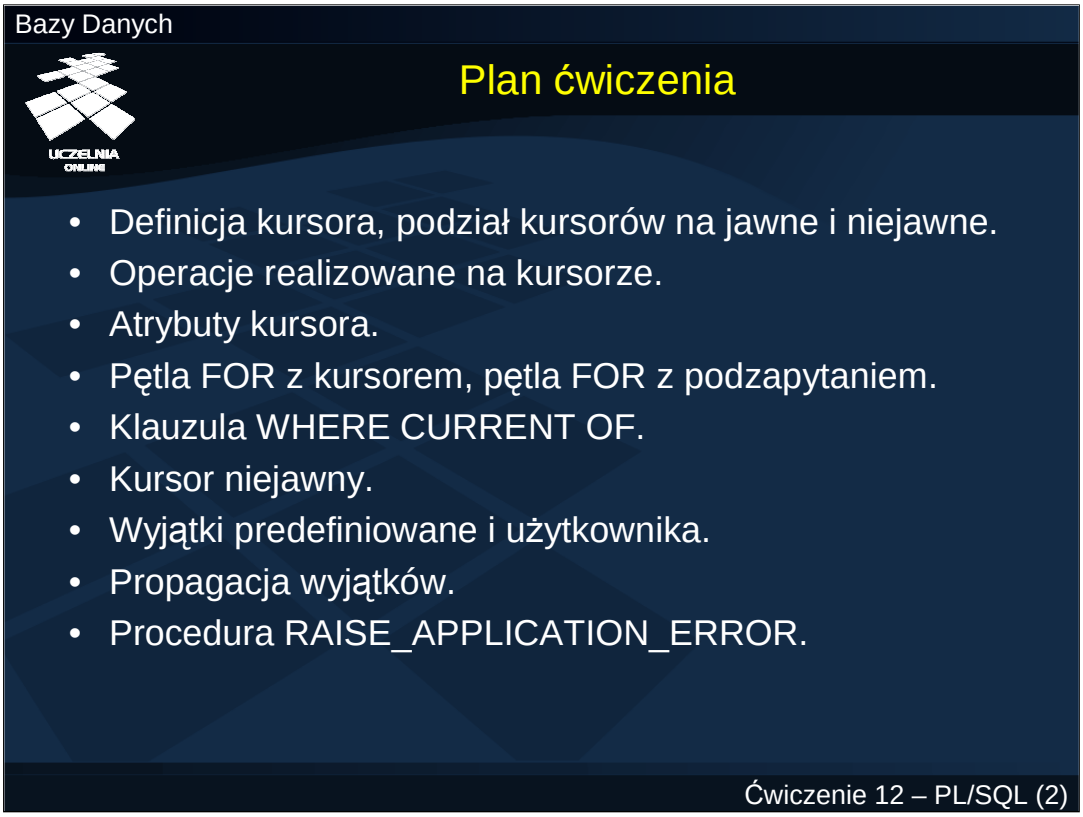
Niniejsze ćwiczenie dotyczy dwóch odrębnych zagadnień języka PL/SQL. Pierwsze z nich to kursory, będące konstrukcjami pozwalającymi na odczyt w programie PL/SQL zbioru rekordów. Drugie zagadnienie to obsługa wyjątków w programie PL/SQL.

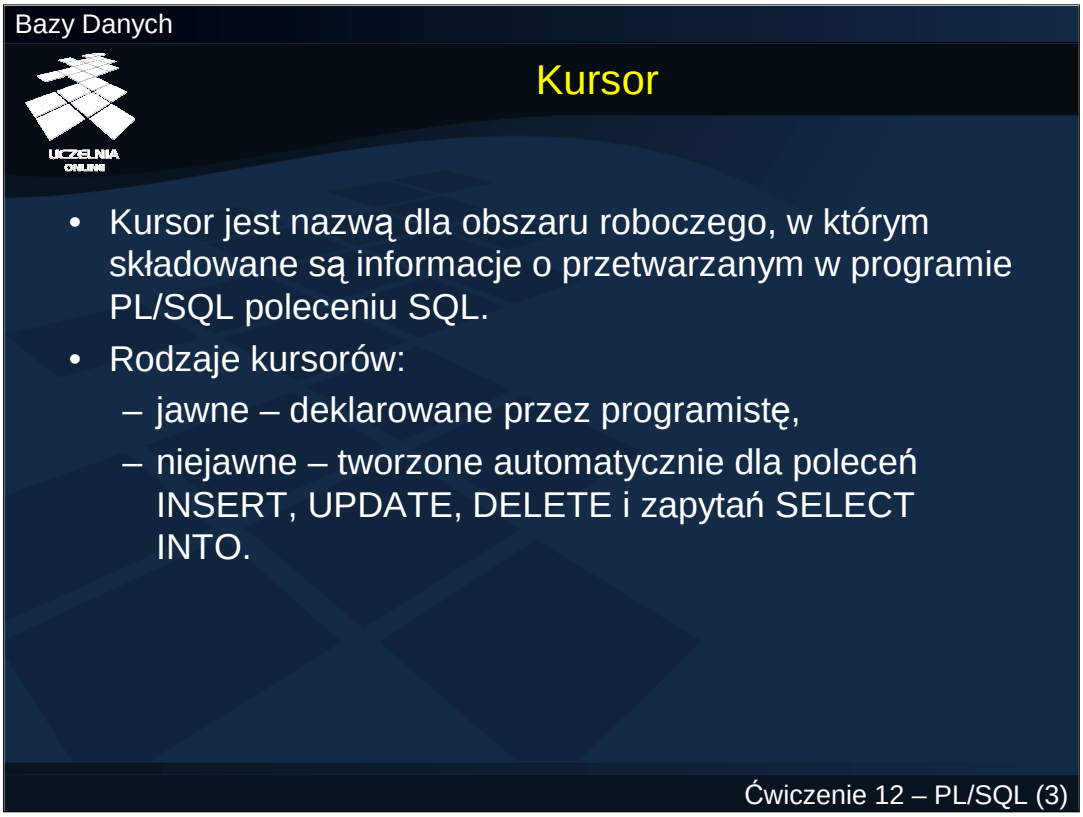
Wymagania:

Umiejętność tworzenia zapytań w języku SQL, znajomość operacji z grup DML i DDL, umiejętność tworzenia prostych programów PL/SQL.



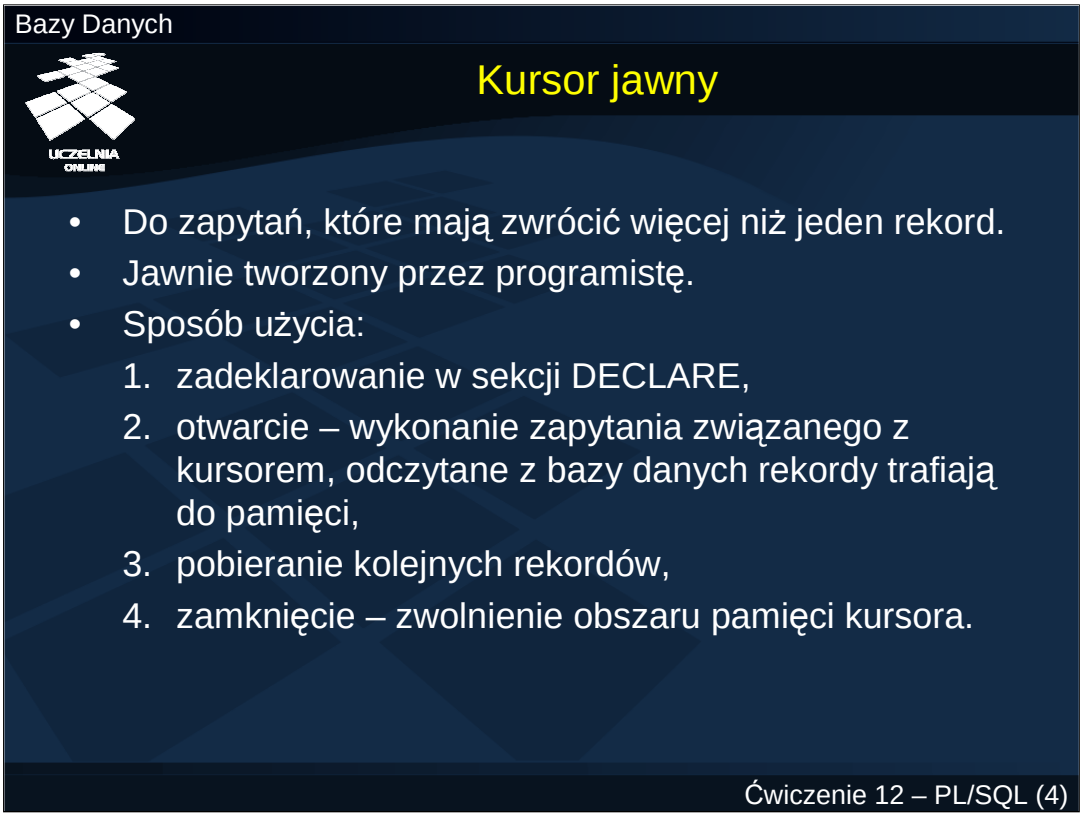
Na początku omówiona zostanie definicja kursora, wyjaśnione zostaną różnice pomiędzy kursorem jawnym i niejawnym. Następnie zaprezentowane zostaną operacje realizowane na kursorze: deklarowanie kursora, otwieranie kursora, pobieranie rekordów z kursora oraz zamykanie kursora. W dalszej części ćwiczenia przedstawione zostaną atrybuty kursora i ich zastosowanie. Dalej omówimy pętle FOR z kursorem i podzapytaniem oraz wykorzystanie klauzuli WHERE CURRENT OF. Problematykę dotyczącą kursorów zakończymy zaprezentowaniem kursora niejawnego.

Dalej omówiona zostanie obsługa wyjątków w programie PL/SQL. Zaprezentujemy działanie programu z obsługą wyjątków, wprowadzimy podział na wyjątki predefiniowane i użytkownika. Następnie dokładnie wyjaśnimy propagację wyjątków w bloku PL/SQL. Ćwiczenie zakończymy omówieniem procedury RAISE_APPLICATION_ERROR.



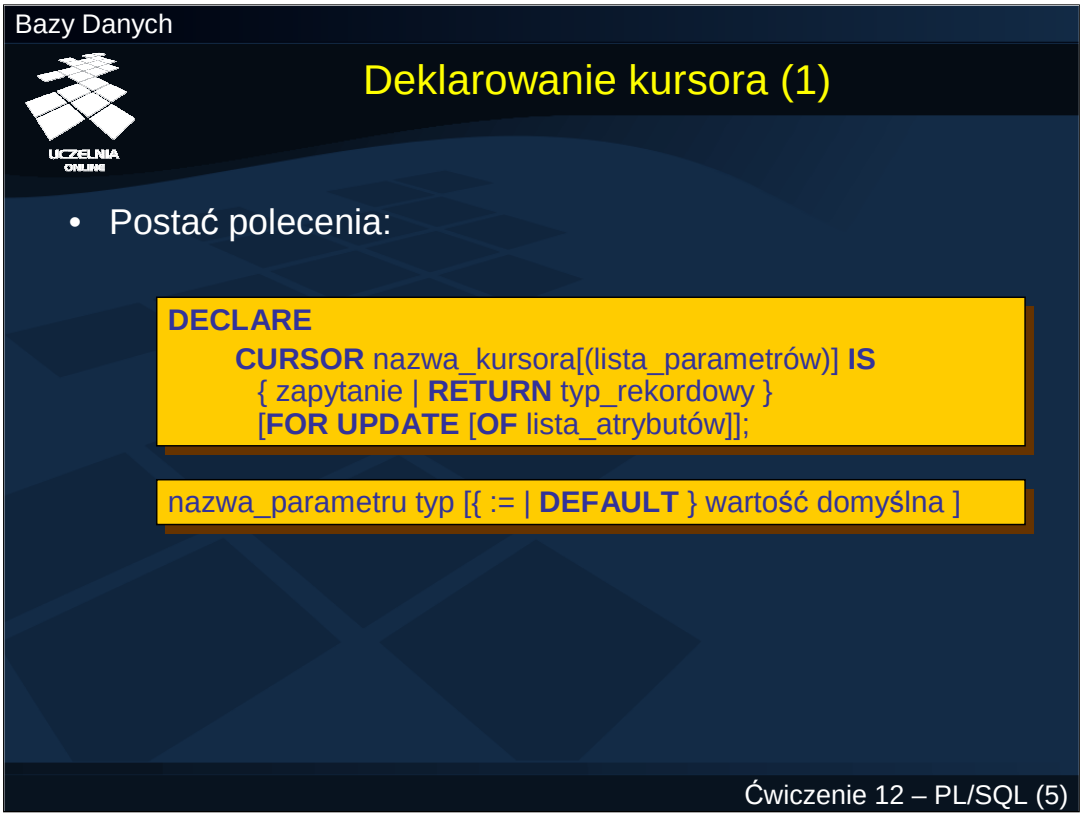
Każde polecenie SQL, umieszczone w programie PL/SQL, w trakcie wykonania zostaje skojarzone z obszarem pamięci, w którym przechowywane są informacje o przetwarzanym poleceniu: m.in. status wykonania polecenia i zbiór odczytanych z bazy danych rekordów w przypadku zapytania. Dostęp w programie PL/SQL do tego obszaru pamięci, nazywanego również obszarem roboczym, jest możliwy za pomocą specjalnej struktury, tzw. kursora. Każdy kursor posiada swoją własną nazwę i może służyć np. do odczytania liczby rekordów, jakie zmodyfikowało polecenie UPDATE w programie PL/SQL, pobierania po kolei rekordów, jakie z bazy danych odczytało zapytanie, czy też sprawdzenia, czy polecenie DELETE usunęło rekordy.

Wyróżniamy dwa rodzaje kursorów: jawne i niejawne. Kursory jawne są deklarowane przez programistę i służą do odczytu zbioru rekordów z bazy danych (jak pamiętamy z poprzedniego ćwiczenia, zwykłe polecenie SELECT umieszczone w programie PL/SQL może odczytać z bazy danych tylko jeden rekord). Z kolei kursory niejawne są tworzone automatycznie dla każdego z poleceń UPDATE, INSERT, DELETE i SELECT INTO, jakie zostaje umieszczone w programie. Kursory niejawne najczęściej służą do sprawdzenia stanu polecenia, dla którego kursor niejawny został utworzony.



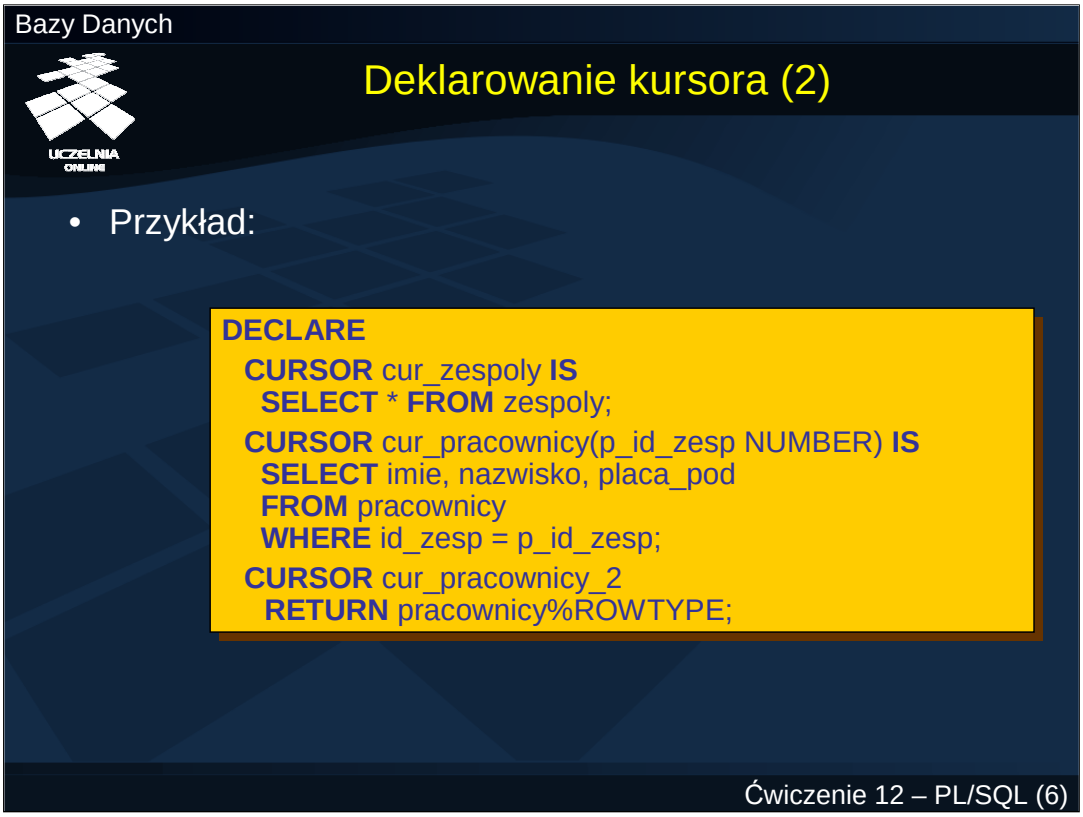
Przejdziemy teraz do omawiania kursorów jawnych. Kursory jawne są jedyną metodą, umożliwiającą w programie PL/SQL odczyt zbioru rekordów.

