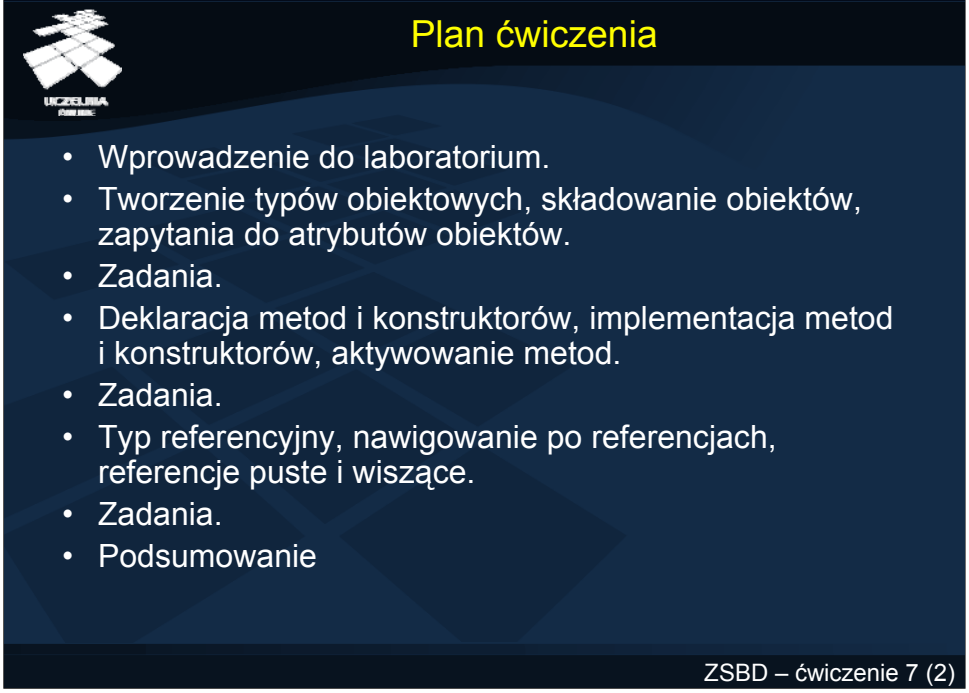
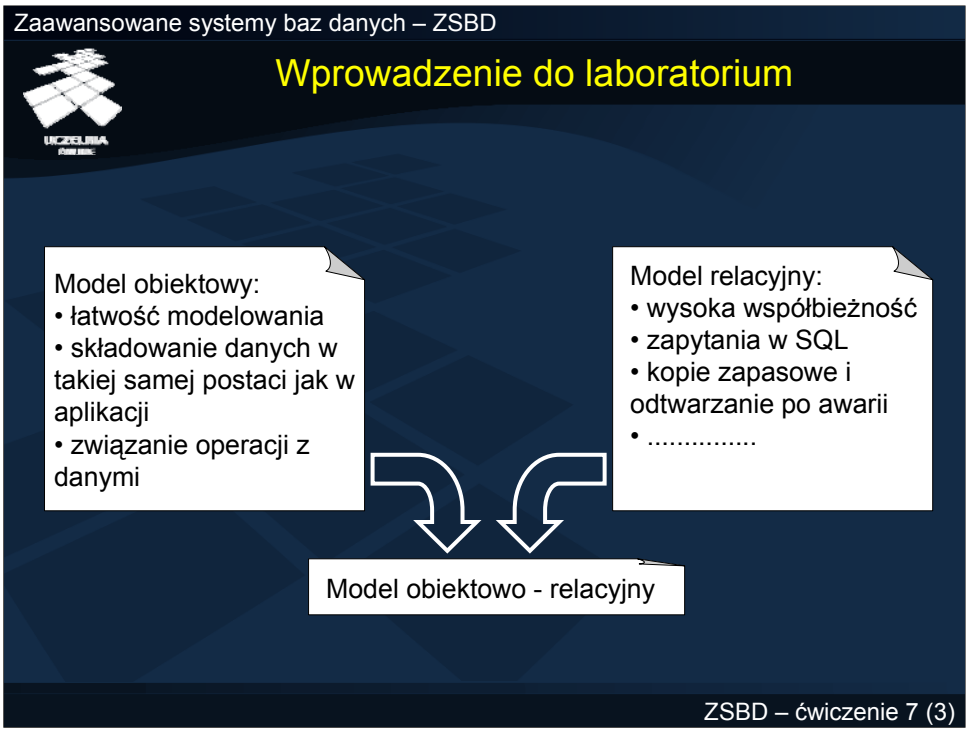
Prócz obiektowych systemów zarządzania bazą danych, z którymi zapoznaliście się Państwo na poprzednich ćwiczeniach, istnieją również obiektowo – relacyjne systemy zarządzania bazą danych (ORSZBD). Są to systemy łączące możliwości obiektowych i relacyjnych systemów zarządzania bazą danych (OSZBD i RSZBD). Celem obecnych i dwóch kolejnych ćwiczeń jest pokazanie państwu jak wygląda praca z takim systemem i pokazanie czym różni się praca z ORSZBD od pracy z OSZBD i RSZBD. Obecne ćwiczenie zapozna Państwa z podstawami pracy z ORSZBD tj. z tworzeniem typów obiektowych i ich instancji, oraz ze składowaniem obiektów w bazie danych, ze składnią języka SQL pozwalającą na przeszukiwanie obiektów, z pisaniem metod i z posługiwaniem się typami referencyjnymi.

Wymagania:

Do wykonania ćwiczenia konieczna jest dobra znajomość języka SQL i przeciętna PL/SQL.

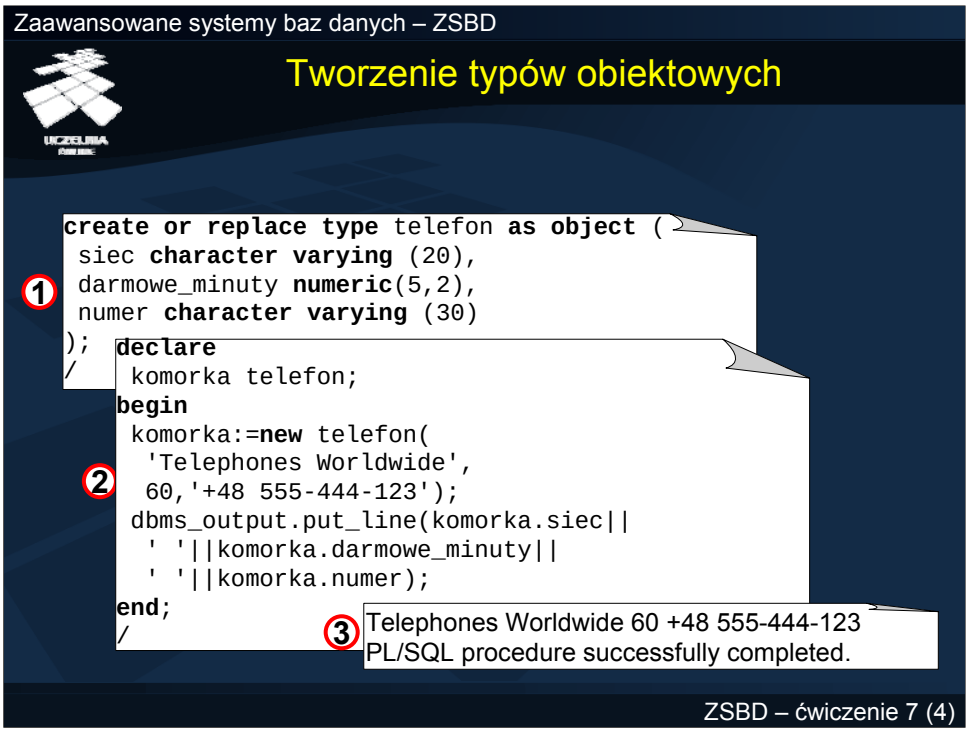


Ćwiczenie rozpocznie się od wprowadzenia motywacji będącej przyczyną powstania obiektowo relacyjnych systemów zarządzania bazą danych. Następnie, zostanie omówiona składnia poleceń pozwalających na tworzenie typów obiektowych, składowanie obiektów w tabelach, oraz wykonywanie zapytań do tabel składujących obiekty. W kolejnej części ćwiczenia pokazane zostaną polecenia pozwalające na deklarację i implementację metod i konstruktorów oraz sposób aktywowania zaimplementowanych metod. W ostatniej części ćwiczenia zapoznacie się państwo z tworzeniem typów referencyjnych, tworzeniem referencji na obiekty składowane w bazie danych oraz nawigowaniem po referencjach. Każda z ww. części zakończona jest kilkoma zadaniami do samodzielnego wykonania. Ćwiczenie zakończymy slajdem podsumowującym przedstawiony materiał.



Model obiektowo - relacyjny powstał jako rozszerzenie modelu relacyjnego o własności modelu obiektowego. Podstawową modyfikacją jest umożliwienie składowania wartości o złożonych typach (obiektów i kolekcji) w pojedynczych komórkach tabeli. Model obiektowo - relacyjny łączy w sobie zalety modelu obiektowego jakimi są: łatwość modelowania, możliwość składowania danych w takiej samej postaci w jakiej są one reprezentowane w nowoczesnych obiektowych aplikacjach i powiązanie operacji z danymi, z zaletami modelu relacyjnego, w tym: wysokiej współbieżności wynikającej z wydajnych algorytmów mechanizmów transakcyjnych, wygodnym i znanym od lat językiem zapytań SQL i rozbudowanymi mechanizmami ochrony danych na wypadek awarii.

Istnieje wiele różnych ORSZBD. Wśród najbardziej popularnych można wymienić: DB2, Informix, Oracle, Sybase i PostgreSQL. Ćwiczenia poświęcone ORSZBD oparto na systemie Oracle, gdyż zaimplementowano w nim większość podstawowych cech modelu obiektowo – relacyjnego. W terminologii Oracle klasy nazywa się „typami obiektowymi” („object types”), a pola klas „atrybutami” („attributes”). W celu zachowania jak największej zgodności z dostępną literaturą w temacie cech obiektowo – relacyjnych SZBD Oracle, postanowiono pozostać przy tej terminologii.



W SZBD Oracle typy obiektowe (klasy) są składowane w bazie danych jako obiekty bazy danych podobnie jak perspektywy, wyzwalacze albo pakiety. Typy obiektowe można utworzyć za pomocą polecenia CREATE [OR REPLACE] TYPE nazwa AS OBJECT. Przykładowe użycie tego polecenia demonstruje przykład (1). Analizując przykładowe polecenie łatwo zauważyć, że składnia tego polecenia jest podobna do składni polecenia CREATE TABLE. Po rozpoczęciu polecenia słowami kluczowymi CREATE, opcjonalnie OR REPLACE, podaniu nazwy tworzonego typu i słowach kluczowych AS OBJECT podaje się w nawiasach listę atrybutów. Listę atrybutów podaje się w sposób analogiczny do polecenia CREATE TABLE, nie można jednak definiować żadnych ograniczeń integralnościowych. Polecenie CREATE TYPE .. AS OBJECT kończy się średnikiem.

