewnić, że wartości kolumn customer_id są ze sobą zgodne. Dzięki temu możliwe są złączenia równościowe między tabelami customer i preference.

```
SQL> INSERT INTO customer (customer_name) VALUES ('Pan Nowak');
SQL> INSERT INTO preference (preference_name) VALUES ('Jasne mocne');
```

Oba wiersze są więc wstawiane bez bezpośrednio podanych wartości klucza sztucznego. Zobaczmy, czy baza Oracle Database 12*c* poprawnie radzi sobie z taką sytuacją. Proste zapytanie ze złączeniem:

```
SQL> SELECT  *
  2  FROM    customer c INNER JOIN preference p USING(customer_id);
```

powinno dać następujący wynik:

```
CUSTOMER_ID CUSTOMER_NAME PREFERENCE_ID PREFERENCE_NAME
----------- ------------- ------------- --------------------
          1 Pan Nowak                 1 Jasne mocne
```

Wyniki są dowodem na to, że to podejście działa. Z dalszego punktu „Kolumny IDENTITY" dowiesz się, jak używać takich kolumn.

Wartości domyślne przy bezpośrednim wstawianiu null

Baza Oracle Database od dawna umożliwia podawanie wartości domyślnych dla dowolnej kolumny. Taką wartość można było jednak zmienić w wyniku bezpośredniego podania wartości `null` w instrukcji `INSERT`. W bazie Oracle Database 12*c* taką bezpośrednio podaną wartość `null` można zastępować wartością domyślną.

```
SQL> CREATE TABLE essay
  2 ( essay_name VARCHAR2(30) DEFAULT essay_s.nextval
  3 , essayist   VARCHAR2(30)
  4 , published  DATE DEFAULT TRUNC(SYSDATE)
  5 , text       CLOB
  6 , CONSTRAINT essay_pk
  7   PRIMARY KEY (essay_name, essayist, published));
```

Wiersz 4. gwarantuje, że próby pominięcia danych w kolumnie `published` spowodują wstawienie bieżącej daty. W dodatku C (poświęconym funkcjom wbudowanym SQL-a) wyjaśniono, że funkcja `TRUNC` usuwa godziny i minuty z typów danych obejmujących datę i czas. Wszystkie typy danych `DATE` w bazach Oracle obejmują datę i czas.

Poniższa instrukcja `INSERT` wstawia wiersz do tabeli `essay`. Ta instrukcja działa tak samo w wersjach 11*g* i 12*c* bazy Oracle Database — wstawia bieżącą datę (bez godzin, minut i sekund) do kolumny `published`. Dzieje się tak, ponieważ kolumna `published` nie znajduje się na liście kolumn w zmodyfikowanej sygnaturze.

```
INSERT INTO essay
( essay_name
, essayist
, text )
VALUES
('Dlaczego chcę zostać Supermanem?'
,'21-SEP-2011'
,'W pewnym okresie swojego życia każdy chciałby stać się kimś innym...');
```

Jeśli do zmodyfikowanej sygnatury dodasz kolumnę `published`, będziesz mógł bezpośrednio wstawić wartość `null`. Powoduje ona zastąpienie standardowej wartości domyślnej. W wersjach starszych niż Oracle Database 12*c* nie można było zapobiec takiemu zastępowaniu wartości. W Oracle Database 12*c* dostępna jest instrukcja `ON NULL`, która pozwala uniemożliwić bezpośrednie wstawianie wartości `null` do kolumny.

Aby uzyskać ten efekt, należy wprowadzić następującą zmianę w wierszu 4. wcześniejszej instrukcji `CREATE TABLE`:

```
4 , published DATE DEFAULT ON NULL TRUNC(SYSDATE)
```

Instrukcja `ON NULL` powoduje, że w bazie Oracle Database 12*c* w kolumnie `published` nie można wstawić wartości `null`.

Kolumny IDENTITY

Społeczność związana z bazami danych (czyli konkurencja) wskazywała na słabość baz Oracle, ponieważ nie obejmowały one kolumn `IDENTITY` (kolumn identyfikacyjnych). Takie kolumny obsługują automatyczne numerowanie wierszy i zwykle pełnią funkcję klucza sztucznego, tworzonego na podstawie sekwencji kolejnych wartości.

W bazie Oracle Database 12*c* dostępny jest operator tożsamościowy. Jeszcze lepszą wiadomością jest to, że ta wersja bazy umożliwia generowanie wartości identyfikacyjnych. Podstawowa kolumna `IDENTITY` zwykle ma nazwę `id`, a bazy Oracle obsługują tę konwencję (chyba że programista zmieni nazwę takiej kolumny).

W nazwach kolumn `IDENTITY` lepiej stosować przyrostek `_id` zamiast `id`.

Poniższy kod tworzy tabelę o dwóch kolumnach — identyfikacyjnej i tekstowej:

```
SQL> CREATE TABLE identity
  2  ( id NUMBER GENERATED ALWAYS AS IDENTITY
  3  , text VARCHAR2(10));
```

Ta przykładowa tabela pozwala pominąć kolumnę `id` w instrukcjach `INSERT`. Gdyby tabela obejmowała tylko jedną kolumnę, podanie wartości kolumny `id` byłoby konieczne. Łatwiej napisać zmodyfikowaną sygnaturę, która jest jednym z przykładów stosowania *notacji opartej na nazwach*. W takich sygnaturach należy podać *listę kolumn* między nazwą tabeli a klauzulą `VALUES` lub podzapytaniem.

Przykładowa tabela `identity` jest bardzo prosta, ponieważ domyślne ustawienie kolumny identyfikacyjnej to `ALWAYS`, co oznacza, że nie można ręcznie wprowadzać wartości kolumny `id`. Dlatego jeśli tabela nie obejmuje innych kolumn, nie można w niej wstawiać wierszy. Wstawianie wierszy do tabeli z kolumną identyfikacyjną jest możliwe tylko wtedy, gdy dana tabela ma więcej niż jedną kolumnę (tak jak w przykładzie).

Poprawny sposób wywoływania instrukcji `INSERT` polega na pominięciu kolumny `id` na liście kolumn:

```
SQL> INSERT INTO identity (text) VALUES ('Jeden');
```

Dlaczego firma Oracle wybrała ustawienie `ALWAYS` jako domyślne? W dokumentacji baz Oracle nie jest to wyjaśnione, jednak podejrzewam, że chodziło o następującą kwestię: jeśli programista użyje opcji `BY DEFAULT` i poda liczbę wyższą niż obecnie wygenerowana wartość sekwencji, może zduplikować wartość kolumny bez klucza głównego lub ograniczenia niepowtarzalności i spowodować nieudane wstawianie, jeśli tabela ma klucz główny lub ograniczenie niepowtarzalności.

W dodatku B znajduje się punkt „Kolumny IDENTITY”, w którym opisano, jak korzystać z takich kolumn. Tu warto wspomnieć, że należy zapoznać się z klauzulą `RETURNING INTO` instrukcji `INSERT`, ponieważ sekwencja wartości dla kolumn `IDENTITY` jest generowana przez system i trudno dostępna. Szczegółowe informacje znajdziesz w ramce „Wiązanie kolumn IDENTITY z sekwencjami” w dodatku B.

Kolumny `IDENTITY` zmieniają sposób pracy w bazach Oracle — przynajmniej wtedy, gdy nie musisz zapewniać obsługi starszego kodu, na przykład opartego na pakiecie Oracle E-Business Suite. Podejście zastosowane w bazie Oracle Database 12*c* sprawia, że można zaprzestać korzystania z pseudokolumn `sequence.nextval` i `sequence.currval`. Dzięki temu można zarządzać wartościami sztucznych kluczy głównych i zewnętrznych w zasięgu transakcji.

Kolumny `IDENTITY` wymagają stosowania klauzuli `RETURNING INTO` w instrukcjach `INSERT`. Pozwala to zapisać w zmiennej lokalnej ostatnią wartość sekwencji z instrukcji `INSERT`. Następnie można ponownie wykorzystać tę zmienną i przypisać ją jako klucz obcy do zależnej tabeli. Oczywiście oparte jest to na założeniu, że wstawianie wartości do tabel odbywa się w jednostce transakcyjnej w bloku języka PL/SQL.

Większa pojemność typów łańcuchowych i typu RAW

Maksymalną pojemność typów `VARCHAR2`, `NVARCHAR2` i `RAW` można teraz skonfigurować w SQL-u. Aby pozostawić limit 4000 bajtów, ustaw parametr `max_string_size` na wartość `STANDARD`. Inna możliwość to ustawienie tego parametru na wartość `EXTENDED`. Wtedy maksymalna pojemność tych typów to 32 767 bajtów.