Aby kursor jawny mógł być użyty w programie, konieczne jest wykonanie szeregu operacji. Pierwszą z nich jest zadeklarowanie kursora w sekcji deklaracji bloku PL/SQL. Deklarując kursor programista podaje jego nazwę i zapytanie, które odczyta z bazy danych rekordy, jakie kursor ma udostępniać. Następnie, już w sekcji wykonywalnej programu, kursor musi zostać otwarty. W tym momencie zostaje wykonane zapytanie związane z kursorem, zbiór rekordów, odczytany z bazy danych przez zapytanie, zostaje składowany w związanym z kursorem obszarze pamięci. Po otwarciu kursora można pobierać kolejno rekordy z odczytanego zbioru. Gdy kursor przestaje być potrzebny, należy go zamknąć, co powoduje zwolnienie obszaru pamięci, uprzednio przydzielonego do kursora.



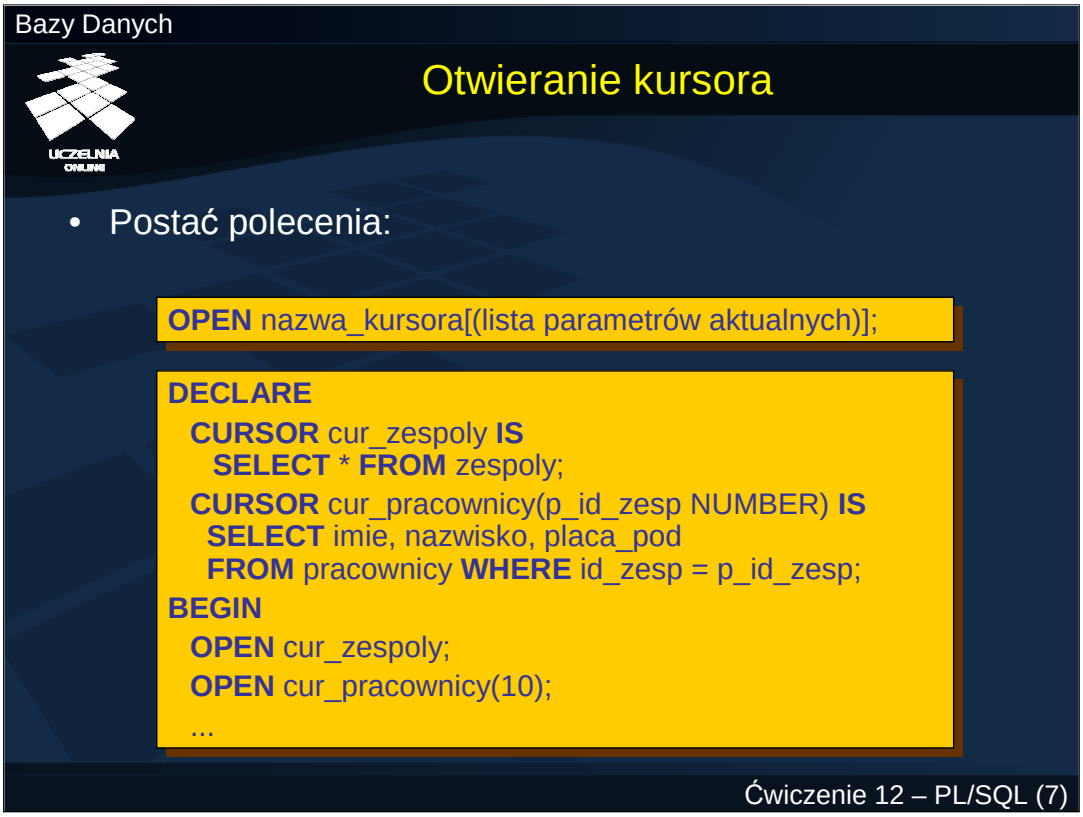
Na bieżącym slajdzie przedstawiono postać deklaracji kursora. Deklaracja rozpoczyna się od słowa kluczowego CURSOR, za którym podaje się nazwę kursora. Dobrą praktyką jest tworzenie nazwy dla kursora z przedrostkiem „cur_”. Jeśli kursor ma być sparametryzowany, listę parametrów kursora umieszczamy w nawiasach po nazwie kursora. Parametry służą do deklarowania kursorów elastycznych, które odczytują różne zbiory rekordów w zależności od wartości parametrów, podanych przy otwieraniu kursora. Każdy parametr musi mieć nadaną nazwę oraz zdefiniowany typ wartości przekazywanej do wnętrza kursora. Uwaga! Dla typu nie podajemy nigdy długości. Np. definiując parametr, będący ciągiem znaków, podajemy tylko nazwę typu, a więc VARCHAR2, natomiast nie określamy długości ciągu znaków. Dodatkowo dla parametru można podać wartość domyślną, jaką zostanie przyjęta dla parametru w przypadku, gdy przy otwarciu kursora nie podano wartości dla parametru. Wartość domyślną podaje się po słowie DEFAULT lub operatorze przypisania. Jeśli kursor ma więcej parametrów, oddzielone muszą być one od siebie przecinkami.

Kolejna część deklaracji kursora, następująca po słowie IS, to zapytanie, które odczyta zbiór rekordów dla kursora. Jeśli kursor jest sparametryzowany, wówczas w zapytaniu należy użyć zdefiniowanych parametrów (np. w klauzuli WHERE). Niekiedy w momencie deklaracji kursora nie znamy postaci zapytania kursora. W takim przypadku po słowie kluczowym RETURN podaje się jedynie nazwę typu rekordowego o strukturze identycznej ze strukturą rekordów, jakie będzie odczytywał z bazy danych kursor.



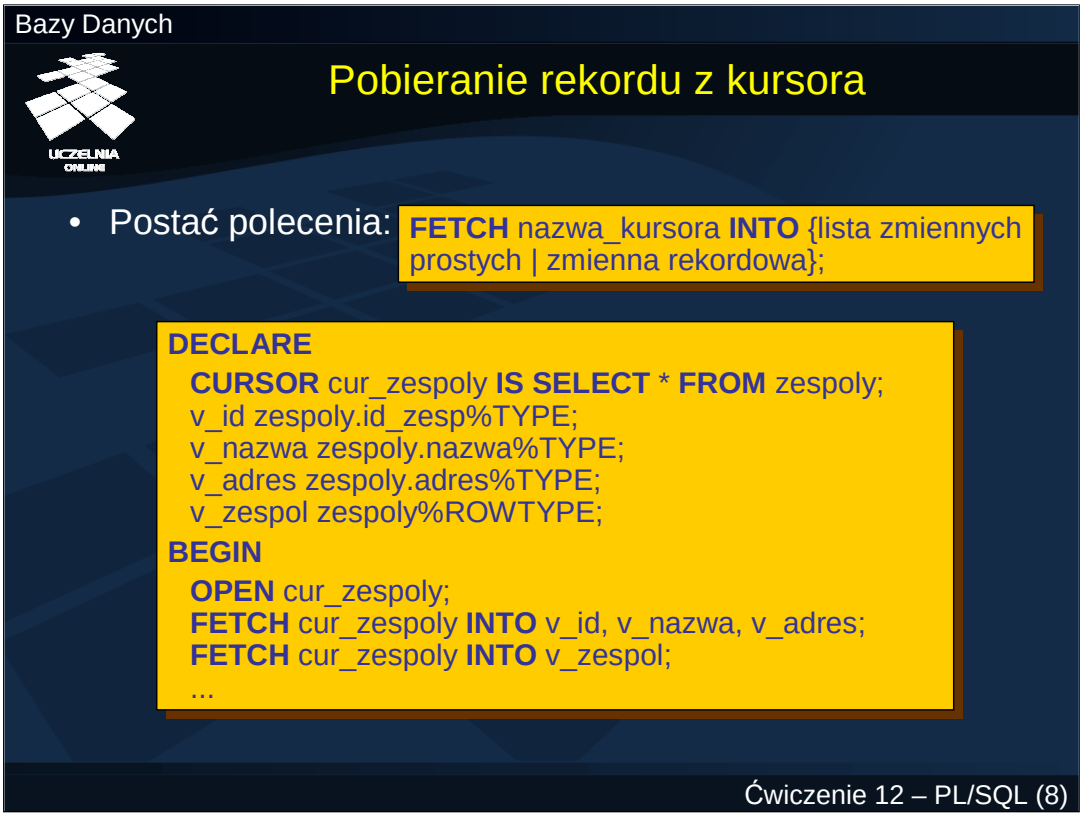
Cd. z poprzedniego slajdu. Ostatnim, opcjonalnym elementem deklaracji kursora jest klauzula FOR UPDATE, która określa, że rekordy, odczytywane z bazy danych przez kursor, mogą być zmodyfikowane w programie przez zlecenia UPDATE lub usunięte przez DELETE. Należy mieć świadomość, że przy otwieraniu takiego kursora na rekordach, odczytanych przez zapytanie kursora, zostają założone blokady. Pełną postać klauzuli, „FOR UPDATE OF lista_atrybutów", używa się wtedy, gdy zapytanie kursora zawiera połączenie, a zablokowane mają zostać rekordy tylko jednej z relacji, biorących udział w połączeniu. W takim przypadku lista_atrybutów powinna zawierać dowolny atrybut z relacji, która ma być blokowana. W przypadku pominięcia listy atrybutów zablokowane zostaną rekordy wszystkich relacji, biorących udział w połączeniu.

W przykładzie, zaprezentowanym na bieżącym slajdzie, zadeklarowano trzy kursory. Pierwszy kursor o nazwie cur_zespoly jest kursorem bezparametrowym, odczytującym wszystkie informacje z relacji ZESPOLY. Kolejny kursor, cur_pracownicy, posiada jeden parametr liczbowy o nazwie p_id_zesp, a zapytanie kursora odczytuje imiona, nazwiska i płace pracowników, zatrudnionych w zespole, którego identyfikator zawiera parametr (parametr jest użyty w klauzuli WHERE do filtrowania wyniku zapytania). Ostatni kursor, c_pracownicy_2, to przykład kursora, którego zapytanie nie jest jeszcze znane. Kursor będzie odczytywał z bazy danych rekordy, których struktura będzie taka sama, jak struktura jednego rekordu relacji PRACOWNICY.



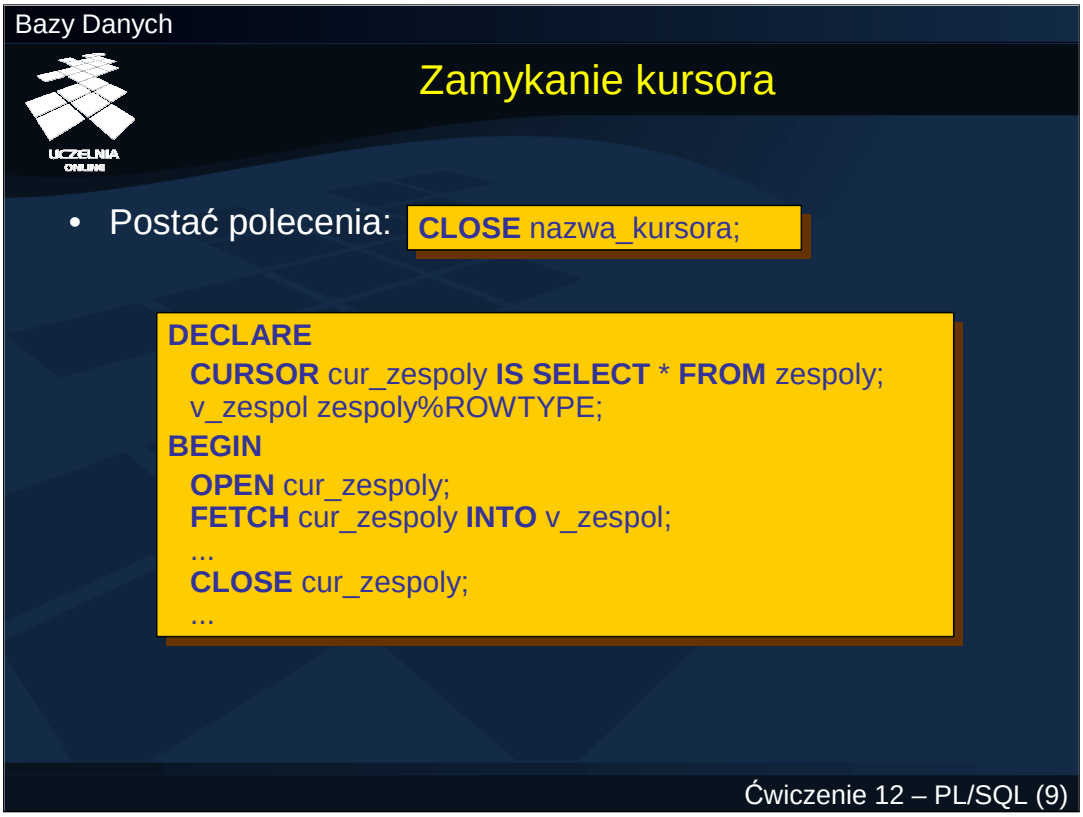
Po zadeklarowaniu kursora, przed jego użyciem, kursor musi zostać otwarty. Służy do tego polecenie OPEN, po którym podajemy nazwę otwieranego kursora i, jeśli kursor posiada parametry, w nawiasie listę wartości dla parametrów. W momencie wykonania polecenia OPEN zapytanie kursora jest wykonywane a odczytane z bazy danych rekordy zostają zapisane w obszarze roboczym kursora. Wskaźnik bieżącego rekordu kursora zostaje ustawiony na pierwszym rekordzie odczytanego zbioru.

W przykładzie powtórzono deklarację kursorów cur_zespoly i cur_pracownicy z poprzedniego przykładu. Następnie w sekcji wykonywalnej bloku otwarto kursor cur_zespoly, a następnie kursor cur_pracownicy z wartością 10 dla parametru p_id_zesp. Spowoduje to odczytanie przez kursor danych pracowników z zespołu o identyfikatorze równym 10.



Z otwartego kursora można zacząć pobierać rekordy. Służy do tego polecenie FETCH. Po słowie kluczowym FETCH podajemy nazwę kursora, następnie po słowie INTO umieszczamy albo listę zmiennych prostych, które odbiorą wartości od kolejnych atrybutów rekordu kursora, albo jedną zmienną rekordową, której struktura musi być identyczna ze strukturą rekordu kursora. Wykonanie operacji FETCH przesuwa wskaźnik bieżącego rekordu na następny rekord zbioru kursora.

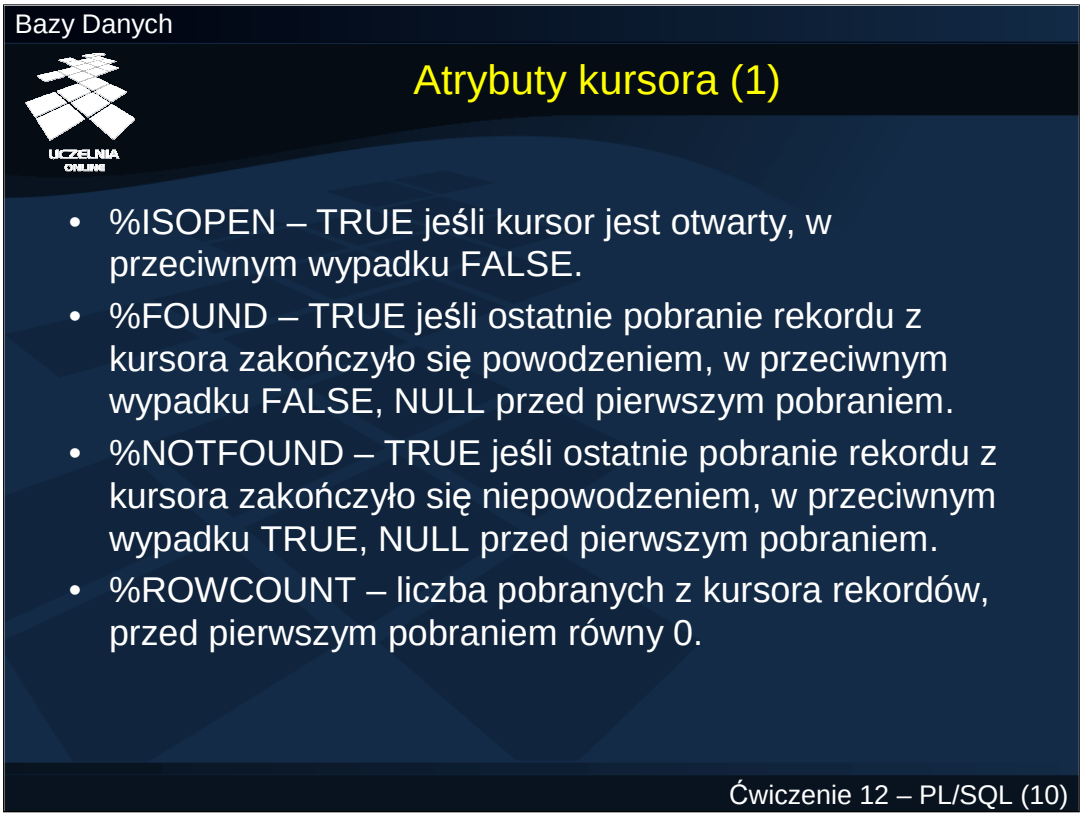
W przykładzie ponownie zadeklarowano kursor cur_zespoly oraz cztery zmienne: trzy proste: v_id, v_nazwa i v_adres i jedną rekordową v_zespol. W sekcji wykonywalnej bloku otwarto kursor, następnie pierwsze polecenie FETCH odczytuje pierwszy rekord kursora cur_zespoly i umieszcza wartości atrybutów ID_ZESP, NAZWA i ADRES rekordu, odczytanego z relacji ZESPOLY przez kursor, w zmiennych v_id, v_nazwa i v_adres. Druga operacja FETCH pobiera kolejny rekord ze zbioru kursora i umieszcza wartości atrybutów w zmiennej rekordowej v_zespol.