Przykład (2) zawiera program, który jest prostą demonstracją wykorzystania wcześniej utworzonego typu obiektowego. Pokazuje on jak można utworzyć obiekt tego typu i wyświetlić wartości wszystkich jego atrybutów na konsoli. Przeanalizujmy program na przykładzie (2) krok po kroku. W sekcji DECLARE deklarowana jest zmienna typu TELEFON, który został utworzony wcześniej za pomocą polecenia (1). Zmienna ta jest domyślnie inicjowana na wartość NULL. Następnie, w pierwszej linijce bloku kodu, wywoływany jest konstruktor. Jest to domyślny konstruktor, który zawsze istnieje, nawet jeżeli się go jawnie nie zaimplementuje. Konstruktor ten posiada tyle parametrów, ile zdefiniowano atrybutów w typie obiektowym. Obiekty utworzone za pomocą tego konstruktora mają wartości atrybutów zainicjowane wartościami kolejnych parametrów konstruktora. Z tego powodu, domyślny konstruktor bywa nazywany czasem konstruktorem atrybutowym. Wartością zwracaną przez konstruktor jest nowy obiekt, który jest przypisywany do zmiennej KOMORKA.

Warto zwrócić tutaj uwagę na dwie rzeczy. Po pierwsze, podobnie jak w językach C++ i JAVA, konstruktor ma nazwę identyczną z nazwą typu obiektowego. Po drugie, przed nazwą konstruktora umieszczono słowo kluczowe NEW. Słowo kluczowe NEW przed wywołaniem konstruktora jest opcjonalne i można je pominąć, ale jego użycie poprawia czytelność programu. Kolejny wiersz programu wyświetla wartości wszystkich atrybutów typu. Warto tutaj zwrócić uwagę na fakt, iż dostęp do atrybutów w obiekcie jest dokonywany za pomocą operatora kropkowego, podobnie jak w językach C++ i JAVA. Wynik działania programu pokazano na fragmencie slajdu oznaczonym przez (3).

Raz utworzony typ obiektowy można usunąć za pomocą polecenia DROP TYPE nazwa_typu [FORCE]. Słowo kluczowe FORCE oznacza, że typ obiektowy ma zostać usunięty niezależnie od jakichkolwiek przeciwwskazań.

Uwaga! Niektóre narzędzia (np. sqlplus) wymagają, aby polecenia CREATE TYPE oraz programy w PL/SQL były zakończone pojedynczym znakiem „/” w ostatniej linii (1) (2). Znak ten został umieszczony we wszystkich przykładach, w których jest to konieczne.

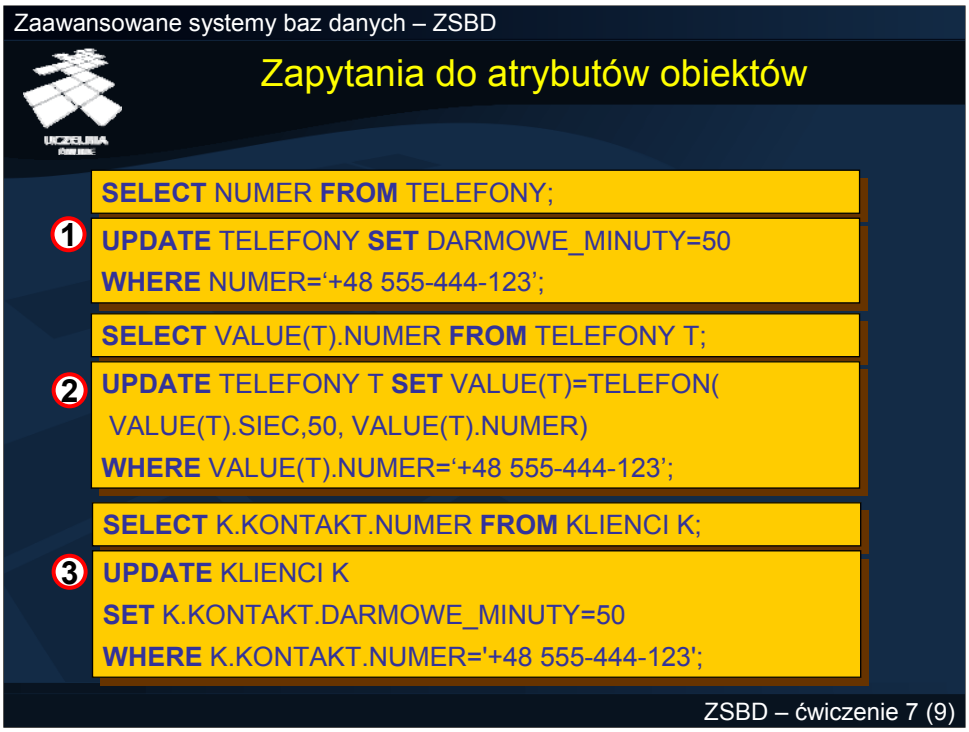
Obiekty w bazie danych mogą być składowane w dwóch rodzajach tabel. Pierwszy rodzaj tabel, to tzw. tabele obiektowe. Tabele obiektowe, podobnie jak zwykłe tabele, tworzone są za pomocą polecenia CREATE TABLE, w którym jednak zamiast listy atrybutów z ograniczeniami integralnościowymi podaje się słowo kluczowe OF i nazwę typu obiektowego. Utworzone w ten sposób tabele mogą przechowywać obiekty podanego typu i wszystkich jego podtypów. Polecenie tworzące przykładową tabelę obiektową pokazano na (1). Charakterystyczną cechą tabel obiektowych jest to, iż przechowują one w każdej „krotce" tylko jeden obiekt, tzw. obiekt krotkowy. Wstawianie obiektów do tabeli obiektowej wykonywane jest za pomocą standardowego polecenia INSERT INTO (2), w którym, w klauzuli VALUES, podaje się jedynie wyrażenie, którego wartością jest obiekt (np. wywołanie konstruktora albo zmienna PL/SQL).

Drugim rodzajem tabel są zwykłe tabele, w których jeden albo więcej atrybutów jest typu obiektowego (przykład (3)). Tworzenie tabel z atrybutami obiektowymi, jest analogiczne do tworzenia zwykłych tabel. Jedyną różnicą jest tutaj podanie zamiast standardowego typu danych, typu obiektowego. Wstawianie krotek również praktycznie niczym się nie różni od wstawiania krotek do zwykłych tabel (przykład (4)). Jedyną różnicą jest to, iż na liście wartości podawanej za słowem kluczowym VALUES w poleceniu INSERT INTO, należy umieścić również wyrażenia, których wartością jest obiekt (np. wywołanie konstruktora albo zmienna PL/SQL).

Tabele obiektowe posiadają jedną ważną własność, której nie posiadają tabele z atrybutami obiektowymi. Otóż obiekty składowane w tabelach obiektowych posiadają tożsamość. Tożsamością nazywamy globalnie unikalny identyfikator obiektu (OID). Tożsamość jest bardzo ważna przy tworzeniu referencji, które zostaną opisane później.

Charakterystyczną cechą tabel obiektowych jest to, że mogą być one traktowane jak zwykłe tabele, których atrybuty stanowią atrybuty typu obiektowego. Rozważmy przykładową tabelę obiektową TELEFONY, która może zostać utworzona za pomocą przedstawionego na poprzednim slajdzie polecenia (1). Typ TELEFON posiada trzy atrybuty: SIEC, DARMOWE_MINUTY i NUMER. Aby odczytać wartości tych atrybutów dla wszystkich obiektów zapisanych w tabeli TELEFONY, można zastosować zwykłe zapytanie, które wymienia te atrybuty w klauzuli SELECT (przykład (1)). Należy tutaj jednak zwrócić uwagę na fakt, iż takie zapytanie zwraca jedynie krotki z wartościami, a nie obiekty. Aby odczytać obiekty z tabeli obiektowej, należy zastosować operator VALUE. Operator VALUE jako parametr przyjmuje alias do tabeli obiektowej, a zwraca obiekty zapisane w tej tabeli. Zapytanie (2) demonstruje sposób użycia tego operatora. W klauzuli SELECT tego zapytania umieszczono wyrażenie VALUE(T), które oznacza „obiekt z tabeli obiektowej, której aliasem jest T”, a w klauzuli FROM podano tabelę obiektową TELEFONY i zdefiniowano jej alias T. W wyniku takiego zapytania SZBD zwróci wszystkie obiekty zapisane w tabeli obiektowej TELEFONY. Odczytanych obiektów nie można normalnie przedstawić w postaci tabelarycznej. Dlatego też, narzędzia takie jak sqlplus wyświetlają odczytane obiekty, jako wywołania konstruktorów atrybutowych, które utworzyłyby obiekty, które właśnie zostały odczytane. Przykładowo, w tabeli TELEFONY znajduje się obiekt wstawiony przez przedstawione na poprzednim slajdzie polecenie (2). W wyniku zapytania (2), na konsoli pojawia się wywołanie konstruktora atrybutowego tworzącego obiekt z takimi samymi wartościami atrybutów. Jeżeli zapytanie zostałoby wykonane z poziomu jakiegoś języka programowania, np. poprzez interfejs JDBC z poziomu języka JAVA, to odczytany zostałby rzeczywiście obiekt. Tabele obiektowe mogą być traktowane jako zwykłe tabele nie tylko w poleceniu SELECT, ale również poleceniach INSERT, UPDATE i DELETE, co zostanie przedstawione na kolejnym slajdzie.

Przykład (3) przedstawia zapytanie do tabeli z atrybutem obiektowym. Jak łatwo zauważyć, zwykłe atrybuty (takie jak NAZWISKO) są wyświetlane w sposób standardowy, a atrybuty obiektowe, podobnie jak w zapytaniu (2), jako wywołanie konstruktora atrybutowego. Oczywiście, w sytuacji, gdyby wykonać takie zapytanie z poziomu języka programowania, odczytano by odpowiednią strukturę danych.



