Relacje nie są jedynymi obiektami bazy danych, które można utworzyć ja pomocą języka definicji danych. Celem ćwiczenia ósmego jest przedstawienie państwu innych obiektów bazy danych, które są kluczowe ze względu poprawności działania, wydajności, bezpieczeństwa i przenaszalności aplikacji korzystających z systemów baz danych. Ćwiczenie omawia polecenia języka SQL służące do konstrukcji i wykorzystania sekwencji oraz tworzenia indeksów. W dalszej części ćwiczenia przedstawione zostaną polecenia tworzące perspektywy relacyjne. Ćwiczenie omawia również warunki modyfikowalności perspektyw.

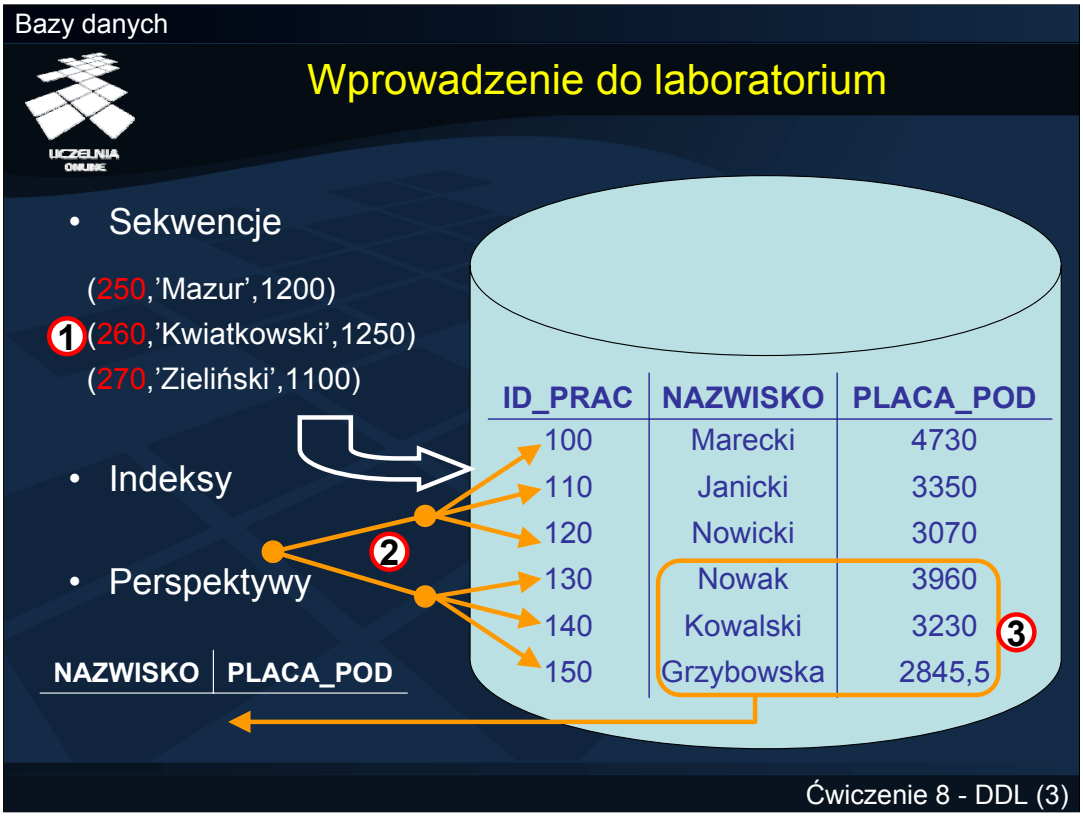
Wymagania:

Umiejętność konstrukcji zapytań, tworzenia relacji oraz wstawiania, modyfikacji i usuwania danych z relacji. Wykonanie ćwiczeń od 1 do 7 z systemów baz danych.

