Obiekty nie są jedynymi wartościami o złożonej strukturze, które mogą być składowane w tabelach w modelu obiektowo-relacyjnym. Prócz obiektów, w pojedynczych komórkach tabel można składować również kolekcje. Celem ćwiczenia jest przedstawienie państwu dwóch rodzajów kolekcji zaimplementowanych w SZBD Oracle: tablic o zmiennej długości (varray) i zagnieżdżonych tabele (nested table).

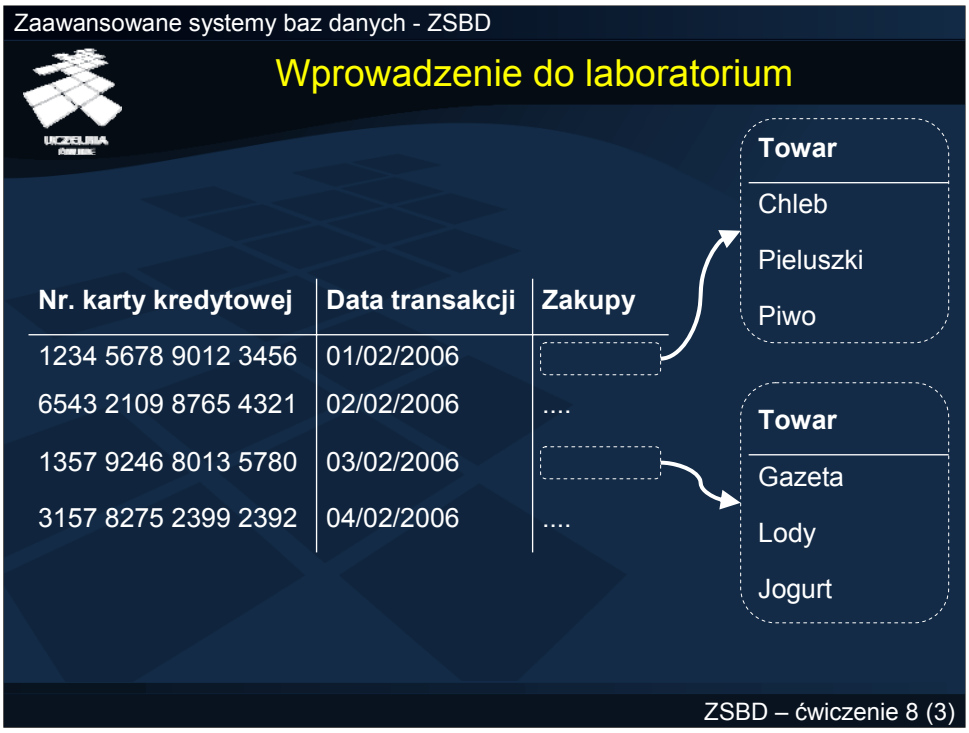
Wymagania:

Do wykonania ćwiczenia konieczna jest dobra znajomość języka SQL i przeciętna PL/SQL oraz tematyka omawiana na siódmych ćwiczeniach z ORSZBD.

Plan ćwiczenia

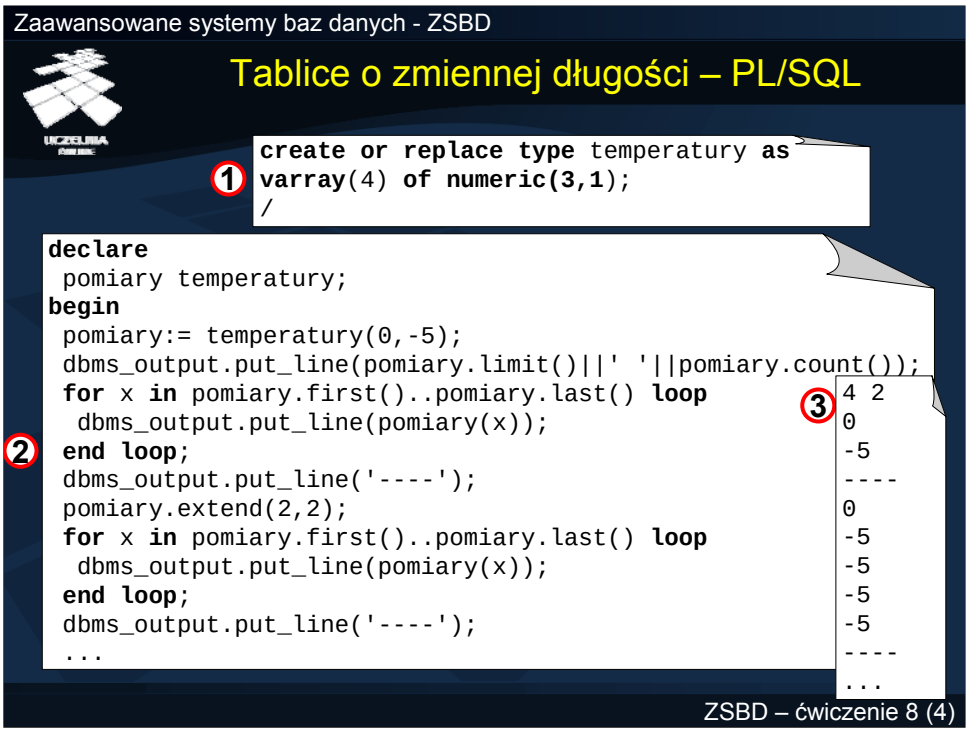
- Wprowadzenie do laboratorium.
- Tablice o zmiennej długości w PL/SQL.
- Zadanie.
- Tablice o zmiennej długości w SQL.
- Zadanie.
- Zagnieżdżone tabele w PL/SQL.
- Zadanie.
- Zagnieżdżone tabele w SQL.
- Zadania.
- Konstrukcja CAST(MULTISET(.
- Zadania.
- Podsumowanie.

Ćwiczenie rozpocznie się od przedstawienia motywacji będącej przyczyną wprowadzenia typów reprezentujących kolekcje wartości. Następnie omówione zostaną tablice o zmiennej długości, oraz sposób, w jaki mogą zostać wykorzystane z poziomu SQL i z poziomu PL/SQL. Po omówieniu tablic o zmiennej długości zajmiemy się drugim typem kolekcji – zagnieżdżonymi tabelami. Zostaną opisane różnice pomiędzy obydwoma rodzajami kolekcji, oraz wprowadzone nowe cechy charakterystyczne dla tabel zagnieżdżonych. Podobnie jak poprzednio omówione zostaną zagadnienia związane z pracą z tym typem kolekcji zarówno z poziomu SQL, jak i z poziomu PL/SQL. Ostatnim tematem omawianym na zajęciach będzie konstrukcja CAST(MULTISET(, która pozwala na zamianę wyniku zapytania w kolekcję. Każde z ww. zagadnień zakończone jest zadaniami do samodzielnego wykonania. Ćwiczenie zakończymy slajdem podsumowującym przedstawiony materiał.



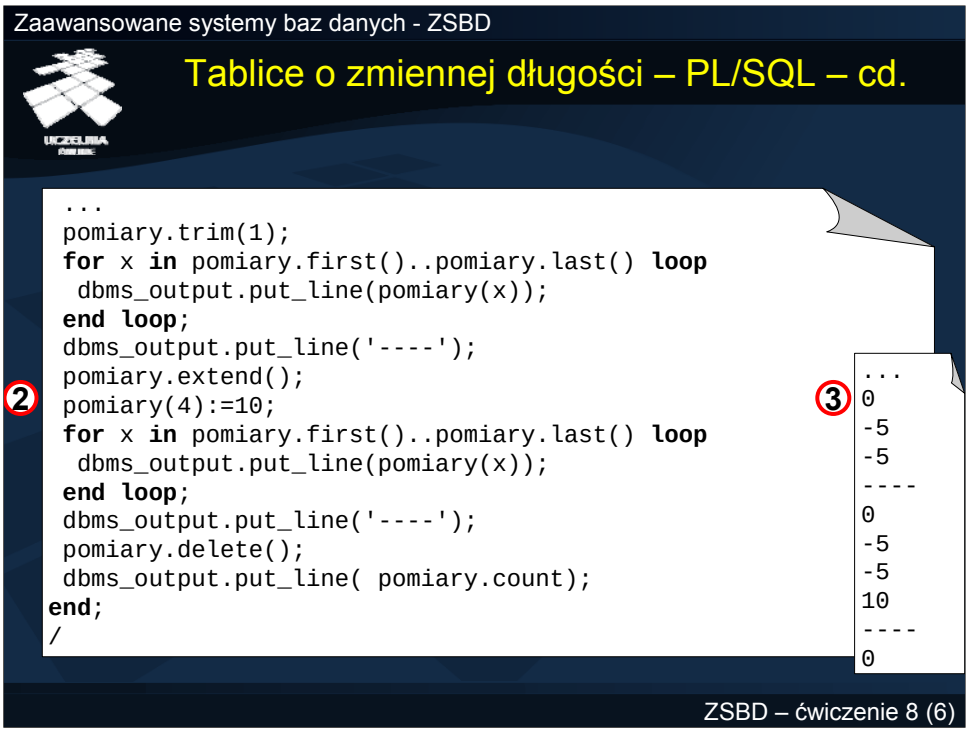
Kolekcje są naturalnym sposobem reprezentacji wielu różnych aspektów świata rzeczywistego takich jak na przykład: zawartość koszyka zakupów, lista wypożyczonych książek w bibliotece, oceny studenta, lista operacji na rachunku bankowym, zbiór pracowników pracujących w jednym zespole albo płyty DVD dostępne w wypożyczalni. Ponieważ w modelu obiektowo relacyjnym możliwe jest składowanie złożonych struktur danych (obiektów) w pojedynczych komórkach tabeli, to nic nie stoi na przeszkodzie aby umożliwić składowanie innych złożonych struktur, takich jak kolekcje. W SZBD Oracle zaimplementowano dwa rodzaje kolekcji : tablice o zmiennej długości (varray) i zagnieżdżone tabele (nested table).

Na slajdzie przedstawiono przykładową tabelę przechowującą dane o zakupach dokonanych przez klienta supermarketu, w której w każdej krotce, w atrybucie zakupy przechowywana jest kolekcja zakupionych towarów.



Aby móc przechowywać kolekcje w tabelach, należy najpierw utworzyć typ danych reprezentujący kolekcję. Typy kolekcji tworzone są za pomocą polecenia CREATE TYPE. Początek polecenia jest podobny jak przy tworzeniu typu obiektowego: „CREATE [OR REPLACE] TYPE nazwa AS...”. W przypadku typów obiektowych po słowie kluczowym AS występowało słowo kluczowe OBJECT i lista atrybutów oraz deklaracji metod. W przypadku tablic o zmiennej długości pojawia się tutaj słowo kluczowe VARRAY, a za nim maksymalna liczba elementów jaka mogą być przechowywane w kolekcji, później słowo kluczowe OF i typ przechowywanych w kolekcji wartości. Sposób utworzenia przykładowego typu, reprezentującego tablicę o zmiennej długości, pokazuje przykład (1). Typ ten reprezentuje tablicę, która może maksymalnie przechowywać 4 wartości liczbowe. W tym przypadku są to wartości pomiarów temperatury w ciągu dnia. Instancje typu kolekcji, a więc konkretne kolekcje przechowujące wartości, są traktowane z poziomu PL/SQL jak obiekty. Na rzecz każdej kolekcji można aktywować kilka metod, których działanie zostało przedstawione w programie na przykładzie (2). Przykład (2) rozpoczyna się od deklaracji pomocniczej zmiennej typu kolekcji. Ponieważ zmienna nie jest inicjowana, to posiada pomyślną wartość równą NULL. Pierwsza linijka przykładowego programu tworzy nową kolekcję. Kolekcje są tworzone za pomocą wywołania czegoś, co przypomina konstruktor atrybutowy. Wywołanie to składa się z podania nazwy typu kolekcji, za którą w nawiasie podaje się listę kolejnych elementów, które mają się znaleźć w nowo utworzonej kolekcji. W przypadku tablic o zmiennej długości liczba podanych parametrów aktualnych nie może przekroczyć maksymalnego rozmiaru kolekcji. Tutaj tworzona jest nowa kolekcja przechowująca dwie liczby: 0 i –5. Ponieważ kolekcje są indeksowane od 1, to pod indeksem 1 w kolekcji znajdzie się liczba 0, a pod indeksem 2, liczba –5.