Slajd pokazuje trzy sposoby dostępu z poziomu SQL do atrybutów obiektów składowanych w bazie danych. W przykładzie (1) pokazano dostęp do atrybutów przy wykorzystaniu dualizmu tabel obiektowych i potraktowaniu ich jako zwykłe tabele. W związku z tym, zapytanie odczytujące numer telefonu sprowadza się do wykonania standardowego zapytania: SELECT NUMER FROM TELEFONY;. Modyfikacja wartości atrybutów za pomocą polecenia UPDATE również może się odbywać tak, jakby poszczególne atrybuty typu obiektowego były atrybutami tabeli. Przykład (2) pokazuje dostęp do atrybutów, kiedy zawartość tabeli traktowana jest jako obiekty. Przeanalizujmy najpierw zapytanie w tym przykładzie. W klauzuli SELECT umieszczono wyrażenie VALUE(T).NUMER. Jak wspominano wcześniej, wyrażenie VALUE(T) oznacza „obiekt z tabeli obiektowej, o aliasie T". Ponieważ jest wyrażenie reprezentujące obiekt, to można się dostać do wartości jego atrybutów za pomocą operatora kropkowego, stąd druga część wyrażenia: .NAZWISKO. Modyfikacja wartości atrybutów za pomocą polecenia UPDATE w podejściu „obiektowym" jest niestety dość skomplikowane. Wynika to z faktu, że, w klauzuli SET, po lewej stronie operatora przypisania nie może wystąpić odwołanie do pojedynczego atrybutu (np. VALUE(T).DARMOWE_MINUTY), a jedynie cały modyfikowany obiekt, czyli VALUE(T). W takiej sytuacji, po prawej stronie operatora, również musi znaleźć się wyrażenie, którego wartością jest obiekt, np. wywołanie konstruktora. Tutaj wywoływany jest konstruktor atrybutowy, któremu jako kolejne parametry podawane są: stara wartość atrybutu SIEC (VALUE(T).SIEC), nowa wartość atrybutu DARMOWE_MINUTY (jest to jedyna zmieniana wartość w tym poleceniu), oraz stara wartość atrybutu NUMER (VALUE(T).NUMER).

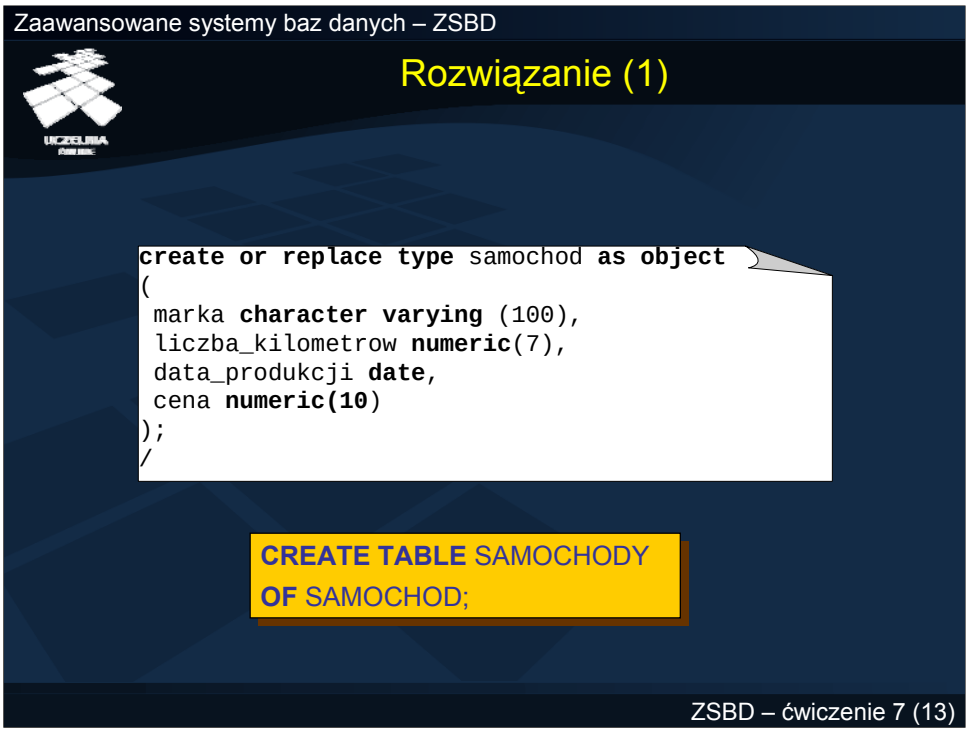
W przypadku tabel z atrybutami obiektowymi, dostęp do atrybutów odbywa się za pomocą operatora kropkowego. Przeanalizujmy zapytanie w przykładzie (3). Atrybutem obiektowym w tabeli KLIENCI jest atrybut KONTAKT, który jest typu TELEFON. Aby odczytać numer telefonu zapisanego w obiekcie znajdującego się w atrybucie kontakt, użyto wyrażenia KONTAKT.NUMER, gdzie KONTAKT reprezentuje obiekt, a NUMER jego atrybut. Należy zwrócić tutaj uwagę na jeszcze jedną ważną rzecz. Otóż, aby odwołać się do atrybutu KONTAKT użyto aliasu tabeli KLIENCI. Normalnie, w sytuacji gdy nie mamy do czynienia z połączeniami i niejednoznacznością w nazwach atrybutów, poprzedzanie nazwy atrybutu aliasem nie jest konieczne. W przypadku, kiedy chcemy uzyskać dostęp do wartości atrybutu obiektu zapisanego w atrybucie obiektowym, użycie aliasu jest wymagane. Podobnie jest z poleceniami UPDATE i DELETE. Dlatego też, w poleceniu UPDATE na przykładzie (3), modyfikowanej tabeli również nadano alias, który jest później wykorzystywany przy odwołaniach do konkretnych atrybutów.

Zadanie (1)

- Zdefiniuj typ obiektowy reprezentujący SAMOCHOD. Każdy samochód powinien mieć markę, liczbę przejechanych kilometrów oraz datę produkcji i cenę. Stwórz tabelę obiektową SAMOCHODY przechowującą obiekty tego typu, wprowadź kilka przykładowych obiektów i obejrzyj zawartość tabeli w postaci obiektowej i w postaci relacyjnej.

Zadanie (2)

- Stwórz tabelę WLASCICIELE zawierającą imiona i nazwiska właścicieli oraz atrybut obiektowy SAMOCHOD. Wprowadź do tabeli przykładowe dane i wyświetl jej zawartość.



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

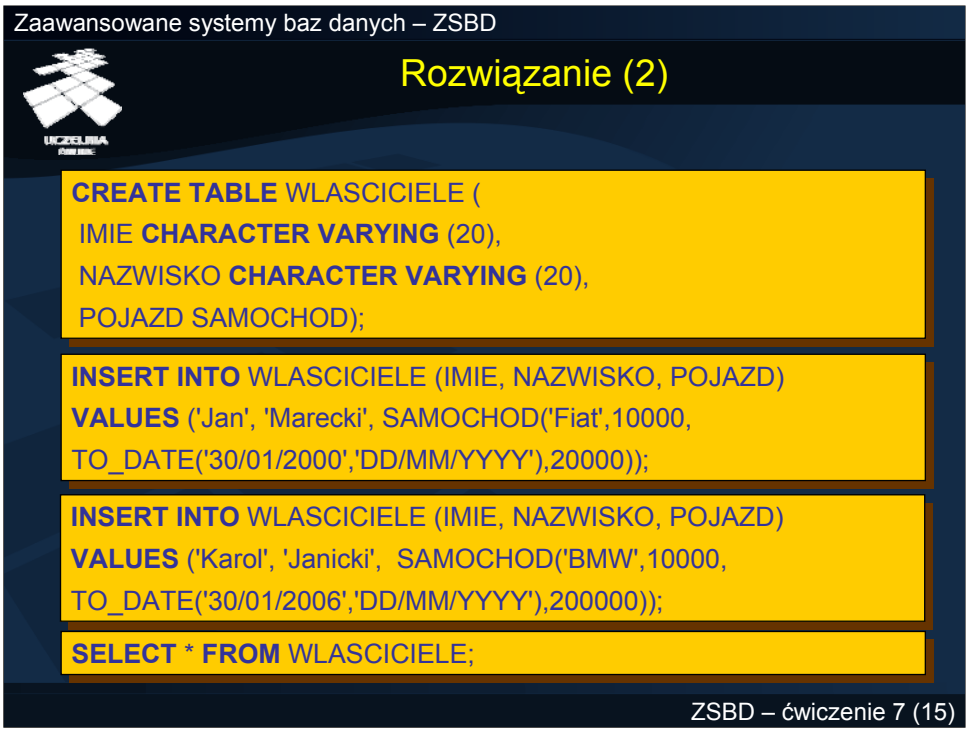
Zdefiniuj typ obiektowy reprezentujący SAMOCHOD. Każdy samochód powinien mieć markę, liczbę przejechanych kilometrów oraz datę produkcji i cenę. Stwórz tabelę obiektową SAMOCHODY przechowującą obiekty tego typu, wprowadź kilka przykładowych obiektów i obejrzyj zawartość tabeli w postaci obiektowej i w postaci relacyjnej.

Rozwiązanie (1) – cd.

```
INSERT INTO SAMOCHODY VALUES(SAMOCHOD('Ford',100000,
 TO_DATE('01/01/2000','DD/MM/YYYY'),25000));
INSERT INTO SAMOCHODY VALUES(SAMOCHOD('Ferrari',230000,
 TO_DATE('01/02/2002','DD/MM/YYYY'),30000));
INSERT INTO SAMOCHODY VALUES(SAMOCHOD('Toyota',150000,
 TO_DATE('01/07/2003','DD/MM/YYYY'),20000));
INSERT INTO SAMOCHODY VALUES(SAMOCHOD('Tarpan',500000,
 TO_DATE('01/01/1980','DD/MM/YYYY'),15000));
```