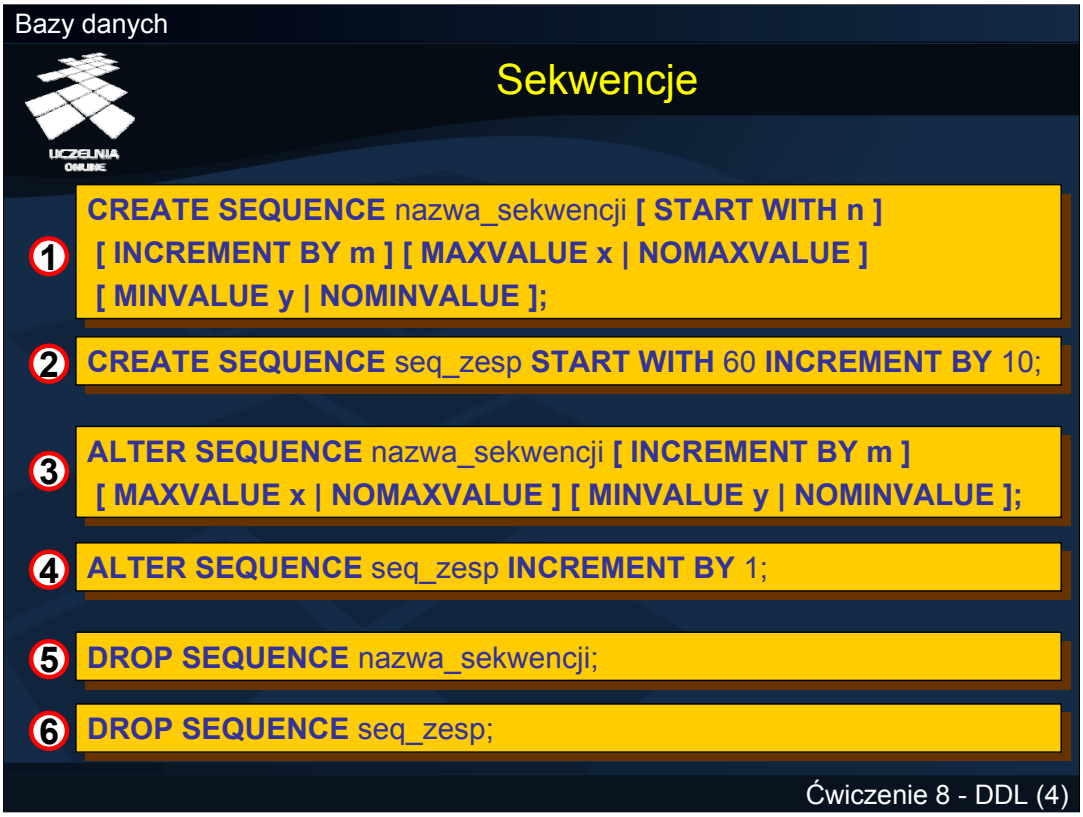
Plan ćwiczenia

- Wprowadzenie do laboratorium.
- Sekwencje.
- Indeksy.
- Perspektywy (proste, złożone, modyfikowalne, niemodyfikowalne, weryfikujące ograniczenia i wbudowane)
- Zadania
- Podsumowanie.

Ćwiczenie rozpoczniemy od wprowadzenia do laboratorium, na którym przedstawimy zarys omawianej tematyki. Następnie opiszemy składnię poleceń pozwalających na tworzenie i wykorzystanie: sekwencji, indeksów i perspektyw. Każdy z wyżej wymienionych tematów zostanie zakończony prostym zadaniem do samodzielnego wykonania. Na końcu ćwiczenia przedstawimy kilka dodatkowych ćwiczeń utrwalających omówione tematy.



Na ćwiczeniu zapoznacie się Państwo z sekwencjami, indeksami i perspektywami. Sekwencje stanowią mechanizm ułatwiający automatyczne generowanie kluczy podstawowych (1), który znajduje zastosowanie wszędzie tam, gdzie kluczem podstawowym nie jest jakiś atrybut opisujący fragment świata rzeczywistego (takim „naturalnym" kluczem podstawowym, który nie wymaga automatycznego wygenerowania, jest na przykład PESEL) ale sztucznie utworzony atrybut stworzony tylko w celu identyfikacji krotki (najczęściej jest to jakiś numer porządkowy). Indeksy są obiektami bazy danych, które pozwalają na znaczne przyspieszenie wykonywanie zapytań na dużych bazach danych (2). Perspektywy stanowią swego rodzaju „okno", które przetwarza i odsłania część danych z relacji (3). Takie okna pozwalają na ograniczenie dostępu do danych dla różnych użytkowników, jak również pozwalają na dostosowywanie schematu relacji do schematu wymaganego przez nową aplikację do pracy z istniejącą już bazą danych.