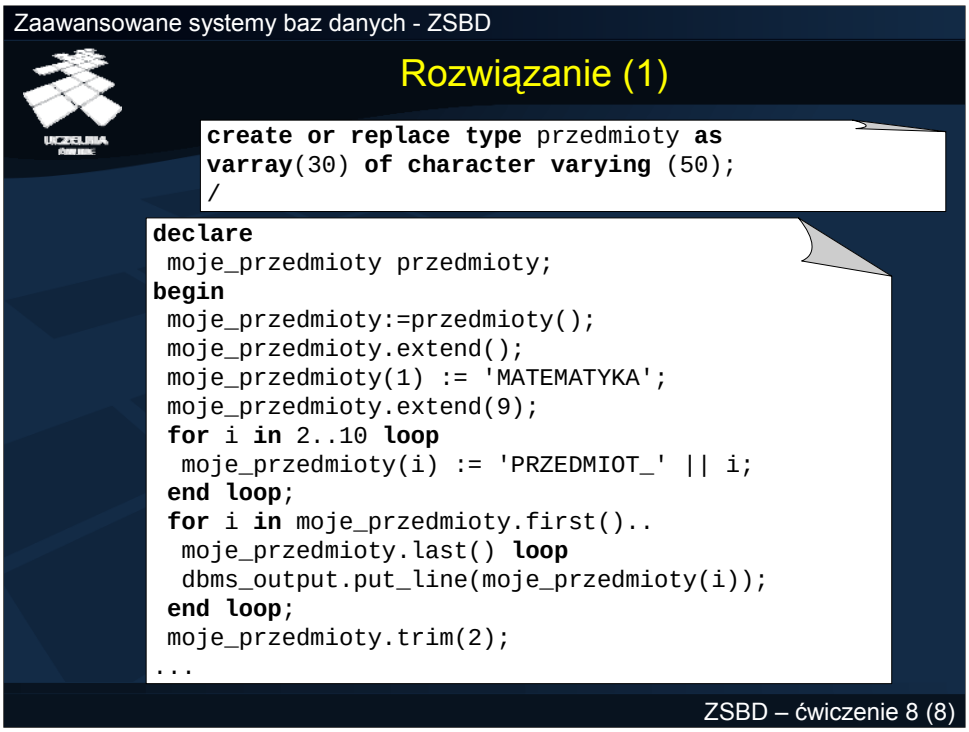
Kolejny wiersz przykładowego programu wypisuje na konsoli wartości zwrócone przed dwie metody kolekcji: LIMIT i COUNT. Metoda LIMIT zwraca maksymalną liczbę elementów kolekcji (tutaj 4), a metoda COUNT aktualną liczbę elementów (tutaj 2). Jeżeli spojrzeć na wynik działania przykładowego programu (3) można zauważyć, że zwracane wartości są poprawne. Kolejny fragment programu to pętla przeglądająca całą kolekcję. W pętli tej zademonstrowano dwie rzeczy: metody FIRST i LAST, które zwracają pierwszy i ostatni poprawny indeks w kolekcji, oraz sposób odwołania się do pojedynczego elementu kolekcji za pomocą indeksu. Tutaj, indeks podaje się w nawiasach okrągłych w przeciwieństwie do takich języków jak np. C++ albo Java, gdzie indeks elementu w tablicy podaje się w nawiasach kwadratowych. Jak łatwo zatem zauważyć, zmienna kontrolna pętli przyjmuje kolejne wartości z przedziału poprawnych indeksów kolekcji, a w ciele pętli wyświetlane są na konsoli kolejne wartości z kolekcji. Jeżeli spojrzeć na wynik działania przykładowego programu można zauważyć, że wyświetlono liczby 0 i –5, które zostały umieszczone w kolekcji za pomocą konstruktora. Po wyżej omówionej pętli pokazano działanie metody EXTEND. Metoda ta dodaje na końcu kolekcji nowe elementy. Metoda ta przyjmuje 0, 1 lub 2 parametry. Jeżeli nie poda się żadnych parametrów, to na końcu kolekcji dodawany jest jeden element o wartości NULL. Jeżeli poda się jeden parametr, to określa on liczbę elementów, jaka ma być dodana na końcu kolekcji. Parametry te będą miały wartość NULL. Ostatecznie, jeżeli poda się dwa parametry, to pierwszy parametr określa liczbę dodawanych elementów, a drugi parametr określa indeks istniejącego elementu, którego wartość zostanie skopiowana do nowych elementów kolekcji. W przykładowym programie użyto tej ostatniej wersji metody. Po wykonaniu tej metody kolekcja zostanie rozszerzona o dwa elementy, które zostaną wypełnione wartością elementu o indeksie 2 (czyli –5). Po aktywacji metody EXTEND pojawia się znana już pętla wyświetlająca zawartość kolekcji. Jeżeli spojrzeć na rozwiązanie można zauważyć, że zawartość kolekcji jest zgodna z oczekiwaniami. ...



... W kolejnym kroku program demonstruje użycie metody TRIM, która służy do usuwania elementów z końca kolekcji. Jej jedynym parametrem jest liczba usuwanych elementów. Po aktywacji metody TRIM znowu wypisywana jest na konsoli zawartość kolekcji i, zgodnie z oczekiwaniami, ostatni element kolekcji został usunięty. Kolejne dwie instrukcje demonstrują użycie innej wersji metody EXTEND (bez parametrów), oraz przypisania jakiejś wartości do atrybutu kolekcji. Tutaj, elementowi o indeksie 4, dopiero co utworzonemu za pomocą metody EXTEND, przypisywana jest liczba 10. Po przypisaniu pojawia się znana już pętla wyświetlająca zawartość kolekcji na konsoli i, jak łatwo zauważyć, w kolekcji pojawił się czwarty element, o wartości 10. Ostatnie dwie instrukcje demonstrują działanie metody DELETE. W wyniku aktywacji tej metody, wszystkie elementy z kolekcji są usuwane i dlatego wartość funkcji COUNT wyświetlana na konsoli wynosi 0.

Zadanie (1)

- Wykonaj kilka eksperymentów na tablicach o zmiennym rozmiarze. Przykładowo: stwórz typ kolekcji zawierający informacje o przedmiotach nauczanych na studiach (łańcuchy znaków). Napisz program w PL/SQL, który tworzy kolekcję, wstawia do kolekcji przykładowe przedmioty, rozszerza kolekcję, wyświetla zawartość kolekcji, usuwa elementy z końca kolekcji, wyświetla informacje o długości kolekcji i o limicie na liczbę elementów.



Slajd pokazuje rozwiązanie zadania (1), którego treść przytoczono poniżej. Rozwiązanie jest kontynuowane na kolejnych slajdach.

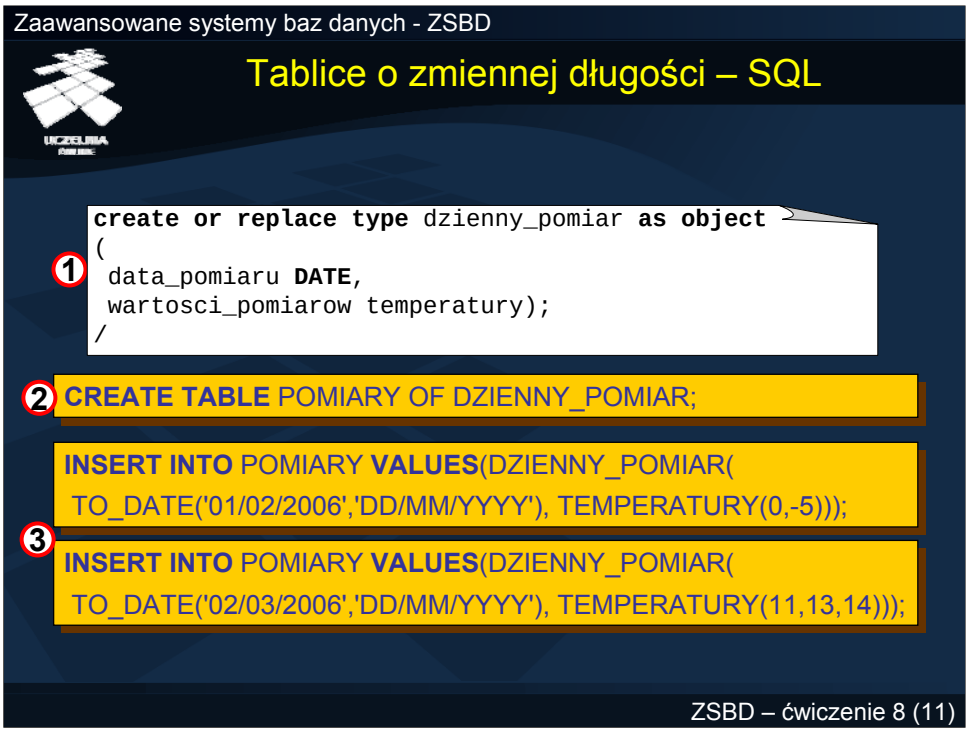
Wykonaj kilka eksperymentów na tablicach o zmiennym rozmiarze. Przykładowo: stwórz typ kolekcji zawierający informacje o przedmiotach nauczanych na studiach (łańcuchy znaków). Napisz program w PL/SQL, który tworzy kolekcję, wstawia do kolekcji przykładowe przedmioty, rozszerza kolekcję, wyświetla zawartość kolekcji, usuwa elementy z końca kolekcji, wyświetla informacje o długości kolekcji i o limicie na liczbę elementów.

Rozwiązanie (1) – cd.