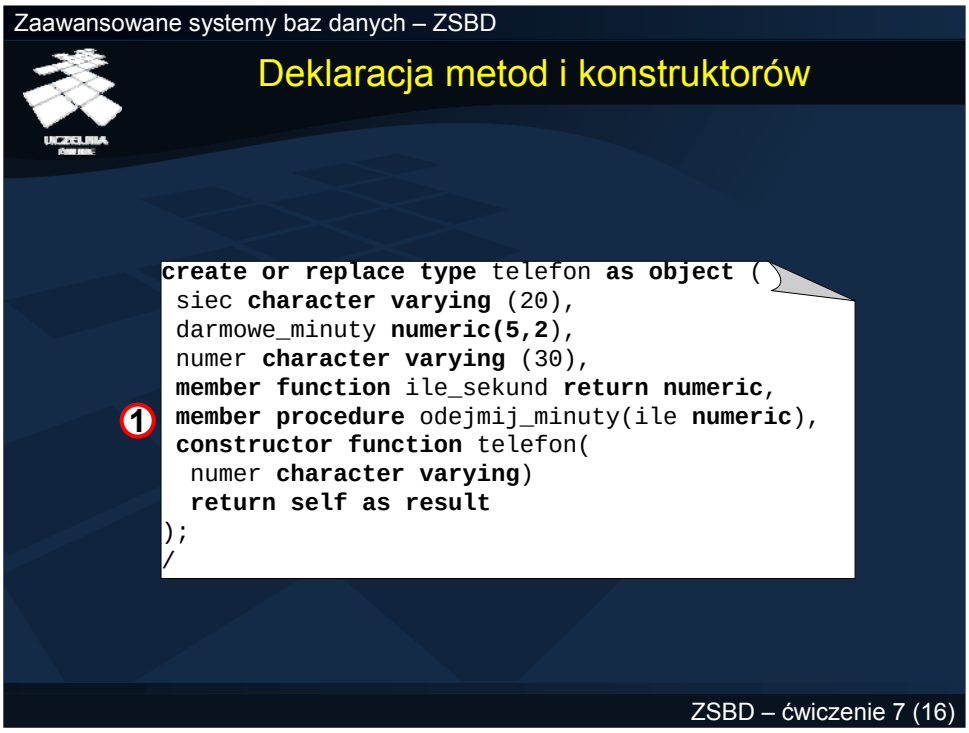
```
SELECT VALUE(S) FROM
SAMOCHODY S;
```

```
SELECT *
FROM SAMOCHODY;
```



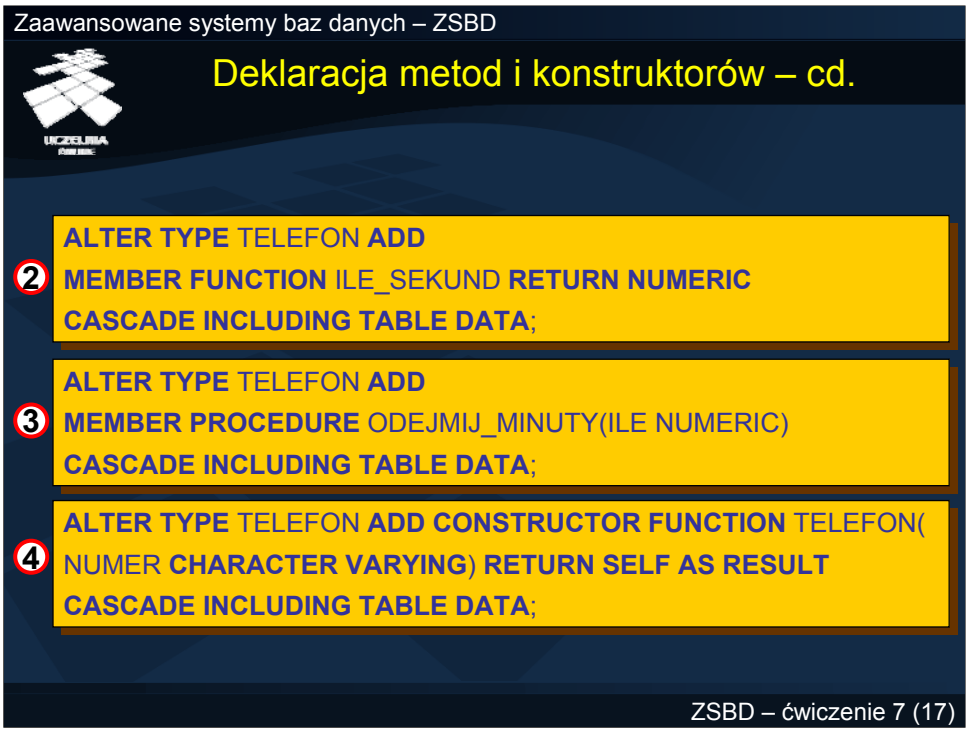
Slajd pokazuje rozwiązanie zadania (2), którego treść przytoczono poniżej.

Stwórz tabelę WLASCICIELE zawierającą imiona i nazwiska właścicieli oraz atrybut obiektowy SAMOCHOD. Wprowadź do tabeli przykładowe dane i wyświetl jej zawartość.



Zajmiemy się obecnie rozbudowaniem poznanego przez państwa modelu obiektowego o metody. Podobnie jak w języku C++ deklaracja i implementacja metod jest rozdzielona. Deklaracja metod znajduje się razem z opisem atrybutów na liście podawanej podczas tworzenia typu obiektowego. Na przykładzie (1) pokazano, jak można zadeklarować trzy różne rodzaje metod w typie obiektowym TELEFON. Deklarację zwykłych metod (pierwsze dwie metody na przykładzie (1)) rozpoczyna słowo kluczowe MEMBER, za którym znajduje się kolejne słowo kluczowe określające, czy deklarowana metoda jest funkcją (FUNCTION) czy procedurą (PROCEDURE). Po tych słowach kluczowych podaje się nazwę metody, za którą można opcjonalnie umieścić listę parametrów formalnych. Składnia listy parametrów formalnych jest taka sama jak w przypadku składowanych procedur i funkcji PL/SQL. Ostatecznie, w przypadku metod – funkcji, podaje się jeszcze typ zwracanych wartości za pomocą słowa kluczowego RETURN, po którym podaje się nazwę typu. Deklaracje metod są od siebie oddzielone przecinkiem. Warto tutaj zwrócić uwagę na fakt, że dla typów parametrów formalnych oraz typów zwracanych wartości nie określa się precyzji składowanych wartości. Przykładowo, zamiast NUMERIC(10) używa się samej nazwy typu: NUMERIC.

Przedstawiony przed chwilą opis nie obejmuje trzeciej zadeklarowanej metody. Metodą tą jest konstruktor użytkownika. Deklarację konstruktora rozpoczyna się od słów kluczowych CONSTRUCTOR FUNCTION (konstruktory są zawsze funkcjami), następnie podaje się nazwę konstruktora, która musi być taka sama jak nazwa typu obiektowego. Za nazwą konstruktora można opcjonalnie umieścić listę parametrów formalnych. Deklarację konstruktora kończą słowa kluczowe RETURN SELF AS RESULT, co znaczy, że w wyniku wywołania konstruktora zostanie zwrócony nowy obiekt.

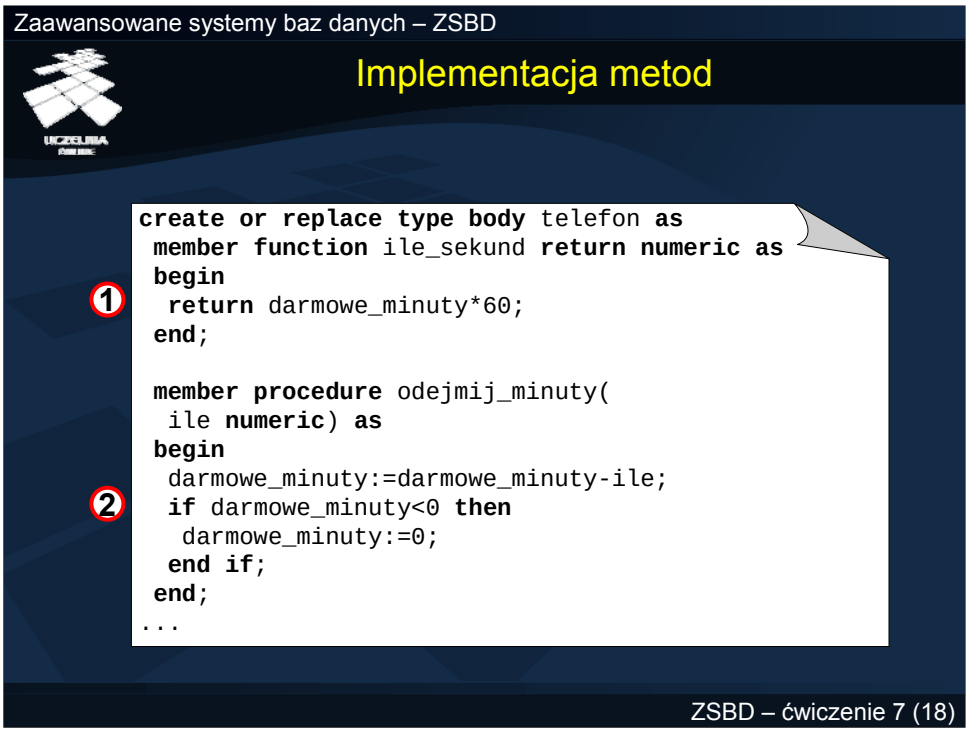


Przykład (1) z poprzedniego slajdu pokazuje jak można utworzyć nowy typ, który zawiera deklarację metod. Nie jest możliwe jednak utworzenie nowego typu, jeżeli istnieje stary typ o tej samej nazwie i posiada on podtypy, bądź istnieje tabela obiektowa składująca obiekty tego typu. Aby dodać deklarację metody do istniejącego typu obiektowego należy użyć polecenia ALTER TYPE, tak jak pokazano na przykładach (2), (3) i (4). Składnia polecenia ALTER TYPE, w przypadku dodawania deklaracji metod do typu wygląda następująco. Polecenie zaczyna się od słów kluczowych ALTER TYPE, za którymi podaje się nazwę typu, słowo kluczowe ADD, i ostatecznie dodawaną deklarację metody. Jeżeli istnieją tabele obiektowe przechowujące obiekty modyfikowanego typu, bądź tabele z atrybutami obiektowymi, należy dodać jeszcze słowa kluczowe CASCADE INCLUDING TABLE DATA, które mówią, aby wszystkie składowane obiekty modyfikowanego typu również zostały poddane odpowiednim modyfikacjom. W podobny sposób można dodawać atrybuty do typu obiektowego. W tym przypadku, po słowie kluczowym ADD należy podać jeszcze słowo kluczowe ATTRIBUTE, za którym należy umieścić opis atrybutu (nazwa i typ), np. :

ALTER TYPE TELEFON ADD ATTRIBUTE MARKA CHARACTER VARYING (50) CASCADE INCLUDING TABLE DATA;

Polecenie ALTER TYPE pozwala również na usuwanie atrybutów i deklaracji metod, jeżeli zamiast słowa kluczowego ADD użyje się słowa kluczowego DROP, np.:

ALTER TYPE TELEFON DROP ATTRIBUTE MARKA CASCADE INCLUDING TABLE DATA;