Zaleta większego limitu pojemności tych typów powinna być oczywista dla programistów używających wcześniej bazy Oracle Database 11*g*. W owej wersji w języku PL/SQL typy `VARCHAR2`, `NVARCHAR2` i `RAW` miały pojemność 32 767 bajtów, jednak tak dużych wartości nie można było zapisać w kolumnach tych samych typów. Obecnie jest to możliwe.

Przekazywanie wartości z instrukcji SQL-a do zewnętrznych programów

W wersjach starszych niż Oracle Database 12*c* wyniki instrukcji `SELECT` trzeba było zapisywać za pomocą typu danych języka SQL lub PL/SQL. To wymagało dodatkowych kroków przy zagnieżdżaniu

zapytań w programach w języku PL/SQL. Aby w programach zewnętrznych uzyskać dostęp do wyników, trzeba było stosować pasujące skalarne lub złożone typy danych. Typami złożonymi były zwykle tabele SQL-a, systemowe kursory referencyjne języka PL/SQL lub zbiory wyników z potokowych funkcji SQL-a.

W Oracle Database 12*c* dostępna jest nowa procedura `return_result` z pakietu `dbms_sql`. Z tego punktu dowiesz się, jak korzystać z tej procedury i z tego pakietu.

Nowy mechanizm działa podobnie jak technologia Shared Source Common Language Infrastructure (CLI) Microsoftu. Według artykułu „The Microsoft Shared Source CLI Implementation" Davida Stutza (ten tekst został opublikowany w marcu 2002 roku w sieci MSDN) „Microsoft opracował technologię Shared Source CLI, aby naukowcy, studenci, profesorowie i inni zainteresowani programiści mogli poznawać zaawansowaną infrastrukturę języka komputerowego, uczyć jej innych i eksperymentować z nią". Z tego samego artykułu wynika, że Microsoft udostępnia licencję na implementację technologii Shared Source CLI każdemu, kto obieca, że będzie modyfikował kod CLI tylko na potrzeby użytku niekomercyjnego. Jednak w 2009 roku Microsoft dodał język C# i technologię CLI do listy specyfikacji objętych programem Community Promise. To powinno oznaczać (choć przyznaję, że nie jestem prawnikiem), iż każdy może bezpiecznie implementować te technologie bez obaw o pozew patentowy ze strony Microsoftu.

W Wikipedii, pod poniższym adresem, znajduje się ciekawy artykuł na temat CLI:

http://en.wikipedia.org/wiki/Common_Language_Infrastructure

Ponieważ w dokumentacji baz Oracle nie ma żadnych informacji o kwestiach licencyjnych, wygląda na to, że firma Oracle musi ufać zapewnieniom Microsoftu z programu Community Promise lub w inny sposób rozwiązała kwestie korzystania z CLI. W CLI parametry funkcji podaje się tak:

```
CREATE FUNCTION mydb.getConquistador
(@nationality AS VARCHAR(30))
RETURNS TABLE
RETURN SELECT * FROM mydb.conquistador WHERE nationality = @nationality;
```

Funkcja zgodna z Shared Source CLI przekazuje referencję do zbioru wyników jako wartość zwracaną funkcji. W bazach Oracle stosuje się inne podejście. Tu używane są procedury `get_next_result` i `return_result` z pakietu `dbms_sql`. Przyjmują one parametry przekazywane przez referencję. Specyfikacje tych procedur znajdziesz w tabeli 2.1.

Tabela 2.1. *Procedury z pakietu dbms_sql przekazujące niejawnie zbiory wyników*

Procedura	Opis

`get_next_result`	Procedura `get_next_result` przyjmuje dwa parametry. Pierwszy to parametr typu `IN` przekazywany przez wartość. Jest nim referencja do kursora `dbms_sql`. Drugi to przeciążony parametr typu `OUT` przekazywany przez referencję. Przyjmuje albo jeden systemowy kursor referencyjny języka PL/SQL, albo referencję do takiego kursora. Programista nie może bezpośrednio używać parametru `rc` w trybie `OUT`. Oto prototypy tej procedury:
	`GET_NEXT_RESULT( c IN INTEGER, rc OUT SYS_REFCURSOR)`
	`GET_NEXT_RESULT( c IN INTEGER, rc OUT INTEGER)`

`return_result`	Procedura `return_result` przyjmuje dwa parametry. Pierwszy to przeciążony parametr typu `IN OUT` przekazywany przez referencję. Parametr `rc` to albo jeden systemowy kursor referencyjny języka PL/SQL, kolekcja takich kursorów, referencja do takiego kursora, albo kolekcja referencji do takich kursorów. Drugi to przekazywany przez wartość parametr logiczny o wartości domyślnej `TRUE`. Oto prototypy tej procedury:
	`RETURN_RESULT( rc IN OUT SYS_REFCURSOR,`
	`to_client IN BOOLEAN [DEFAULT TRUE] )`
	`RETURN_RESULT( rc IN OUT INTEGER,`
	`to_client IN BOOLEAN [DEFAULT TRUE] )`

Poniżej przedstawiony jest program w postaci bloku anonimowego, ilustrujący zwracanie wyniku z kursora niejawnego:

```
SQL> COLUMN item_title    FORMAT A30
SQL> COLUMN item_subtitle FORMAT A40
SQL> DECLARE
  2    /* Deklarowanie kursora. */
  3    lv_cursor SYS_REFCURSOR;
  4  BEGIN
  5    /* Otwieranie statycznego kursora. */
  6    OPEN lv_cursor FOR
  7      SELECT i.item_title
  8      , i.item_subtitle
  9      FROM item i
 10      WHERE REGEXP_LIKE(i.item_title,'^Star.*');
 11
 12    /* Wywołanie procedury dbms_sql.return_result. */
 13    dbms_sql.return_result(lv_cursor);
 14  END;
 15  /
```

W wierszu 3. zadeklarowany jest systemowy kursor referencyjny języka PL/SQL. W wierszach od 6. do 10. tworzone jest statyczne zapytanie powiązane z tym kursorem W wierszu 13. ten lokalny kursor jest zwracany do klienta.

Przedstawiony blok anonimowy wyświetla poniższe informacje, ponieważ wyniki z kursora są niejawnie przekazywane przez referencję z powrotem do zasięgu wywołania:

```
ITEM_TITLE                     ITEM_SUBTITLE
------------------------------ ----------------------
Star Wars I                    Phantom Menace
Star Wars II                   Attack of the Clones
Star Wars III                  Revenge of the Sith
```

Gdy anonimowy blok obejmuje dwa systemowe kursory referencyjne (lub większą ich liczbę) i dwa wywołania procedury `return_result` z pakietu `dbms_sql` (lub większą ich liczbę), zwracane są przynajmniej dwa zbiory wyników. Procedura `get_next_result` zwraca jeden zbiór wyników.

Pojawiły się też nowe funkcje bibliotek zewnętrznych przeznaczone do pracy z niejawnymi zbiorami wyników. Na przykład w bibliotece OCI8 2.0 pojawiła się funkcja `oci_get_implicit_resultset()`. Można z niej korzystać razem z wszystkimi funkcjami z rodziny `oci_fetch_*`.

Dzięki temu pojawiają się ciekawe alternatywy do stosowania systemowych kursorów referencyjnych oraz potokowych i obiektowych funkcji tabelowych. Od wersji Oracle Database 10*g* można tworzyć obiektowe funkcje tabelowe oraz używać funkcji `TABLE` do zwracania kolekcji elementów typów skalarnych i złożonych z baz Oracle jako relacyjnych zbiorów wyników.

Określanie liczby wyników i zwracanych przedziałów wyników w natywnym SQL-u

W wersjach starszych od Oracle Database 12*c* można było tylko ograniczyć liczbę zwracanych wierszy za pomocą operacji `ROWNUM`. Nowe klauzule `FETCH FIRST` i `OFFSET` dają większe możliwości. W Oracle Database 12*c* dostępny jest rozbudowany zbiór opcji do pobierania pierwszych n wierszy.

Aby ograniczyć liczbę wyników zapytania do jednego wiersza, zastosuj następujący kod:

```
SQL> SELECT  i.item_title
  2 FROM     item i
  3 FETCH FIRST 1 ROWS ONLY;
```

Wiersz 3. pokazuje, jak za pomocą klauzuli `FETCH FIRST` pobrać tylko jeden wiersz. Najzabawniejsze jest to, że trzeba tu zastosować słowa kluczowe w liczbie mnogiej — `ROWS ONLY` (czyli tylko wiersze).