Po zakończeniu korzystania z kursora powinien on zostać zamknięty. Wykonuje to polecenie CLOSE, po którym podajemy nazwę zamykanego kursora. Wykonanie polecenia CLOSE zwalnia obszar roboczy kursora, nie można już z niego pobierać kolejnych rekordów. Jeśli wykonamy operację FETCH dla zamkniętego kursora, program zostanie przerwany z błędem INVALID_CURSOR.

Po zamknięciu kursor może być ponownie otwarty omawianym już poleceniem OPEN. Zapytanie kursora zostaje wtedy ponownie wykonane a do obszaru roboczego kursora zostaje ściągnięty nowy zbiór rekordów.

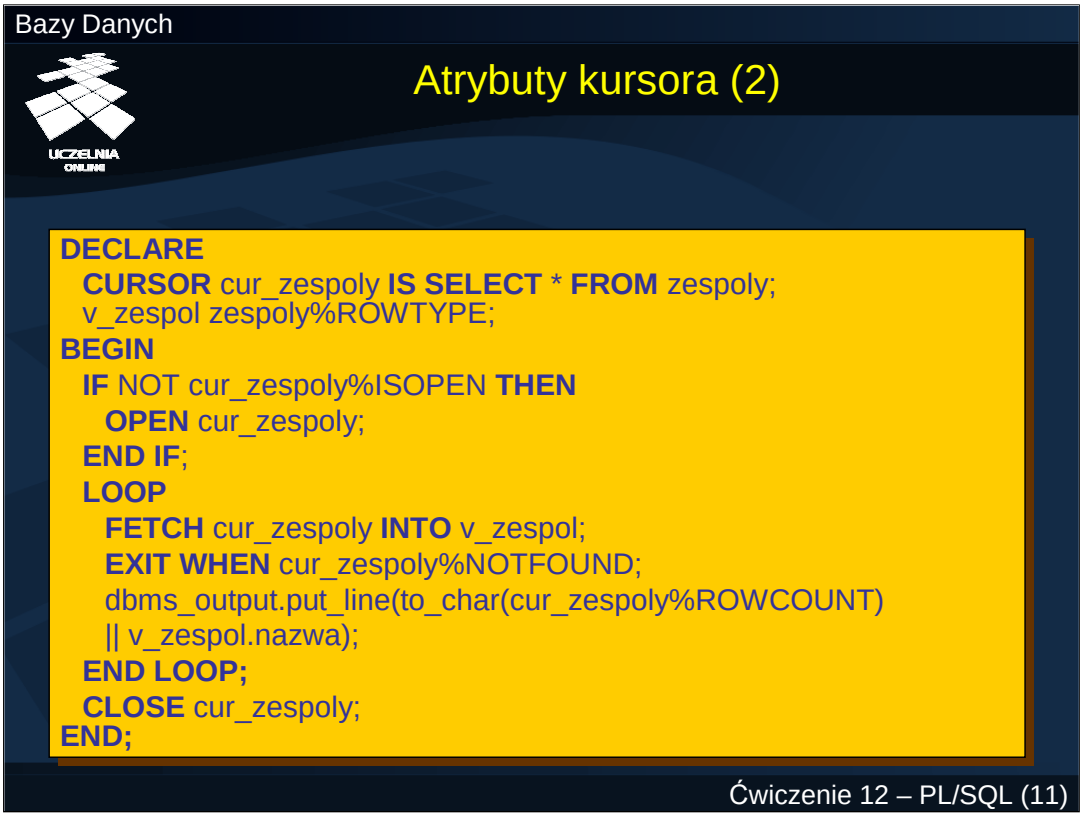
W zaprezentowanym na bieżącym slajdzie przykładzie otwieramy zadeklarowany kursor cur_zespoly, pobieramy z niego rekord poleceniem FETCH, następnie zamykamy kursor poleceniem CLOSE.



Każdy kursor posiada cztery atrybuty, dzięki którym mamy możliwość sprawdzenia statusu kursora. Aby użyć atrybuty w programie należy nazwę atrybutu poprzedzić nazwą kursora.

Atrybut %ISOPEN posiada wartość TRUE jeśli kursor jest otwarty, w przeciwnym wypadku wartością atrybutu jest FALSE. Wartością atrybut %FOUND jest TRUE jeśli ostatnia operacja pobrania rekordu z kursora (polecenie FETCH) zakończyło się powodzeniem (rekord został odczytany). Gdy odczytanie rekordu nie zakończyło się powodzeniem (zbiór rekordów kursora jest pusty lub wskaźnik bieżącego rekordu dotarł do końca zbioru rekordów kursora), wartością atrybutu %FOUND jest wartość FALSE. Wartości atrybutu %NOTFOUND są komplementarne w stosunku do wartości atrybutu %FOUND: TRUE gdy ostatni odczyt nie zakończył się powodzeniem i FALSE, gdy odczyt się powiódł. Należy pamiętać, że wartości atrybutów %FOUND i %NOTFOUND są ustawiane po próbie pobrania pierwszego rekordu z kursora, wcześniej wartościami obu atrybutów jest wartość pusta.

Wykorzystując atrybut %ROWCOUNT możemy dowiedzieć się, ile rekordów odczytano dotąd z kursora. Przed pierwszym pobraniem rekordów z kursora wartość tego atrybutu to 0.



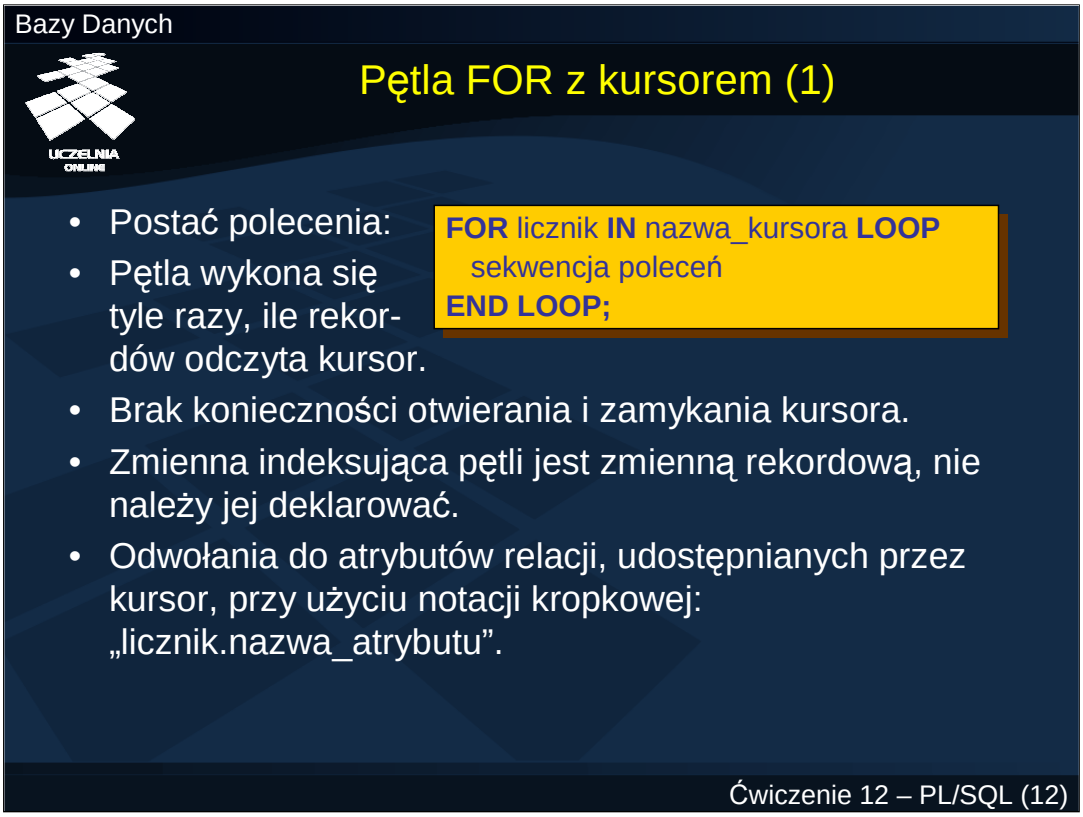
Przykład zaprezentowany na bieżącym slajdzie pokazuje sposób wykorzystania atrybutów kursora.

W przykładzie zadeklarowano kursor cur_zespoly i zmienną rekordową v_zespol. Na początku programu sprawdzamy, czy kursor nie został już otwarty, odczytując wartość atrybutu %ISOPEN. Jeśli wartość atrybutu to FALSE, otwieramy kursor poleceniem OPEN.

Następnie rozpoczynamy pętlę LOOP, w której będziemy po kolei pobierać rekordy z kursora. Pierwsze polecenie wewnątrz pętli pobiera rekord z kursora i umieszcza wartości atrybutów rekordu w zmiennej rekordowej v_zespol. Dalej sprawdzany jest warunek wyjścia z pętli. Kończymy pętlę w momencie, gdy wartość atrybutu %NOTFOUND kursora jest równa TRUE, czyli wtedy, gdy wcześniejsze polecenie FETCH nie odczytało żadnego rekordu (czyli albo kursor nie odczytał żadnych rekordów z bazy danych albo wskaźnik bieżącego rekordu znajduje się poza ostatnim rekordem ze zbioru, czyli zostały odczytane już wszystkie rekordy kursora). Jeśli atrybut %NOTFOUND ma wartość FALSE, wówczas kontynuujemy pętlę, wypisując na konsoli liczbę pobranych dotąd rekordów z kursora (wartość atrybutu %ROWCOUNT, która tutaj jest wykorzystywana jako liczba porządkowa kolejnych rekordów) i nazwę zespołu z bieżącego rekordu kursora.

Po zakończeniu pętli kursor zostaje zamknięty poleceniem CLOSE

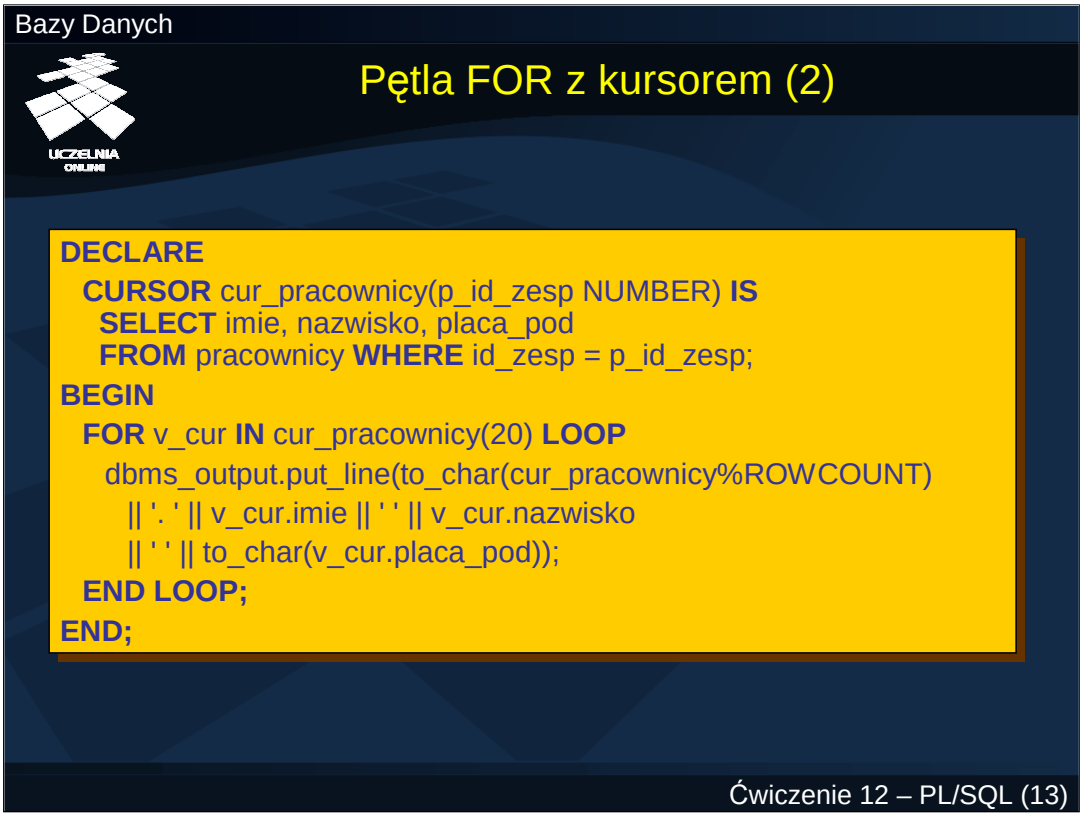
Zaprezentowany przykład pokazuje typową konstrukcję pętli LOOP z kursorem.



Wygodniejszym niż pętla LOOP rozwiązaniem, umożliwiającym pobieranie rekordów z kursora, jest pętla FOR z kursorem.

Pętla rozpoczyna się słowem kluczowym FOR, po którym umieszczamy zmienną licznikową. Zmiennej tej nie należy deklarować, jest to zmienna rekordowa o strukturze odpowiadającej strukturze rekordu kursora. Następnie po słowie IN podajemy nazwę wcześniej zadeklarowanego kursora. Po słowie LOOP podajemy sekwencję poleceń, która kończy się słowem END LOOP. Pętla wykona się tyle razy, ile rekordów odczytało zapytanie kursora z bazy danych. Uwaga! Stosując pętle FOR nie wykonujemy operacji otwierania i zamykania kursora. Kursor jest automatycznie otwierany przy wejściu do pętli, a zostaje zamknięty po wykonaniu ostatniej iteracji.

Wewnątrz pętli odwołujemy się do wartości atrybutów bieżącego rekordu kursora bezpośrednio bez konieczności wykonania polecenia FETCH. Odwołanie ma postać „licznik.nazwa_atrybutu”, możliwe jest tylko odczytanie wartości atrybutów (zmiana wartości jest zabroniona).

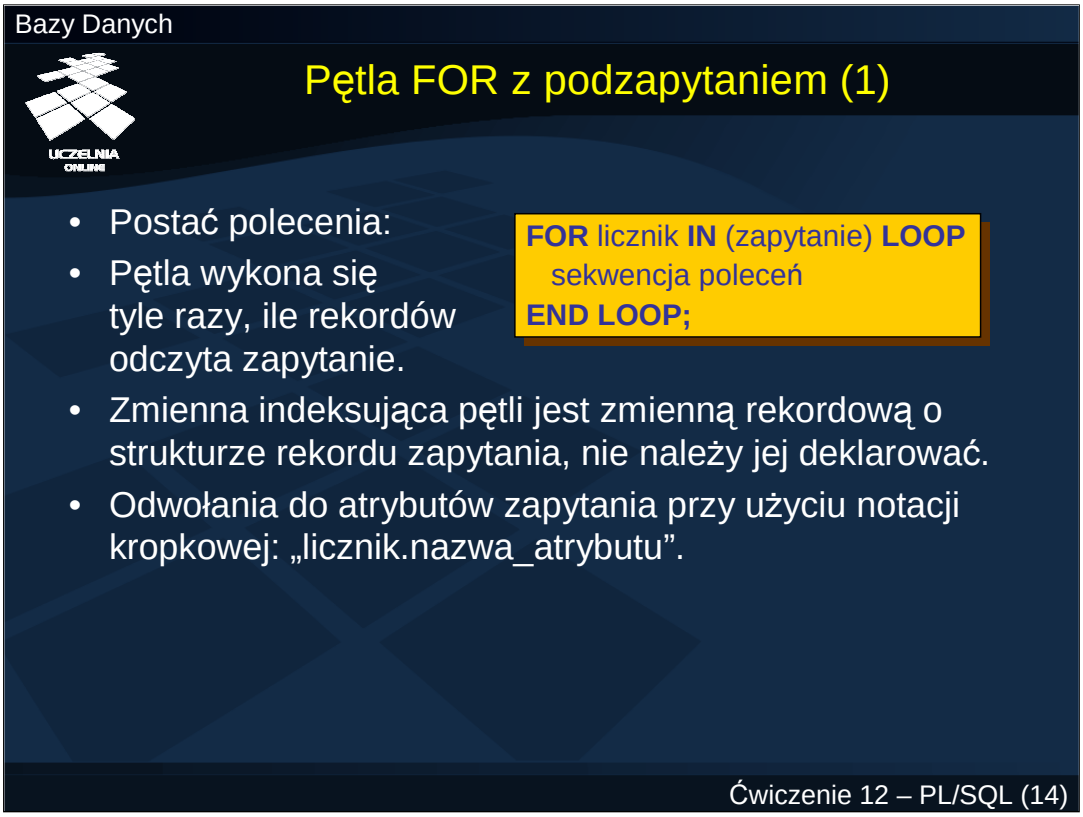


Bieżący slajd prezentuje użycie pętli FOR z kursorem.