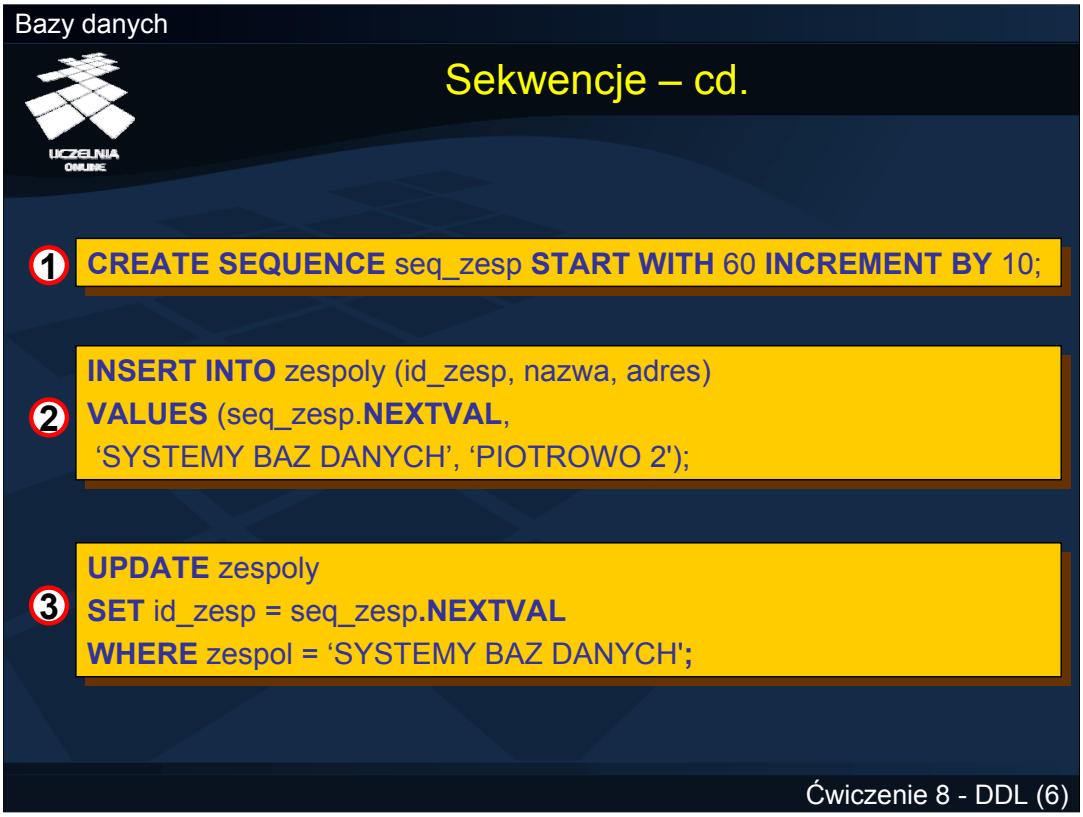


Na poprzednim ćwiczeniu pokazaliśmy Państwu sposób generowania nowych kluczy, który polegał na tym, iż znajdowano za pomocą podzapytania w poleceniu INSERT największą wartość klucza w tabeli i zwiększano go o jakąś stałą. Niestety, sposób ten jest poprawny jedynie w sytuacji, gdy w danym momencie do tabeli wstawia dane tylko jeden użytkownik. Rozważmy sytuację, w której dwóch użytkowników próbuje równocześnie wstawić nową krotkę. Obu użytkowników równocześnie znajduje maksymalną wartość klucza w tabeli i zwiększa go o ustaloną wcześniej liczbę. Ponieważ obydwu tych użytkowników widzi takie same dane w relacji, oblicza taki sam klucz podstawowy dla nowej krotki. Ponieważ ograniczenie integralnościowe „klucz podstawowy” zabrania wstawiania dwóch krotek o takich samych wartościach na atrybutach wchodzących w skład klucza podstawowego, tylko jednemu z tych użytkowników uda się wstawić krotkę. Dla drugiego użytkownika operacja wstawienia danych zakończy się błędem. Użytkownik, któremu operacja się nie udała, może oczywiście próbować ponowić operację zapisania krotki do relacji, ale zawsze istnieje możliwość, że pojawi się nowy użytkownik, z którym ten pierwszy znowu przegra. W ogólności jeden użytkownik może nigdy nie być w stanie wstawić krotki, gdyż zawsze jakiś inny użytkownik z nim będzie wygrywał i wstawiał krotkę o takim samym kluczu pierwszy.

Aby rozwiązać wyżej opisany problem, można użyć sekwencji. Sekwencja jest obiektem przechowującym pewną wartość liczbową. Każda próba odczytania tej wartości powoduje wpierw zwiększenie jej o ustaloną wielkość. Zwiększenie i odczytanie tej wartości odbywa się w sposób atomowy, dzięki czemu nie jest możliwe, aby jakikolwiek użytkownik korzystający z SZBD w danym momencie odczytał taką samą wartość z sekwencji. Odczytane z sekwencji wartości mogą następnie zostać wykorzystane jako wartość klucza podstawowego nowych krotek, dzięki czemu problem równoczesnej pracy kilku użytkowników jest rozwiązany.

W ogólności problem generowania nowych wartości klucza jest rozwiązywany w różny sposób w różnych SZBD. W Oracle, DB2 i PostgreSQL można zastosować sekwencje. Prócz sekwencji, w DB2 można zastosować również tzw. atrybuty tożsamościowe (ang. *identity columns*). Są to atrybuty, do których automatycznie wstawiana jest nowa wartość przy wstawianiu nowej krotki do relacji. W PostgreSQL istnieje również podobne rozwiązanie. Jest nim „typ danych" o nazwie SERIAL, którego użycie jest równoważne utworzeniu sekwencji i domyślnemu zapisywaniu do odpowiedniego atrybutu wartości odczytanej z sekwencji. W MySQL również istnieje rozwiązanie analogiczne do atrybutów tożsamościowych z DB2, ale jest to realizowane poprzez dopisanie słowa kluczowego AUTO_INCREMENT w definicji atrybutu, w sposób podobny do definicji ograniczenia atrybutu.

Na tym ćwiczeniu przedstawiono składnię poleceń dotyczących sekwencji używaną w Oracle SZBD, jednak składnia tych poleceń jest prawie identyczna dla wszystkich SZBD, w których można korzystać z sekwencji. Przykład (1) pokazuje ogólną składnię polecenia CREATE SEQUENCE tworzącego sekwencje w bazie danych. Polecenie rozpoczyna się od słów kluczowych CREATE SEQUENCE, za którymi podaje się nazwę sekwencji, a następnie klauzule pozwalające określić: pierwszą wartość licznika w sekwencji (START WITH), o ile ma być zwiększany licznik przy każdym odczycie (INCREMENT BY), maksymalną i minimalną wartość licznika, lub ich brak (MAXVALUE, NOMAXVALUE, MINVALUE, NOMINVALUE). Przykład (2) pokazuje polecenie CREATE SEQUENCE tworzące sekwencję o nazwie SEQ_ZESP, której pierwszą zwracaną wartością będzie 60, a każda kolejna wartość będzie większa o 10 od poprzedniej. Można wpływać na parametry sekwencji za pomocą polecenia ALTER SEQUENCE, którego ogólną składnię przedstawiono na przykładzie (3). Składnia polecenia jest prawie identyczna z poleceniem CREATE SEQUENCE. Pierwszą różnicą jest zastąpienie słowa kluczowego CREATE słowem ALTER, a drugą jest to, iż nie można modyfikować początkowej wartości sekwencji. Wszystkie pozostałe parametry można zmieniać. Przykład (4) pokazuje polecenie ALTER SEQUENCE modyfikujące utworzoną wcześniej sekwencję SEQ_ZESP tak, aby kolejne wartości zwracane przez sekwencję były tylko o jeden większe od poprzednich. Sekwencję można usunąć za pomocą polecenia DROP SEQUENCE, którego jedynym parametrem jest nazwa usuwanej sekwencji (5). Przykład (6) pokazuje jak można usunąć sekwencję o nazwie SEQ_ZESP.