Łatwo można się domyślić, że w celu zwrócenia pierwszych pięciu wierszy należy zmodyfikować wiersz 3. w następujący sposób:

```
3 FETCH FIRST 5 ROWS ONLY;
```

Załóżmy, że nie wiesz, ile wierszy zwróci zapytanie, a nie chcesz ograniczać liczby zwracanych wyników do 20, 50, 100 lub 500 (są to najczęściej stosowane wartości). Oracle udostępnia składnię, która

pozwala pobrać określoną część wszystkich wierszy. W tym celu należy dodać do klauzuli FETCH FIRST słowo kluczowe PERCENT, tak jak w poniższej wersji wiersza 3.:

```
3 FETCH FIRST 20 PERCENT ROWS ONLY;
```

Baza Oracle Database 12*c* umożliwia też pominięcie zbioru wierszy przed wczytaniem ich określonej liczby. W ten sposób pobieranych jest n wierszy od określonego miejsca zbioru wyników. Potrzebną składnię przedstawia zmodyfikowana wersja wiersza 3.:

```
3 OFFSET 20 ROWS FETCH FIRST 20 ROWS ONLY;
```

Minimalna wartość klauzuli OFFSET to 0. Warto o tym pamiętać przy podawaniu parametrów dla takich zapytań.

Do zastosowania parametrów w takiej instrukcji można wykorzystać zmienne wiązane:

```
3 OFFSET :bv_offset ROWS FETCH FIRST :bv_rows ROWS ONLY;
```

Jeśli chcesz zastosować parametry w zapytaniu pobierającym określoną liczbę wierszy, zawsze powinieneś używać klauzuli OFFSET, ponieważ pozwala napisać jedną instrukcję mającą dwie funkcje. Jedna polega na odczycie danych od początku zbioru rekordów (wtedy klauzula OFFSET ma wartość 0). Druga pozwala wczytać dane od dowolnego miejsca innego niż początek zbioru. Dane są wczytywane do końca tylko wtedy, gdy wartość :bv_rows jest większa od liczby pozostałych rekordów.

Klauzule FETCH FIRST i OFFSET są też dostępne w blokach PL/SQL. Te klauzule można stosować w instrukcjach SELECT INTO i jako definicje kursorów statycznych. W anonimowych blokach PL/SQL można też korzystać ze zmiennych wiązanych.

Poniższy fragment pokazuje, jak zastosować zapytanie SELECT INTO.

```
SQL> DECLARE
  2    /* Deklarowanie zmiennej lokalnej. */
  3    lv_item_title VARCHAR2(60);
  4  BEGIN
  5    /* Pobieranie danych do zmiennej lokalnej. */
  6    SELECT i.item_title
  7    INTO lv_item_title
  8    FROM item i
  9    FETCH FIRST 1 ROWS ONLY;
 10    dbms_output.put_line('['||lv_item_title||']');
 11  END;
 12  /
```

Wiersz 9. pobiera tylko pierwszy wiersz z zapytania. W wierszu 9. można też zastosować klauzulę OFFSET:

```
9    OFFSET 5 ROWS FETCH FIRST 1 ROWS ONLY;
```

Jak wspomniano, w anonimowych blokach PL/SQL można stosować zmienne wiązane. Jeśli wartość zmiennej :bv_size to 1, można wywołać następujący kod:

```
9 OFFSET :bv_offset ROWS FETCH FIRST :bv_size ROWS ONLY;
```

Zmienna :bv_size musi mieć wartość 1, ponieważ instrukcja SELECT INTO może zwrócić tylko jeden wiersz. Dlatego przy większej wartości wystąpi wyjątek ORA-01422, informujący, że zapytanie zwróciło zbyt wiele wierszy.

Za pomocą dynamicznego zapytania umieszczonego w zewnętrznym programie możesz wyeliminować ryzyko zwrócenia zbyt wielu wierszy. W tym celu należy wykorzystać biblioteki ODBC lub JDBC.

Poniższy fragment ilustruje technikę dynamicznego pobierania n wierszy w języku PHP:

```
15  // Deklaracja instrukcji SQL-a.
16  $sql = "SELECT   i.item_title "
17       . "FROM     item i "
18       . "OFFSET :bv_offset ROWS FETCH FIRST :bv_rows ROWS ONLY";
19
20  // Przygotowanie instrukcji i powiązanie dwóch łańcuchów znaków.
21  $stmt = oci_parse($c,$sql);
22
23  // Powiązanie zmiennych lokalnych z instrukcją języka PHP.
```

```
24  oci_bind_by_name($stmt, ":bv_offset", $offset);
25  oci_bind_by_name($stmt, ":bv_rows", $rows);
26
27  // Wykonanie instrukcji języka PL/SQL.
28  if (oci_execute($stmt)) {
```

Następny przykład pokazuje, jak pobrać n wierszy ze środka zbioru w statycznym kursorze:

```
SQL> DECLARE
  2  /* Deklaracja zmiennej lokalnej. */
  3  lv_item_title VARCHAR2(60);
  4  /* Deklaracja statycznego kursora. */
  5  CURSOR c IS
  6    SELECT i.item_title
  7    FROM item i
  8    OFFSET 10 ROWS FETCH FIRST 1 ROWS ONLY;
  9  BEGIN
 10    /* Otwieranie, pobieranie, wyświetlanie i zamykanie kursora. */
 11    OPEN c;
 12    FETCH c INTO lv_item_title;
 13    dbms_output.put_line('['||lv_item_title||']');
 14    CLOSE c;
 15  END;
 16  /
```

W wierszu 8. użyty jest literał do ustawienia wartości klauzuli `OFFSET` i liczby zwracanych wierszy. Nie można podstawiać zmiennych za literały (a przynajmniej nie w produkcyjnej wersji bazy Oracle Database 12*c* Release 1).

Oto próba użycia kursora dynamicznego:

```
SQL> DECLARE
  2    /* Deklaracja zmiennej lokalnej. */
  3    lv_item_title VARCHAR2(60);
  4    /* Deklaracja statycznego kursora. */
  5    CURSOR c
  6    ( cv_offset NUMBER
  7    , cv_size NUMBER ) IS
  8      SELECT i.item_title
  9      FROM item i
 10      OFFSET cv_offset ROWS FETCH FIRST cv_size ROWS ONLY;
 11  BEGIN
 12    NULL;
 13  END;
 14  /
```

W wierszu 10. ustawiane są ograniczenia zapytania zwracającego wybraną liczbę wierszy. Używane są do tego parametry kursora: `cv_offset` i `cv_size`. Wiersz 12. zapobiega błędowi parsowania. W bloku wykonania musi znajdować się instrukcja; w przeciwnym razie parsowanie zakończy się niepowodzeniem. Ten blok jednak i tak powoduje błąd parsowania. Występuje wyjątek i odłączenie programu od aktywnej sesji. Oto wyświetlany stos błędu:

```
ERROR:
ORA-03114: not connected to ORACLE

DECLARE
*
ERROR at line 1:
ORA-03113: end-of-file on communication channel
Process ID: 4148
Session ID: 31 Serial number: 3187
```

Jest to nieprzechwycony wyjątek. Takie wyjątki zwykle są szybko eliminowane. Z powodu takiego, a nie innego typu błędu podejrzewam, że Oracle w przyszłości rozwiąże ten problem. Możliwe też, że ten błąd zostanie udokumentowany jako ograniczenie. Do czasu opublikowania tej książki kwestia powinna zostać rozwiązana.

Sterownik bazy Oracle dla aplikacji dla baz MySQL

Baza Oracle Database 12*c* udostępnia sterownik dla aplikacji dla baz MySQL. Zastępuje on bibliotekę kliencką MySQL 5.5. Dzięki niemu można korzystać z aplikacji i narzędzi opartych na językach, które wykorzystują interfejs API w języku C bazy MySQL. Są to na przykład języki: PHP, Ruby, Perl i Python. Zaletą tego podejścia jest to, że użytkownicy mogą korzystać z aplikacji dla baz MySQL zarówno w takich bazach, jak i w bazach Oracle. Zwiększa to przenośność rozwiązań opartych na wymienionych językach skryptowych.