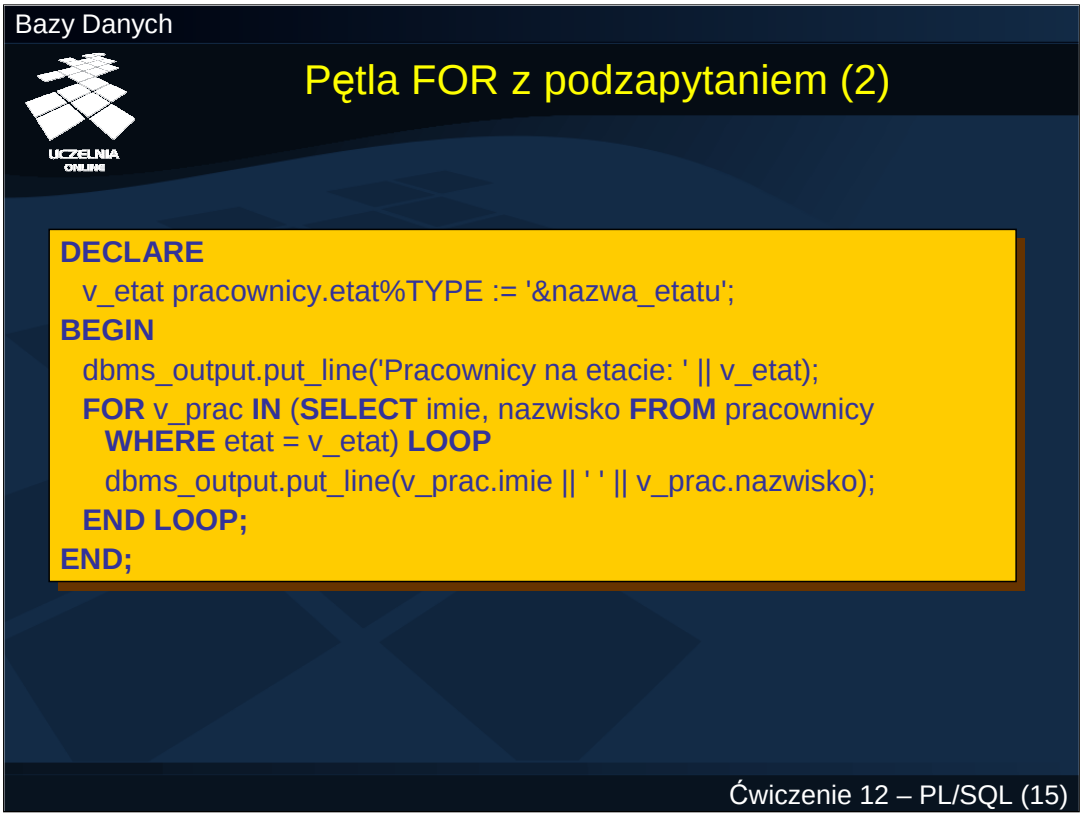
W sekcji deklaracyjnej bloku zostaje zadeklarowany sparametryzowany kursor cur_pracownicy, który odczytuje imiona, nazwiska i płace podstawowe pracowników z zespołu, którego identyfikator jest przekazywany przez parametr.

W sekcji wykonywalnej bloku konstruujemy pętlę FOR, zauważmy, że po słowie IN podano nazwę kursora z wartością parametru – w pętli będą odczytywane rekordy opisujące kolejnych pracowników zespołu o numerze 20. Zmienną licznikową pętli jest zmienna rekordowa v_cur. Wewnątrz pętli wypisujemy imiona, nazwiska i płace podstawowe pracowników, kolejne rekordy numerowane są wartością atrybutu %ROWCOUNT kursora.

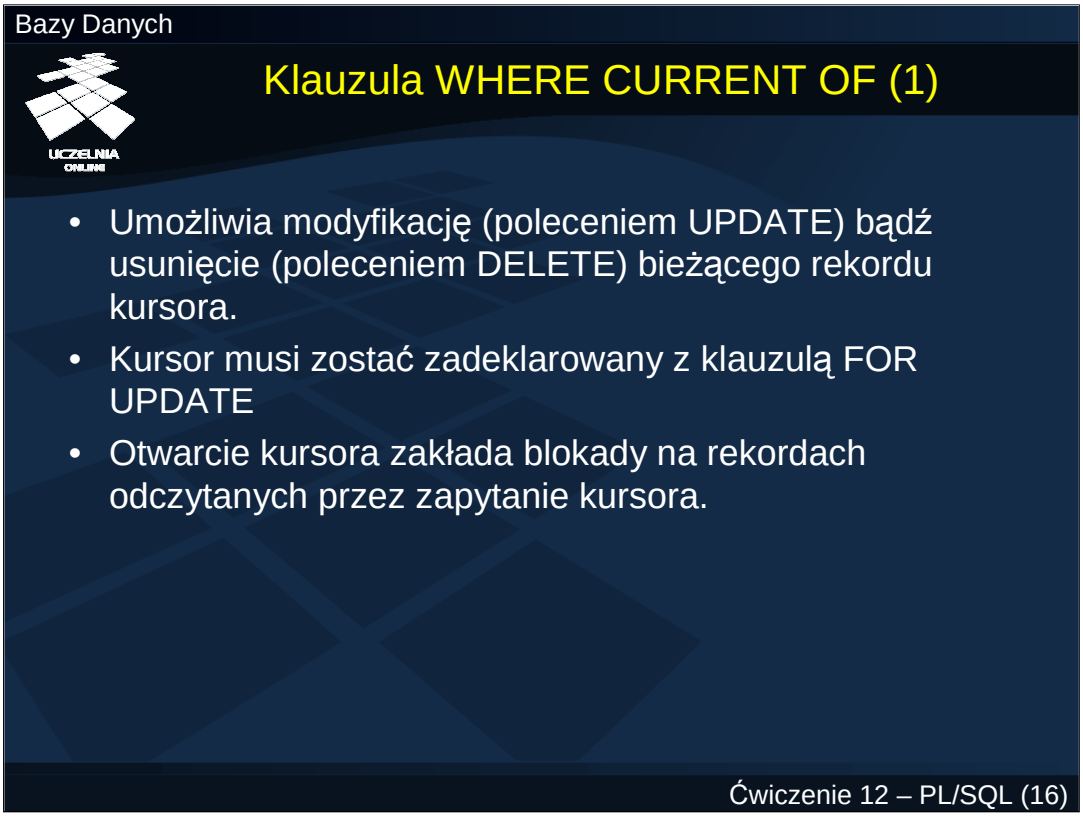
Zauważmy, że w programie nie ma poleceń otwarcia i zamknięcia kursora.



W programie PL/SQL można stosować odmianę pętli FOR z kursorem, mianowicie pętlę FOR z podzapytaniem. Obie pętle działają identycznie, jedyna różnica między nimi jest taka, że pętla FOR z podzapytaniem nie odczytuje rekordów wcześniej zadeklarowanego kursora, ale rekordy zapytania, umieszczonego w nawiasach zaraz po słowie IN. Pętla wykona się tyle razy, ile rekordów odczyta zapytanie z bazy danych.

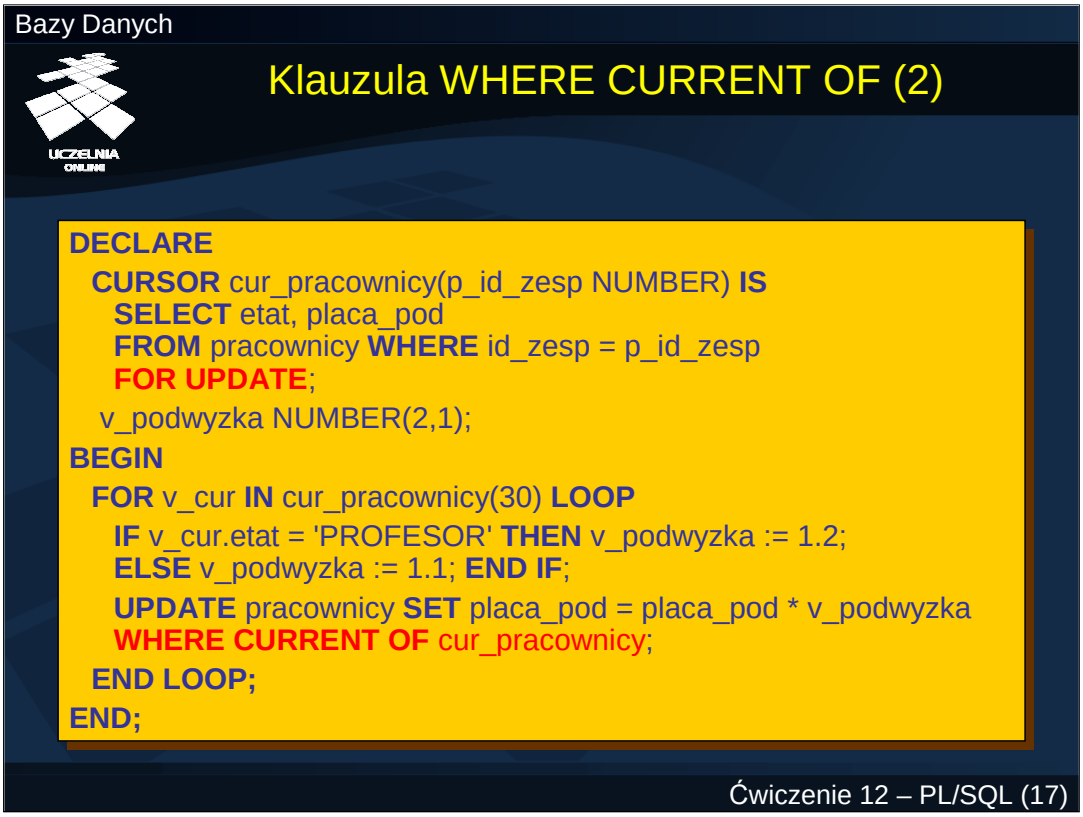


Bieżący slajd demonstruje użycie pętli FOR z podzapytaniem. W sekcji deklaracyjnej zadeklarowano zmienną v_etat, która jest inicjowana wartością podaną przez użytkownika (wykorzystano tutaj zmienną podstawienia). Pierwsze polecenie programu wyświetla na konsoli tekst „Pracownicy na etacie <nazwa etatu>”, gdzie nazwa etatu jest wartością zmiennej v_etat. Następnie rozpoczyna się pętla FOR z podzapytaniem, odczytującym imiona i nazwiska pracowników, pracujących na etacie określonym zmienną v_etat (patrz warunek w klauzuli WHERE zapytania). Zmienną licznikową pętli jest zmienna rekordowa v_prac. Wewnątrz pętli wartości atrybutów IMIE i NAZWISKO rekordów zapytania zostają wypisane na konsoli. Zauważmy, w jaki sposób realizowane jest odwołanie do atrybutów – wykorzystujemy notację kropkową (odpowiednio v_prac.imie i v_prac.nazwisko).



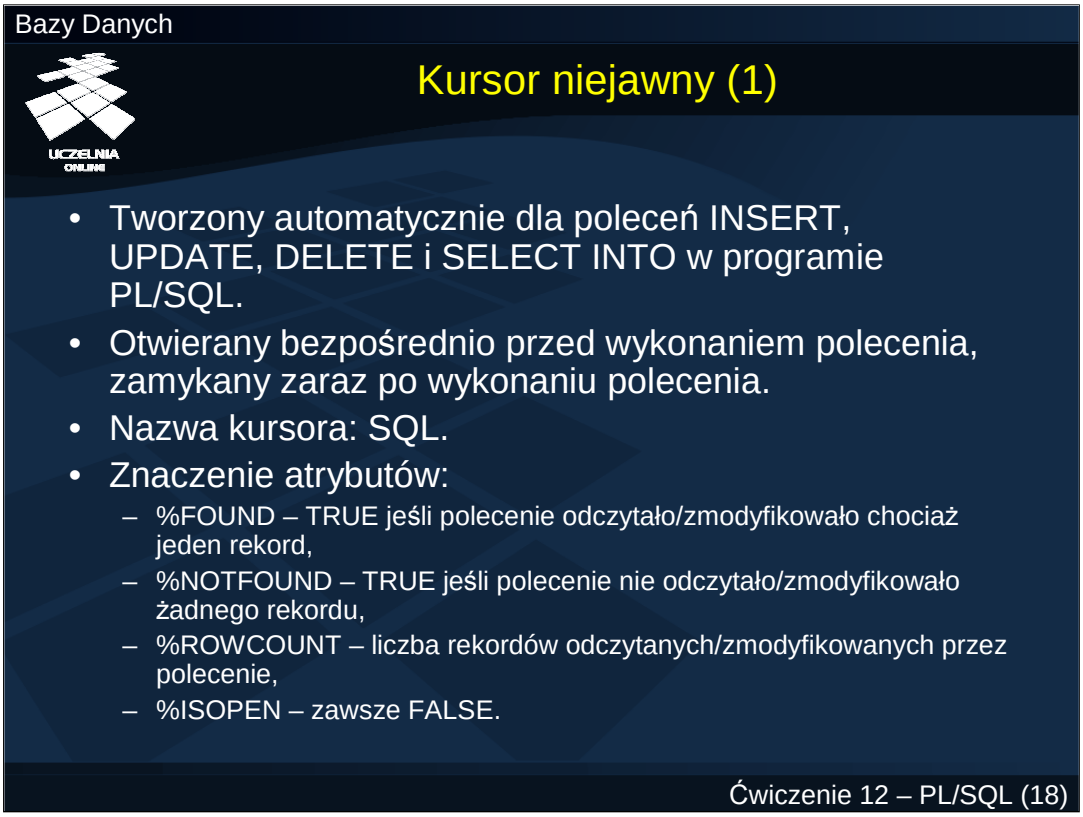
Przejdziemy teraz do omówienia konstrukcji, umożliwiających modyfikację i usuwanie rekordów z użyciem kursora.

Jak już wcześniej wspomniano, jeśli w deklaracji kursora umieszczono klauzulę FOR UPDATE, rekordy, odczytane z bazy danych przez kursor, zostają zablokowane i przygotowane do ewentualnej modyfikacji czy też usunięcia. Jeśli bieżący rekord kursora ma zostać zmodyfikowany, wówczas operację tą realizuje się, używając standardowego polecenia UPDATE, skierowanego do relacji, z której kursor odczytał rekord. Selekcję w relacji rekordu, który zostanie zmodyfikowany (musi to być bieżący rekord kursora), zapewnia użycie klauzuli WHERE CURRENT OF <nazwa kursora>. Analogicznie wygląda sytuacja, gdy chcemy usunąć z relacji rekord, będący bieżącym rekordem kursora. Stosujemy wówczas polecenie DELETE z klauzulą WHERE CURRENT OF <nazwa kursora>.



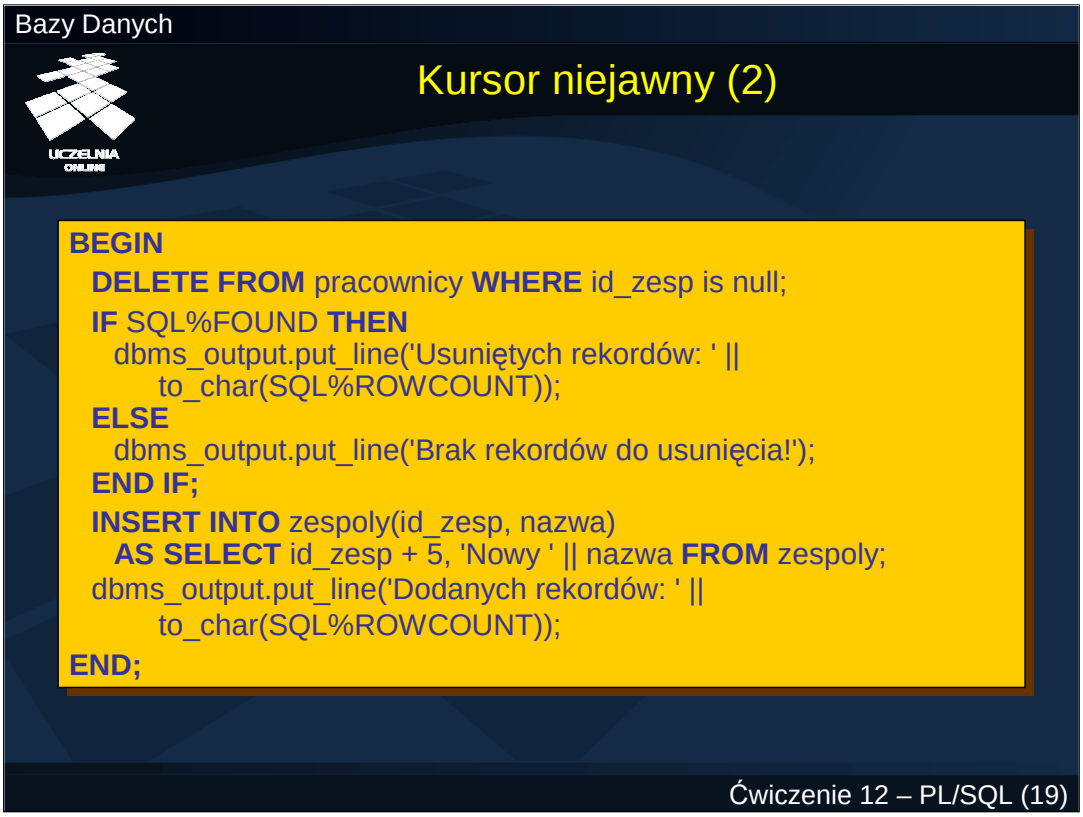
Przykład na bieżącym slajdzie prezentuje użycie klauzuli WHERE CURRENT OF.

Sparametryzowany kursor cur_pracownicy został zadeklarowany z klauzulą FOR UPDATE. Dodatkowo zadeklarowano zmienną liczbową o nazwie v_podwyzka. W sekcji wykonywalnej programu umieszczono pętlę FOR z kursorem. Wewnątrz pętli dla bieżącego rekordu kursora sprawdzana jest wartość atrybutu ETAT. Jeśli jest to „PROFESOR”, wówczas do zmiennej v_podwyzka przypisywana jest wartość 1.2, w przeciwnym wypadku v_podwyzka jest równa 1.1. Następnie dokonujemy modyfikacji rekordu relacji PRACOWNICY, który jest bieżącym rekordem kursora, zwiększając wartość atrybutu PLACA_POD o procent określony w zmiennej v_podwyzka. Polecenie modyfikujące to standardowe polecenie UPDATE, klauzula WHERE CURRENT OF cur_pracownicy znajduje w relacji PRACOWNICY rekord, który jest bieżącym rekordem kursora cur_pracownicy.