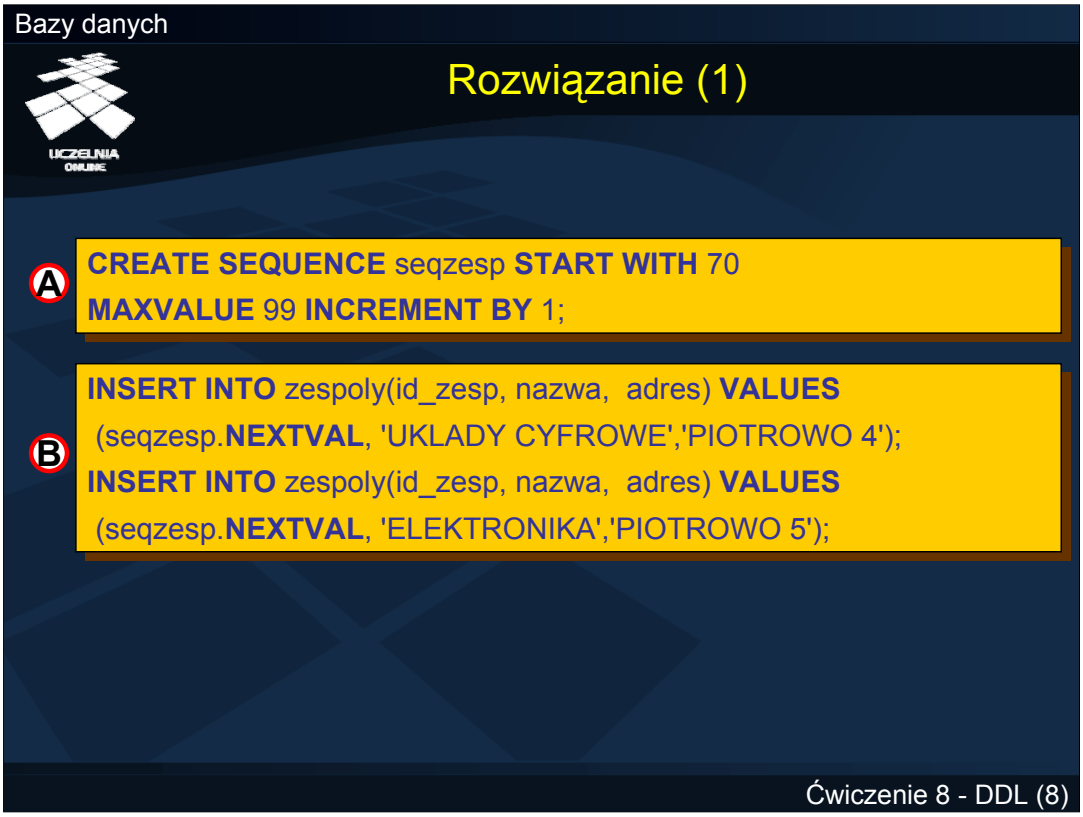
Na poprzednim slajdzie omówiliśmy sposób tworzenia, modyfikacji i usuwania sekwencji. Obecnie pokażemy państwu w jaki sposób można odczytać wartość sekwencji. W przeciwieństwie do poleceń służących do tworzenia i usuwania sekwencji, które są prawie identyczne w wielu SZBD, sposób odczytu wartości z sekwencji już nie jest. Na slajdzie przedstawiono składnię charakterystyczną dla Oracle. W poniższych opisach przedstawimy również składnię dla dwóch innych SZBD.

Zacznijmy od rozważenia przykładu (1). Jest on przypomnieniem polecenia tworzącego przykładową sekwencję SEQ_ZESP. Przykład (2) pokazuje sposób użycia sekwencji do wygenerowania klucza podstawowego dla nowo tworzonej krotki. Jak łatwo zauważyć, kolejną wartość z sekwencji odczytuje się podając najpierw nazwę sekwencji, a potem kropkę i słowo NEXTVAL. Prócz NEXTVAL, przy odwoływaniu się do sekwencji można użyć słowa CURRVAL, które powoduje odczytanie aktualnej wartości licznika sekwencji. Przy korzystaniu z CURRVAL należy bardzo uważać, gdyż wielokrotne odczytanie aktualnej wartości licznika sekwencji nie musi zawsze zwrócić tą samą wartość (wartość licznika może zostać zmieniona przez innego, pracującego równocześnie użytkownika). Przykład (3) pokazuje, że ze sekwencji można korzystać nie tylko w poleceniu INSERT. Można z nich korzystać również w innych poleceniach języka SQL, w tym np. UPDATE i SELECT. Polecenie UPDATE na przykładzie (3) zmienia wartość klucza zespołu o nazwie „SYSTEMY BAZ DANYCH”.

W DB2 kolejną wartość z sekwencji o nazwie SEQ_ZESP (odpowiednik seq_zesp.NEXTVAL) odczytuje się za pomocą wyrażenia: „NEXTVAL FOR seq_zesp”, a aktualną wartość sekwencji (odpowiednik seq_zesp.CURRVAL) odczytuje się za pomocą wyrażenia: „PREVVAL FOR seq_zesp”. W PostgreSQL odpowiednikiem seq_zesp.NEXTVAL jest: „nextval('seq_zesp')”.