Instrukcje CROSS APPLY, OUTER APPLY i LATERAL w SQL-u

Instrukcja `APPLY` SQL-a umożliwia wywoływanie funkcji tabelowych (zwracających kolekcje, których można używać jak tabel) dla każdego wiersza zwróconego przez zewnętrzne wyrażenie tabelowe zapytania. Złączenie traktuje wtedy funkcję tabelową jako prawy operand, a zewnętrzne wyrażenie tabelowe jako lewy operand. Złączenie analizuje każdy wiersz z prawej strony dla każdego wiersza z lewej strony i łączy wyniki w ostateczny zbiór wyników.

Istnieją dwie odmiany tej operacji. Instrukcja `CROSS APPLY` to wersja złączenia wewnętrznego. Zwraca wiersze z tabeli (lub ze zbioru tabel) podanej po lewej stronie instrukcji `CROSS APPLY` pasujące do klauzuli `WHERE` zapisanej wewnątrzwierszowo po stronie prawej.

Poniżej pokazano przykładowy kod ze złączeniem `CROSS APPLY`:

```
SQL> SELECT i.item_title
  2  FROM   item i CROSS APPLY
  3                (SELECT *
  4                 FROM rental_item ri
  5                 WHERE i.item_id = ri.item_id
  6                 OFFSET 0 ROWS FETCH FIRST 1 ROWS ONLY);
```

Instrukcja `OUTER APPLY` to odmiana złączenia lewostronnego. Tworzy ona złączenie zewnętrzne między tabelą lub zbiorem złączanych tabel a zapisanym wewnątrzwierszowo widokiem. Ten widok musi obejmować klauzulę `WHERE`, która określa relację między zbiorem wyników z lewej strony a podanym po prawej widokiem. Zwracane są wszystkie wiersze z tabeli podanej po lewej stronie złączenia i pasujące wyniki z kolekcji lub wartości `null`.

Oto przykładowy kod, w którym zastosowano złączenie `OUTER APPLY`:

```
SQL> SELECT i.item_title
  2  FROM   item i OUTER APPLY
  3                (SELECT *
  4                 FROM rental_item ri
  5                 WHERE i.item_id = ri.item_id
  6                 OFFSET 0 ROWS FETCH FIRST 1 ROWS ONLY);
```

Klauzula `LATERAL` służy do podawania podzapytań w lateralnych widokach wewnątrzwierszowych. W klauzuli `FROM` zapytania można podać tabele, z którymi korelowany jest dany lateralny widok wewnątrzwierszowy. Przy stosowaniu takich widoków obowiązują pewne ograniczenia:

- Nie można stosować klauzul `PIVOT` i `UNPIVOT` ani klauzul z referencjami tabel.
- Nie można stosować lewostronnej korelacji, gdy lateralny widok wewnątrzwierszowy zawiera klauzulę `PARTITION` w zapytaniu i pojawia się po prawej stronie złączenia.
- W widoku lateralnym nie można stosować lewostronnej korelacji dla pierwszej tabeli ze złączenia prawostronnego lub pełnego złączenia zewnętrznego.

Klauzula `LATERAL` jest częścią standardu ANSI SQL i stanowi rozszerzenie składni widoków wewnątrzwierszowych baz Oracle. Choć poniższe zapytanie można łatwo przekształcić na złączenie `INNER JOIN`, ta wersja pozwala zademonstrować ograniczenia wyeliminowane dzięki klauzuli `LATERAL` z bazy Oracle Database 12*c*.

```
SQL> SELECT *
  2  FROM   contact c CROSS JOIN
  3          (SELECT *
```

```
  4           FROM   member m
  5           WHERE  c.member_id = m.member_id);
```

To zapytanie próbuje zapisać widok wewnątrzwierszowy obejmujący skorelowane podzapytanie. W efekcie generowany jest następujący komunikat o błędzie:

```
    WHERE c.member_id = m.member_id)
          *
ERROR at line 5:
ORA-00904: "C"."MEMBER_ID": invalid identifier
```

Ten błąd oznacza, że nie można znaleźć aliasu c tabeli `contact`. Widok wewnątrzwierszowy nie potrafi znaleźć tego aliasu, ponieważ staje się on dostępny dopiero po zakończeniu parsowania klauzuli `FROM`. To dlatego Oracle zgłasza błąd nieprawidłowego identyfikatora (szczegółowe omówienie identyfikatorów znajdziesz w rozdziale 4.). Błąd tego samego rodzaju występuje w złączeniach `CROSS APPLY` i `OUTER APPLY`.

Klauzula `LATERAL` umożliwia korzystanie w widoku wewnątrzwierszowym z tabel podanych po lewej stronie instrukcji `CROSS JOIN`. Jest tak, ponieważ cały kod do słowa kluczowego `LATERAL` jest parsowany osobno. Rozdzielenie parsowania na dwa etapy pozwala widokowi wewnątrzwierszowemu umieszczonemu z prawej strony słowa kluczowego `LATERAL` na poprawne zinterpretowanie identyfikatora. To oznacza, że teraz w widokach wewnątrzwierszowych można stosować korelacje:

```
SQL> SELECT *
  2  FROM contact c CROSS JOIN
  3        LATERAL(SELECT  *
  4                FROM    member m
  5                WHERE   c.member_id = m.member_id);
```

Słowo kluczowe `LATERAL` z wiersza 3. pozwala podzapytaniu znaleźć tabele podane po lewej stronie instrukcji `CROSS JOIN`. Operacje są tu wykonywane od lewej do prawej, dlatego ta technika nie zadziała, jeśli niezrozumiały identyfikator znajduje się po prawej stronie.

Widoki z ustawieniem BEQUEATH CURRENT_USER

W wersjach starszych niż Oracle Database 12*c* widoki zawsze działały z uprawnieniami jednostki definiującej. Takie uprawnienia są stosowane domyślnie dla funkcji, procedur, pakietów i typów. Przy definiowaniu jednostek programów składowanych można zastosować klauzulę `AUTHID DEFINER`, przy czym nie jest to konieczne, ponieważ to ustawienie jest domyślne.

W Oracle Database 12*c* pojawiła się możliwość ustawiania uprawnień widoków. Domyślne ustawienie to `BEQUEATH DEFINER`, które działa jak opcja `AUTHID DEFINER` stosowana do jednostek programów składowanych. Aby zmienić domyślne uprawnienia, należy zastosować ustawienie `BEQUEATH CURRENT_USER`.

Powtórzenie materiału

W tym podrozdziale przedstawiono następujące informacje na temat nowych funkcji SQL-a dostępnych w bazie Oracle Database 12*c*:

- W Oracle Database 12*c* oprócz wskazywania zmiennych środowiskowych w instrukcjach `LIBRARY` można też stosować katalogi wirtualne.
- Oracle Database 12*c* umożliwia bezpośrednie i pośrednie definiowanie wymiarów VT, co zwiększa możliwości administratorów w zakresie analiz retrospektywnych.
- W Oracle Database 12*c* wzbogacono możliwości złączeń `LEFT OUTER JOIN`. Obecnie mogą one obejmować wiele tabel po lewej stronie.
- W Oracle Database 12*c* wprowadzono instrukcje `CROSS APPLY`, `OUTER APPLY` i `LATERAL`, przeznaczone do pracy z zagnieżdżonymi tabelami.
- Oracle Database 12*c* udostępnia domyślne kolumny przechowujące wartości pseudokolumn `.nextval` i `.currval` dla nazwanych sekwencji.

- W Oracle Database 12*c* wprowadzono kolumny `IDENTITY`, zawierające automatycznie zwiększane sekwencje liczb. Takie kolumny pełnią funkcję klucza sztucznego.
- W Oracle Database 12*c* do wartości domyślnych można dodać klauzulę `ON NULL`, co uniemożliwia bezpośrednie ustawianie wartości `null` przy wstawianiu lub aktualizowaniu wierszy tabeli.
- Oracle Database 12*c* pozwala ustawić parametr w celu zwiększenia pojemności typów `VARCHAR2`, `NVARCHAR2` i `RAW` do 32 767 bajtów, co odpowiada ich pojemności w języku PL/SQL.
- Oracle Database 12*c* udostępnia słowo kluczowe `BEQUEATH`, służące do ustawiania dla widoków uprawnień jednostki definiującej lub wywołującej.