Omówimy teraz kursory niejawne. Są one automatycznie tworzone dla każdego polecenia INSERT, UPDATE, DELETE i SELECT INTO, jakie zostaje umieszczone w programie PL/SQL. Kursor niejawny nosi zawsze nazwę „SQL” i jest otwierany bezpośrednio przed wykonaniem polecenia i zamykany zaraz po jego zakończeniu. Kursora niejawnego używa się tylko do sprawdzenia stanu ostatnio realizowanego polecenia. Dla polecenia UPDATE możemy sprawdzić, ile rekordów zostało uaktualnionych, dla polecenia DELETE ile rekordów usunięto, dla polecenia INSERT – ile rekordów zostało wstawionych. W tym celu wykorzystuje się znane nam już atrybuty kursora. Atrybut %FOUND dla kursora niejawnego ma wartość TRUE jeśli ostatnie polecenie odczytało w przypadku SELECT INTO lub przetworzyło dla pozostałych poleceń chociaż jeden rekord, wartość FALSE gdy polecenie nie odczytało/nie przetworzyło żadnego rekordu. Wartości atrybutu %NOTFOUND są komplementarne w stosunku do wartości atrybutu %FOUND – TRUE jeśli ostatnie polecenie nie odczytało/nie przetworzyło żadnego rekordu, FALSE w przeciwnym wypadku. Jeśli interesuje nas liczba rekordów odczytanych/ przetworzonych przez ostatnie polecenie, posługujemy się atrybutem %ROWCOUNT. Atrybutu %ISOPEN dla kursora niejawnego nie używamy – ma on zawsze wartość FALSE.

Korzystając z atrybutów kursora niejawnego należy pamiętać, że opisują one stan ostatniego polecenia – każde polecenie ustawia na nowo wartości atrybutów.



Przykład na bieżącym slajdzie pokazuje użycie atrybutów kursora niejawnego.

Pierwsze polecenie usuwa z relacji PRACOWNICY rekordy, w których wartość atrybutu ID_ZESP jest pusta. Jeśli usunięto chociaż jeden rekord (wartość atrybutu %FOUND kursora niejawnego jest równa TRUE), na konsoli wypisana zostanie liczba usuniętych rekordów (wartość atrybutu %ROWCOUNT kursora niejawnego). W przeciwnym wypadku na konsoli pojawi się komunikat „Brak rekordów do usunięcia!”.

Dalej wykonywane jest polecenie wstawienia do relacji ZESPOLY wyniku podzapytania. Po wykonaniu operacji na konsoli wypisana zostanie liczba dodanych przez polecenie rekordów (wartość atrybutu %ROWCOUNT kursora niejawnego).

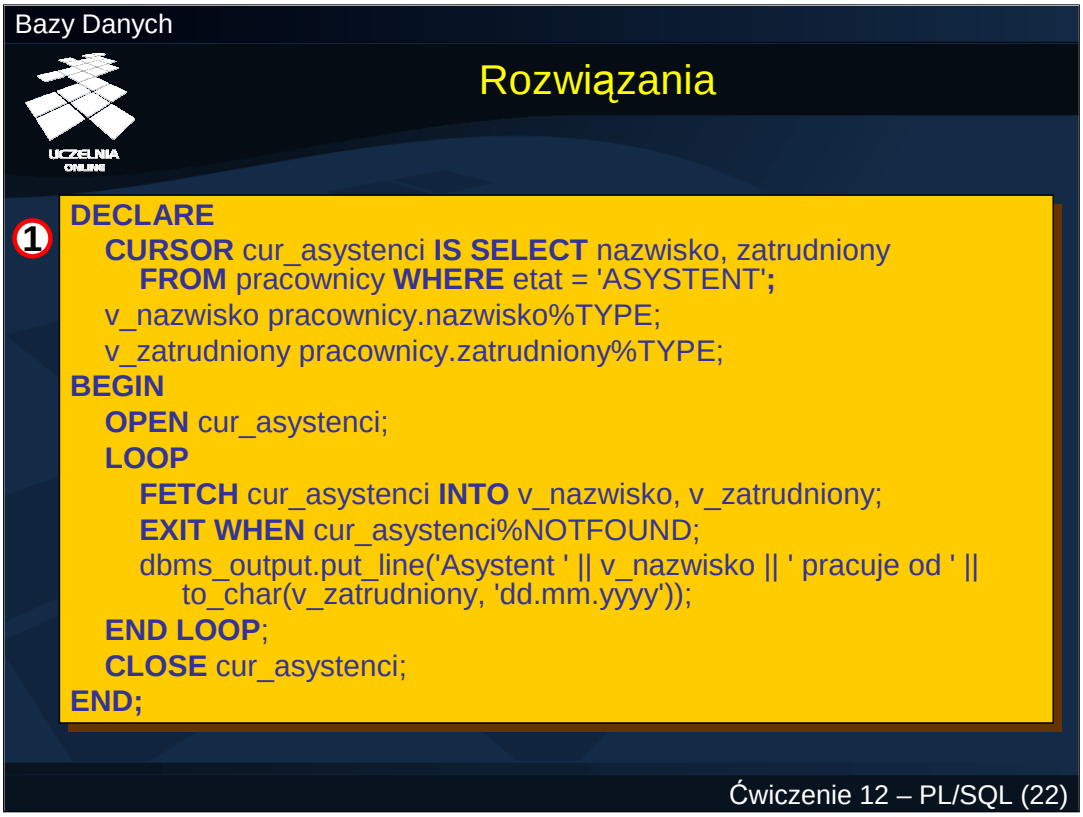
Zadania

1. Zdefiniuj kursor zawierający nazwiska i daty zatrudnienia wszystkich asystentów. Posłuż się tym kursorem do wyświetlenia rekordów w formie zdań „Asystent <nazwisko> pracuje od <data_zatrudnienia>”. Posłuż się poleceniem FETCH.
2. Zdefiniuj kursor, dzięki któremu będzie można wyświetlić trzech najlepiej zarabiających pracowników. Posłuż się atrybutem kursora %ROWCOUNT.

Bieżący slajd rozpoczyna zbiór zadań, których celem jest utrwalenie wiadomości o kursorach w programach PL/SQL.

Zadania

3. Zdefiniuj kursor, który posłuży do dokonania następującej modyfikacji: pracownikom zespołu „ALGORYTMY” podnieś płacę dodatkową o 100 złotych, pracownikom zespołu „ADMINISTRACJA” podnieś płacę dodatkową o 150 złotych, w pozostałych zespołach usuń doktorantów.
4. Napisz program, który zapyta się użytkownika o etat a następnie wyświetli nazwiska wszystkich pracowników posiadających dany etat. Zastosuj pętlę FOR z kursorem sparametryzowanym.

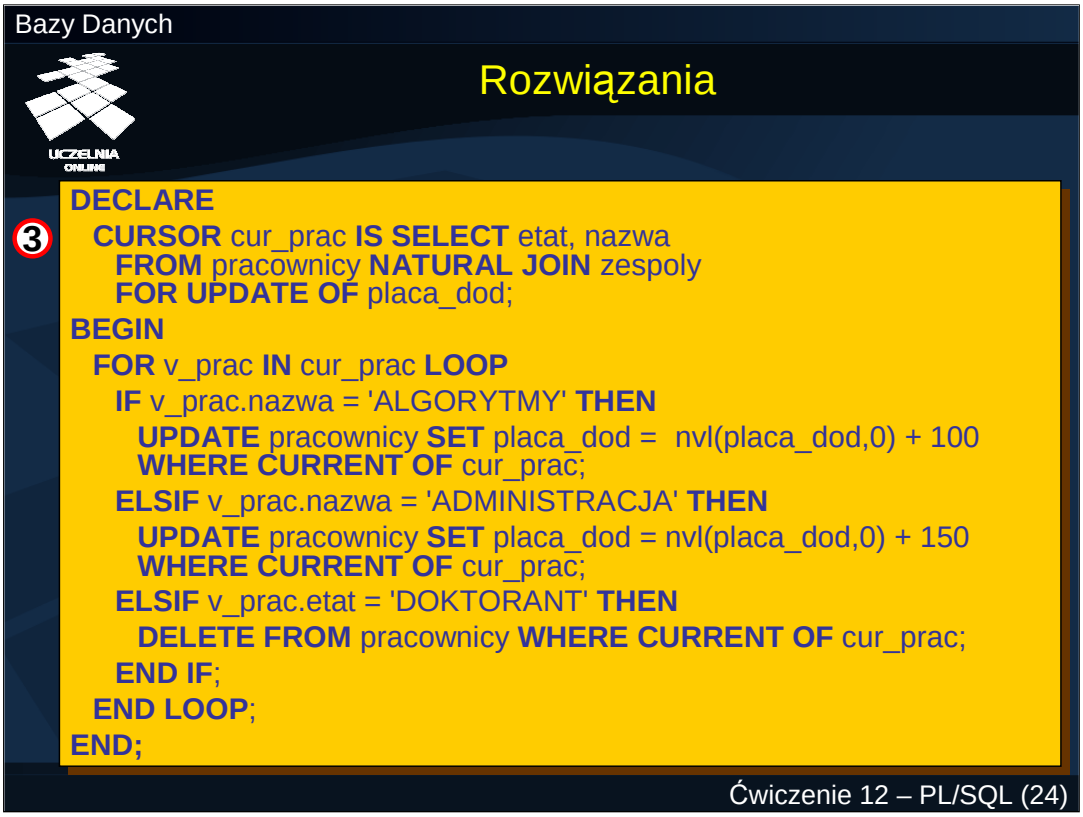


Bieżący slajd przedstawia rozwiązanie zadania (1), którego treść zacytowano poniżej.

(1) Zdefiniuj kursor zawierający nazwiska i daty zatrudnienia wszystkich asystentów. Posłuż się tym kursorem do wyświetlenia rekordów w formie zdań „Asystent <nazwisko> pracuje od <data_zatrudnienia>”. Posłuż się poleceniem FETCH.

Bieżący slajd przedstawia rozwiązanie zadania (2), którego treść zacytowano poniżej.

(2) Zdefiniuj kursor, dzięki któremu będzie można wyświetlić trzech najlepiej zarabiających pracowników. Posłuż się atrybutem kursora %ROWCOUNT.

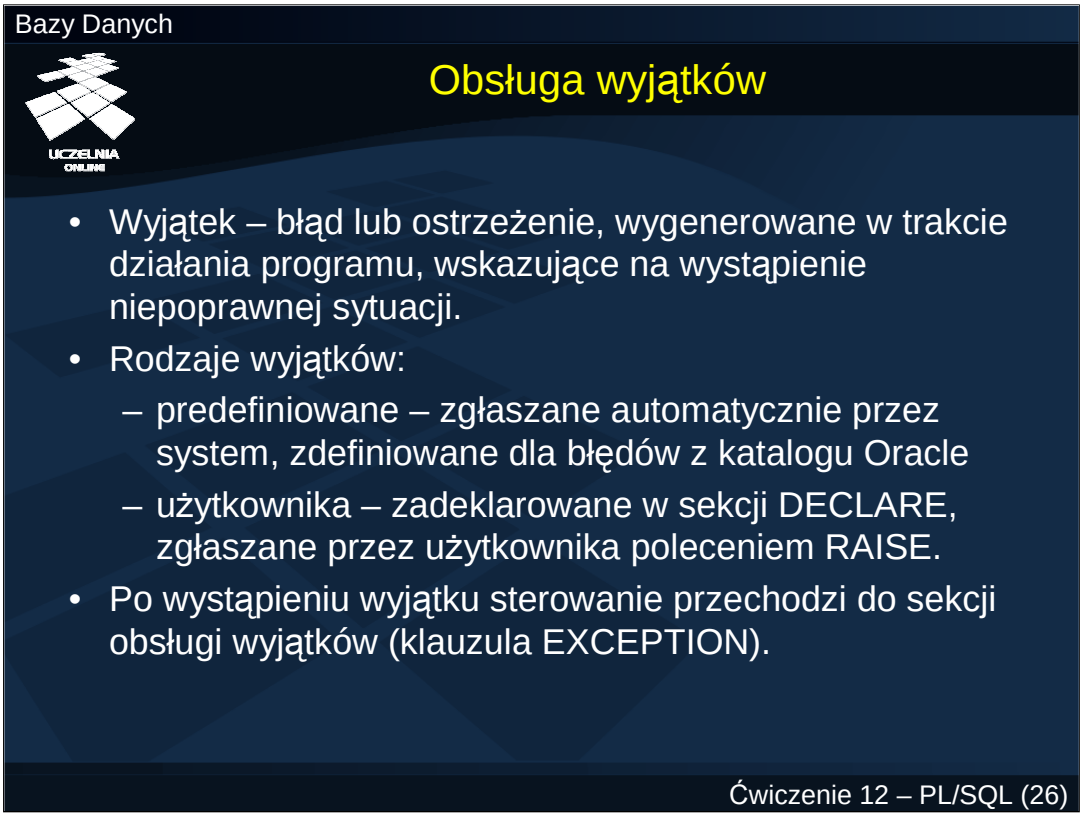


Bieżący slajd przedstawia rozwiązanie zadania (3), którego treść zacytowano poniżej.

(3) Zdefiniuj kursor, który posłuży do dokonania następującej modyfikacji: pracownikom zespołu „ALGORYTMY” podnieś płacę dodatkową o 100 złotych, pracownikom zespołu „ADMINISTRACJA” podnieś płacę dodatkową o 150 złotych, w pozostałych zespołach usuń doktorantów.

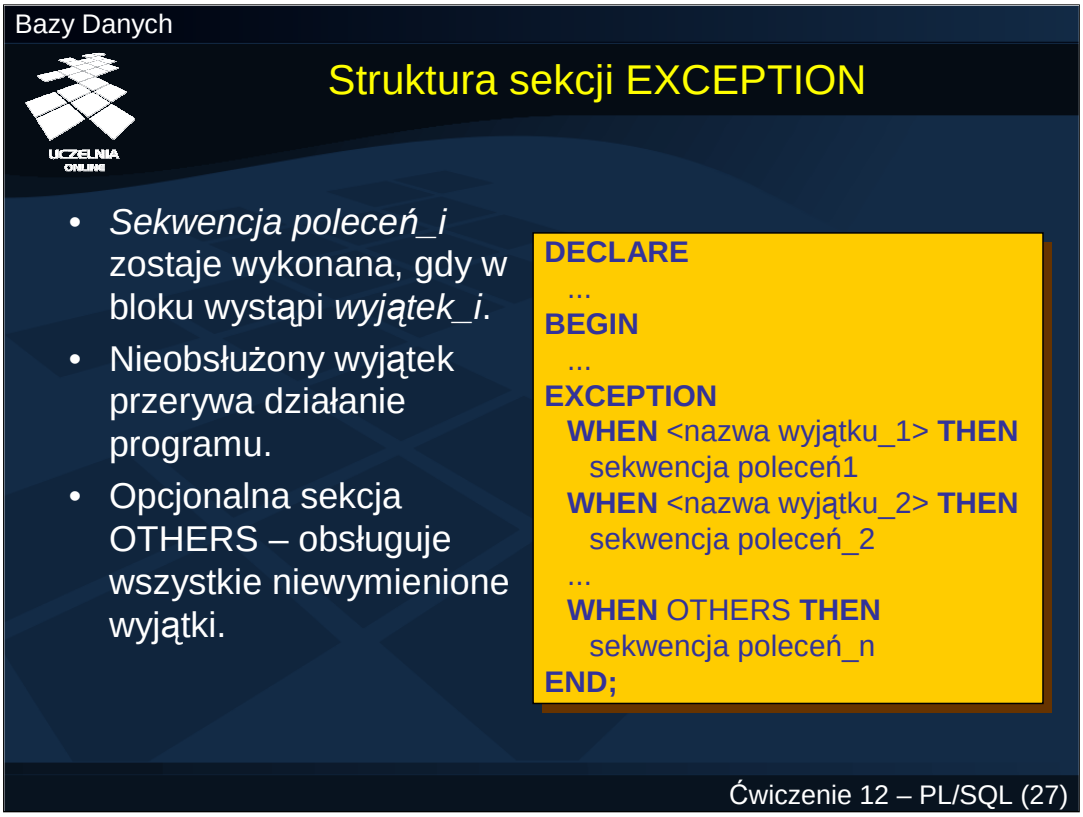
Bieżący slajd przedstawia rozwiązanie zadania (4), którego treść zacytowano poniżej.

(4) Napisz program, który zapyta się użytkownika o żądany etat a następnie wyświetli nazwiska wszystkich pracowników posiadających dany etat. Zastosuj pętlę FOR z kursorem sparametryzowanym.