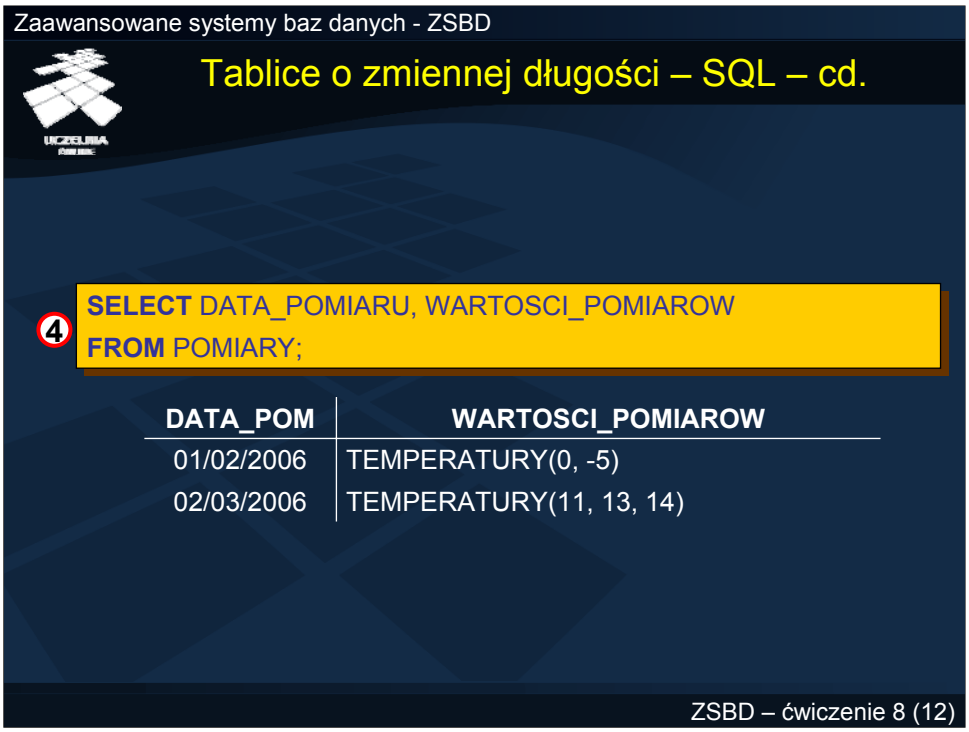
```
...
 for i in moje_przedmioty.first()..
  moje_przedmioty.last() loop
  dbms_output.put_line(moje_przedmioty(i));
 end loop;
 dbms_output.put_line('Limit: ' || moje_przedmioty.limit());
 dbms_output.put_line('Liczba elementow: '||
  moje_przedmioty.count());
 moje_przedmioty.extend();
 moje_przedmioty(9) := 9;
...
```

Rozwiązanie (1) – cd.

```
...
 dbms_output.put_line('Limit: ' || moje_przedmioty.limit());
 dbms_output.put_line('Liczba elementow: ' ||
  moje_przedmioty.count());
 moje_przedmioty.delete();
 dbms_output.put_line('Limit: ' || moje_przedmioty.limit());
 dbms_output.put_line('Liczba elementow: ' ||
  moje_przedmioty.count());
end;
/
```

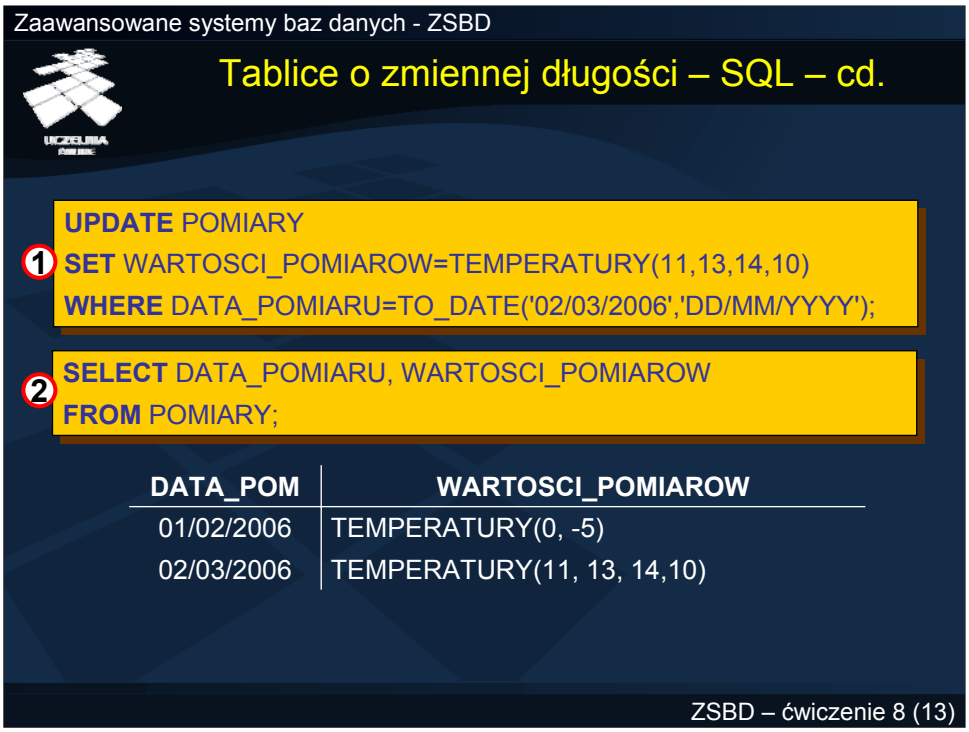


Na slajdzie przedstawiono użycie tablic o zmiennej długości z poziomu SQL. Z perspektywy języka SQL tablice o zmiennej długości są atomowymi wartościami – nie ma możliwości dostępu do pojedynczych elementów składowanych wewnątrz kolekcji tego typu, ani też nie działają żadne metody. Przeanalizujmy obecnie przykłady przedstawione na slajdzie. Polecenie (1) tworzy typ obiektowy DZIENNY_POMIAR reprezentujący wyniki dziennych pomiarów temperatury. Wartości pomiarów są składowane w kolekcji zapisanej w atrybucie WARTOSCI_POMIAROW. Polecenie (2) tworzy tabelę obiektową pozwalającą na składowanie obiektów typu DZIENNY_POMIAR, a polecenia INSERT na przykładzie (3) wstawiają do tej tabeli dwa obiekty reprezentujące pomiary, które zawierają kolekcję. Jak łatwo zauważyć, podano dwa parametry aktualne dla konstruktora atrybutowego typu obiektowego DZIENNY_POMIAR. Pierwszy parametr konstruktora odpowiada dacie pomiaru i dlatego jako parametr aktualny podano wywołanie funkcji TO_DATE zwracającej datę. Drugi parametr odpowiada atrybutowi WARTOSCI_POMIAROW, który przechowuje kolekcję. Aby wstawić jakąś konkretną kolekcję, umieszczono w tym miejscu wywołanie konstruktora kolekcji, który poznaliście państwo przy okazji omawiania tablic o zmiennej długości w PL/SQL. ...



... Przykład (4) pokazuje zapytanie odczytujące wartości obu atrybutów składowanych obiektów. Jak łatwo zauważyć, sqlplus przedstawia kolekcje jako wywołania konstruktorów kolekcji, które utworzyłyby kolekcję taką samą jak odczytana. Oczywiście, z poziomu języka programowania, odczytywana jest rzeczywiście kolekcja, a nie łańcuch taki, jak wyświetlany przez sqlplus.

Należy tutaj wspomnieć o jednej ważnej rzeczy związanej ze składowaniem tablic o zmiennej długości. Otóż są one fizycznie składowane razem z resztą krotki, z zachowaniem porządku wewnątrz kolekcji. Tym samym, po zapisaniu kolekcji i jej ponownym odczytaniu, elementy kolekcji nie zmienią kolejności.

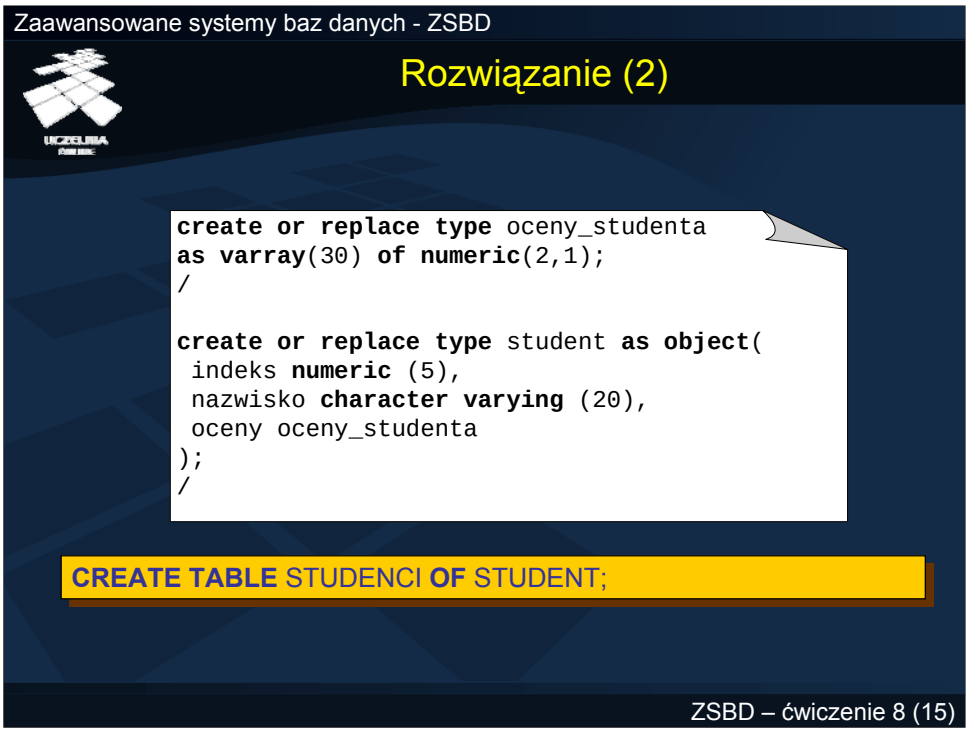


Jak wspominano wcześniej, z poziomu SQL nie można dostać się do pojedynczych elementów tablicy o zmiennym rozmiarze. Nie można ich także zmodyfikować. Stąd, jedyną możliwością modyfikacji kolekcji z poziomu SQL jest wstawienie całkowicie nowej kolekcji z innym zestawem przechowywanych wartości. Przykładowe polecenie UPDATE (1) pokazuje sposób modyfikacji wartości atrybutu przechowującego tablicę o zmiennej długości. Jest on analogiczny do modyfikacji atrybutu o wartości atomowej. Atrybutowi WARTOSCI_POMIAROW przypisywana jest cała nowa kolekcja utworzona za pomocą konstruktora kolekcji. Wynik zapytania (2) pokazuje, że odpowiednia kolekcja rzeczywiście się zmieniła.