Nowe funkcje języka PL/SQL

W Oracle Database 12*c* do języka PL/SQL dodano liczne nowe funkcje:

- Można zapisywać w pamięci podręcznej wyniki funkcji z uprawnieniami jednostki wywołującej.
- Ważne usprawnienie pozwala tworzyć białe listy jednostek wywołujących dla funkcji składowanych, procedur, pakietów i typów.
- Dostępna jest natywna obsługa wiązania pakietów PL/SQL i typów logicznych jako parametrów. Pojawiła się też natywna obsługa klienckich interfejsów API w zakresie typów danych języka PL/SQL.
- Dodany został pakiet `utl_call_stack`.
- W pakiecie `dbms_utility` znalazła się nowa procedura `expand_sql_text`.
- Procedura `parse` ma nowy formalny schemat określania niepełnych nazw obiektów.
- W klauzuli `WITH` SQL-a można stosować funkcje języka PL/SQL.
- Można definiować lokalne typy języka PL/SQL i stosować je w zagnieżdżonych instrukcjach SQL-a.
- Dostawca ODP.NET (ang. *Oracle Data Provider for .NET*) może teraz wiązać parametry typu `REF CURSOR` z procedurami składowanymi.

W dalszych punktach znajdziesz omówienie wszystkich tych nowych funkcji języka PL/SQL.

Zapisywanie wyników funkcji z uprawnieniami jednostki wywołującej

Oracle Database 12*c* umożliwia zapisanie w pamięci podręcznej wyników funkcji z uprawnieniami jednostki wywołującej. Do zapisywanych wyników dołączana jest tożsamość bieżącego użytkownika. To pozwala zapisać różne wyniki dla jednego programu z uprawnieniami jednostki wywołującej. To oznacza, że można zapisywać w pamięci podręcznej wyniki deterministycznych funkcji z uprawnieniami jednostki wywołującej, czyli funkcji korzystających z wartości `CURRENT_USER`.

Wprowadzenie funkcji z uprawnieniami jednostki wywołującej zmienia sposób rozwiązywania problemów. W środowisku rozproszonym (na przykład z bazami PDB) takie funkcje pozwalają uzyskać wyższą przepustowość.

Tworzenie białych list jednostek wywołujących w programach języka PL/SQL

Oracle Database 12*c* umożliwia przechowywanie białych list użytkowników, którzy mogą wywoływać funkcje, procedury, pakiety i typy obiektowe. Umieszczenie użytkownika na białej liście umożliwia mu wywoływanie procedur składowanych. Jest to uzupełnienie innych zabezpieczeń. Użytkownik, który ma uprawnienia do wykonywania procedury składowanej ze schematu, musi też znajdować się na liście uprawnionych użytkowników.

W dokumentacji bazy Oracle Database 12*c* przedstawiono nowy sposób opisywania procedur składowanych. Ogólne określenie `rodzaj_jednostki` oznacza tu funkcje, procedury, pakiety i typy obiektowe. Klauzula `ACCESSIBLE BY` służy do określania białych list dla tworzonych lub zastępowanych programów.

W dokumentacji bazy Oracle przedstawiony jest następujący prototyp:

```
[ACCESSIBLE BY (rodzaj_jednostki [schema.]nazwa_jednostki
[,              rodzaj_jednostki [schema.]nazwa_jednostki]
[,... ]])]
```

Jest to bezpośredni i krótki zapis, jednak bardziej zrozumiały może okazać się rozszerzony prototyp, ponieważ słowo kluczowe zastępujące określenie `rodzaj_jednostki` poprzedza w nim nazwę programu składowanego:

```
[ACCESSIBLE BY
( [{FUNCTION | PROCEDURE | PACKAGE | TYPE}] [schema.]nazwa_jednostki)
[,[{FUNCTION | PROCEDURE | PACKAGE | TYPE}] [schema.]nazwa_jednostki)]
[,... ]]]
```

Poniższy krótki przykład ilustruje, jak napisać funkcję dodającą jednostki do białej listy. Ta funkcja umieszcza na białej liście funkcję, procedurę, pakiet i typ, co pozwala przedstawić kompletny opis tego mechanizmu.

```
SQL> CREATE OR REPLACE FUNCTION library
  2  ( pv_message VARCHAR2 ) RETURN VARCHAR2
  3  ACCESSIBLE BY
  4  ( FUNCTION  video.gateway
  5  , PROCEDURE video.backdoor
  6  , PACKAGE   video.api
  7  , TYPE      video.hobbit ) IS
  8    lv_message VARCHAR2(20) := 'Witaj, ';
  9  BEGIN
 10    lv_message := lv_message || pv_message || '!';
 11    RETURN lv_message;
 12  END;
 13  /
```

W wierszach od 3. do 7. zadeklarowana jest lista uprawnionych jednostek wywołujących. Każda z nich może z powodzeniem wywołać funkcję biblioteczną, natomiast nie mogą tego zrobić żadne inne funkcje, procedury, pakiety ani typy. Próba utworzenia nowej funkcji wywołującej funkcję z ustawioną białą listą, na przykład:

```
SQL> CREATE OR REPLACE FUNCTION black_knight
  2  ( pv_message VARCHAR2 ) RETURN VARCHAR2 IS
  3  BEGIN
  4    RETURN library(pv_message);
  5  END;
  6  /
```

spowoduje błąd kompilacji. Można wtedy wyświetlić informacje o błędzie:

```
SQL> show errors
Errors for FUNCTION BLACK_KNIGHT:
LINE/COL ERROR
-------- -----------------------------------------------------------
4/3      PL/SQL: Statement ignored
4/10     PLS-00904: insufficient privilege to access object LIBRARY
```

Tworzenie białych list jednostek wywołujących to przydatne i od dawna oczekiwane usprawnienie, niedostępne na razie w żadnej innej bazie danych.

Natywna obsługa klienckich interfejsów API w zakresie typów języka PL/SQL

Ta funkcja umożliwia klienckim interfejsom API baz Oracle opisywanie i wiązanie typów z pakietów języka PL/SQL i typów logicznych. Używane są do tego interfejsy API OCI i JDBC. Do wiązania i wykonywania funkcji oraz procedur języka PL/SQL można też używać aplikacji opartych na języku C.

Nowy pakiet utl_call_stack języka PL/SQL

W Oracle Database 12*c* pojawił się pakiet `utl_call_stack`. Udostępnia on liczne funkcje usprawniające obsługę stosu błędów. *Stos błędów* to sekwencja zgłoszonych wyjątków przekazanych w górę łańcucha wywołań. Zawartość tego pakietu i sposób korzystania z niego opisano w rozdziale 6.

Nowa procedura expand_sql_text w pakiecie dbms_utility

W Oracle Database 12*c* w pakiecie `dbms_utility` dostępna jest nowa procedura `expand_sql_text`. Ta procedura umożliwia przekształcenie widoku zależnego od innych widoków w jedno zapytanie. Jest to bardzo przydatne, gdy chcesz zobaczyć kompletny obraz działania kodu.

Procedurę `expand_sql_text` można wykorzystać do ustalenia, jak widoki oparte na innych widokach są powiązane z tabelami. Jest to najprostsze rozwiązanie oprócz ręcznego refaktoryzowania kodu widok po widoku. Problem z procedurą `expand_sql_text` w Oracle polega na tym, że przyjmuje ona obiekt `CLOB` i zwraca taki obiekt, natomiast widoki są przechowywane w kolumnach typu `LONG`. Przekształcenie typu `LONG` na `CLOB` nie jest łatwe. Dlatego napisałem funkcję, która zrobi to za Ciebie. Tę funkcję, `long_to_clob`, znajdziesz w rozdziale 10.