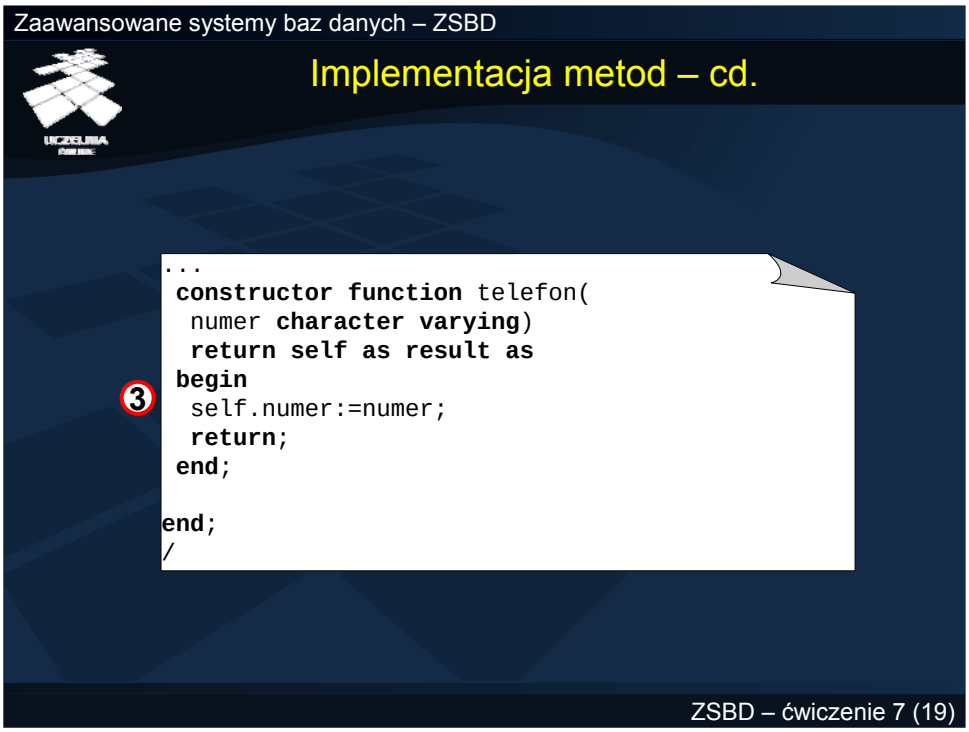
Metody zadeklarowane podczas tworzenia typu są implementowane w tzw. *ciele typu*. Ciało typu jest osobnym obiektem bazy danych, który przypomina budową ciało pakietu. Ciało typu można utworzyć za pomocą polecenia CREATE TYPE BODY. Polecenie rozpoczyna się słowem kluczowym CREATE, za którym można opcjonalnie podać słowa kluczowe OR REPLACE. Kolejnymi słowami kluczowymi są TYPE BODY, za którym podaje się nazwę typu, dla którego tworzone jest ciało, po którym podaje się słowo kluczowe AS. Dalsza część polecenia CREATE TYPE składa się z implementacji zadeklarowanych wcześniej metod i kończy się ostatecznie słowem kluczowym END i średnikiem. Implementacja metody rozpoczyna się od takiego samego wyrażenia, jakie stanowiło deklarację tej metody, zakończonego słowem kluczowym AS. Przykładowo, jeżeli deklaracja metody wyglądała następująco:

MEMBER FUNCTION ILE_SEKUND RETURN NUMERIC

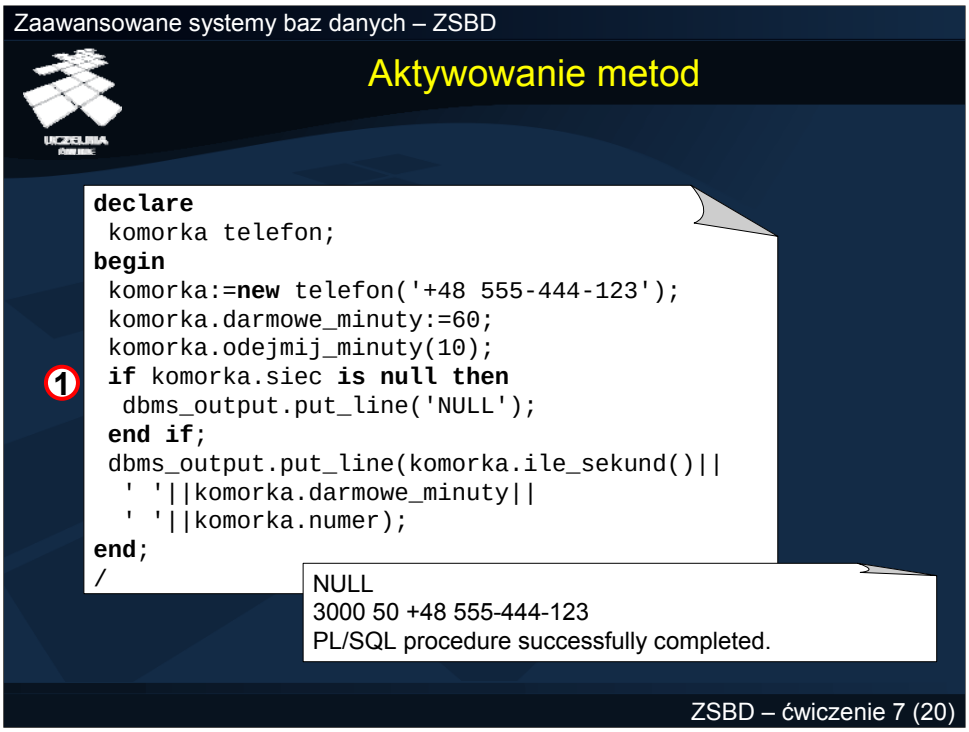
To implementację tejże metody rozpoczyna się od:

MEMBER FUNCTION ILE_SEKUND RETURN NUMERIC AS

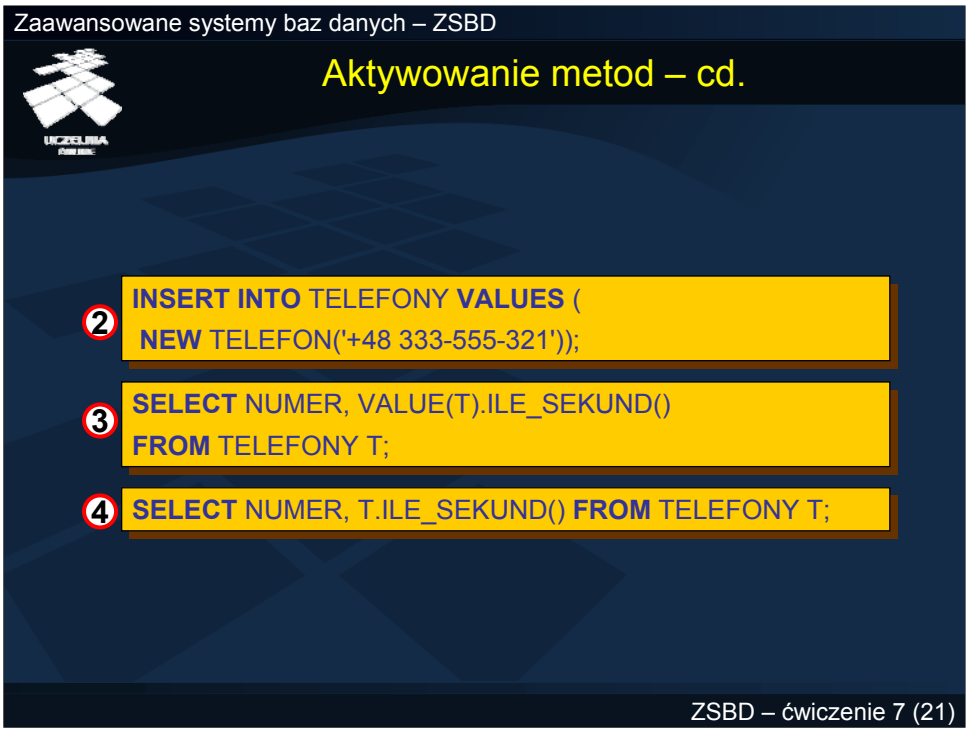
Po tym wyrażeniu, podobnie jak w przypadku procedur składowanych, rozpoczyna się opcjonalna sekcja deklaracji zmiennych lokalnych, po której, począwszy od słowa kluczowego BEGIN rozpoczyna się właściwy kod implementujący logikę metody. Przeanalizujmy przykładowe metody. Metoda ILE_SEKUND (1) jest funkcją obliczającą ile darmowych sekund rozmów zostało dla danego telefonu. Ponieważ w obiekcie typu TELEFON czas darmowych rozmów przechowywany jest w minutach, to funkcja ta mnoży jedynie składowaną liczbę darmowych minut razy 60. Jak łatwo zauważyć, metody mają bezpośredni dostęp do wartości poszczególnych atrybutów w obiekcie, na którego rzecz zostały aktywowane. Metoda ODEJMIJ_MINUTY (2) jest procedurą, która modyfikuje liczbę darmowych minut, odejmując od nich wartość przekazaną jako parametr aktualny metody. Jeżeli, po odjęciu, liczba darmowych minut jest ujemna, to liczba darmowych minut jest zerowana. ...



... Konstruktor (3) pokazuje dwie nowe rzeczy. Pierwszą, jest użycie specjalnego wyrażenia SELF, w wierszu SELF.NUMER:=NUMER. Wyrażenie SELF jest specjalnym wyrażeniem, które reprezentuje obiekt, na którego rzecz została aktywowana metoda. W przypadku konstruktora stanowi on nowo utworzony obiekt. Najbliższym odpowiednikiem SELF znanym z innych języków programowania jest wskaźnik this z języka C++. SELF jednak nie jest wskaźnikiem ani referencją na obiekt, tylko wartością, czyli konkretnym obiektem. W przykładowym konstruktorze został on użyty w celu rozwiązania niejednoznaczności w nazwach parametru formalnego i atrybutu typu obiektowego. Drugą nową rzeczą jest użycie instrukcji RETURN bez żadnego parametru. Takie użycie RETURN, w przypadku konstruktora, świadczy o tym, że w tym punkcie programu działanie konstruktora ma się zakończyć, a nowo utworzony obiekt ma zostać zwrócony jako wynik działania.



Aktywowanie metod odbywa się podobnie jak dostęp do wartości atrybutu przechowywanej w obiekcie, za pomocą operatora kropkowego (za wyjątkiem konstruktora). Zacznijmy od analizy przykładowego bloku PL/SQL (przykład (1)). Blok ten zaczyna się od deklaracji pomocniczej zmiennej KOMORKA typu TELEFON. W pierwszej linijce właściwego kodu, zmiennej KOMORKA przypisywany jest wynik działania konstruktora, czyli nowy obiekt typu TELEFON. Zauważmy, ze jest to wywołanie zdefiniowanego i zaimplementowanego na poprzednich slajdach konstruktora, a nie wywołanie konstruktora atrybutowego. Kolejny wiersz przypisuje atrybutowi DARMOWE_MINUTY liczbę 60. Następnie, za pomocą metody ODEJMIJ_MINUTY, wartość atrybutu DARMOWE_MINUTY jest zmniejszana o 10. Jak widać, aktywacja metody ODEJMIJ_MINUTY odbywa się tutaj podobnie jak w językach C++ albo JAVA, czyli za pomocą operatora kropkowego. W kolejnych trzech wierszach, za pomocą instrukcji warunkowej sprawdzane jest, czy atrybut SIEC w obiekcie przechowywanym w zmiennej KOMORKA zawiera wartość NULL czy nie. Ponieważ zaimplementowany przez nas konstruktor nie modyfikował tej wartości należy się spodziewać, że warunek ten będzie spełniony. Ostatecznie, za pomocą procedury PUT_LINE wypisywane są na konsoli: wynik obliczeń w funkcji ILE_SEKUND, wartość atrybutu DARMOWE_MINUTY, oraz numer telefonu. Jeżeli spojrzeć na wynik działania przykładowego programu, to można zauważyć, że jest on zgodny z oczekiwaniami. Wartość atrybutu SIEC wynosi NULL, liczba zwracana przez metodę ILE_SEKUND jest 60 razy większa od liczby zapisanej w atrybucie DARMOWE_MINUTY (wcześniej nieco pomniejszonej za pomocą metody ODEJMIJ_MINUTY), oraz widać, że numer telefonu przekazany jako parametr aktualny konstruktora został zapisany w atrybucie NUMER zgodnie z jego implementacją. ...