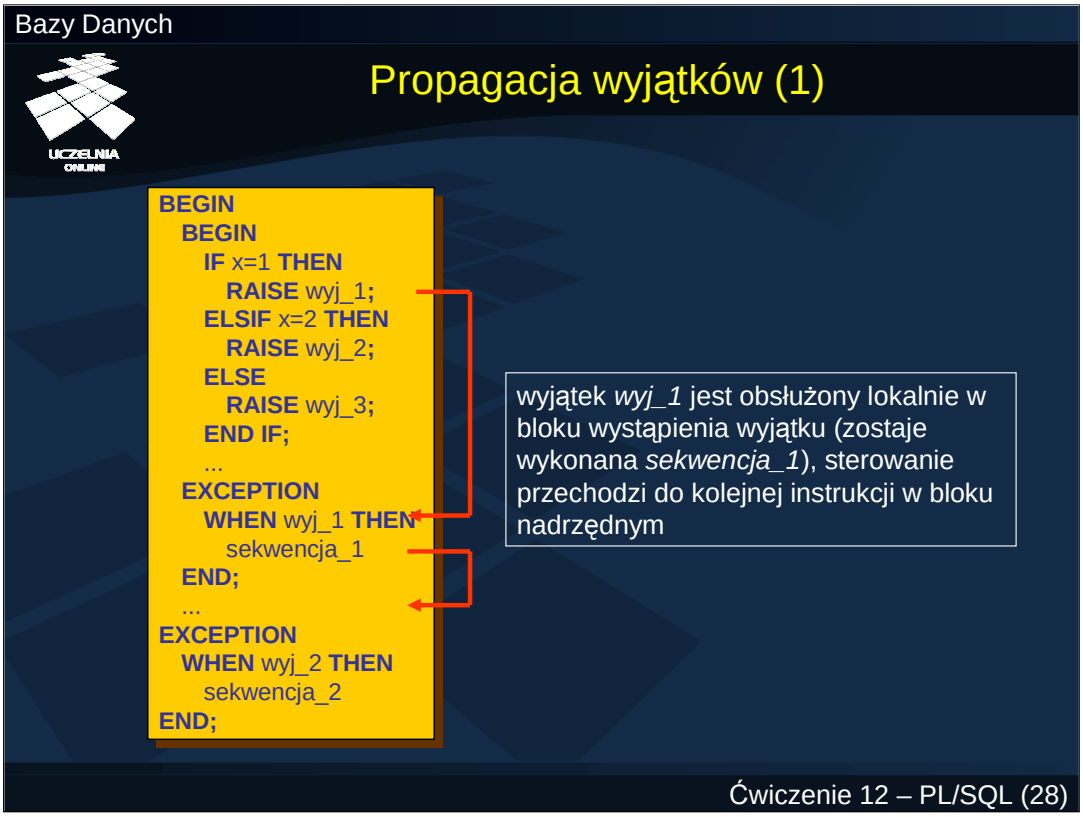


Przejdziemy teraz do omawiania zagadnień związanych z obsługą błędów, jakie mogą pojawić się w trakcie działania programu PL/SQL. Źródłem błędów mogą być niepoprawne operacje (np. próba wstawienia do relacji rekordu z wartościami powielającymi istniejące już w kluczu unikalnym wartości, dzielenie przez 0), awarie SZBD lub utrata połączenia z SZBD. SZBD Oracle ma zdefiniowany tzw. katalog błędów, w którym każdy błąd ma przypisany numer i odpowiedni komunikat (np. błąd ORA-00060 z komunikatem „Wykryto zakleszczenie w trakcie oczekiwania na przydzielenie zasobu” czy też ORA-02264 „Nazwa używana przez istniejące ograniczenie”).

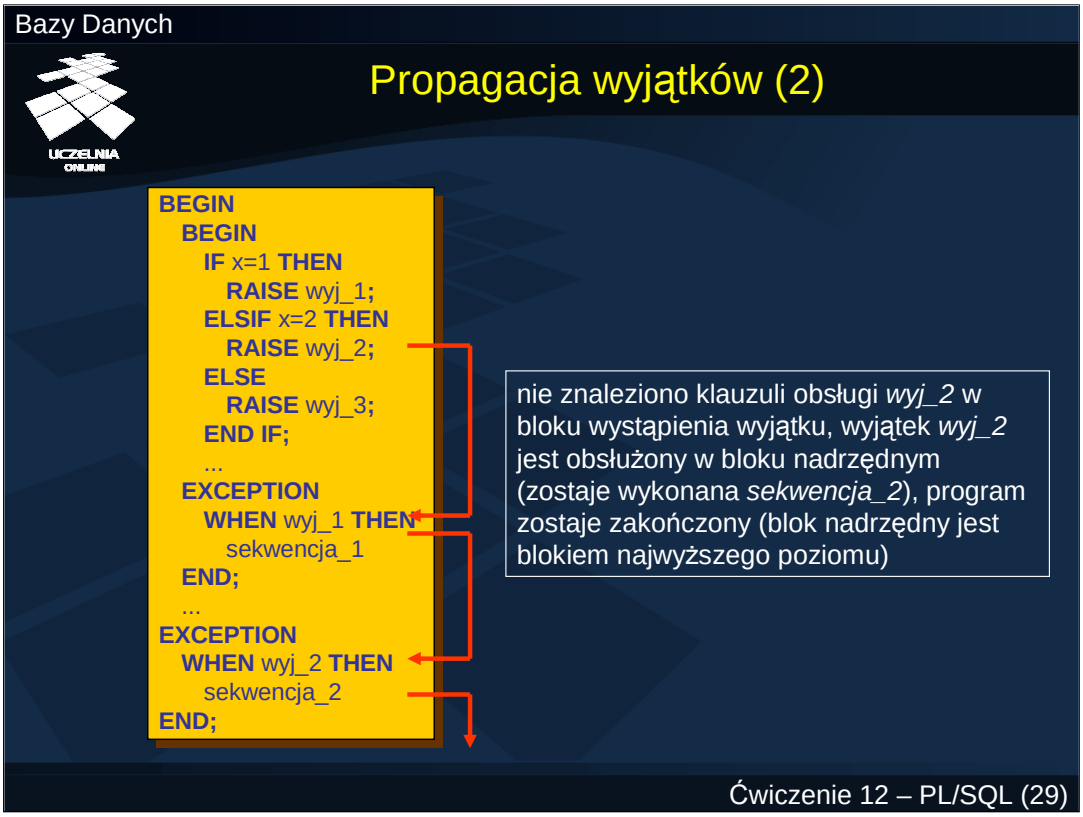
Wystąpienie błędu w trakcie działania programu PL/SQL domyślnie powoduje przerwanie programu i wyświetlenie na konsoli komunikatu o wystąpieniu błędu. Programista ma jednak możliwość konstrukcji programu PL/SQL w taki sposób, aby w razie wystąpienia błędu program nie został przerwany, ale wykonał jakąś alternatywną akcję w postaci sekwencji poleceń. Jest to możliwe dzięki istnieniu predefiniowanych wyjątków, będących „nazwami” dla najczęstszych błędów z katalogu błędów SZBD Oracle. W razie wystąpienia w trakcie działania programu błędu, dla którego istnieje predefiniowany wyjątek, odpowiednio skonstruowany program może ów wyjątek „przechwycić”, nie dopuszczając do przerwania programu i wykonując polecenia, umieszczone w sekcji obsługi wyjątków, rozpoczynającej się od słowa kluczowego EXCEPTION. Oprócz wyjątków predefiniowanych programista może użyć swoich własnych wyjątków, jednak sam musi zapewnić ich wywoływanie w programie przez użycie odpowiedniego polecenia.



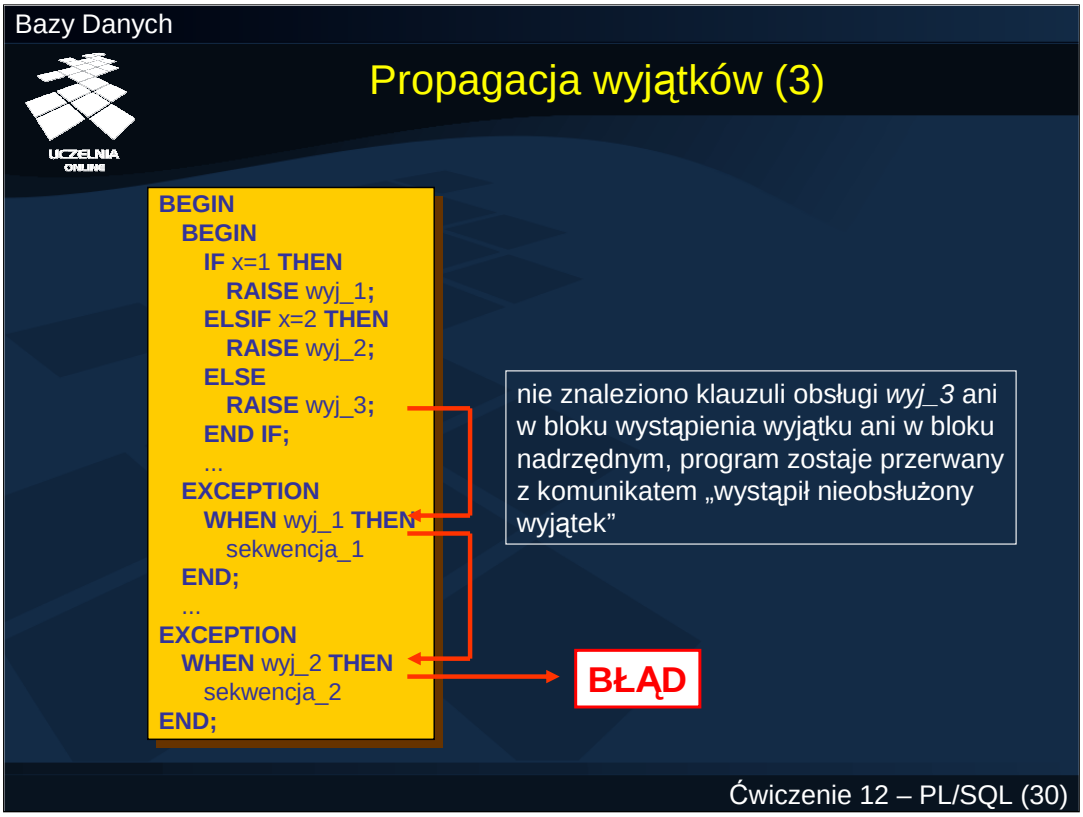
Na bieżącym slajdzie zaprezentowano strukturę sekcji obsługi wyjątków. Sekcja ta rozpoczyna się słowem kluczowym EXCEPTION i występuje na samym końcu bloku PL/SQL, po wszystkich poleceniach części wykonywalnej bloku. Sekcja obsługi wyjątków składa się z szeregu podsekcji (minimum jednej), z których każda odpowiada za obsługę określonego wyjątku. Podsekcja rozpoczyna się słowem kluczowym WHEN po którym umieszcza się nazwę wyjątku, wystąpienie którego uruchamia daną podsekcję, natomiast po słowie THEN znajduje się sekwencja poleceń (również np. blok zagnieżdżony), która ma zostać wykonana w razie wystąpienia wyjątku. Jedna podsekcja może również być związana z kilkoma wyjątkami, w taki przypadku po słowie WHEN umieszczamy nazwy wyjątków połączone spójnikami logicznymi OR (np. WHEN wyj_1 OR wyj_2 OR wyj_3 THEN ...). Opcjonalna podsekcja WHEN OTHERS zostaje uruchomiona dla wszystkich wyjątków, które wystąpiły w trakcie działania programu, a dla których nie zdefiniowano indywidualnych podsekcji. Sekcja obsługi wyjątków kończy się słowem kluczowym END, kończącym również cały blok PL/SQL. Jeśli w programie wystąpi wyjątek, dla którego nie zdefiniowano podsekcji WHEN i nie istnieje również podsekcja WHEN OTHERS, wyjątek taki przerywa działanie programu. Na następnym slajdzie dokładnie przyjrzymy się procesowi propagacji wyjątków.



Na powyższym slajdzie zdefiniowano przykładowy blok PL/SQL („blok zewnętrzny”), w którym zagnieżdżono inny blok („blok wewnętrzny”). Oba bloki posiadają zdefiniowane sekcje obsługi wyjątków, blok wewnętrzny z podsekcją dla wyjątku wyj_1, blok zewnętrzny z podsekcją dla wyjątku wyj_2. Zakładamy teraz, że zmienna x w bloku wewnętrznym ma wartość 1. Zostaje spełniony warunek x = 1, wykonane zostaje polecenie RAISE wyj_1, powodujące wygenerowanie w bloku wewnętrznym wyjątku wyj_1 (polecenie RAISE zostanie dokładnie przedstawione przy omawianiu wyjątków użytkownika). Sterowanie w bloku wewnętrznym zostaje przekazane do sekcji obsługi wyjątków tego bloku, tam zostaje znaleziona podsekcja odpowiedzialna za obsługę wyjątku wyj_1 i wykonane zostają polecenia określone jako sekwencja_1. Następnie sterowanie przechodzi do pierwszego polecenia po bloku, w którym została znaleziona podsekcja obsługi wyjątku, a więc do pierwszego polecenia bloku zewnętrznego, umieszczonego zaraz po słowie END, kończącym blok wewnętrzny. Program kontynuuje wykonanie.

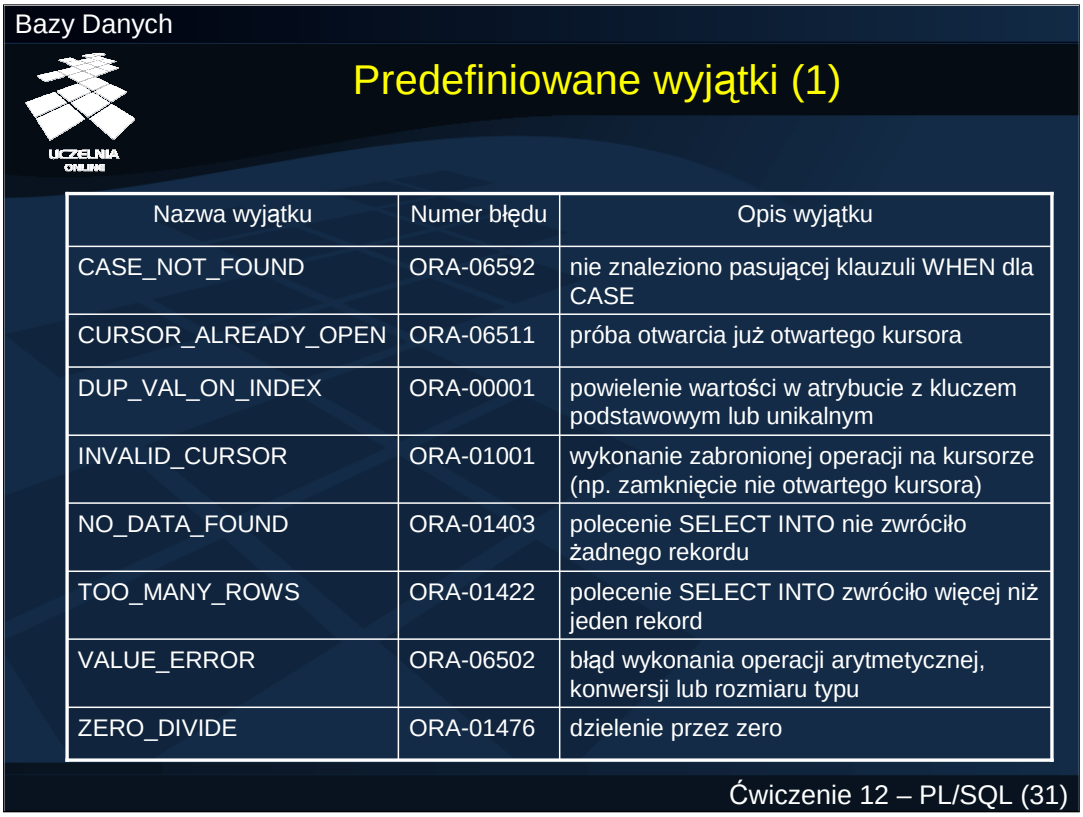


Załóżmy teraz, że zmienna x ma wartość 2, a więc wygenerowany został wyjątek wyj_2. Sterowanie przechodzi do sekcji obsługi wyjątków w bloku wewnętrznym, jednak nie ma tam zdefiniowanej podsekcji do obsługi wyjątku wyj_2. Sterowanie przechodzi do sekcji obsługi wyjątków bloku zewnętrznego, tam zostaje znaleziona podsekcja obsługi wyjątku wyj_2, zostaje wykonana sekwencja poleceń sekwencja_2. Dalej sterowanie jest przekazywane do pierwszej instrukcji po bloku, w którym znaleziono pasującą podsekcję obsługi wyjątku, a więc do pierwszej instrukcji po słowie END kończącym blok zewnętrzny. Jednak blok zewnętrzny nie jest blokiem zagnieżdżonym w innym bloku, nie ma więc żadnej dalszej instrukcji. Program zostaje zakończony, nie ma żadnego komunikatu o wystąpieniu błędu.



Rozważmy ostatnią sytuację. Teraz zmienna x ma wartość różną od 1 i 2, dlatego zostaje wygenerowany wyjątek wyj_3. Sterowanie przechodzi do sekcji obsługi wyjątków bloku, w którym wystąpił wyjątek, czyli bloku wewnętrznego, tam jednak nie ma podsekcji obsługi wyjątku wyj_3. Sterowanie jest przekazywane do sekcji obsługi wyjątków bloku zewnętrznego, ale i tam nie ma pasującej podsekcji obsługi wyjątku wyj_3. Program zostaje przerwany z komunikatem o wystąpieniu nieobsługiwanego wyjątku.