Nawet z funkcją `long_to_clob` używanie procedury `expand_sql_text` wymaga kilku kroków, co ilustruje poniższa funkcja:

```
SQL> CREATE OR REPLACE FUNCTION expand_view
  2  ( pv_view_name     VARCHAR2 ) RETURN CLOB IS
  3
  4    /* Deklaracje kontenerów na widoki. */
  5    lv_input_view   CLOB;
  6    lv_output_view  CLOB;
  7
  8    /* Deklaracja docelowej zmiennej (z powodu ograniczeń instrukcji SELECT INTO). */
  9    lv_long_view     LONG;
 10
 11    /* Deklaracja dynamicznego kursora. */
 12    CURSOR c (cv_view_name VARCHAR2) IS
 13      SELECT   text
 14      FROM     user_views
 15      WHERE    view_name = cv_view_name;
 16
 17  BEGIN
 18    /* Otwieranie, pobieranie i zamykanie kursora w celu pobrania tekstu widoku. */
 19    OPEN c(pv_view_name);
 20    FETCH c INTO lv_long_view;
 21    CLOSE c;
 22
 23    /* Przekształcanie typu LONG na CLOB. */
 24    lv_input_view := long_to_clob(pv_view_name, LENGTH(lv_long_view));
 25
 26    /* Przesyłanie tekstu widoku i pobieranie kompletnego tekstu. */
 27    dbms_utility.expand_sql_text(lv_input_view, lv_output_view);
 28
 29    /* Zwracanie wyjściowej wartości CLOB. */
 30    RETURN lv_output_view;
 31  END;
 32  /
```

Choć nie lubię stosować typu danych `LONG` (jest to prawdziwy „dinozaur” w bazie Oracle), jest on potrzebny do zilustrowania omawianej tu ciekawej funkcji. W wierszu 9. znajduje się deklaracja zmiennej `lv_long_view` typu `LONG`. Choć używanie kursora z parametrami jest tu pewną przesadą, warto konsekwentnie stosować się do dobrych praktyk. Nie można wykorzystać tu instrukcji `SELECT INTO`, ponieważ nie współdziała ona z typem `LONG`. Klauzula `FETCH INTO` pozwala na zastosowanie typu `LONG`, dlatego wykorzystano ją w przypisaniu w wierszu 20.

Dalej znajduje się wywołanie funkcji long_to_clob z parametrem pv_view_name i długością kolumny tekstowej widoku. Ten kod zgłasza dwa zapytania do katalogu (ponieważ funkcja long_to_clob ponawia takie zapytanie), jest to jednak konieczne, aby uniknąć przepisywania danych znak po znaku z typu LONG do typu CLOB. Oracle nie udostępnia wielu możliwości w zakresie pracy z typem LONG. Na przykład funkcja to_clob nie przyjmuje parametrów typu LONG.

Pełne omówienie funkcji long_to_clob znajdziesz w rozdziale 10. Tu wystarczy zauważyć, że wykorzystuje ona pakiety dbms_sql i dbms_lob do przekształcenia danych typu LONG na typ CLOB. Więcej informacji o pakiecie dbms_sql zawiera rozdział 13. W rozdziale 10. opisano pakiet dbms_lob i pracę z dużymi obiektami.

Wiersz 27. to wywołanie procedury expand_sql_text, a w wierszu 30. zwracany jest wyjściowy obiekt CLOB z procedury expand_sql_text. Ostatecznie funkcja zwraca obiekt CLOB z kompletnym zapytaniem opartym na tabelach. Po uzyskaniu tego zapytania można przeanalizować jego wydajność.

Nowy formalny schemat procedury parse w pakiecie dbms_sql

Pakiet dbms_sql obejmuje nowy formalny schemat procedury parse. Ta procedura obecnie interpretuje niepełne nazwy obiektów. Dzięki temu jednostka programu z uprawnieniami jednostki definiującej może sterować określaniem nazw w uruchamianych dynamicznych instrukcjach SQL-a. Za pomocą procedury dbms_sql.parse możesz na przykład wywołać polecenie DROP TABLE w procedurze składowanej.

Funkcje języka PL/SQL w klauzuli WITH SQL-a

W Oracle Database 12*c* można wywoływać funkcje języka PL/SQL w klauzuli WITH. Jedyny problem związany jest z uruchamianiem takich funkcji, ponieważ obejmują średniki. Załóżmy, że chcesz wywołać instrukcję w środowisku SQL*Plus. Najpierw trzeba wyłączyć domyślny symbol kończący SQL-a (średnik — ;). Służy do tego następujące polecenie tego środowiska:

```
SET SQLTERMINATOR OFF
```

Następnie możesz utworzyć funkcję lokalną w instrukcji WITH:

```
SQL> COLUMN person FORMAT A18
SQL> WITH
  2    FUNCTION glue
  3    ( pv_first_name VARCHAR2
  4    , pv_last_name  VARCHAR2) RETURN VARCHAR2 IS
  5      lv_full_name  VARCHAR2(100);
  6    BEGIN
  7      lv_full_name := pv_first_name || ' ' || pv_last_name;
  8      RETURN lv_full_name;
  9    END;
 10    SELECT glue(a.first_name,a.last_name) AS person
 11    FROM  actor a
 12  /
```

Funkcja z wierszy od 2. do 9. złącza dwa łańcuchy znaków i wstawia między nie jedną spację. Średniki są traktowane w zapytaniu jak zwykłe znaki, ponieważ wyłączono domyślny symbol kończący SQL-a. Warto też zauważyć, że w środowisku SQL*Plus instrukcje SQL-a są uruchamiane za pomocą ukośnika i że kompletna instrukcja nie ma kończącego średnika w wierszu 11.

W tym prostym przykładzie tabela actor zawiera nazwiska dwóch aktorów (z filmu *Iron Man*), dlatego zapytanie zwraca następujące dane:

```
PERSON
------------------
Robert Downey
Gwyneth Paltrow
```

Gdy otworzysz takie zapytanie w narzędziu Oracle SQL Developer lub podobnym, mogą wystąpić pewne problemy z parsowaniem przy uruchamianiu zapytania. Najłatwiejsze rozwiązanie polega na umieszczeniu zapytania w widoku, ponieważ widok eliminuje konieczność zmiany ustawienia SQLTERMINATOR w czasie wykonywania programu. Poniższy kod tworzy widok z funkcją języka PL/SQL zagnieżdżoną w instrukcji WITH:

```
SQL> CREATE OR REPLACE VIEW actor_v AS
  2  WITH
  3    FUNCTION glue
  4    ( pv_first_name VARCHAR2
  5    , pv_last_name VARCHAR2) RETURN VARCHAR2 IS
  6    BEGIN
  7      RETURN pv_first_name || ' ' || pv_last_name;
  8    END;
  9    SELECT glue(a.first_name,a.last_name) AS person
 10    FROM actor a
 11  /
```

Widok to nic innego jak składowane zapytanie. Widok actor_v pozwala skrócić funkcję glue o dwa wiersze. Można pominąć deklarację lv_full_name i zastąpić przypisywanie złączanych wartości do zmiennej instrukcją, która bezpośrednio je zwraca (wiersz 7.).

Jeśli zechcesz uruchomić zwykłe polecenia SQL-a, z domyślnym średnikiem, ponownie włącz domyślny symbol kończący SQL-a:

```
SET SQLTERMINATOR ON
```

Oczywistą zaletą klauzuli WITH jest to, że jest uruchamiana raz, po czym można ją wielokrotnie wykorzystać w zasięgu zapytania. Można też zagnieździć funkcje o zasięgu lokalnym ograniczonym do jednego zapytania. Po co stosować klauzulę WITH, skoro można wykorzystać globalną tabelę tymczasową? Tom Kyte odpowiedział na to pytanie w dziale *Ask Tom* (*http://asktom.oracle.com*), gdzie napisał, że optymalizator potrafi scalić klauzulę WITH z resztą instrukcji, natomiast w przypadku globalnej tabeli tymczasowej jest to niemożliwe.

Stosowanie w SQL-u typów danych specyficznych dla języka PL/SQL

Możliwość przekazywania specyficznych dla baz Oracle typów danych z języka PL/SQL to cenna funkcja bazy Oracle Database 12*c*. Aby to zadziałało, potrzebna jest pewna sztuczka — należy zadeklarować zmienną lokalną w programie składowanym, a następnie wykorzystać tę zmienną w zagnieżdżonej instrukcji SQL-a.

Przyjrzyj się działaniu tej techniki na przykładzie kolekcji języka PL/SQL oraz nazwanego i anonimowego bloku tego języka. Zademonstrowanie tego podejścia wymaga pięciu kroków. W szóstym kroku zobaczysz, w jakiej sytuacji pojawiają się problemy, i jednocześnie zrozumiesz, dlaczego w Oracle Database 12*c* potokowe funkcje tabelowe nadal są potrzebne.

Pierwszy krok to utworzenie pozbawionego ciała pakietu type_defs. Specyfikacja *pakietu pozbawionego ciała* obejmuje tylko definicje typu i kursora. Takie pakiety stosuje się w celu współużytkowania typów i kursorów między różnymi jednostkami programu.

W poniższej specyfikacji pakietu tworzona jest tylko tablica asocjacyjna specyficzna dla języka PL/SQL (jest to kolekcja z rzadkimi indeksami):

```
SQL> CREATE OR REPLACE PACKAGE type_defs IS
  2    TYPE plsql_table IS TABLE OF VARCHAR2(20)
  3      INDEX BY BINARY_INTEGER;
  4  END type_defs;
  5  /
```

W drugim kroku tworzona jest tabela honeymooner, z kolumną identyfikacyjną i kolumną person. Kolumna person zawiera dane typu używanego we wspomnianej wcześniej tablicy asocjacyjnej. Dane z tej tabeli posłużą do zapełnienia tablicy asocjacyjnej z języka PL/SQL.