... Przykład (2) pokazuje, że wywołanie konstruktora zdefiniowanego przez użytkownika z poziomu SQL, niczym się nie różni od wywołania konstruktora atrybutowego. Przykład (3) pokazuje w jaki sposób można aktywować metodę w zapytaniu. Za pomocą operatora VALUE uzyskiwany jest obiekt, odczytany z bazy danych, a następnie, za pomocą operatora kropkowego, aktywowana jest jego metoda. Warto tutaj wspomnieć o trzech ważnych rzeczach. Po pierwsze, w ten sposób aktywowane mogą być tylko metody - funkcje. Jest to o tyle oczywiste, że procedury nie zwracają wartości, a zatem ich użycie w klauzuli SELECT nie ma sensu. Drugą ważną rzeczą jest to, iż metody - funkcje nie mogą modyfikować wartości atrybutów obiektu. Mogą jedynie wartości odczytywać, i wykonywać na ich podstawie jakieś obliczenia. Wynika to z faktu, że metody - funkcje mogą być, jak pokazano przed chwilą, aktywowane z wewnątrz polecenia SELECT. Polecenie SELECT reprezentuje zapytanie, a zatem może dane jedynie odczytywać i nie może wprowadzać żadnych modyfikacji do tych danych, a zatem aktywowanie metod-funkcji nie może wprowadzać żadnych ubocznych efektów. Trzecią rzeczą, na którą warto zwrócić uwagę, to fakt, iż nawet aktywacja metody bezparametrowej, wymaga podania za nazwą metody pustej listy parametrów aktualnych („( )”), co można łatwo zauważyć na przykładzie (3) w klauzuli SELECT. Przykład (4) pokazuje nieco skróconą wersję przykładu (3). Zrezygnowano tutaj z użycia operatora VALUE. Jest to dopuszczalne pod warunkiem, że aktywacja metody jest poprzedzona aliasem tabeli obiektowej.

Zadanie (3)

- Wartość samochodu maleje o 10% z każdym rokiem. Dodaj do typu obiektowego SAMOCHOD metodę wyliczającą na podstawie wieku aktualną wartość samochodu (użyj polecenia ALTER, a następnie zaimplementuj ciało typu). Wykonaj zapytanie do tabeli SAMOCHODY, które aktywuje tą metodę.

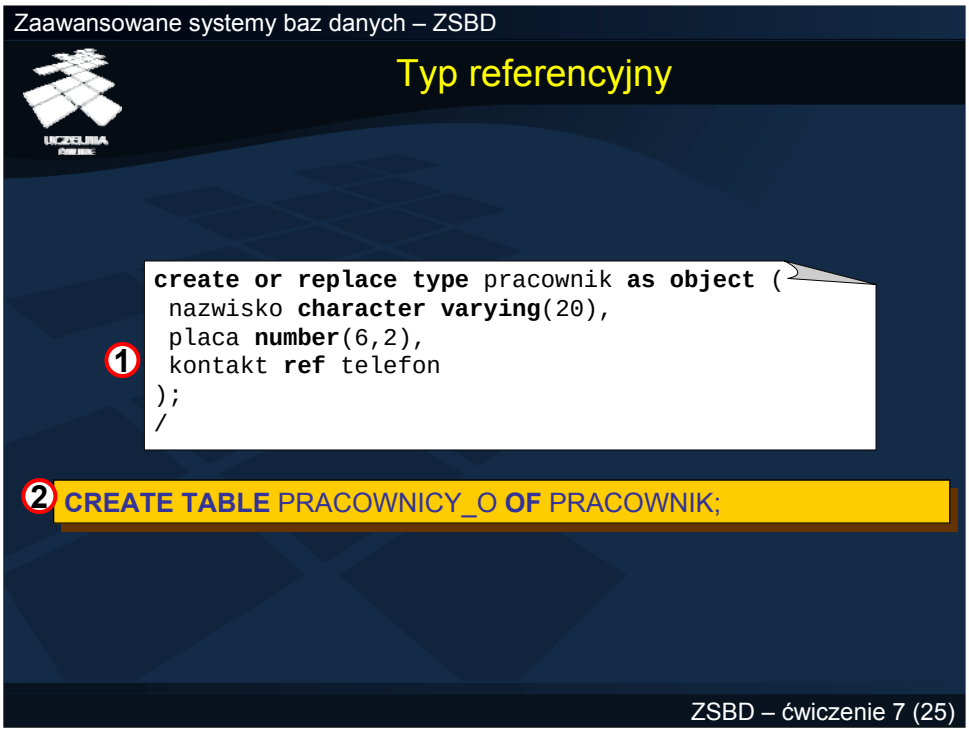
Slajd pokazuje rozwiązanie zadania (3), którego treść przytoczono poniżej. Rozwiązanie jest kontynuowane na kolejnym slajdzie.

Wartość samochodu maleje o 10% z każdym rokiem. Dodaj do typu obiektowego SAMOCHOD metodę wyliczającą na podstawie wieku aktualną wartość samochodu (użyj polecenia ALTER TYPE, a następnie zaimplementuj ciało typu). Wykonaj zapytanie do tabeli SAMOCHODY, które aktywuje tą metodę.

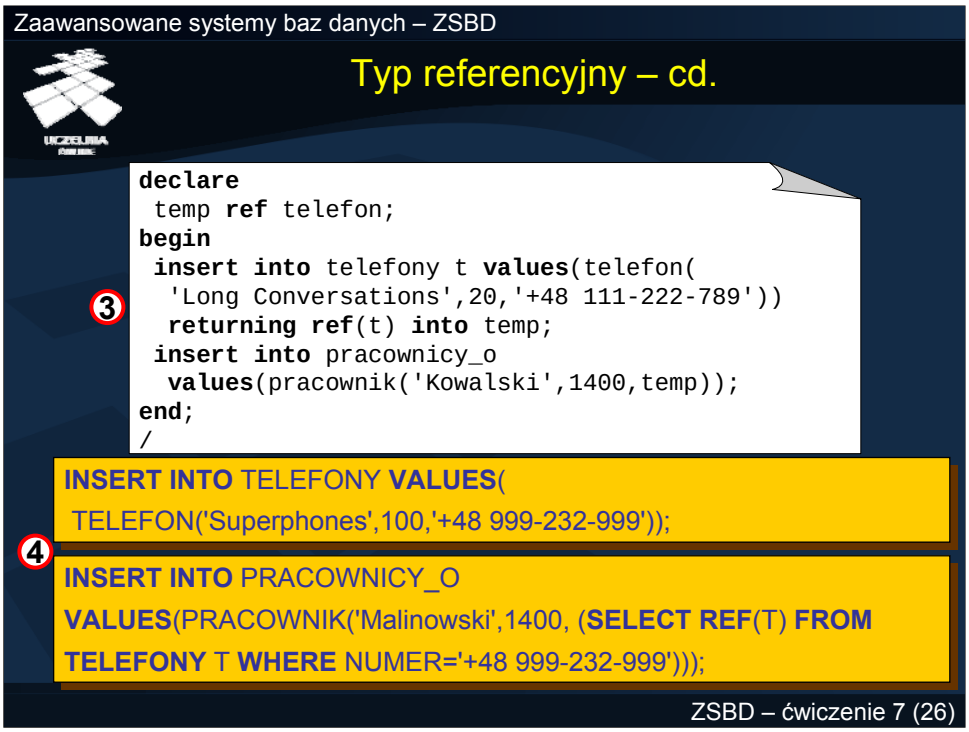
Rozwiązanie (3) – cd.

```
create or replace type body samochod as
 member function wartosc return numeric as
  result numeric;
 begin
  result:=cena*(1-0.1*months_between(
   sysdate,data_produkcji)/12);
  if result>0 then
   return result;
  else
   return 0;
  end if;
 end;
end;
/
```

```
SELECT MARKA,VALUE(S).WARTOSC() FROM SAMOCHODY S;
```



Jedną z cech model obiektowego, które stanowią o jego sile, jest możliwość tworzenia referencji do innych obiektów. Referencje przypominają w przybliżeniu wskaźniki z języka C++. Znając wskaźnik, a więc adres obiektu w pamięci, można go szybko odnaleźć i odczytać z niego dane. Dzięki referencjom możemy szybko odnajdować obiekty, które są składowane w bazie danych. Rozważmy przykłady pokazane na slajdzie. Przykład (1) pokazuje instrukcję tworzącą typ PRACOWNIK, który posiada atrybut KONTAKT, które może przechowywać referencje na obiekty typu TELEFON. Jak łatwo zauważyć, nazwa typu, który stanowi referencję na obiekt składa się ze słowa kluczowego REF oraz nazwy typu obiektowego, na którego referencje mają być składowane. Przykład (2) zawiera polecenie CREATE TABLE tworzące tabelę obiektową przechowującą obiekty typu PRACOWNIK. ...



... Program na przykładzie (3) pokazuje w jaki sposób można utworzyć telefon i pracownika, a następnie powiązać te obiekty za pomocą referencji. Program rozpoczyna się od deklaracji pomocniczej zmiennej przechowującej referencje na obiekty typu telefon. Właściwy kod programu składa się z dwóch poleceń INSERT wstawiających obiekty do tabel TELEFONY i PRACOWNICY_O. Pierwsze polecenie INSERT jest zakończone konstrukcją: „...RETURNING REF(T) INTO TEMP”. Powoduje to zapisanie referencji, wskazującej na wstawiony do tabeli obiekt, do zmiennej TEMP. Parametr operatora REF, którego tutaj użyto, to alias tabeli do której wstawiono obiekt. Zmienna TEMP jest używana w drugim poleceniu INSERT jako parametr aktualny konstruktora atrybutowego typu obiektowego PRACOWNIK. W rezultacie wykonania programu (3) w tabeli TELEFONY zostanie zapisany nowy obiekt reprezentujący telefon, a w tabeli PRACOWNICY_O zostanie zapisany nowy obiekt reprezentujący pracownika, który posiada referencję na wstawiony wcześniej telefon.