Zadanie (1)

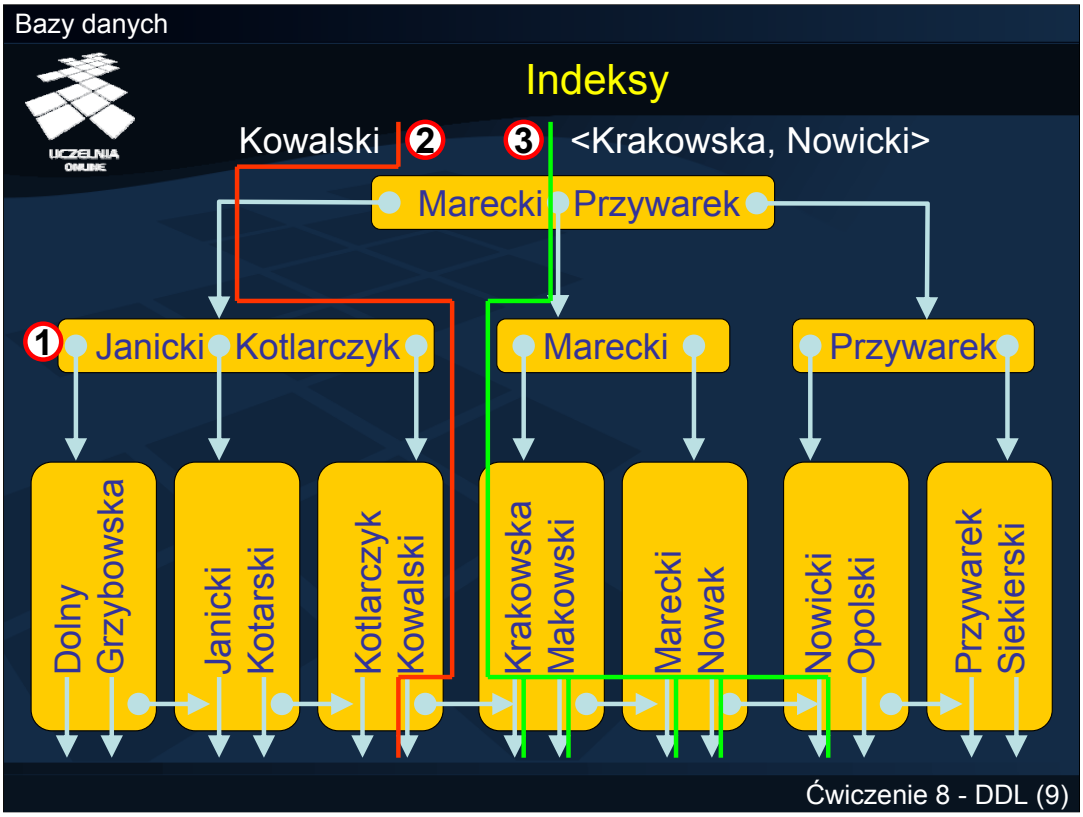
A. Utwórz sekwencję o wartości początkowej równej 70, maksymalnej 99, zwiększającą się w każdym kroku o 1.
B. Korzystając z tej sekwencji wstaw dwa nowe zespoły do relacji ZESPOLY.



Slajd pokazuje rozwiązanie zadania (1), którego treść przytoczono poniżej.