Oto definicja tej tabeli:

```
SQL> CREATE TABLE honeymooner
  2  ( honeymooner_id NUMBER GENERATED ALWAYS AS IDENTITY
  3  , person         VARCHAR2(20));
```

Trzeci krok wymaga wstawienia do tabeli honeymooner czterech wierszy:

```
SQL> INSERT INTO honeymooner (person) VALUES ('Ralph Kramden');
SQL> INSERT INTO honeymooner (person) VALUES ('Alice Kramden');
SQL> INSERT INTO honeymooner (person) VALUES ('Edward Norton');
SQL> INSERT INTO honeymooner (person) VALUES ('Thelma Norton');
```

Trzy pierwsze kroki tworzą pozbawiony ciała pakiet plsql_table i tabelę honeymooner oraz zapełniają tę tabelę czterema wierszami. Czwarty krok wymaga utworzenia funkcji implicit_convert, która wczytuje cztery dodane wcześniej wiersze z tabeli i zapisuje je w tablicy asocjacyjnej języka PL/SQL, a następnie zwraca tę tablicę:

```
SQL> CREATE OR REPLACE FUNCTION implicit_convert
  2    RETURN type_defs.plsql_table IS
  3    lv_index NUMBER := 1;           -- Zmienna licznika.
  4    lv_list TYPE_DEFS.PLSQL_TABLE;  -- Zmienna kolekcji.
  5    CURSOR c IS SELECT person FROM honeymooners;
  6  BEGIN
  7    FOR i IN c LOOP
  8      lv_list(lv_index) := i.person;
  9      lv_index := lv_index + 1;
 10    END LOOP;
 11    RETURN lv_list;  -- Zwracanie kolekcji języka PL/SQL o zasięgu lokalnym.
 12  END;
 13  /
```

W wierszu 2. jako typ zwracanej wartości ustawiona jest tablica asocjacyjna języka PL/SQL. W wierszu 4. znajduje się deklaracja zmiennej lokalnej tego samego typu. Pętla zapełnia tę tablicę, a w wierszu 11. zwracana jest zmienna lokalna zawierająca tę tablicę.

W piątym kroku należy zaimplementować blok anonimowy wywołujący funkcję implicit_convert. W instrukcjach tego bloku *lokalna* tablica asocjacyjna języka PL/SQL jest przekazywana do instrukcji SQL-a, która wczytuje tę tablicę za pomocą funkcji TABLE.

Oto ten blok anonimowy:

```
SQL> DECLARE
  2    list TYPE_DEFS.PLSQL_TABLE;
  3  BEGIN
  4    list := implicit_convert;
  5    FOR i IN (SELECT  column_value
  6              FROM    TABLE(list)) LOOP
  7      dbms_output.put_line(i.column_value);
  8    END LOOP;
  9  END;
 10  /
```

W wierszu 2. zadeklarowana jest zmienna typu plsql_table z pakietu type_defs. Wiersz 4. zawiera wywołanie funkcji implicit_convert i przypisanie zwróconej tablicy asocjacyjnej języka PL/SQL do zmiennej lokalnej. Wiersze 5. i 6. obejmują instrukcję SELECT, która wykorzystuje lokalnie zadeklarowaną zmienną języka PL/SQL w funkcji TABLE.

W wersjach starszych niż Oracle Database 12*c* funkcja TABLE potrafiła tylko przekształcać *tablice VARRAY* lub *tabele zagnieżdżone* na zbiór wyników SQL-a. Obecnie funkcja TABLE potrafi przekształcić lokalną zmienną z tablicą asocjacyjną języka PL/SQL w zasięgu SQL-a.

Jeśli umieścisz wiersz 4. (ze zmienną *lokalną*) w komentarzu i zastąpisz zmienną lokalną wywołaniem funkcji implicit_convert w wierszu 6., wystąpi błąd. Oto te zmiany:

```
4  -- list := implicit_convert;
5  FOR i IN (SELECT column_value
6            FROM  TABLE(implicit_convert)) LOOP
```

Te zmiany prowadzą do następującego stosu błędu:

```
      FROM TABLE(implicit_convert)) LOOP
                *
ERROR at line 6:
ORA-06550: line 6, column 28:
PLS-00382: expression is of wrong type
ORA-06550: line 6, column 22:
PL/SQL: ORA-22905: cannot access rows from a non-nested table item
ORA-06550: line 5, column 13:
PL/SQL: SQL Statement ignored
ORA-06550: line 7, column 26:
PLS-00364: loop index variable 'I' use is invalid
ORA-06550: line 7, column 5:
PL/SQL: Statement ignored
```

Jest też dobra wiadomość — można przekształcić tablicę asocjacyjną języka PL/SQL przez opakowanie jej w potokową funkcję tabelową. Takie funkcje nadal są potrzebne w języku PL/SQL. Załóżmy, że chcesz usunąć pozbawiony ciała pakiet, w którym zdefiniowałeś tablicę asocjacyjną języka PL/SQL. Wymaga to zmodyfikowania kodu w bloku anonimowym:

```
SQL> DECLARE
  2    TYPE local_table IS TABLE OF VARCHAR2(20)
  3      INDEX BY BINARY_INTEGER;
  4    lv_index NUMBER := 1; -- Counter variable.
  5    lv_list LOCAL_TABLE; -- Local PL/SQL collection.
  6    CURSOR c IS SELECT person FROM honeymooners;
  7  BEGIN
  8    FOR i IN c LOOP
  9      lv_list(lv_index) := i.person;
 10      lv_index := lv_index + 1;
 11    END LOOP;
 12    FOR i IN (SELECT column_value
 13              FROM TABLE(lv_list)) LOOP
 14      dbms_output.put_line(i.column_value);
 15    END LOOP;
 16  END;
 17  /
```

Ten blok także nie zadziała, ale nie, jak miało to miejsce wcześniej, z powodu próby przetwarzania tablicy asocjacyjnej zwróconej przez funkcję języka PL/SQL (choć na podstawie stosu błędu można uznać, że w obu sytuacjach przyczyna problemów jest taka sama). Tym razem źródłem błędu jest to, że typ języka PL/SQL nie jest zdefiniowany w katalogu bazy danych i baza Oracle nie potrafi go znaleźć. To oznacza, że baza nie wie, co ma przekształcić na analogiczny typ SQL-a.

Choć firma Oracle nie wyjaśnia szczegółów procesu przekształcania typów, można spróbować domyślić się, jak on przebiega. Zgaduję, że baza Oracle odwzorowuje niejawną kolekcję z języka PL/SQL na jawną tabelę zagnieżdżoną SQL-a. Jeśli na tym etapie książki ten opis wydaje Ci się zbyt techniczny, nie martw się. W rozdziale 6. znajdziesz szczegółowe omówienie złożonych typów danych (kolekcji).

Oto krótkie podsumowanie: można przypisać *lokalną* zmienną języka PL/SQL do lokalnego kontekstu SQL-a. Jednak obecnie nie można przypisać w ten sposób *nielokalnej* tablicy asocjacyjnej języka PL/SQL zwróconej przez funkcję.

Niejawne wiązanie parametrów REF CURSOR

Dostawca ODP.NET potrafi teraz niejawnie wiązać parametry `REF CURSOR` procedur składowanych. Dzieje się tak, gdy programista poda metadane w plikach konfiguracyjnych platformy .NET.

Powtórzenie materiału

W tym podrozdziale przedstawiono następujące zagadnienia związane z nowymi funkcjami języka PL/SQL w bazie Oracle Database 12c:

- Oracle Database 12*c* umożliwia zapisywanie w pamięci podręcznej wyników funkcji o uprawnieniach jednostki wywołującej.
- Oracle Database 12*c* umożliwia tworzenie białych list z jednostkami wywołującymi funkcje składowane, procedury, pakiety i typy obiektowe.
- Oracle Database 12*c* udostępnia natywną obsługę klienckich interfejsów API w zakresie typów danych języka PL/SQL.
- Oracle Database 12*c* udostępnia nowy mechanizm zarządzania stosem błędów — pakiet `utl_call_stack`.
- Oracle Database 12*c* umożliwia rozwinięcie kodu widoków zależnych od innych widoków. Służy do tego procedura `expand_sql_text` z pakietu `dbms_utility`.
- Pakiet `dbms_sql` obejmuje nowy formalny schemat, który pozwala określać niepełne nazwy obiektów.
- Oracle Database 12*c* obsługuje zagnieżdżanie funkcji języka PL/SQL w instrukcjach w klauzuli `WITH` SQL-a.
- Oracle Database 12*c* umożliwia stosowanie w lokalnych instrukcjach SQL-a lokalnych zmiennych o typach danych z języka PL/SQL.
- Oracle Database 12*c* obsługuje w dostawcy ODP.NET niejawne wiązanie typu danych `REF CURSOR` z języka PL/SQL.