Zadanie (2)

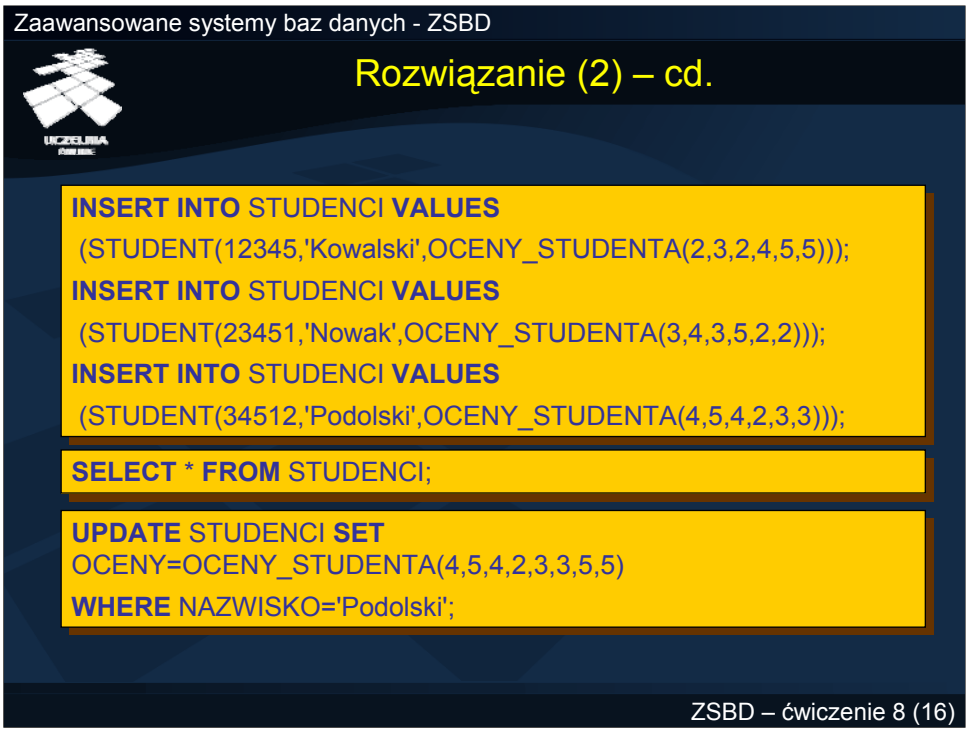
Utwórz typ tablicy o zmiennym rozmiarze przechowujący maksymalnie 30 liczb, który będzie reprezentował listę ocen studenta. Utwórz typ obiektowy Student, który posiada atrybuty: indeks (typ liczbowy), nazwisko (typ łańcuchowy) oraz oceny (typu wcześniej zdefiniowanej kolekcji). Utwórz tabelę obiektową STUDENCI przechowującą obiekty typu STUDENT. Wstaw za pomocą polecenia INSERT kilku studentów, razem z ocenami, do tabeli. Odczytaj tabelę za pomocą polecenia SELECT. Zmień listę ocen jednego ze studentów za pomocą polecenia UPDATE.

(!) Całość zadania wykonaj tylko za pomocą poleceń SQL. Pamiętaj o tym, że tablice o zmiennym rozmiarze są traktowane z poziomu SQL jako całość, a zatem, aby zmienić oceny jakiegoś studenta trzeba w poleceniu UPDATE utworzyć nową kolekcję.

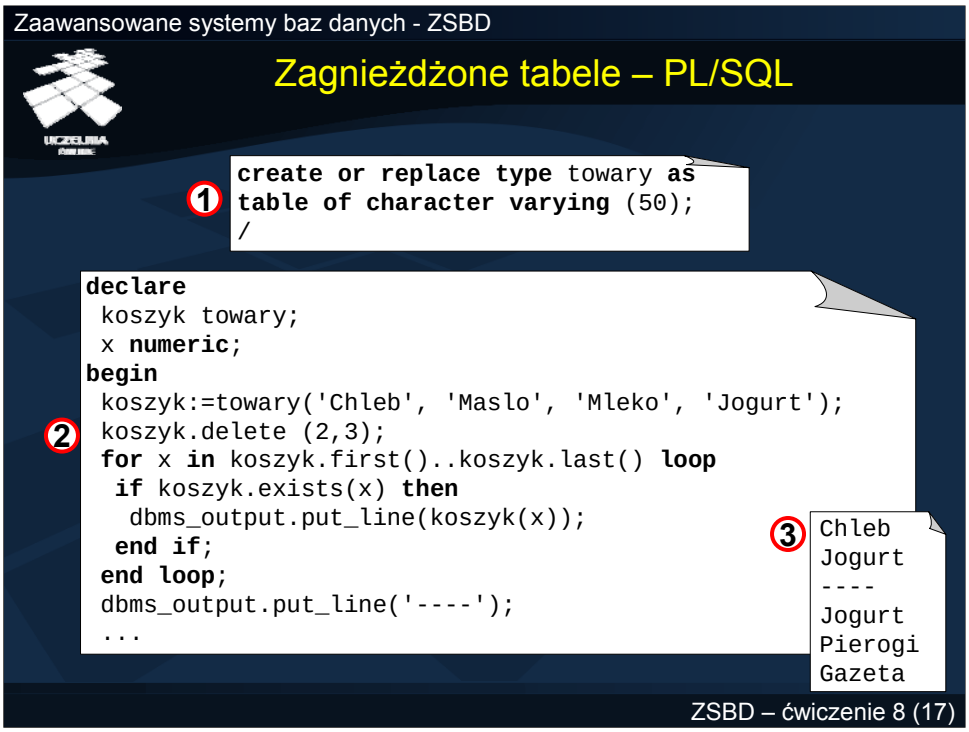


Slajd pokazuje rozwiązanie zadania (2), którego treść przytoczono poniżej. Rozwiązanie jest kontynuowane na kolejnym slajdzie.

Utwórz typ tablicy o zmiennym rozmiarze przechowujący maksymalnie 30 liczb, który będzie reprezentował listę ocen studenta. Utwórz typ obiektowy Student, który posiada atrybuty: indeks (typ liczbowy), nazwisko (typ łańcuchowy) oraz oceny (typu wcześniej zdefiniowanej kolekcji). Utwórz tabelę obiektową STUDENCI przechowującą obiekty typu STUDENT. ...

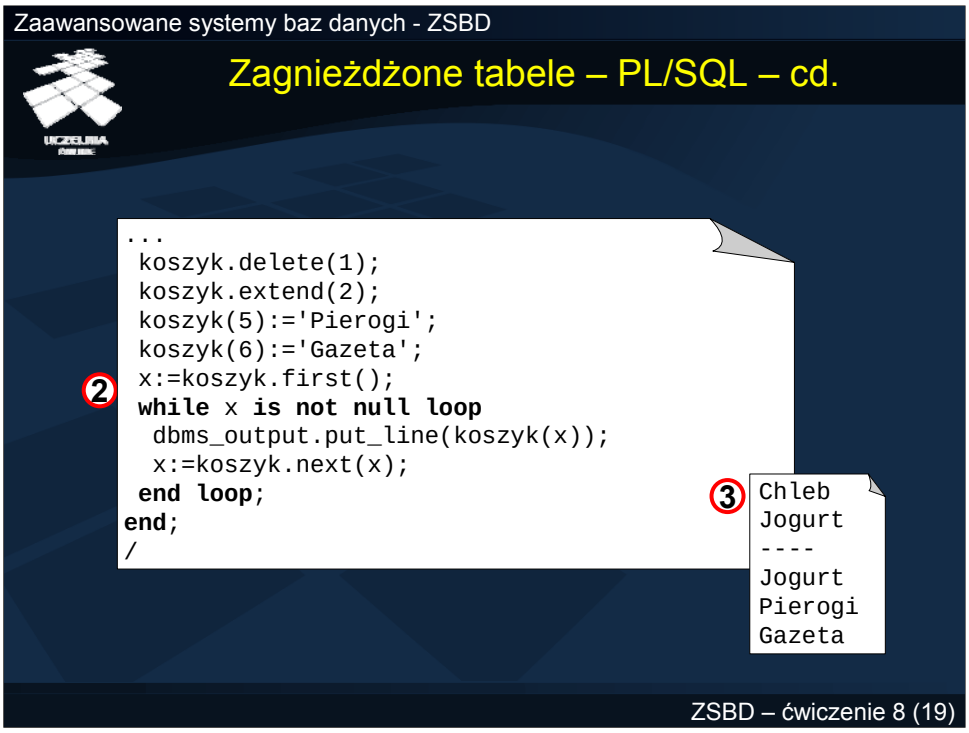


... Wstaw za pomocą polecenia INSERT kilku studentów, razem z ocenami, do tabeli. Odczytaj tabelę za pomocą polecenia SELECT. Zmień listę ocen jednego ze studentów za pomocą polecenia UPDATE.



Prócz tablic o zmiennej długości istnieje również inny rodzaj kolekcji – zagnieżdżone tabele. Podobnie jak w przypadku tablic o zmiennej długości, zanim będzie można korzystać z zagnieżdżonych tabel, konieczne jest utworzenie typu danych reprezentującego kolekcję. Typy zagnieżdżonych tabel są tworzone w analogiczny sposób jak typy tablic o zmiennym rozmiarze. Jedyną różnicą jest tutaj to, iż zamiast wyrażenia VARRAY(maksymalny_rozmiar) używa się słowa kluczowego TABLE (tabele zagnieżdżone mają nieograniczony rozmiar). Przykład (1) pokazuje sposób utworzenia typu kolekcji łańcuchów, który będzie reprezentował zbiór towarów w koszyku w supermarkecie. Korzystanie z zagnieżdżonych tabel w PL/SQL jest bardzo podobne do korzystania z tablic o zmiennym rozmiarze. Kolekcje tego rodzaju są również, podobnie jak tablice o zmiennym rozmiarze, traktowane jako obiekty i posiadają takie prawie takie same metody. Tabele zagnieżdżone posiadają wszystkie metody tablic o zmiennym rozmiarze, ale posiadają one również kilka innych metod, które zostały zademonstrowane na przykładzie (2). Przykładowy program rozpoczyna się od deklaracji zmiennej KOSZYK typu TOWARY, który reprezentuje kolekcję – tabelę zagnieżdżoną oraz zmiennej pomocniczej typu liczbowego o nazwie X. Pierwszy wiersz właściwego programu stanowi wywołanie konstruktora kolekcji. Konstruktory tabel zagnieżdżonych działają analogicznie do konstruktorów tablic o zmiennej długości. Na przykładzie tworzona jest kolekcja przechowująca cztery towary: „Chleb”, „Masło”, „Mleko” i „Jogurt”. Podobnie jak w przypadku tablic o zmiennej długości, tabele zagnieżdżone są również indeksowane od jeden, dlatego „Chleb” będzie miał w kolekcji indeks 1, „Masło” 2 itd. Kolejny wiersz programu demonstruje działanie metody DELETE. Metoda DELETE w tabelach zagnieżdżonych posiada większą funkcjonalność niż w przypadku tablic o zmiennej długości. Prócz wersji bezparametrowej tej metody, istnieją również wersje z 1 albo 2 parametrami. Jeżeli parametr jest jeden, to oznacza on indeks elementu kolekcji, który ma zostać usunięty. Jeżeli użyte zostaną dwa parametry to są to indeksy pierwszego i ostatniego elementu kolekcji do usunięcia. W przykładowym programie podano parametry 2 i 3, co oznacza, że usunięte zostaną z kolekcji „Masło” i „Mleko”. Usuwanie elementów z tabeli zagnieżdżonej jest dosyć charakterystyczne. Otóż, usunięcie elementów ze środka nie powoduje przesunięcia elementów o wyższych indeksach na zwolnione miejsca w kolekcji.