Podobny efekt można uzyskać stosując polecenia DML. Przykład (4) składa się z dwóch poleceń INSERT. Pierwsze polecenie wstawia nowy obiekt reprezentujący telefon do tabeli TELEFONY. Drugie polecenie wstawia nowego pracownika do tabeli PRACOWNICY_O. Zwróćmy tutaj uwagę na to, iż referencja na obiekt reprezentujący telefon pracownika jest tutaj uzyskiwana za pomocą podzapytania:

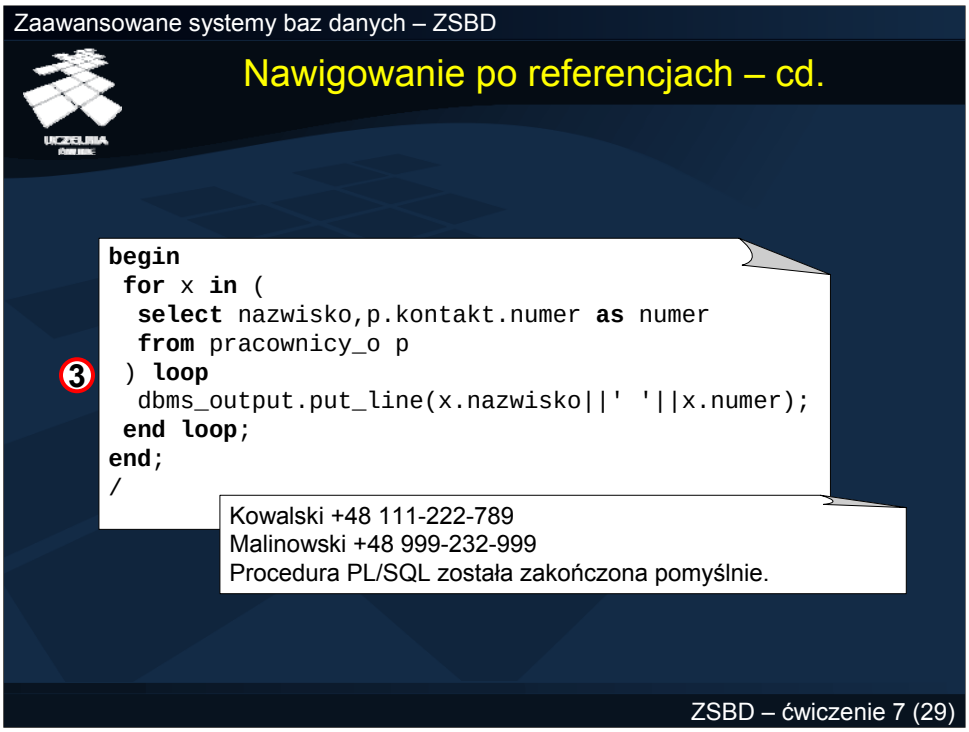
SELECT REF(T) FROM TELEFONY T WHERE NUMER='+48 999-232-999’

Jeżeli pominąć chwilowo zawartość klauzuli SELECT, łatwo zauważyć, że podzapytanie odnajduje wszystkie obiekty o zadanym numerze telefonu. Ponieważ nie ma wielu telefonów o jednym numerze, to zapytanie to powinno zwrócić zero lub jeden wynik, a ponieważ poprzednie polecenie ten telefon wstawiło mamy pewność, że podzapytanie znajdzie jeden obiekt. Wróćmy teraz do klauzuli SELECT. Zawiera ona jedynie wywołanie operatora REF. Wartością tego operatora jest referencja na obiekt pochodzący z tabeli obiektowej, której alias podano jako parametr. Podsumowując, podzapytanie w drugim poleceniu INSERT zwraca referencję na obiekt, który posiada zadany numer telefonu.

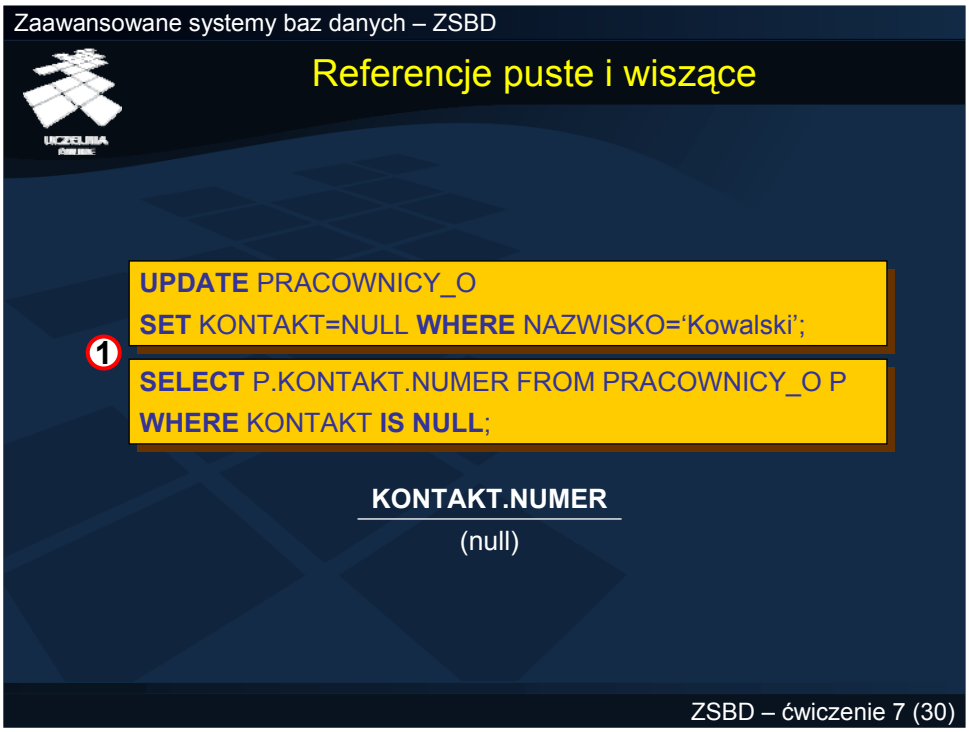
Obecnie warto byłoby przedyskutować jedną kwestię. Tłumacząc działanie podzapytania założono, że nie może być więcej niż jeden obiekt o jednym numerze telefonu w bazie danych. W ogólności jest to jednak możliwe, gdyż nie ma żadnego ograniczenia integralnościowego, które by tego pilnowało. Jeżeli więcej niż jeden obiekt o takim samym numerze telefonu znalazłby się w tabeli TELEFONY, to polecenie INSERT zakończyłoby się błędem. Aby tego uniknąć można stosować dodatkowe warunki selekcji, jednak w ogólności może się zdarzyć, że będzie wiele obiektów o takich samych wartościach na wszystkich atrybutach. Jest to kolejna cecha charakterystyczna modelu obiektowego. W przeciwieństwie do modelu relacyjnego, w którym krotki są identyfikowane poprzez przechowywane dane, w modelu obiektowym obiekty mogą mieć takie same dane, ale różną tożsamość. Dlatego też jedynym całkowicie poprawnym sposobem odnajdywania referencji, jest sposób przedstawiony na przykładzie (3).

Należałoby tutaj przypomnieć o jeszcze jednej rzeczy. Otóż, jak wspominano wcześniej, jedynie obiekty składowane w tabelach obiektowych posiadają tożsamość. Referencja jest, logicznie, wartością tożsamości (czyli unikalnego identyfikatora) wskazywanego obiektu. W praktyce często zawiera również fizyczną lokalizację wskazywanego obiektu na dysku. Konsekwencją tego, że referencja zawiera tożsamość wskazywanego obiektu, jest to, że referencję można uzyskać jedynie na obiekty składowane w tabeli obiektowej.

Aby uzyskać dostęp do obiektu wskazywanego przez referencję należy wykonać operację dereferencji. Dereferencji można dokonać w sposób jawny, bądź niejawny. Zapytania (1) i (2) wykonują to samo zapytanie, jednak zapytanie (1) wykonuje jawną dereferencję, a zapytanie (2) niejawną. Przeanalizujmy zapytanie (1). Zapytanie to odczytuje wartość atrybutu NAZWISKO, oraz wartość wyrażenia DEREF(KONTAKT).NUMER, którym jest numer telefonu należącego do danego pracownika. Operator DEREF jest operatorem dereferencji. Jako parametr przyjmuje on referencję na obiekt, a jego wartością jest wskazywany obiekt. Atrybut KONTAKT stanowi referencję na obiekt typu TELEFON, dlatego też, dla każdego pracownika, operator DEREF zwraca wskazywany przez referencję KONTAKT telefon. Mając dany obiekt typu TELEFON można w łatwy sposób, za pomocą operatora kropkowego, odczytać jego numer („.NUMER”). W niektórych sytuacjach operator DEREF można pominąć. Jeżeli nawigujemy za pomocą operatora kropkowego, poprzez referencję, do wartości atrybutów wskazywanego obiektu, to wykonywana jest automatyczna dereferencja i operator DEREF nie jest tutaj potrzebny. Operator DEREF znajduje zastosowanie właściwie jedynie w sytuacjach, kiedy chcemy uzyskać w wyniku zapytania całe obiekty, na które mamy dane referencje. Należy tutaj jeszcze zauważyć, że nazwę atrybutu przechowującego referencje poprzedzono aliasem tabeli obiektowej. W sytuacji gdy korzystamy z niejawnej dereferencji, jest to obowiązkowe.