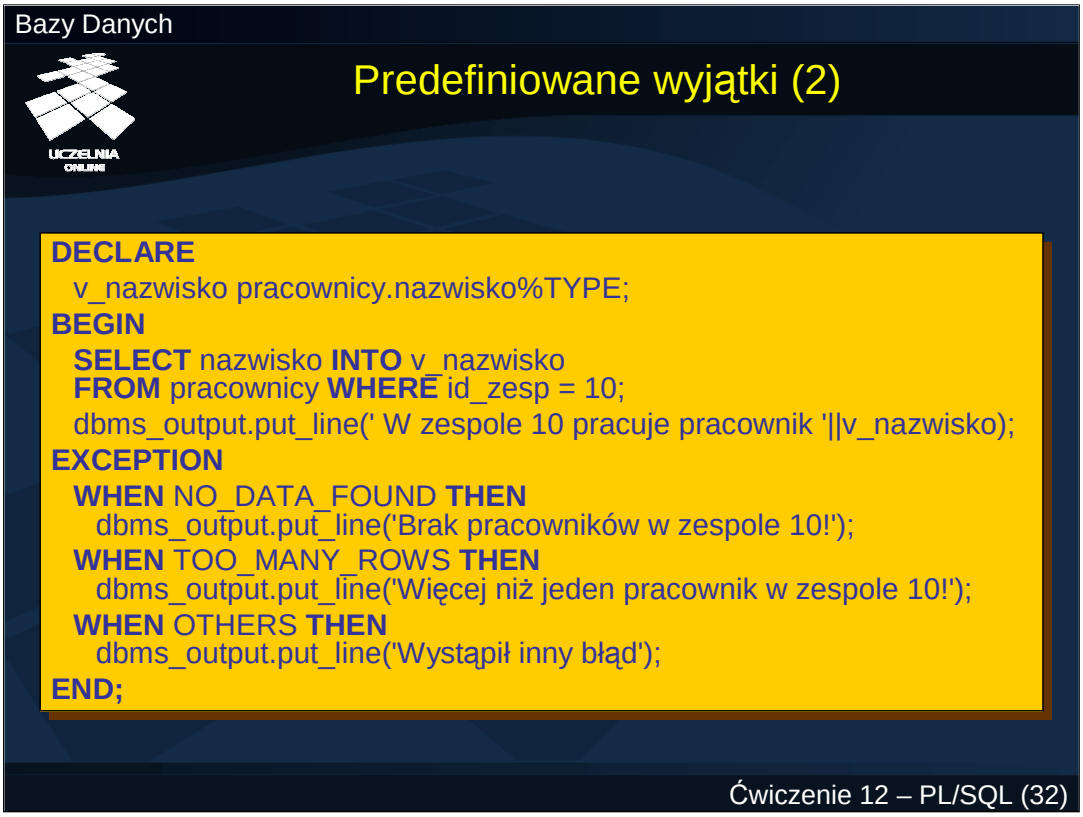
Zaprezentowane przykłady powinny wyjaśnić ogólną zasadę propagacji wyjątków. Jeśli w bloku wystąpił wyjątek, sterowanie jest przekazywane do sekcji obsługi wyjątków tego bloku, tam poszukiwana jest podsekcja odpowiedzialna za obsługę danego wyjątku. Jeśli podsekcja zostanie znaleziona, wykonywana jest sekwencja poleceń w niej umieszczona. Następnie sterowanie przechodzi do pierwszej instrukcji po bloku, w którym obsłużono wyjątek. W przypadku nie znalezienia pasującej podsekcji obsługi wyjątków, poszukiwanie pasującej podsekcji jest kontynuowane w sekcji obsługi wyjątków bloku nadrzędnego, itd. Jeśli blok nie ma bloku nadrzędnego lub przeszukano sekcje obsługi wyjątków wszystkich bloków i nie znaleziono żadnej pasującej podsekcji, program zostaje przerywany z komunikatem o błędzie.



Nazwa wyjątku	Numer błędu	Opis wyjątku
CASE_NOT_FOUND	ORA-06592	nie znaleziono pasującej klauzuli WHEN dla CASE
CURSOR_ALREADY_OPEN	ORA-06511	próba otwarcia już otwartego kursora
DUP_VAL_ON_INDEX	ORA-00001	powielenie wartości w atrybucie z kluczem podstawowym lub unikalnym
INVALID_CURSOR	ORA-01001	wykonanie zabronionej operacji na kursorze (np. zamknięcie nie otwartego kursora)
NO_DATA_FOUND	ORA-01403	polecenie SELECT INTO nie zwróciło żadnego rekordu
TOO_MANY_ROWS	ORA-01422	polecenie SELECT INTO zwróciło więcej niż jeden rekord
VALUE_ERROR	ORA-06502	błąd wykonania operacji arytmetycznej, konwersji lub rozmiaru typu
ZERO_DIVIDE	ORA-01476	dzielenie przez zero

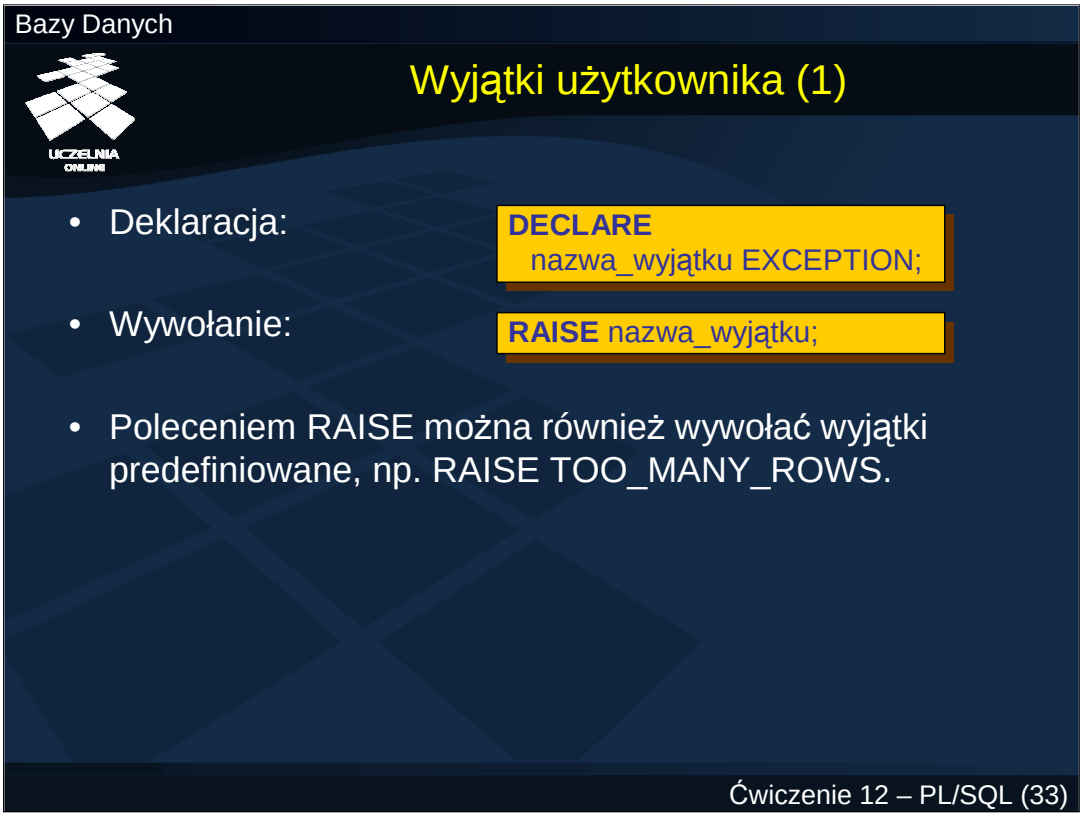
Na bieżącym slajdzie przedstawiono niektóre z predefiniowanych wyjątków SZBD Oracle wraz z numerami błędów odpowiadających błędów z katalogu błędów.

Wyjątek CASE_NOT_FOUND zostaje wygenerowany w przypadku, gdy dla wyrażenia w poleceniu CASE nie znaleziono żadnej pasującej klauzuli WHEN a w wyrażeniu nie zdefiniowano klauzuli ELSE. Następny wyjątek, CURSOR_ALREADY_OPEN, powstaje w przypadku, gdy programista próbuje poleceniem OPEN otworzyć kursor, który już jest otwarty. Z kolei wyjątek INVALID_CURSOR zostaje wygenerowany w przypadku wykonania zabronionej operacji na kursorze, np. próba pobrania rekordu z kursora, który nie został otwarty. Wyjątek DUP_VAL_ON_INDEX powstaje w sytuacji, gdy wykonanie zlecenia INSERT lub UPDATE powoduje powielenie wartości w atrybucie (atrybutach) wchodzącym w skład klucza podstawowego lub unikalnego. Dwa następne wyjątki, NO_DATA_FOUND i TOO_MANY_ROWS, są związane z poleceniem SELECT INTO i zostają wygenerowane w przypadku, gdy polecenie, odpowiednio, nie zwróci żadnego rekordu lub zwróci więcej niż jeden rekord. Wyjątek VALUE_ERROR powstaje w przypadku wykonania błędnej operacji arytmetycznej, błędnej operacji konwersji lub w przypadku próby przypisania do obiektu wartości, która przekracza rozmiar typu obiektu. Ostatni z prezentowanych wyjątków, ZERO_DIVIDE, zostaje wygenerowany w przypadku wykonania w programie dzielenia przez zero.

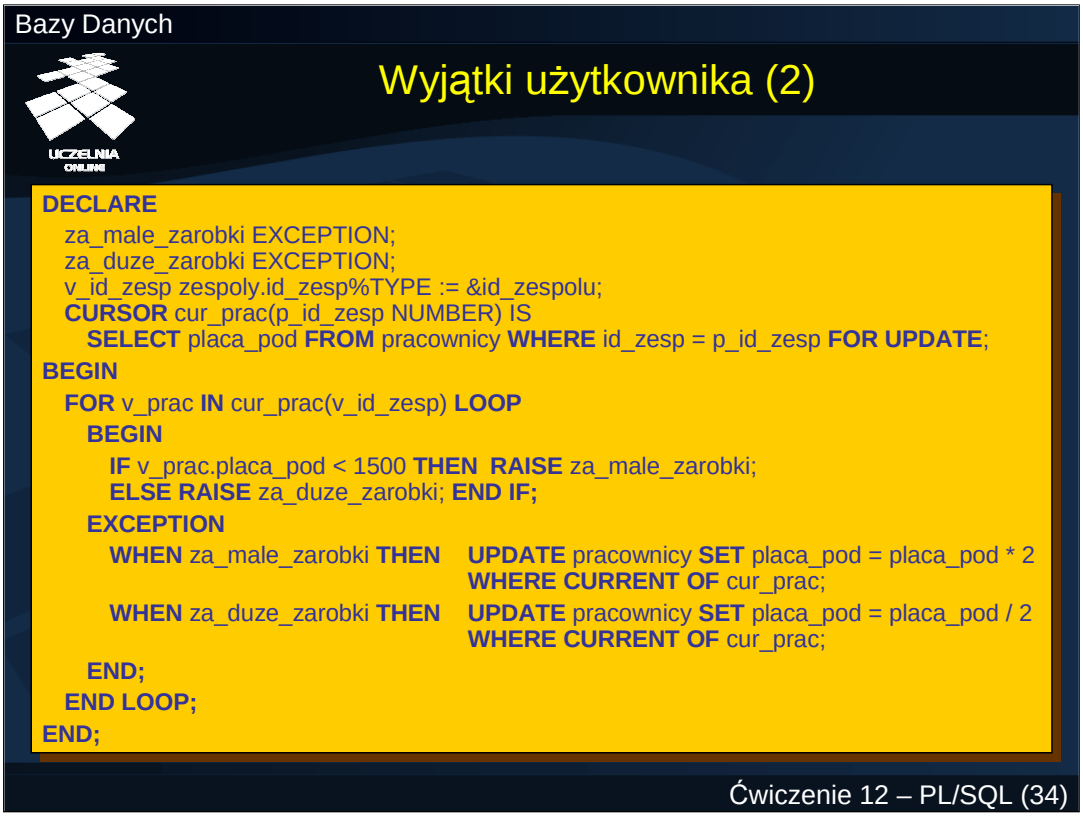


Powyższy slajd prezentuje przykład użycia predefiniowanych wyjątków. W sekcji deklaracyjnej programu zadeklarowano zmienną v_nazwisko. Pierwsze polecenie w sekcji wykonywalnej, SELECT INTO, próbuje odczytać nazwisko pracownika, należącego do zespołu o numerze 10. Nazwisko pracownika ma zostać zapamiętane w zmiennej v_nazwisko. Jeśli w zespole o numerze 10 jest tylko jeden pracownik, odczyt powiedzie się i zostanie wykonana następna operacja, która wypisze na konsoli tekst „W zespole 10 pracuje pracownik <nazwisko pracownika>”. Następnie program zakończy działanie. Jeśli w zespole 10 nie ma żadnego pracownika, polecenie SELECT INTO wygeneruje wyjątek NO_DATA_FOUND. Sterowanie zostanie przekazane do sekcji obsługi wyjątków, mamy tam zdefiniowaną podsekcję dla wygenerowanego wyjątku. Z podsekcją związane jest tylko jedno polecenie, wypisze ono na konsoli komunikat „Brak pracowników w zespole 10”, następnie program zostanie zakończony. Jeśli w zespole 10 jest więcej niż jeden pracownik, polecenie SELECT INTO wygeneruje wyjątek TOO_MANY_ROWS. Sterowanie przejdzie do sekcji obsługi wyjątków, tam znajdzie podsekcję odpowiedzialną za ten wyjątek, polecenie w tej sekcji wypisze na konsoli tekst „Więcej niż jeden pracownik w zespole 10”. Program zostanie zakończony.

W sekcji obsługi wyjątków przykładowego bloku zdefiniowano dodatkową klauzulę WHEN OTHERS, która obsłuży ewentualne inne wyjątki, które zostaną wygenerowane podczas działania programu.

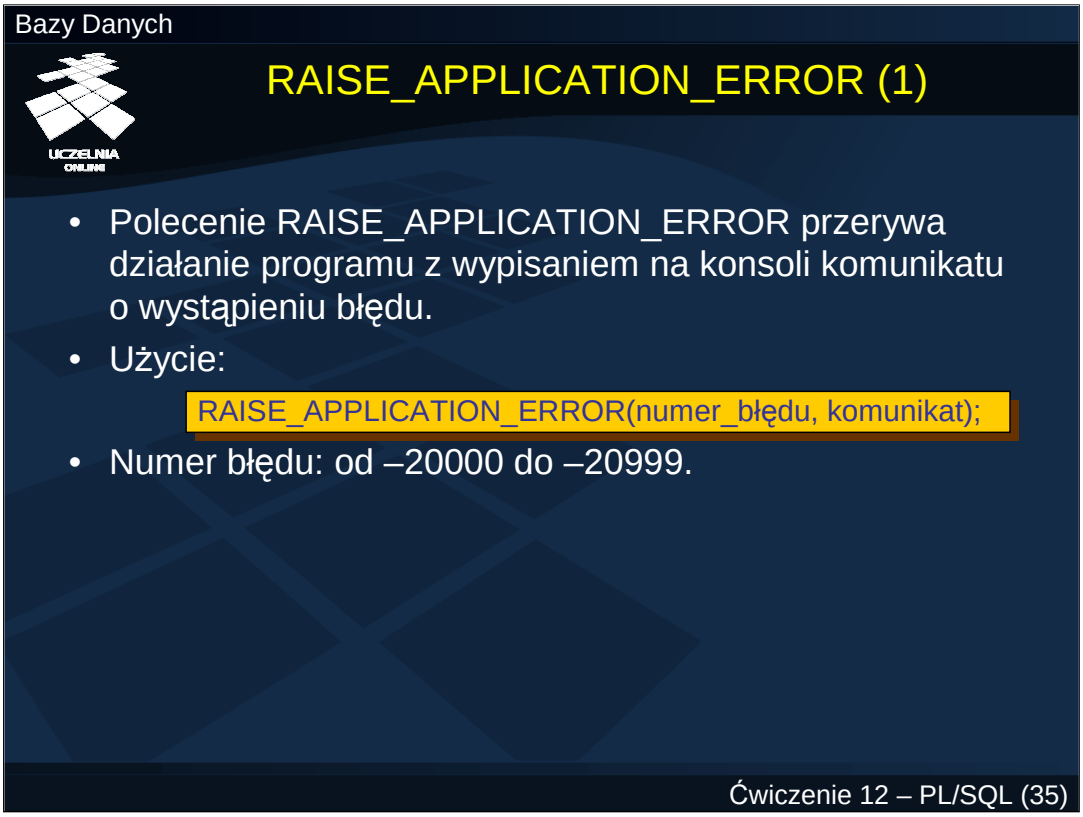


Przejdziemy teraz do omawiania wyjątków użytkownika. Taki wyjątek musi zostać zadeklarowany w sekcji deklaracyjnej bloku PL/SQL. Postać deklaracji to nazwa wyjątku, za którym umieszcza się słowo EXCEPTION. Wyjątki użytkownika nigdy nie są zgłaszane automatycznie – nie ma żadnego zdarzenia, z którym byłyby one związane. To sam użytkownik musi taki wyjątek zgłosić – kiedy to zrobi zależy to od logiki programu. Poleceniem zgłaszającym wyjątek użytkownika jest polecenie RAISE, za którym umieszczamy nazwę zgłaszanego wyjątku. Działanie programu w przypadku zgłoszenia wyjątku użytkownika jest takie samo jak w przypadku wystąpienia wyjątku predefiniowanego – sterowanie zostaje przekazane do sekcji obsługi wyjątków. Należy wspomnieć, że poleceniem RAISE można w programie również zgłosić wyjątki predefiniowane.