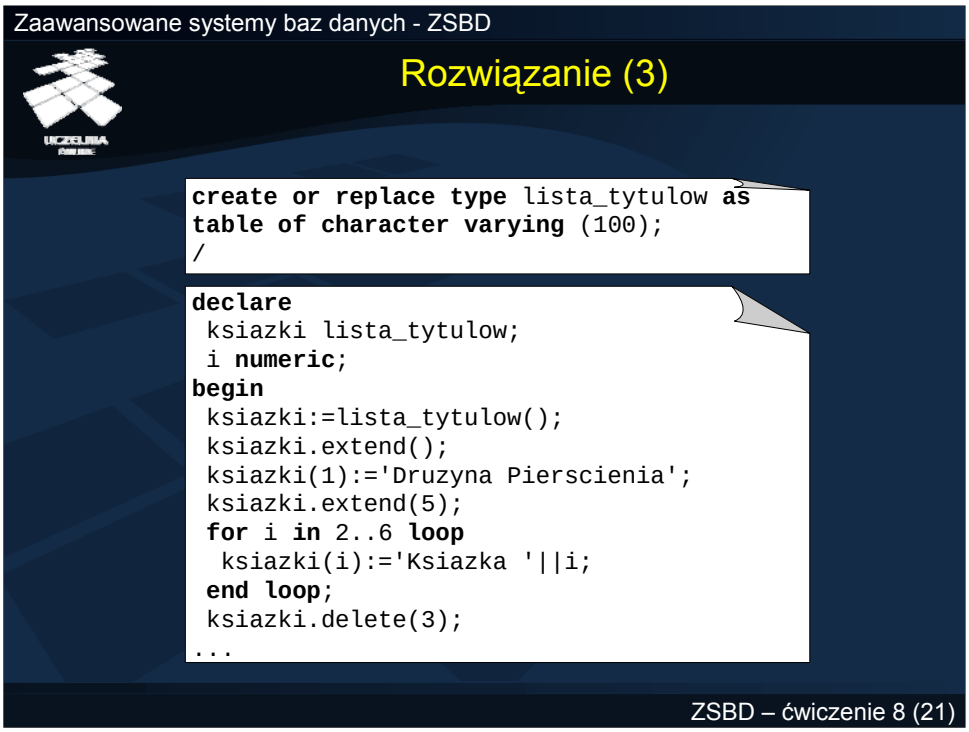
W naszej przykładowej kolekcji, po usunięciu elementów „Masło” i „Mleko”, pozostanie „Chleb” pod indeksem 1 i „Jogurt” pod indeksem 4. Próba odwołania do indeksów 2 albo 3 skończy się błędem. Po wywołaniu metody DELETE przedstawiono pętlę odczytującą kolejne wartości z kolekcji. Podobnie, jak w przypadku tablic o zmiennej długości, pętla wykorzystuje metody FIRST i LAST kolekcji w celu określenia poprawnego zakresu indeksów. Niestety, ponieważ usunęliśmy ze środka kolekcji jakieś elementy, zakres ten nie jest ciągły (indeksy 2 i 3 nie są poprawne). Aby sprawdzić, czy indeks jest poprawny (czyli, czy istnieje jakiś element pod danym indeksem) należy zastosować metodę kolekcji o nazwie EXISTS. Parametrem tej metody jest indeks, a metoda zwraca prawdę, jeśli jakiś element pod danym indeksem istnieje, bądź fałsz jeśli nie. Wykorzystując metodę EXISTS można przeczytać całą tabelę zagnieżdżoną bez błędnych odwołań do nieistniejących elementów kolekcji. ...



... Po pętli demonstrowane jest użycie metody DELETE z jednym parametrem. Tutaj usuwany jest pierwszy element kolekcji („Chleb”), a następnie za pomocą metody EXTEND, kolekcja jest rozszerzana o dwa dodatkowe elementy, do których zapisywane są nowe towary: „Pierogi” i „Gazeta”. Nowa zawartość kolekcji jest przeglądana za pomocą innej wersji pętli pozwalającej odczytać zawartość całej tabeli. Pętla ta wykorzystuje metodę NEXT. Metoda NEXT posiada parametr, który jest indeksem jakiegoś elementu w kolekcji. Jej wartością jest następny poprawny indeks, o ile takowy istnieje. W przeciwnym wypadku wartością metody jest NULL. Istnieje również metoda PRIOR o podobnym działaniu, która zwraca poprzedni poprawny indeks. Działanie pętli wygląda następująco. Do zmiennej x przypisywany jest pierwszy poprawny indeks (tutaj będzie to 4). Następnie tak długo, jak x nie jest równy NULL odczytywana jest wartość elementu z kolekcji pod indeksem zapisanym w zmiennej x, a następnie, za pomocą metody NEXT znajdowany jest następny poprawny indeks. Wynik działania programu (2) przedstawiono na przykładzie (3).

Zadanie (3)

- Wykonaj kilka eksperymentów na zagnieżdżonych tabelach. Przykładowo: stwórz typ kolekcji reprezentujący listę tytułów książek. Napisz program w PL/SQL, który tworzy kolekcję, rozszerza kolekcję i wstawia kilka książek, usuwa jakieś elementy ze środka oraz wyświetla wszystkie książki z kolekcji na konsoli z wykorzystaniem pętli FOR i metod EXISTS, FIRST i LAST oraz z pętli WHILE i metod FIRST i NEXT.

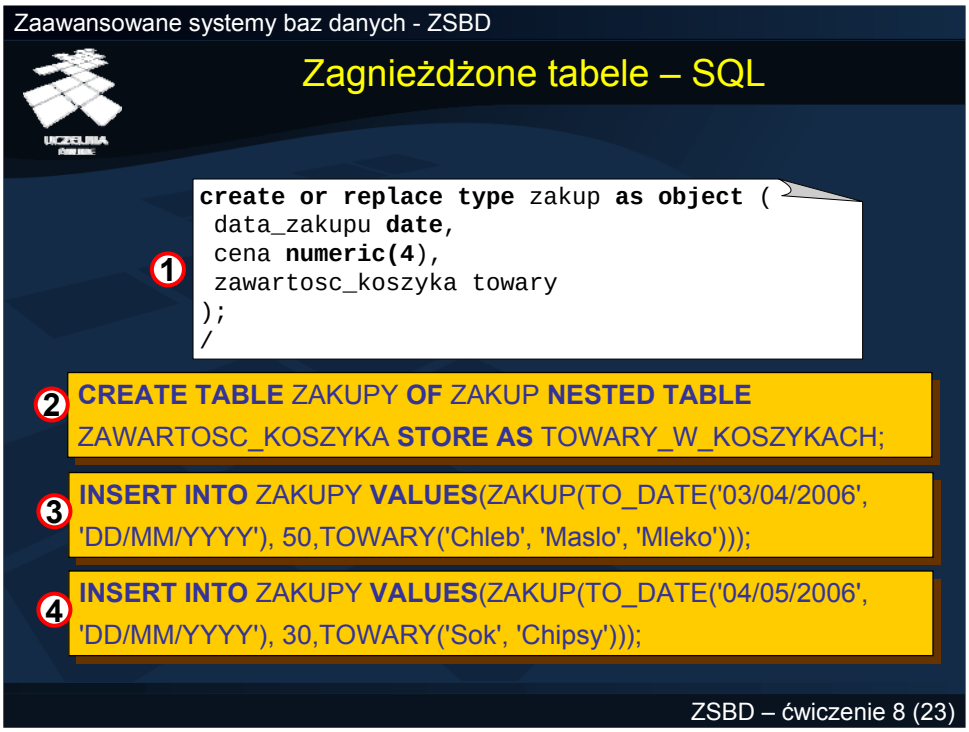


Slajd pokazuje rozwiązanie zadania (3), którego treść przytoczono poniżej. Rozwiązanie jest kontynuowane na kolejnym slajdzie.

Wykonaj kilka eksperymentów na zagnieżdżonych tabelach. Przykładowo: stwórz typ kolekcji reprezentujący listę tytułów książek. Napisz program w PL/SQL, który tworzy kolekcję, rozszerza kolekcję i wstawia kilka książek, usuwa jakieś elementy ze środka oraz wyświetla wszystkie książki z kolekcji na konsoli z wykorzystaniem pętli FOR i metod EXISTS, FIRST i LAST oraz z pętli WHILE i metod FIRST i NEXT.

Rozwiązanie (3) – cd.

```
...
 for i in ksiazki.first() .. ksiazki.last() loop
  if (ksiazki.exists(i)) then
   dbms_output.put_line(ksiazki(i));
  end if;
 end loop;
 ksiazki.trim(2);
 i:=ksiazki.first();
 while i is not null loop
  dbms_output.put_line(ksiazki(i));
  i:=ksiazki.next(i);
 end loop;
end;
/
```



Z poziomu języka SQL, z zagnieżdżonych tabel można korzystać w sposób analogiczny do tablic o zmiennej długości. Wszystkie przykłady przedstawione wcześniej dla tablic o zmiennej długości są poprawne również dla zagnieżdżonych tabel. Zagnieżdżone tabele mają jednak więcej możliwości, które zostaną zademonstrowane na tym, oraz na kolejnych slajdach. Demonstrację zacznijmy od utworzenia typu obiektowego ZAKUP (przykład (1)), który reprezentuje pojedynczy zakup zarejestrowany przez kasę w supermarkecie. Polecenie (2) tworzy tabelę obiektową pozwalającą na składowanie obiektów typu ZAKUP. Należałoby w tym miejscu wytłumaczyć po co w poleceniu tym dodano „... NESTED TABLE ZAWARTOSC_KOSZYKA STORE AS TOWARY_W_KOSZYKACH;”. W przeciwieństwie do tablic o zmiennej długości, zawartość tabel zagnieżdżonych jest składowana w osobnej tabeli zarządzanej przez bazę danych. Tabela ta nie jest dostępna w standardowy sposób, a do jej wartości można się dostać jedynie poprzez odczytywanie zawartości kolekcji. Przy tworzeniu tabeli obiektowej, która posiada atrybut typu zagnieżdżonej tabeli, konieczne jest podanie nazwy osobnej tabeli, w której będą składowane dane z kolekcji. Do tego służy właśnie wyrażenie NESTED TABLE. Po słowach kluczowych NESTED TABLE podaje się nazwę atrybutu, który przechowuje kolekcje, następnie podaje się słowa kluczowe STORE AS i ostatecznie nazwę dodatkowej tabeli. Ponieważ dane z kolekcji są przechowywane w tabeli, to po zapisaniu kolekcji do bazy danych i odczytaniu jej, porządek w danych może nie zostać zachowany. Polecenia (3) i (4) wstawiają dane do tabeli ZAKUPY. Jak łatwo zauważyć, wstawianie całych tabel zagnieżdżonych niczym się nie różni od wstawiania tablic o zmiennym rozmiarze.

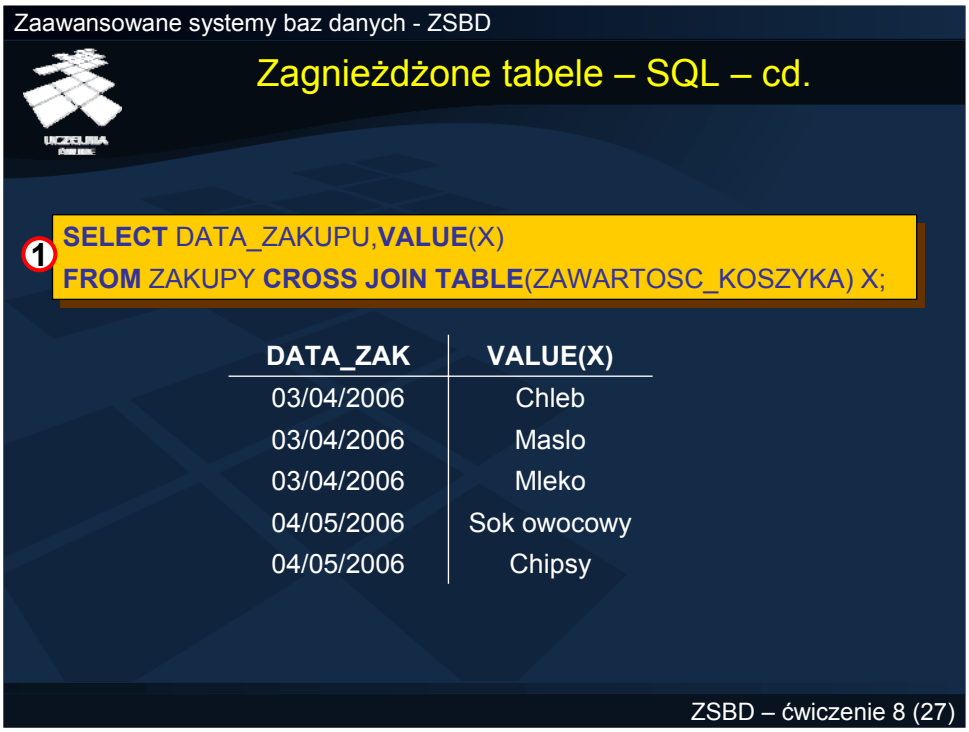
Przykład (1) przedstawiony na slajdzie pokazuje jak można odczytać całą kolekcję z tabeli. Jak łatwo zauważyć, zarówno sposób odczytania, jak i sposób wyświetlenia przez sqlplus niczym się nie różni od tablic o zmiennej długości.