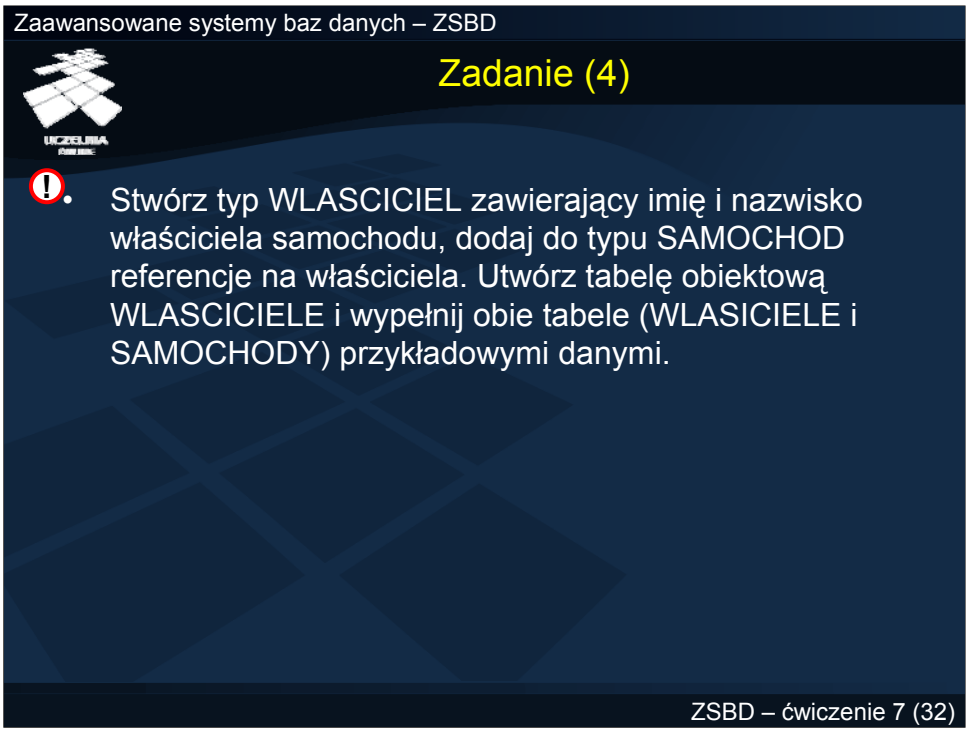


W PL/SQL nie jest możliwe wykonywanie dereferencji, zarówno jawnych, jak i niejawnych. Aby wykonać dereferencję należy zagnieździć polecenie SQL w bloku PL/SQL. Przykład (3) demonstruje właśnie takie zagnieżdżenie.



Jeżeli mamy do czynienia z referencjami, to zawsze należy rozważyć dwa problemy. Po pierwsze: „Co się stanie jeżeli spróbujemy nawigować poprzez referencję o wartości NULL?”. Po drugie: „Co się stanie, jeżeli usuniemy obiekt, na który inny obiekt w bazie danych przechowuje referencję?”. Zacznijmy od pierwszego problemu. Zachowanie się SZBD Oracle w takiej sytuacji demonstruje przykład (1). W przykładzie tym, referencji na telefon Kowalskiego przypisywana jest wartość NULL, a następnie, korzystając z konstrukcji poznanej na poprzednim slajdzie, próbujemy odczytać numer telefonu wskazywanego przez pustą referencję. Nawigacja poprzez taką referencję musi się z definicji skończyć niepowodzeniem. W większości języków taka próba kończy się zgłoszeniem przez program wyjątku. W SZBD Oracle, w wyniku nawigacji poprzez pustą referencję zawsze otrzymamy wartość pustą. Stąd, przedstawiony na slajdzie wynik zapytania sklada się jedynie z wartości pustej.

Zachowanie się SZBD Oracle w sytuacji, gdy zostanie usunięty obiekt na który istnieją referencje w bazie danych, demonstruje przykład (2). Polecenie DELETE usuwa obiekt reprezentujący telefon Malinowskiego. Pierwszym podejrzeniem byłoby to, iż atrybutowi stanowiącemu referencję na ten obiekt zostanie automatycznie przypisana wartość pusta. Pierwsze polecenie SELECT w przykładzie (2) pokazuje że tak nie jest. Nie istnieje w tabeli PRACOWNICY_O żaden obiekt, który by posiadał wartość referencji na telefon równą NULL i wartość atrybutu NAZWISKO równą „Malinowski". Do wykrywania, czy referencja wskazuje na nieistniejący obiekt (potocznie „czy referencja 'wisi'?") służy operator IS DANGLING. Sposób użycia operatora demonstruje drugie zapytanie w przykładzie (2). Zapytanie to odszukuje wszystkie obiekty, które posiadają referencję zapisaną w atrybucie KONTAKT wskazującą na nieistniejący telefon. Zapytanie zwraca w wyniku Malinowskiego, któremu, na początku przykładu, telefon usunięto .



(!) W celu wykonania zadania należy usunąć tabelę WLASCICIELE utworzoną w poprzednim zadaniu. Do typu SAMOCHOD należy dodać referencję na obiekt typu WLASCICIEL za pomocą polecenia ALTER. Samochody w tabeli obiektowej SAMOCHODY można albo usunąć i wstawić nowe, tym razem z referencją na właściciela, bądź zmodyfikować istniejące za pomocą polecenia UPDATE.

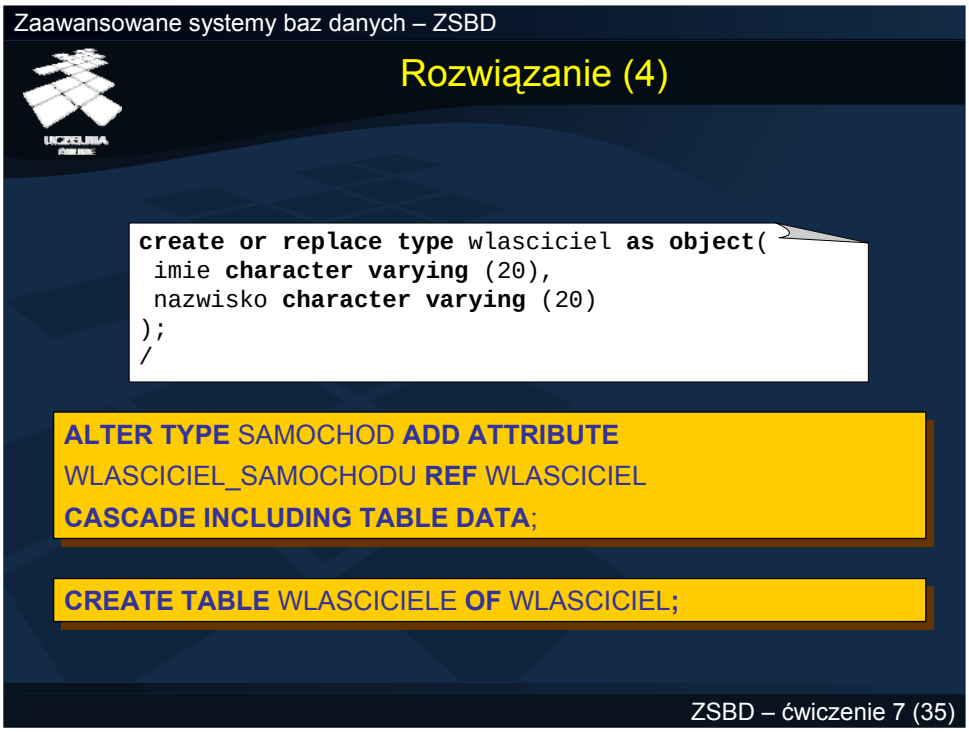
Zadanie (5)

- Wypisz marki wszystkich samochodów i nazwiska ich właścicieli korzystając z nawigacji poprzez referencję (nie wykorzystuj połączeń).

Zadanie (6)

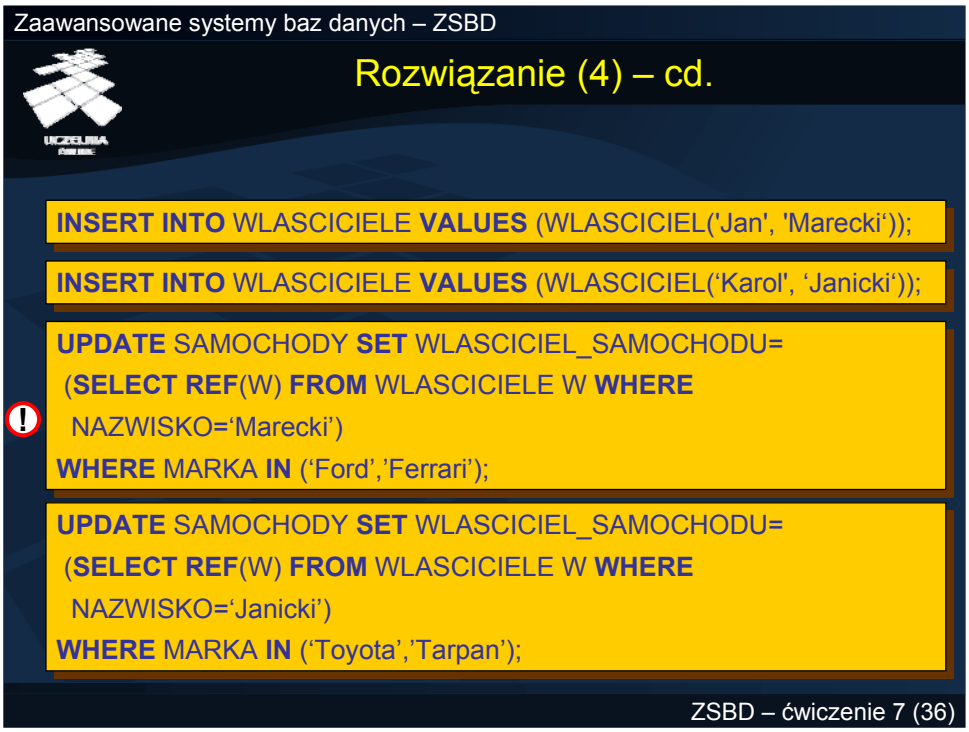
- Usuń z tabeli WLASCICIELE jeden obiekt. Wyszukaj w tabeli SAMOCHODY samochody, które mają wiszące referencje na WLASCICIELA i przypisz tym referencjom NULL. Zadanie należy rozwiązać za pomocą pojedynczego polecenia DELETE i pojedynczego UPDATE.

(!)



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

Stwórz typ WLASCICIEL zawierający imię i nazwisko właściciela samochodu, dodaj do typu SAMOCHOD referencje na właściciela. Utwórz tabelę obiektową WLASCICIELE i ...



... wypełnij obie tabele (WLASICIELE i SAMOCHODY) przykładowymi danymi.

(!) Wykorzystano tutaj samochody utworzone w poprzednim zadaniu.

Slajd pokazuje rozwiązania zadań (5) i (6), których treść przytoczono poniżej.

- (5)Wypisz marki wszystkich samochodów i nazwiska ich właścicieli korzystając z nawigacji poprzez referencję (nie wykorzystuj połączeń).
- (6)Usuń z tabeli WLASCICIELE jeden obiekt. Wyszukaj w tabeli SAMOCHODY samochody, które mają wiszące referencje na WLASCICIELA i przypisz tym referencjom NULL.

Podsumowanie

- W trakcie zajęć zapoznaliście się Państwo z podstawami funkcjonalności obiektowo relacyjnych systemów zarządzania bazą danych.
- Dowiedzieliście się jak można: stworzyć typy obiektowe, składować obiekty, zaimplementować metody w typach obiektowych, aktywować te metody z poziomu SQL i z poziomu PL/SQL, stosować referencje i jak unikać związanych z referencjami problemów.