A. Utwórz sekwencję o wartości początkowej równej 70, maksymalnej 99, zwiększającą się w każdym kroku o 1.

B. Korzystając z tej sekwencji wstaw dwa nowe zespoły do relacji ZESPOLY.

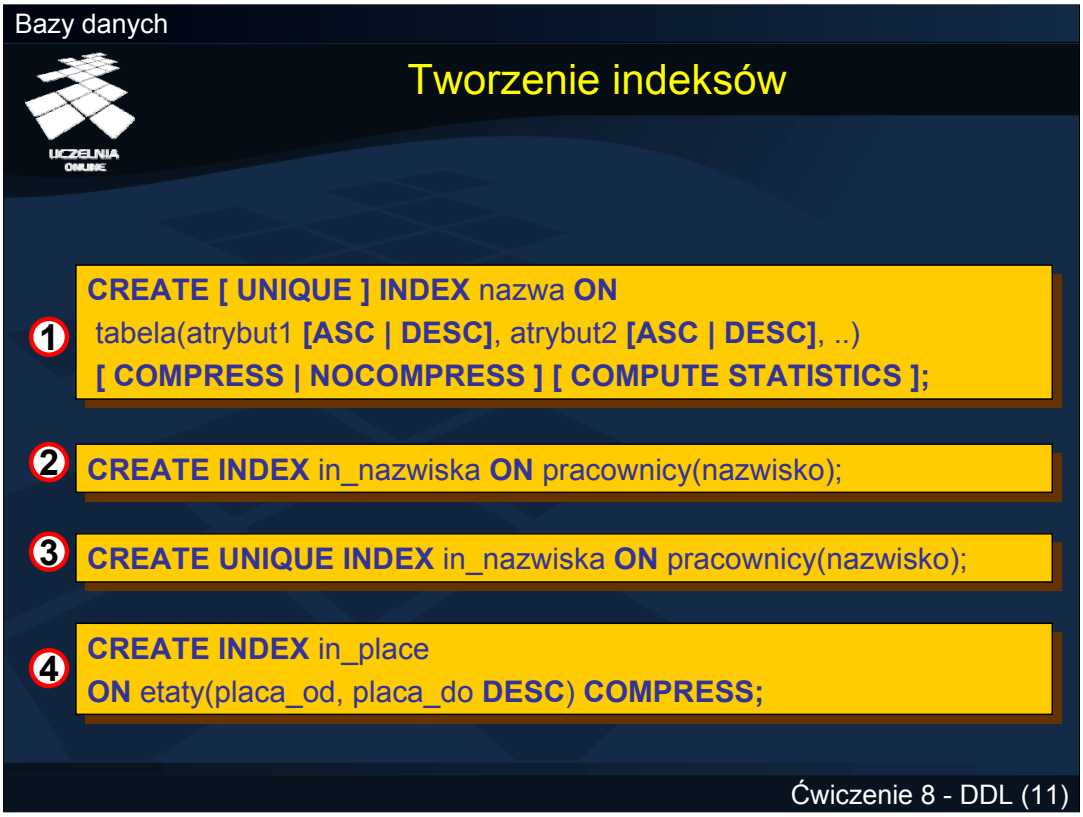


Rozważmy sytuację, w której w bazie danych znajdują się relacje, składujące dane, dla których nie założono żadnych struktur wspomagających przeszukiwanie tych relacji. W takiej sytuacji, realizacja dowolnego zapytania wymaga przynajmniej jednokrotnego przeczytania całej relacji w poszukiwaniu krotek spełniających warunek selekcji. Jak łatwo zauważyć, nawet jednokrotne przeczytanie relacji może zająć dużo czasu, gdyż bazy danych potrafią osiągać olbrzymie rozmiary. W celu przyspieszenia realizacji zapytań i ograniczenia wymaganej liczby odczytów z dysku stosuje się indeksy. W zależności od typów wspieranych zapytań indeksy mogą mieć różną strukturę. Ponieważ omawianie różnych typów indeksów nie wchodzi w zakres tego ćwiczenia omówimy jedynie pokrótce strukturę indeksu typu B+drzewo, który jest jednym z najpopularniejszych indeksów i jest stosowany w prawie każdym SZBD.

Na rysunku przedstawiono strukturę przykładowego indeksu typu B+drzewo, zbudowanego w celu wspierania realizacji zapytań na atrybucie NAZWISKO relacji PRACOWNICY. Pomarańczowe prostokąty na rysunku reprezentują pojedyncze bloki dyskowe, a zawarte w nich napisy, dane składowane w tym blokach. Niebieskie strzałki reprezentują wskaźniki, czyli lokalizacje innego bloku na dysku. Jak łatwo zauważyć, struktura przedstawiona na rysunku jest zrównoważonym drzewem. Każdy węzeł drzewa jest reprezentowany przez jeden blok dyskowy, a w każdym bloku dyskowym (w niniejszym przykładzie) można zapisać maksymalnie dwa nazwiska i trzy wskaźniki. W ogólności zawsze w węźle składowane jest n wartości atrybutu i n+1 wskaźników. Zacznijmy analizę węzłów tworzących liście B+drzewa. W kolejnych węzłach liści składowane są posortowane wartości atrybutu na którym założono indeks. Z każdą wartością atrybutu skojarzony jest wskaźnik, który wskazuje na położenie na dysku krotki, z której wartość ta pochodzi. Dodatkowy wskaźnik w liściu wskazuje na kolejny blok tworzący liść.

Liście B+drzewa tworzą zatem posortowaną listę wartości atrybutu na którym założono indeks. Wskaźniki w węzłach pośrednich składowane są na przemian z wartościami atrybutu. Każdy z tych wskaźników wskazuje na węzeł niższego poziomu. Pierwszy wskaźnik w węźle pośrednim wskazuje na węzeł niższego poziomu, w którym wszystkie wartości są „mniejsze" od pierwszej wartości atrybutu w węźle. Przykładowo, pierwszy wskaźnik w węźle oznaczonym przez (1) wskazuje na węzeł, w którym znajdują się nazwiska „Dolny" i „Grzybowska", które są w porządku alfabetycznym wcześniej niż nazwisko „Janicki", które jest pierwszą wartością w węźle (1). Kolejne wskaźniki w węźle pośrednim B+drzewa wskazują na węzły niższego poziomu, w których znajdują się wartości większe lub równe wartości przed wskaźnikiem i mniejsze od wartości znajdującej się za wskaźnikiem. Przykładowo, drugi wskaźnik w węźle (1) wskazuje na węzeł zawierający wartości większe lub równe wartości „Janicki" i mniejsze od wartości „Kotlarczyk". Ostatni wskaźnik w węźle pośrednim wskazuje na węzeł, w którym znajdują się wartości większe lub równe ostatniej wartości w węźle. Przykładowo, ostatni wskaźnik w węźle (1) wskazuje na węzeł, w którym znajdują się wartości większe lub równe wartości „Kotlarczyk".