Dużo ciekawszym przykładem jest przykład (2). W przeciwieństwie do tablic o zmiennym rozmiarze, w przypadku zagnieżdżonych tabel możliwy jest dostęp za pomocą poleceń DML do pojedynczych elementów kolekcji. Umożliwia to operator TABLE. Operator TABLE zamienia dowolną kolekcję, która jest zagnieżdżoną tabelą, na tabelę, która może zostać odczytana np. za pomocą polecenia SELECT. Zacznijmy od analizy klauzuli FROM przykładu (2). W klauzuli tej umieszczono operator TABLE, któremu jako parametr przekazano podzapytanie. Wynikiem tego podzapytania jest kolekcja towarów zakupionych w dniu 04/05/2006. Oczywiście, w praktycznych zastosowaniach tego typu warunek selekcji spowodowałby, że podzapytanie zwróci więcej niż jedną krotkę (co skończyłoby się błędem), ale w naszym przykładzie taki warunek selekcji jest wystarczający. Operator TABLE zamienia zwróconą przez podzapytanie kolekcję na tabelę, której nadawany jest alias X. Ponieważ atrybut w utworzonej przez operator TABLE tabeli nie posiada nazwy, należy użyć operatora VALUE, aby odczytać wartości z tej tabeli. W tym kontekście, operator VALUE reprezentuje wartość zapisaną w tabeli utworzonej przez operator TABLE. Podobnie jak w przypadku odczytywania obiektów z tabeli obiektowej, parametrem operatora VALUE jest alias tabeli z której odczytywane są dane. Ponieważ VALUE reprezentuje wartości z zagnieżdżonej tabeli, to można go umieścić w klauzuli SELECT. W wyniku zapytania (2) zostaną odczytane krotki, z których każda odpowiada jednemu elementowi kolekcji zapisanej w obiekcie typu ZAKUP z datą 04/05/2006.

Operator TABLE pozwala nie tylko na odpytywanie tabel zagnieżdżonych, ale również na wstawianie, modyfikację i usuwanie danych z tabel zagnieżdżonych. Przykładowe użycie polecenia INSERT pokazuje przykład (1). W przykładzie tym, w miejscu, gdzie normalnie podaje się nazwę tabeli, podano operator TABLE, z podzapytaniem takim samym jak w przykładzie na poprzednim slajdzie. Tym razem do odczytywanej przez podzapytanie tabeli zagnieżdżonej wstawiany jest łańcuch „Salsa”. Prócz zastosowania operatora TABLE polecenie INSERT niczym się nie różni od polecenia INSERT wstawiającego dane do zwykłej tabeli. Pod poleceniem pokazano zawartość tabeli obiektowej ZAKUPY, na której można zauważyć, że „Salsa” rzeczywiście została dodana do listy zakupionych towarów.

Przykład (2) pokazuje jak można użyć polecenia UPDATE na zagnieżdżonej tabeli. Podobnie jak w przypadku polecenia INSERT, zamiast nazwy tabeli podano operator TABLE z podzapytaniem. Podzapytanie w przykładzie zwraca tą samą tabelę zagnieżdżoną co w poprzednich. Przejdźmy do klauzul SET i WHERE. W klauzulach tych, ponownie skorzystano z operatora VALUE, ponieważ atrybut tabeli utworzonej za pomocą operatora TABLE nie ma nazwy. Zgodnie z warunkiem selekcji, polecenie UPDATE odszukuje wszystkie elementy kolekcji, które mają wartość „Sok” i przypisuje im wartość „Sok owocowy”. Pod poleceniem pokazano zawartość tabeli obiektowej ZAKUPY, na której można zauważyć, że „Sok” został rzeczywiście zamieniony na „Sok owocowy”.

Przykład przedstawiony na slajdzie pokazuje jak można usuwać elementy z zagnieżdżonej tabeli. Użycie polecenia DELETE jest analogiczne do poprzednio demonstrowanych poleceń INSERT i UPDATE. W miejscu gdzie zwykle podaje się nazwę tabeli, podano operator TABLE z podzapytaniem, które zwraca kolekcję, którą chcemy zmodyfikować. Aby móc odczytać wartości elementów kolekcji użyto operatora VALUE, któremu przekazano jako parametr alias modyfikowanej tabeli. Warunek selekcji określa że usunięty ma zostać towar „Salsa”. Analizując wynik polecenia pokazany na slajdzie można łatwo zauważyć, że towar ten został usunięty.



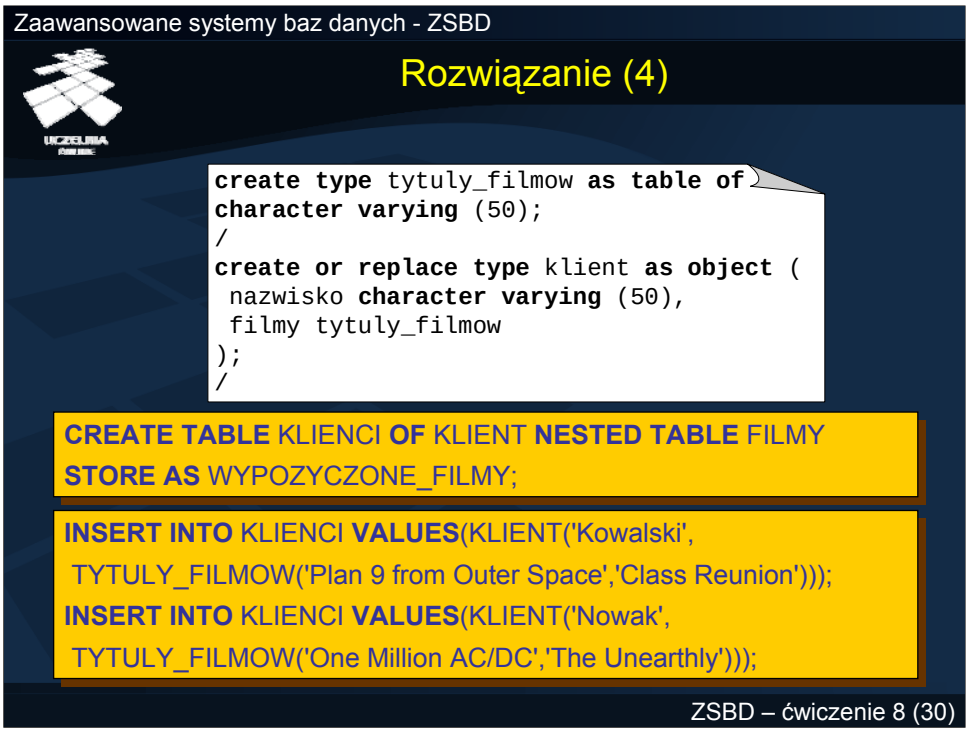
Aby skorzystać z operatora TABLE nie trzeba koniecznie podawać mu jako parametr podzapytania. Ten slajd demonstruje inną, bardzo przydatną konstrukcję z wykorzystaniem operatora TABLE. W zapytaniu pokazanym na slajdzie łączymy każdy obiekt z tabeli ZAKUPY, z wartościami zapisanymi w składowanych w nich kolekcjach. Przykładowo, zakupy dokonane 03/04/2006 to „Chleb”, „Masło” i „Mleko”. W wyniku połączenia obiekt reprezentujący zakupy dokonane 03/04/2006 zostanie zwrócony tyle razy ile jest elementów w kolekcji, ale za każdym razem z tym obiektem zostanie zwrócona inna wartość z kolekcji. Z kolei 04/05/2006, zakupiono „Sok owocowy” i „Chipsy”, a zatem obiekt ten zostanie zwrócony dwukrotnie, przy czym raz z obiektem zostanie zwrócona wartość „Sok owocowy”, a raz „Chipsy”.

Zadanie (4)

- Utwórz typ zagnieżdżonej tabeli przechowującej listę tytułów wypożyczonych filmów. Utwórz typ obiektowy reprezentujący klienta wypożyczalni o atrybutach: nazwisko (typu łańcuchowego) oraz filmy (typu wcześniej zdefiniowanej kolekcji). Utwórz tabelę obiektową KLIENCI przechowującą obiekty typu KLIENT. Wstaw przynajmniej dwóch klientów z wypożyczonymi filmami. Wykorzystując operator TABLE i polecenia INSERT, UPDATE i DELETE, wstaw, zmień i usuń jakieś filmy z tabeli zagnieżdżonej skojarzonej z jednym z klientów.

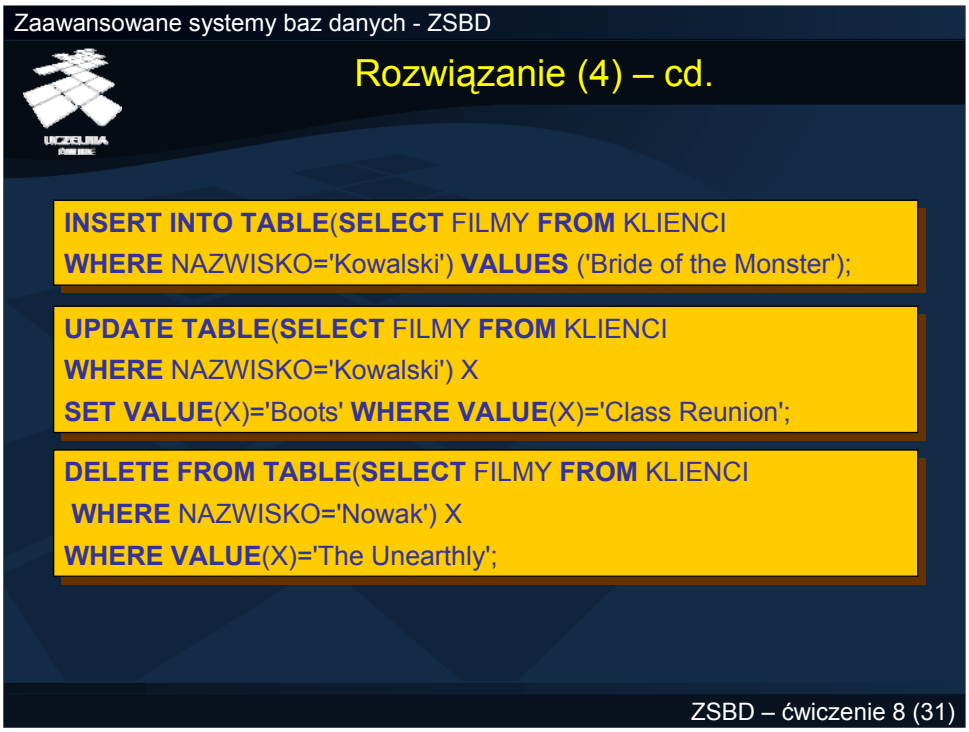
Zadanie (5)

- Napisz zapytanie, które wypisze tabelę składającą się z dwóch kolumn: nazwisko klienta i wypożyczony film. Wykorzystaj połączenie tabeli obiektowej z zagnieżdżoną tabelą.