Skrypty pomocnicze

W tym podrozdziale opisane są powiązane z tą książką programy dostępne w witrynach wydawnictw McGraw-Hill Professional i Helion.

- Plik *dynamic_topnquery.php* zawiera kompletny przykładowy program, którego fragmenty przedstawiono w tym rozdziale.
- Plik *white_list.sql* obejmuje wszystkie funkcje, procedury, pakiety i typy potrzebne w pokazanych w tym rozdziale przykładach ilustrujących białe listy.
- Plik *expanding_view.sql* zawiera funkcje potrzebne do przekształcania typu `LONG` na typ `CLOB` i wywoływania omówionej w tym rozdziale procedury `dbms_utility.expand_sql_text`.

Podsumowanie

W tym rozdziale zapoznałeś się z nowymi funkcjami bazy Oracle Database 12*c*. W dalszych rozdziałach także znajdziesz informacje o różnicach między starszymi wersjami baz Oracle a bazą Oracle Database 12*c*.

Test wiedzy

Test wiedzy to zestaw pytań typu „prawda czy fałsz” i wielokrotnego wyboru, dzięki którym sprawdzisz, jak dobrze opanowałeś materiał z poszczególnych rozdziałów. Odpowiedzi na pytania znajdziesz w dodatku I.

Prawda czy fałsz?

1. __Wymiar VT określa moment zatwierdzenia transakcji.
2. __Można zdefiniować kolumnę z wartościami domyślnymi, w której do generowania sekwencji używana jest pseudokolumna `.nextval`.
3. __Można zdefiniować kolumnę z wartościami domyślnymi, w której do generowania sekwencji używana jest pseudokolumna `.currval`.
4. __Pseudokolumna `.currval` nie jest już zależna od wcześniejszego wywołania pseudokolumny `.nextval` w sesji.

5. __Oracle Database 12*c* nie umożliwia zapobiegania bezpośredniemu ustawieniu wartości `null` w instrukcji `INSERT`, co oznacza, że można zastąpić wartość domyślną kolumny.
6. __Kolumny identyfikacyjne umożliwiają automatyczne generowanie wartości kolumny z kluczem sztucznym.
7. __W bazie Oracle Database 12*c* typy danych `VARCHAR2`, `NVARCHAR2` i `RAW` zawsze mają 32 767 bajtów.
8. __Dzięki zmianom wprowadzonym w bazie Oracle Database 12*c* funkcja języka PL/SQL może zwrócić tablicę asocjacyjną tego języka bezpośrednio do instrukcji SQL-a.
9. __Oracle Database 12*c* obsługuje pobieranie n pierwszych wyników zapytania bez konieczności określania przedziału pobieranych danych.
10. __Można umieścić funkcję języka PL/SQL w klauzuli `WITH` zapytania i wywoływać ją w programach zewnętrznych.

Pytania wielokrotnego wyboru

11. Które z poniższych słów kluczowych można stosować przy definiowaniu widoku? Poprawnych może być kilka odpowiedzi.
    - **A.** Słowa kluczowe `AUTHID DEFINER`.
    - **B.** Słowa kluczowe `BEQUEATH INVOKER`.
    - **C.** Słowa kluczowe `AUTHID CURRENT_USER`.
    - **D.** Słowa kluczowe `BEQUEATH DEFINER`.
    - **E.** Wszystkie z powyższych.
12. Które z poniższych stwierdzeń dotyczących zapisywania w pamięci podręcznej wyników funkcji z uprawnieniami jednostki wywołującej są prawdziwe? Poprawnych może być kilka odpowiedzi.
    - **A.** Dla poszczególnych jednostek wywołujących istnieją różne zbiory wyników.
    - **B.** Dla każdej jednostki wywołującej istnieje ten sam zbiór wyników.
    - **C.** Funkcja z uprawnieniami jednostki wywołującej, której wyniki są zapisywane w pamięci podręcznej, musi być deterministyczna.
    - **D.** Funkcja z uprawnieniami jednostki wywołującej, której wyniki są zapisywane w pamięci podręcznej, może być niedeterministyczna.
    - **E.** Wszystkie z powyższych.
13. Które z poniższych stwierdzeń dotyczących przekształcania tekstu rozwijanych instrukcji SQL-a z typu `LONG` na typ `CLOB` są prawdziwe w kontekście pracy z widokami `CDB_`, `DBA_`, `ALL_` i `USER_VIEWS` w bazie Oracle Database 12*c*? Poprawnych może być kilka odpowiedzi.
    - **A.** Możesz używać funkcji wbudowanej `to_lob` do przekształcania typu `LONG` na typ `CLOB`.
    - **B.** Możesz używać funkcji wbudowanej `to_clob` do przekształcania typu `LONG` na typ `CLOB`.
    - **C.** Możesz używać pakietu `dbms_sql` do przekształcania typu `LONG` na typ `VARCHAR2`.
    - **D.** Możesz używać funkcji wbudowanej `length` do określania długości danych typu `LONG`.
    - **E.** Możesz używać pakietu `dbms_lob` do tworzenia tymczasowych danych typu `CLOB`.
14. Które z poniższych stwierdzeń dotyczących typów danych języka PL/SQL dostępnych w zagnieżdżonych instrukcjach SQL-a są prawdziwe? Poprawnych może być kilka odpowiedzi.
    - **A.** Typ danych języka PL/SQL musi być zadeklarowany w pakiecie.
    - **B.** Instrukcja SQL-a musi być zagnieżdżona w bloku języka PL/SQL, w którym zdefiniowany jest określony typ.
    - **C.** Typ danych języka PL/SQL musi być zdefiniowany lokalnie.
    - **D.** Typ danych języka PL/SQL może być zwracany przez funkcję tego języka.
    - **E.** Wszystkie z powyższych.

15. Która z poniższych instrukcji umożliwia dostęp do sztucznego klucza głównego z kolumny identyfikacyjnej w celu użycia wartości tego klucza w dalszej instrukcji `INSERT` jako wartości klucza obcego?

    A. Instrukcja `RETURN INTO`.

    B. Instrukcja `RETURNING INTO`.

    C. Instrukcja `.nextval`.

    D. Instrukcja `.currval`.

    E. Żadne z powyższych.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

Obowiązkowa wiedza każdego bazodanowca!

Oracle to jedna z najlepszych baz danych. Najczęściej korzystają z niej firmy i instytucje. W jej tabelach przechowywane są gigantyczne ilości danych, przetwarzane każdego dnia w celu wyłuskania kluczowych informacji. Wykonywanie operacji na danych bezpośrednio w bazie jest możliwe dzięki rozszerzeniu języka SQL, które pozwala na tworzenie konstrukcji znanych z innych języków programowania. To właśnie PL/SQL!

Jeżeli sięgniesz po tę książkę, będziesz mieć niepowtarzalną okazję błyskawicznego poznania jego potencjału. Na kolejnych stronach znajdziesz informacje na temat podstaw pracy z PL/SQL, a następnie przejdziesz do bardziej zaawansowanych zagadnień. Zdobędziesz wiedzę na temat struktur sterujących, kolekcji oraz pracy z dużymi obiektami. Ponadto nauczysz się obsługiwać błędy oraz budować wyzwalacze. Książka ta jest doskonałą lekturą dla osób chcących poznać możliwości języka PL/SQL!

Dzięki tej książce:

- **zaznajomisz się ze strukturami sterującymi**
- **zbudujesz pętle, instrukcje warunkowe oraz kolekcje**
- **błyskawicznie zlokalizujesz błędy w Twoim kodzie**
- **opanujesz możliwości języka PL/SQL**

Helion
28952 numer katalogowy
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900
Informatyka w najlepszym wydaniu

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-246-9923-0
9 788324 699230
cena: 149,00 zł

Oracle Press