Indeks B+drzewo wspiera zapytania punktowe (ang. *unique scan*), np. SELECT * FROM pracownicy WHERE nazwisko='Kowalski', zapytania przedziałowe (ang. r*ange scan*), np. SELECT * FROM pracownicy WHERE nazwisko BETWEEN 'Krakowska' AND 'Nowicki' oraz wspomaga sortowanie danych w relacji według atrybutu na którym założono indeks. Przykładowy sposób realizacji zapytania punktowego pokazuje czerwona linia oznaczona na rysunku przez (2). W zapytaniu tym szukamy krotki, w której wartość atrybutu nazwisko jest równa „Kowalski". Realizację zapytania rozpoczyna się od korzenia. Ponieważ nazwisko „Kowalski" jest alfabetycznie przed nazwiskiem „Marecki" odczytujemy kolejny węzeł wskazywany przez wskaźnik umieszczony przed wartością „Marecki". W kolejnym węźle okazuje się, że nazwisko „Kowalski" znajduje się w porządku alfabetycznym za nazwiskiem „Kotlarczyk", a zatem wędrujemy wzdłuż wskaźnika znajdującego się za wartością „Kotlarczyk". Po dotarciu do liścia jesteśmy w stanie sprawdzić, czy nazwisko „Kowalski" znajduje się w ogóle w relacji, i jeżeli tak, to można w łatwy sposób odczytać wskaźnik na krotkę z tym nazwiskiem. Zapytania przedziałowe realizowane są w podobny sposób. Odszukiwana jest pierwsza wartość przedziału szukanych wartości, a następnie, wykorzystując fakt, iż liście są zorganizowane w postaci listy, odczytywane są kolejne wartości, aż do napotkania wartości stanowiącej koniec wyszukiwanego przedziału. Realizację przykładowego zapytania przedziałowego pokazuje zielona linia, oznaczone na rysunku przez (3). W zapytaniu tym szukamy wszystkich krotek, w których nazwisko jest alfabetycznie większe lub równe nazwisku „Krakowska" i mniejsze lub równe nazwisku „Nowicki". W pierwszym etapie odszukiwane jest w indeksie nazwisko „Krakowska", a następnie, wędrując wzdłuż wskaźników do kolejnych liści, odczytywane są kolejne wartości i skojarzone z nimi wskaźniki do krotek. Napotkanie nazwiska „Nowicki" kończy realizację zapytania. Wspieranie sortowania polega na odczycie kolejnych wartości z listy utworzonej z liści B+drzewa, które są zawsze posortowane.



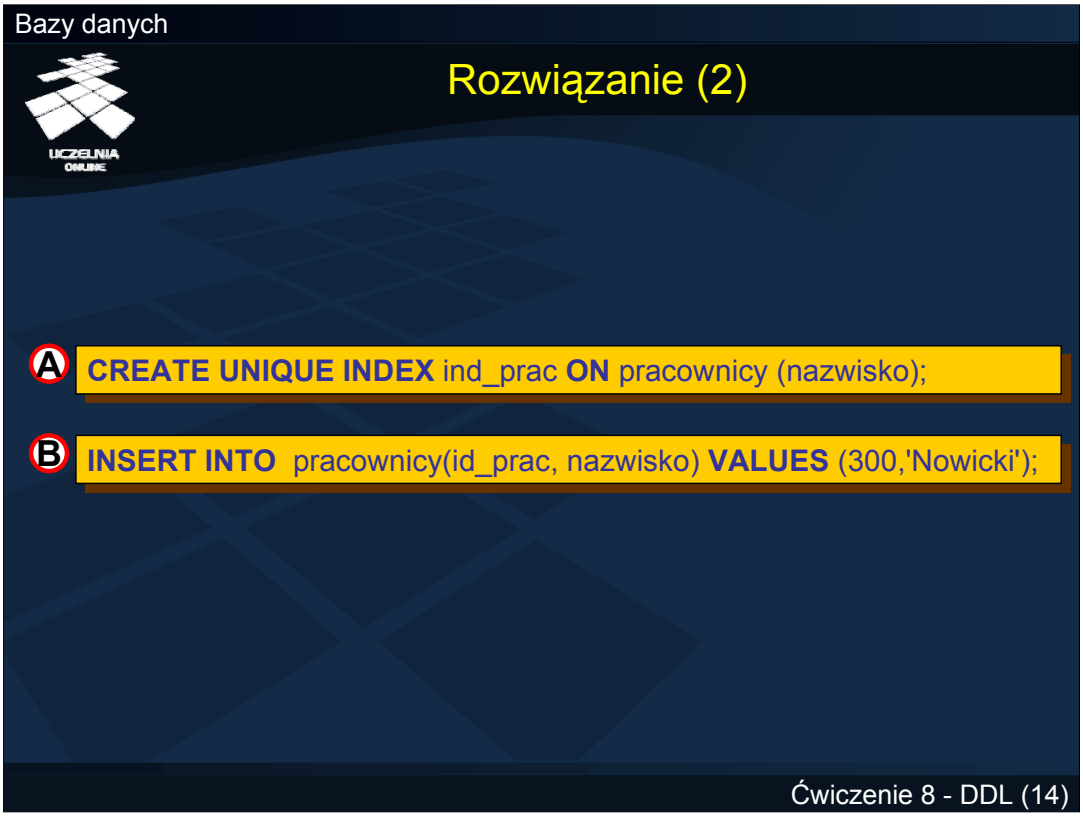
Składnia poleceń tworzących indeksy różni się pomiędzy SZBD. Na slajdzie przedstawiono składnię charakterystyczną dla SZBD Oracle, ale w innych SZBD jest ona bardzo podobna. Przykład (1) pokazuje ogólną składnię polecenia tworzącego indeks na wybranych atrybutach tabeli. W przypadku SZBD Oracle polecenie to jest dużo bardziej rozbudowane, ale omawianie pozostałych opcji jest poza zakresem tego ćwiczenia. Przykład (2) przedstawia polecenie tworzące indeks na atrybucie NAZWISKO relacji PRACOWNICY. Po słowach kluczowych CREATE INDEX podano tutaj nazwę indeksu, a następnie słowo kluczowe ON i nazwę relacji oraz w nawiasie atrybut, na którym zakładany jest indeks. Przykład (3) jest nieco bardziej skomplikowaną wersję przykładu (2). Użyto tutaj dodatkowego słowa kluczowego UNIQUE. Oznacza ono, że wartości atrybutów składowane w indeksie muszą być unikalne. SZBD pilnuje aby ten warunek był spełniony. W praktyce, jest to równoważne założeniu indeksu na atrybucie z ograniczeniem integralnościowym „wartość unikalna”. Przykład (4) pokazuje kilka nowych rzeczy. Po pierwsze, za nazwą relacji w nawiasie wymieniono kilka atrybutów. Tworzone w ten sposób indeksy są zakładane na konkatenacji wartości tych atrybutów i noszą nazwę indeksów skonkatenowanych. Skonkatenowany indeks B+drzewo wspiera zapytania punktowe na kombinacji wartości atrybutów (np. SELECT * FROM pracownicy WHERE placa_od=1000 AND placa_do=2000) oraz niektóre typy zapytań przedziałowych: SELECT * FROM pracownicy WHERE placa_od BETWEEN 1000 AND 2000 albo SELECT * FROM pracownicy WHERE placa_od =1000 AND placa_do BETWEEN 2000 AND 30000. Drugą nowością na przykładzie (4) jest dopisanie słowa kluczowego DESC do jednego z atrybutów na których zakładany jest indeks. Oznacza on porządek sortowania na tym atrybucie. Słowo kluczowe DESC oznacza, że wartości tego atrybutu w indeksie będą sortowane malejąco. Domyślnie wartości te są sortowane rosnąco, co można również zaznaczyć explicite za pomocą słowa kluczowego ASC.