Slajd pokazuje rozwiązanie zadania (4), którego treść przytoczono poniżej. Rozwiązanie jest kontynuowane na kolejnym slajdzie.

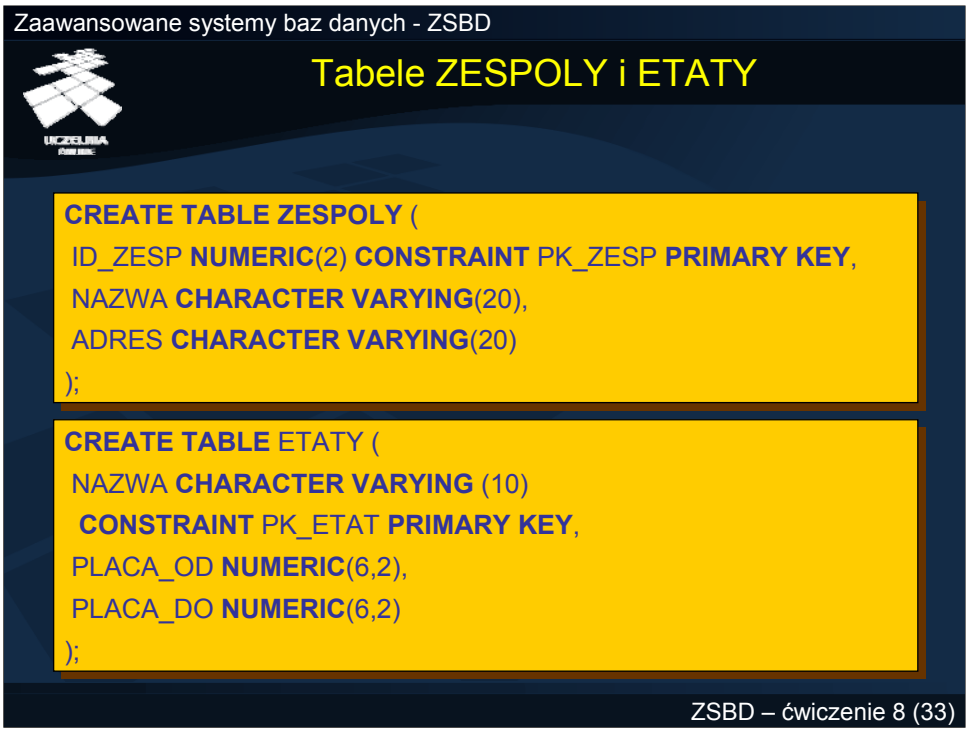
Utwórz typ zagnieżdżonej tabeli przechowującej listę tytułów wypożyczonych filmów. Utwórz typ obiektowy reprezentujący klienta wypożyczalni o atrybutach: nazwisko (typu łańcuchowego) oraz filmy (typu wcześniej zdefiniowanej kolekcji). Utwórz tabelę obiektową KLIENCI przechowującą obiekty typu KLIENT. Wstaw przynajmniej dwóch klientów z wypożyczonymi filmami.



...Wykorzystując operator TABLE i polecenia INSERT, UPDATE i DELETE, wstaw, zmień i usuń jakieś filmy z tabeli zagnieżdżonej skojarzonej z jednym z klientów.

Slajd pokazuje rozwiązanie zadania (5), którego treść przytoczono poniżej.

Napisz zapytanie, które zwróci tabelę składającą się z dwóch kolumn: nazwisko klienta i wypożyczony film.



W celu demonstracji ostatniego zagadnienia na obecnym ćwiczeniu wykorzystamy schemat tabel, który poznaliście państwo na przedmiocie poświęconemu podstawom języka SQL. Są to tabele ZESPOLY, ETATY i PRACOWNICY. Obecny i kolejny slajd stanowią krótkie przypomnienie tego schematu tabel. Skrypt tworzący tabele ZESPOLY, ETATY i PRACOWNICY załączono do kursu (pracownicy.sql).

Tabela ZESPOLY przechowuje dane dotyczące zespołów pracowników, którzy zajmują się różną tematyką naukową, bądź działalnością administracyjną. Każdy zespół posiada unikalny identyfikator (atrybut ID_ZESP), swoją nazwę (atrybut NAZWA) oraz adres (atrybut ADRES), pod którym znajdują się pomieszczenia zespołu. Tabela ETATY przechowuje dane dotyczące widełek płacowych pracowników na poszczególnych etatach. Atrybut NAZWA określa nazwę etatu, a atrybuty PLACA_OD i PLACA_DO określają jaka jest minimalna i maksymalna miesięczna płaca pracownika na określonym etacie.

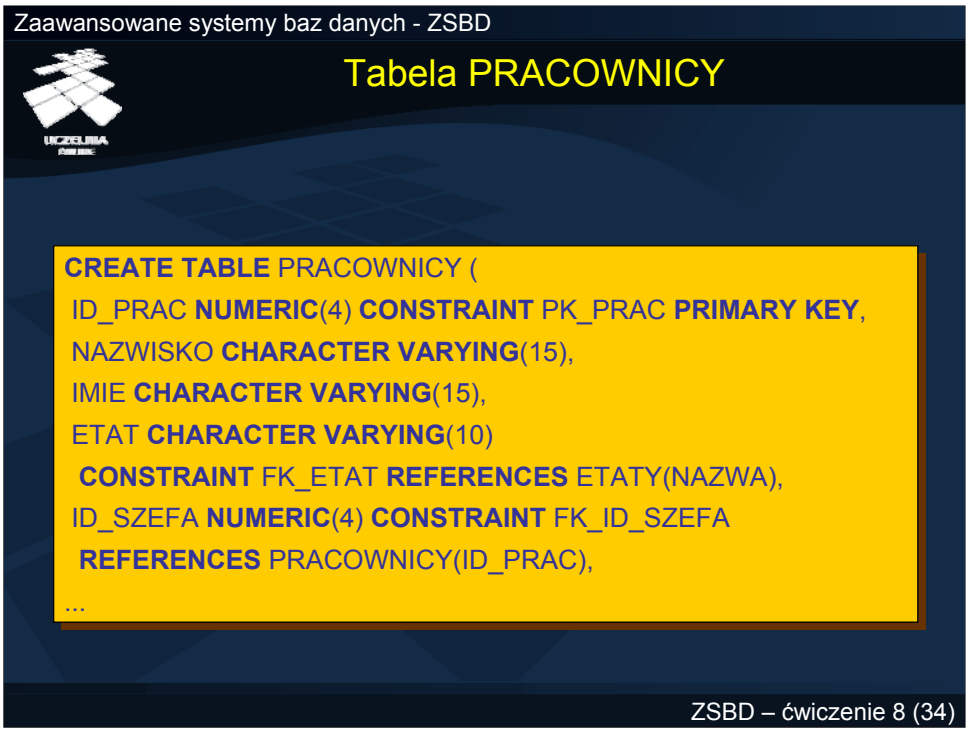
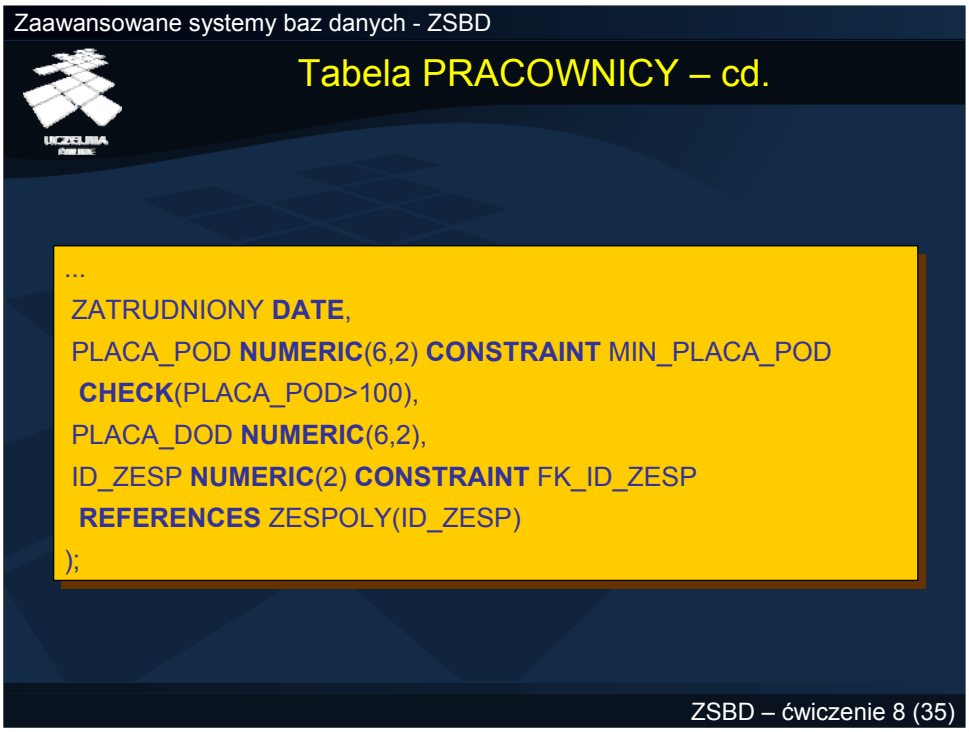


Tabela PRACOWNICY przechowuje dane dotyczące pracowników uczelni. Kolejne atrybuty mają następujące znaczenie: ID_PRAC to unikalny identyfikator pracownika, NAZWISKO i IMIE, to odpowiednio nazwisko i imię pracownika, ETAT to nazwa etatu pracownika, ID_SZEFA to wartość identyfikatora pracownika, który jest bezpośrednim szefem pracownika opisywanego w danej krotce, ...



... ZATRUDNIONY jest datą zatrudnienia pracownika, PLACA_POD, to podstawa miesięcznej pensji pracownika, która bywa niekiedy rozszerzana o płacę dodatkową (atrybut PLACA_DOD). Ostatnim atrybutem jest ID_ZESP, który opisuje wartość unikalnego identyfikatora zespołu, do którego należy pracownik.