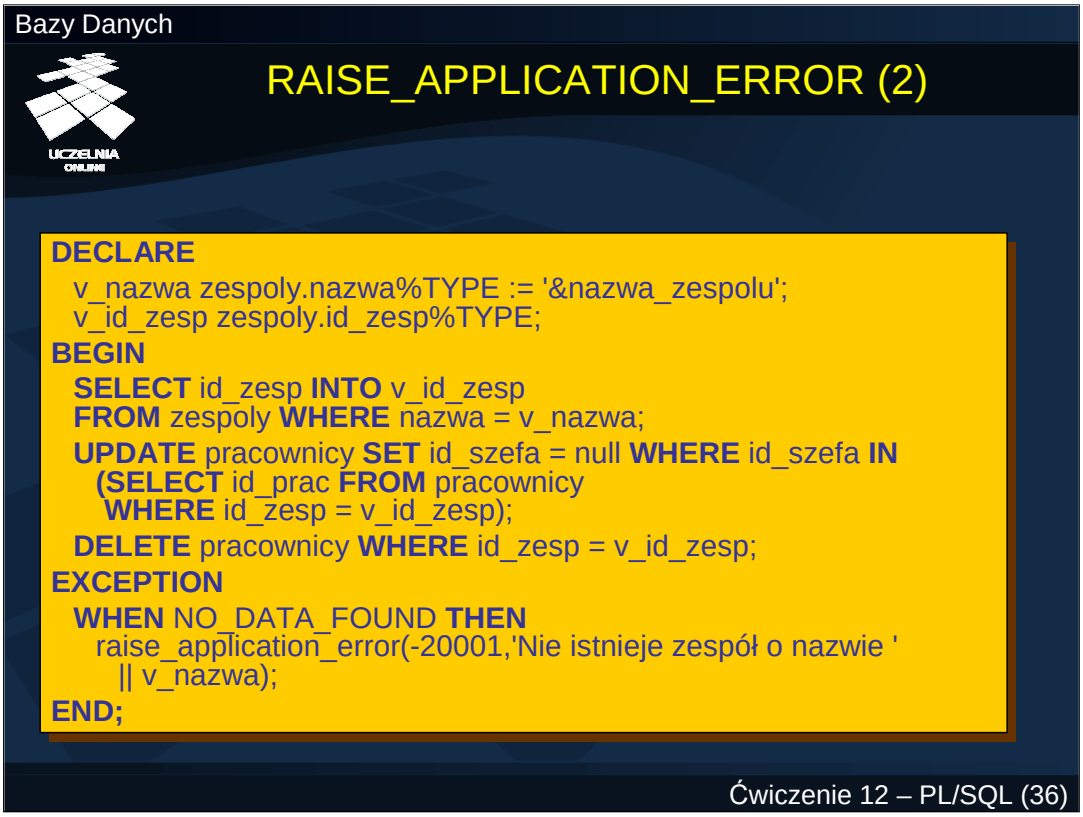


Bieżący slajd prezentuje zastosowanie wyjątków użytkownika. W sekcji deklaracji bloku zadeklarowano: dwa wyjątki: za_male_zarobki i za_duze_zarobki, zmienną v_id_zesp, inicjalizowaną wartością podane przez użytkownika, oraz sparametryzowany kursor cur_prac, przeglądający płace podstawowe pracowników z zespołu o numerze przekazanym przez parametr (zwróćmy uwagę, że kursor przygotowuje zbiór rekordów do modyfikacji – zawiera klauzulę FOR UPDATE). Omówmy część wykonywalną programu. Rekordy kursora cur_prac są przeglądane w pętli FOR. Zauważmy, że wewnątrz pętli FOR mamy zagnieżdżony blok. Instrukcja warunkowa porównuje wartość atrybutu PLACA_POD rekordu kursora z wartością 1500, jeśli wartość placa_pod jest mniejsza od 1500, zostaje wygenerowany wyjątek za_male_zarobki, w przeciwnym wypadku zostaje wygenerowany wyjątek za_duze_zarobki. Sterowanie zostaje przekazane do sekcji obsługi wyjątków bloku zagnieżdżonego w pętli FOR, tam zostaje dopasowana podsekcja do odpowiedniego wyjątku. W przypadku wyjątku za_male_zarobki płaca podstawowa bieżącego pracownika zostaje podwojona, w przypadku wyjątku za_duze_zarobki – zmniejszona o połowę. Po aktualizacji zostaje zrealizowana kolejna iteracja pętli.

Zauważmy, że przykład został tak skonstruowany, że wyjątek nie przerywa wykonania programu, po obsłudze wyjątku program kontynuuje działanie.



Bieżący slajd przedstawia polecenie RAISE_APPLICATION_ERROR, umożliwiające przerwanie programu z wypisaniem na konsoli komunikatu o błędzie. RAISE_APPLICATION_ERROR jest procedurą o dwóch parametrach. Pierwszy z nich określa numer błędu, z jakim zostanie przerwany program. Numer ten musi zawierać się w przedziale od –20000 do –20999. Numer ten wybiera sam użytkownik, przez co może utworzyć swój własny katalog błędów. Drugi parametr to komunikat, jaki zostanie wyświetlony na konsoli po przerwaniu programu. Przykładowo, jeśli wykonana zostanie procedura w postaci: RAISE_APPLICATION_ERROR(-20001, 'Mój błąd'), na konsoli pojawi się komunikat: „Błąd ORA-20001. Mój błąd”.

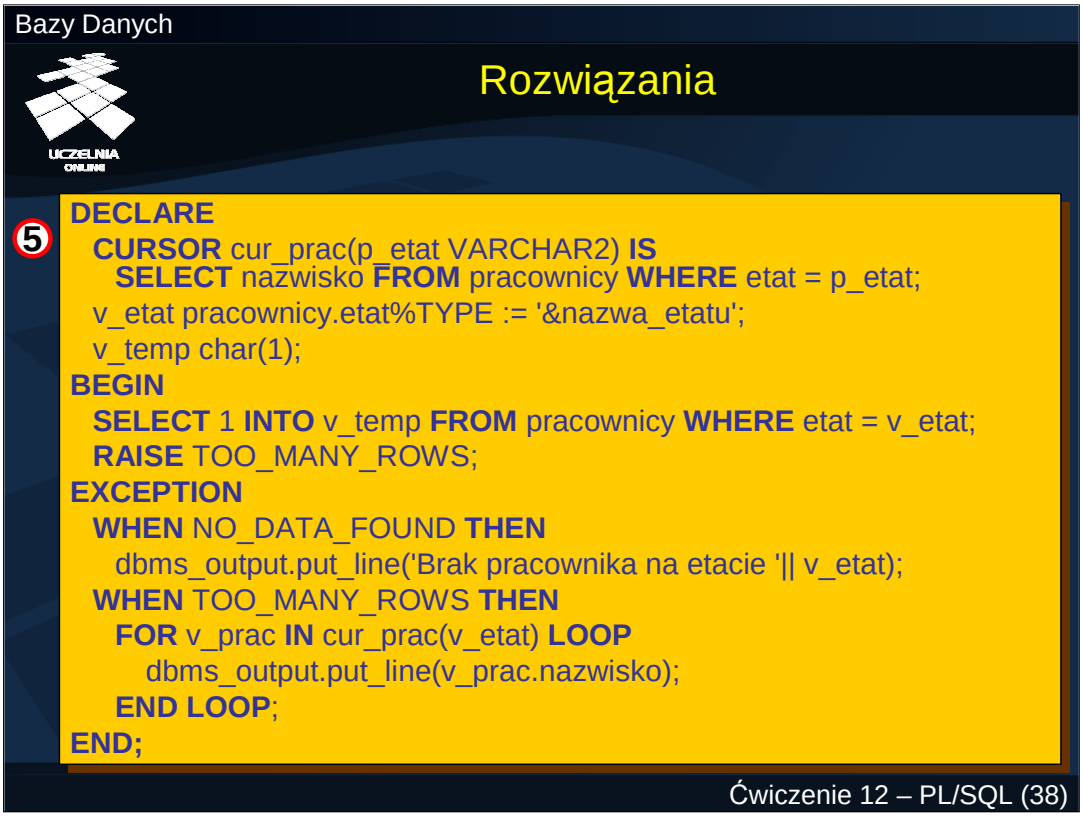


Bieżący slajd prezentuje przykład użycia procedury RAISE_APPLICATION_ERROR. W sekcji deklaracyjnej bloku zadeklarowano dwie zmienne: v_nazwa (inicjowana przez użytkownika) i v_id_zesp. Pierwsze polecenie programu próbuje odczytać z bazy danych identyfikator zespołu, którego nazwę podał użytkownik. Jeśli odczyt się powiedzie, identyfikator zespołu zostaje zapisany w zmiennej v_id_zesp. Następnie wszystkim pracownikom, których szefowie należą do zespołu, którego numer obecnie znajduje się w v_id_zesp, usuwamy wskazania szefów (ustawiamy wartość pustą w atrybucie ID_SZEFA), a dalej usuwamy wszystkich pracowników z zespołu o numerze w v_id_zesp. Jeśli natomiast operacja odczytu identyfikatora zespołu, którego nazwę podał użytkownik, nie zakończyła się powodzeniem (np. użytkownik podał nazwę nieistniejącego zespołu), generowany jest wyjątek NO_DATA_FOUND. Sterowanie zostaje przekazane do sekcji obsługi wyjątków, tam zostaje znaleziona podsekcja dla wyjątku NO_DATA_FOUND z poleceniem RAISE_APPLICATION_ERROR, które przerywa działanie programu zgłoszeniem błędu ORA-20001 i komunikatem „Nie istnieje zespół o nazwie <nazwa zespołu>”.

Zadania

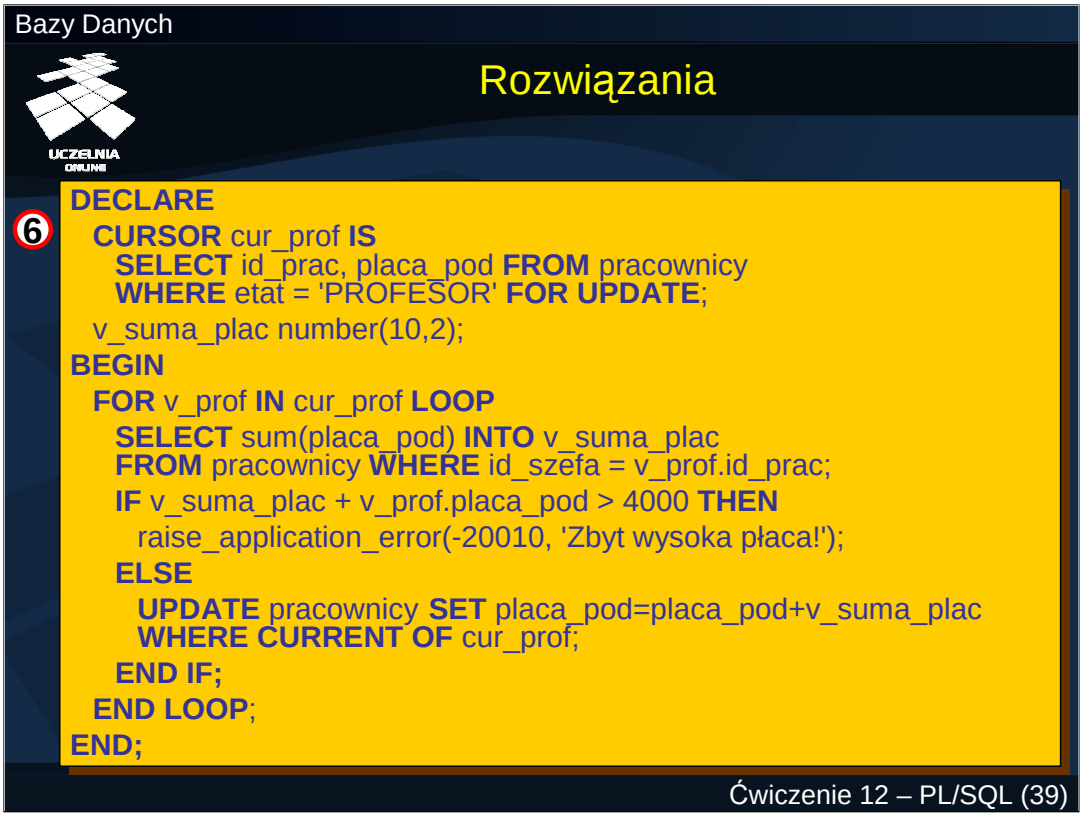
5. Rozszerz program z zadania 4. z kursorów o obsługę sytuacji podania niepoprawnego etatu (etatu, na którym nie pracuje żaden pracownik).
6. Napisz program z użyciem kursora, który odczyta informacje o wszystkich profesorach i przyzna im podwyżkę w wysokości 10% sumy płac podstawowych ich podwładnych. Jeśli po podwyżce pensja któregoś z profesorów przekroczyłaby 4000 złotych, program powinien zgłosić błąd (skorzystaj z procedury RAISE APPLICATION ERROR).

Bieżący slajd rozpoczyna zbiór zadań, których celem jest utrwalenie wiadomości o kursorach w programach PL/SQL.



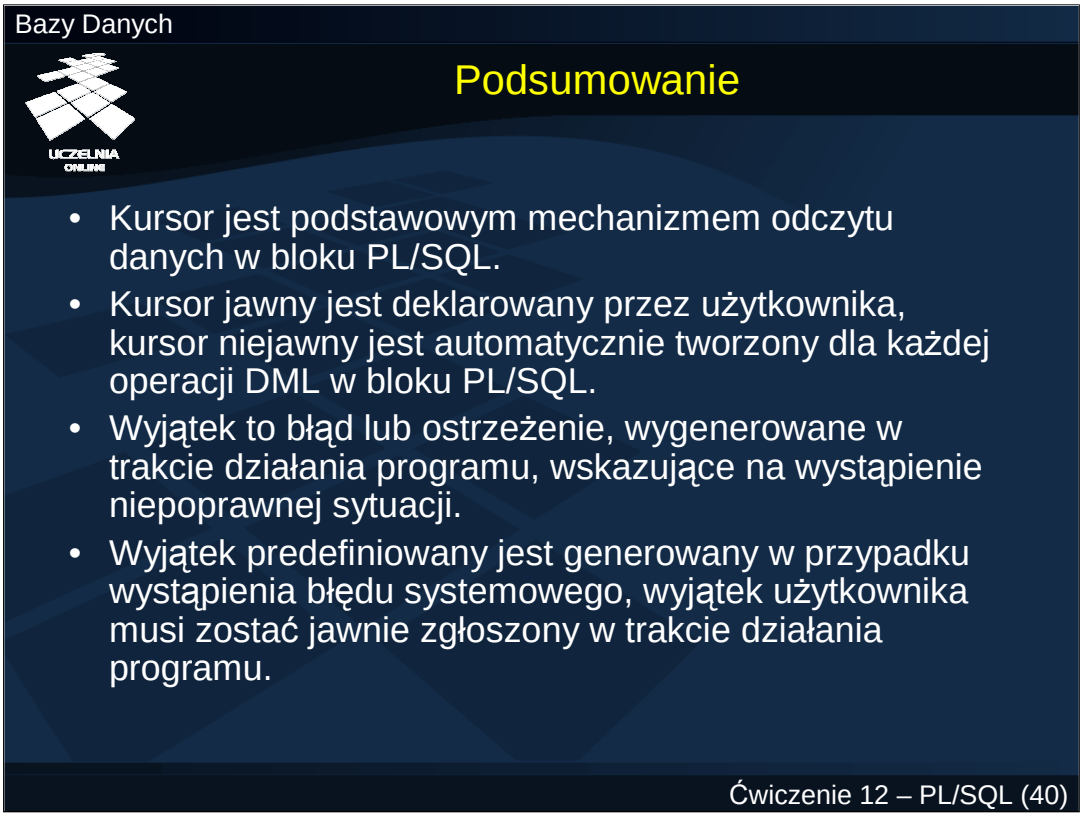
Bieżący slajd przedstawia rozwiązanie zadania (5), którego treść zacytowano poniżej.

(5) Rozszerz program z zadania 4. z kursorów o obsługę sytuacji podania niepoprawnego etatu (etatu, na którym nie pracuje żaden pracownik).



Bieżący slajd przedstawia rozwiązanie zadania (6), którego treść zacytowano poniżej.

(6) Napisz program z użyciem kursora, który odczyta informacje o wszystkich profesorach i przyzna im podwyżkę w wysokości 10% sumy płac podstawowych ich podwładnych. Jeśli po podwyżce pensja któregoś z profesorów przekroczyłaby 4000 złotych, program powinien zgłosić błąd (skorzystaj z procedury RAISE APPLICATION ERROR).



W zakończonym ćwiczeniu zaprezentowano zaawansowane konstrukcje języka PL/SQL: kursory i obsługę wyjątków. Kursor jest podstawowym mechanizmem PL/SQL, umożliwiającym odczyt rekordów z relacji w bazie danych. Kursor jawny musi zostać przed użyciem zadeklarowany przez użytkownika, kursor niejawny jest automatycznie tworzony dla każdej operacji DML, umieszczonej w bloku PL/SQL.

Wyjątek jest błędem lub ostrzeżeniem, generowanym w trakcie wykonania programu, wskazującym na wystąpienie niepoprawnej sytuacji (np. brak rekordów w relacji, dzielenie przez zero, naruszenie ograniczenia integralnościowego). Wyjątek predefiniowany odpowiada na wystąpienie błędu systemowego, natomiast wyjątek użytkownika musi zostać zadeklarowany i jawnie zgłoszony w trakcie działania programu PL/SQL.

Każde z omówionych w ćwiczeniu zagadnień zostało utrwalone przez serię zadań.