Operator TABLE omawiany poprzednio służy do zamiany kolekcji na tabelę, którą można następnie przetwarzać za pomocą poleceń DML. Istnieje również konstrukcja o działaniu odwrotnym. Jest to konstrukcja „CAST(MULTISET(...”. Pozwala ona na zamianę wyniku zapytania w kolekcję. Rozważmy przykład przedstawiony na slajdzie. Polecenie (1) tworzy typ NAZWISKA, który reprezentuje zagnieżdżoną tabelę przechowującą łańcuchy. Typ ten jest potrzebny w zapytaniu (2). Zapytanie (2) odczytuje wszystkie etaty zapisane w tabeli ETATY i zwraca NAZWY tych etatów, oraz wartość zwracaną przez wyrażenie:

CAST(MULTISET(
SELECT NAZWISKO FROM PRACOWNICY
WHERE ETAT=NAZWA) AS NAZWISKA)

Struktura tego wyrażenia jest następująca:

CAST(MULTISET(podzapytanie) AS typ_kolekcji)

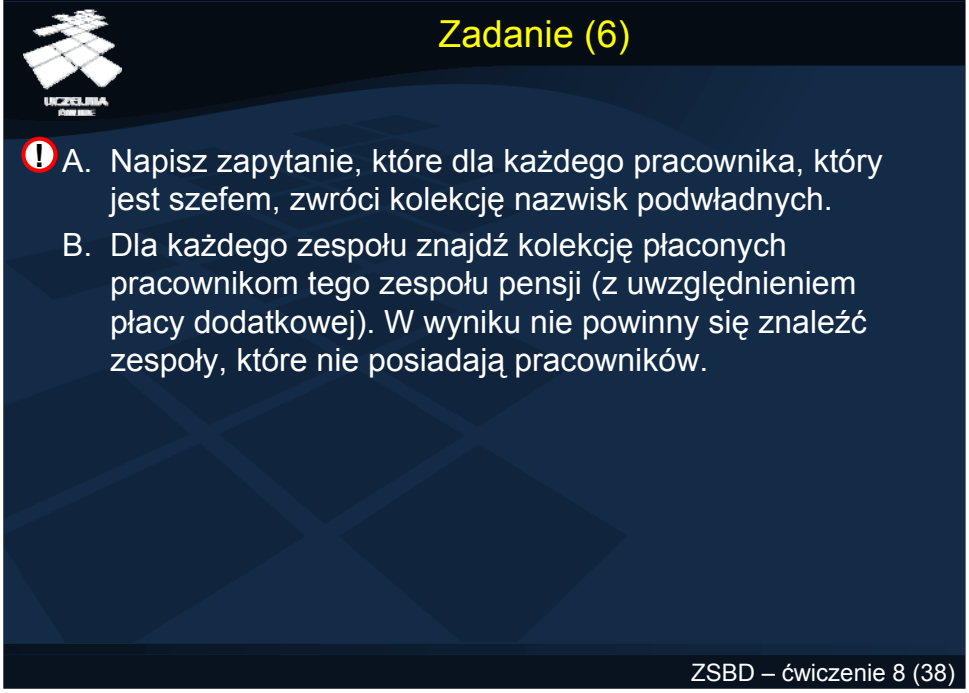
gdzie podzapytaniem jest:

SELECT NAZWISKO FROM PRACOWNICY WHERE ETAT=NAZWA,

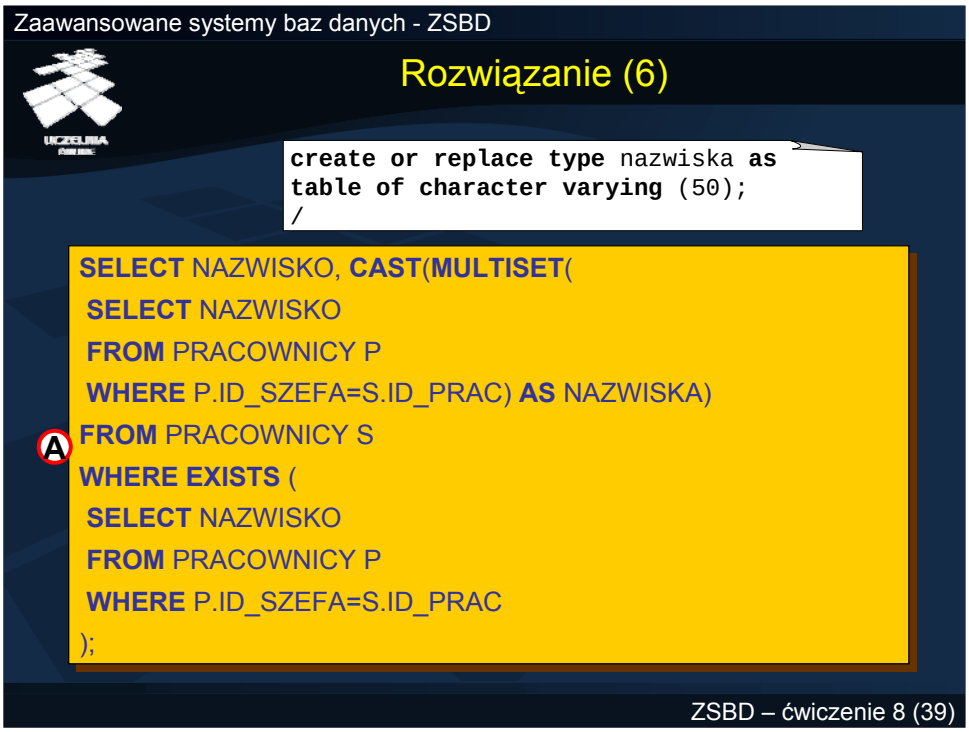
a typem kolekcji jest typ NAZWISKA.

Podzapytanie zagnieżdżone w konstrukcji „CAST(MULTISET(...” to podzapytanie skorelowane, które zwraca zbiór nazwisk pracowników pracujących na etacie, który jest aktualnie analizowany przez zapytanie zewnętrzne. W wyniku działania konstrukcji „CAST(MULTISET(...”, wynik podzapytania jest konwertowany na kolekcję, której typ podano w wyrażeniu. Można zatem powiedzieć, że zapytanie (2) zwraca dla każdego etatu kolekcję nazwisk pracowników pracujących na tym etacie.

Konstrukcji „CAST(MULTISET(...„ można używać nie tylko w zapytaniach, ale również w innych miejscach. Jest to możliwe wszędzie tam, gdzie konieczne jest podanie wyrażenia, które zwraca całą kolekcję, np. w poleceniu INSERT, przy wstawianiu nowego obiektu zawierającego kolekcję do tabeli obiektowej.

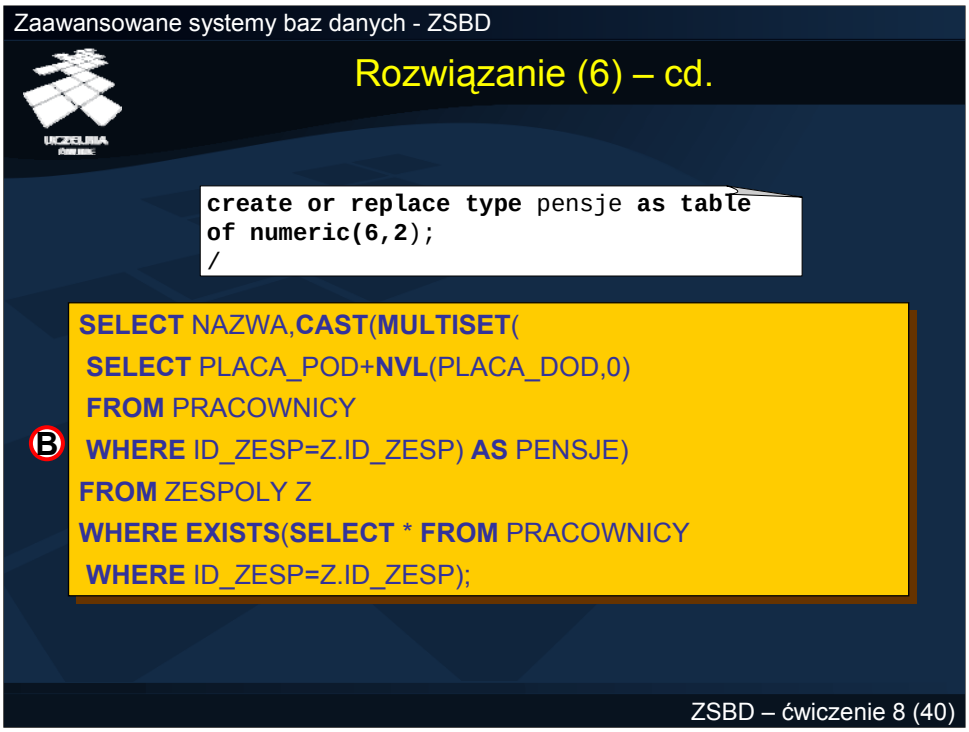


(!) Przed wykonaniem zapytań konieczne będzie utworzenie typów kolekcji, które będą zwrócone w wyniku zapytań.

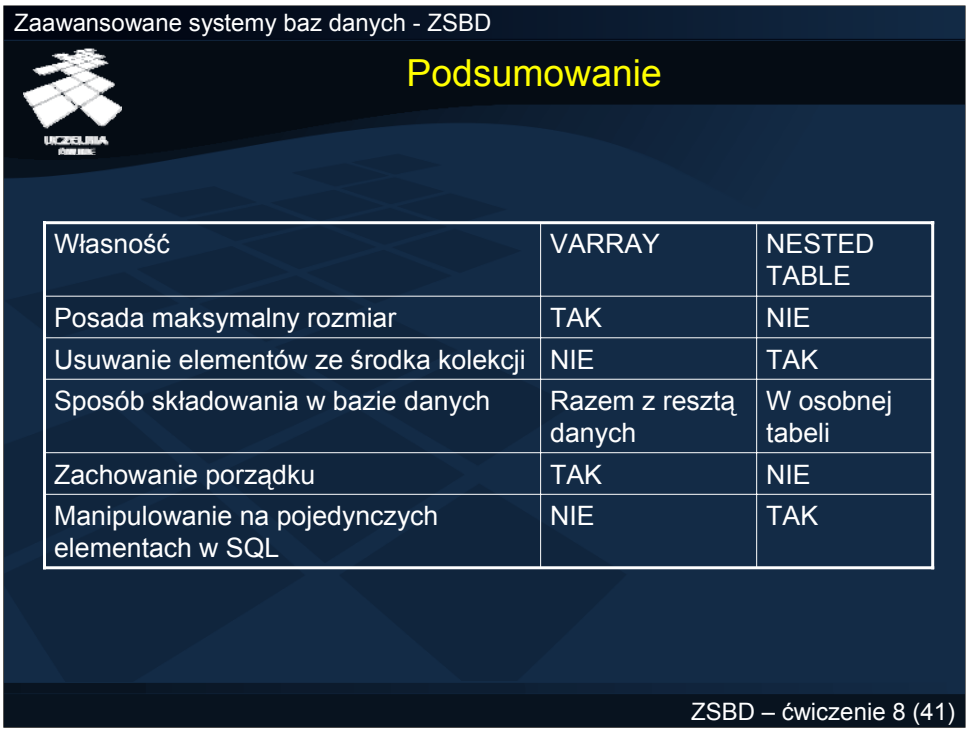


Slajd pokazuje rozwiązanie zadania (6), którego treść przytoczono poniżej. Rozwiązanie jest kontynuowane na następnym slajdzie.

(A) Napisz zapytanie, które dla każdego pracownika, który jest szefem, zwróci kolekcję nazwisk podwładnych.



(B) Dla każdego zespołu znajdź kolekcję płaconych pracownikom tego zespołu pensji (z uwzględnieniem płacy dodatkowej). W wyniku nie powinny się znaleźć zespoły, które nie posiadają pracowników.



Slajd podsumowuje różnice pomiędzy tablicami o zmiennej długości (VARRAY) oraz zagnieżdżonymi tabelami (NESTED TABLE).

Podsumowanie – cd.

- W trakcie zajęć poznaliście Państwo dwa rodzaje kolekcji, które można przechowywać w tabelach jako wartości o złożonej strukturze.
- Dowiedzieliście się jak można utworzyć kolekcje zarówno z poziomu SQL jak i PL/SQL, jak można odczytywać i modyfikować kolekcje za pomocą poleceń SQL, oraz za pomocą instrukcji w języku PL/SQL.
- Dowiedzieliście się jak można zamienić kolekcję na tabelę, oraz jak zamienić wynik zapytania w kolekcję.