Ostatnią nową rzeczą jest dopisanie na końcu polecenia słowa kluczowego COMPRESS. Dotyczy ono indeksów nieunikalnych (takich w, których jedna wartość indeksowanego atrybutu może wystąpić wielokrotnie) i powoduje, że w indeksie każda wartość jest umieszczona tylko raz, ale z każdą z nich jest pamiętana lista wskaźników na wszystkie krotki zawierające tą wartość. Przykład pierwszy pokazuje jeszcze jedną opcję, która nie została zademonstrowana. Jest nią opcja COMPUTE STATISTICS. Pozwala ona na obliczenie tzw. statystyk, czyli pewnych informacji o rozkładzie i liczności różnych klas danych w relacji oraz o ich rozmieszczeniu na dysku. Statystyki są potrzebne SZBD, aby był on w stanie określić, czy warto jest skorzystać z indeksu, czy też raczej bardziej wydajnym rozwiązaniem by było przejrzeć całą relację.

Indeksy usuwa się za pomocą polecenia DROP INDEX, np. polecenie: DROP INDEX in_nazwiska; usunie indeks o nazwie IN_NAZWISKA.

Zadanie (2)

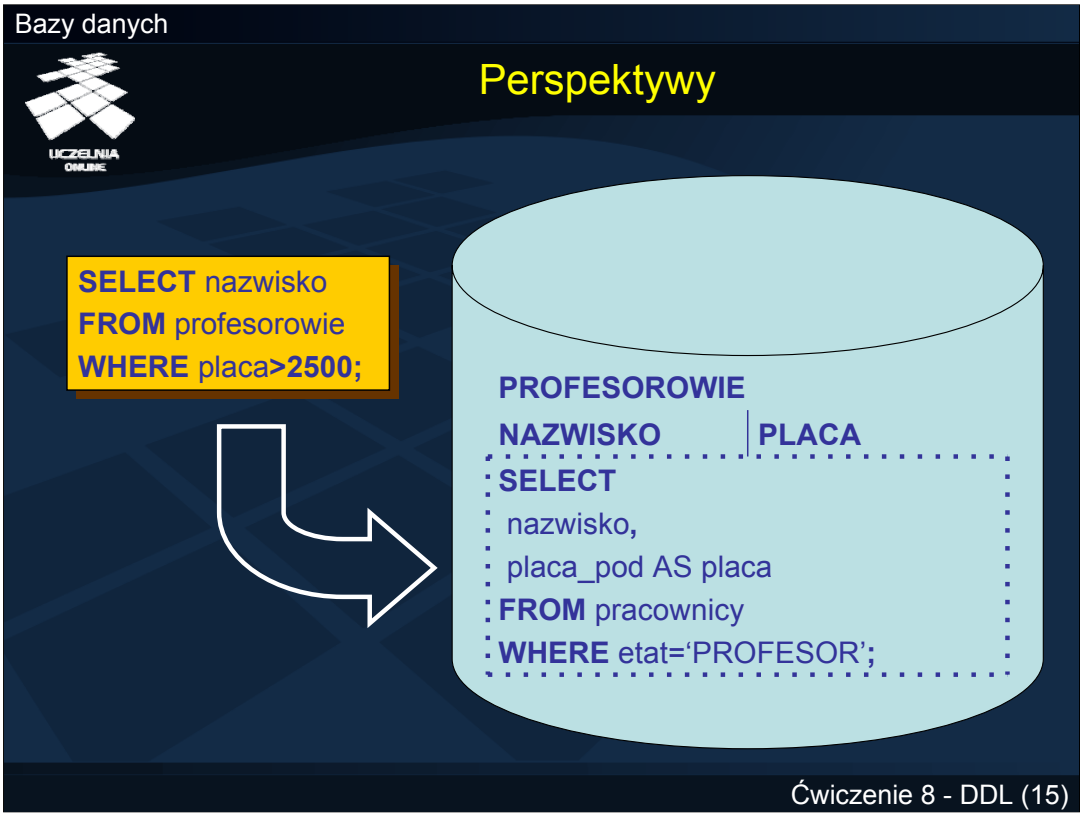
A. Utwórz unikalny indeks na atrybucie nazwisko relacji pracownicy.
B. Spróbuj wstawić krotkę z nazwiskiem, które już znajduje się w relacji. Czy polecenie zakończyło się sukcesem?



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

A. Utwórz unikalny indeks na atrybucie nazwisko relacji pracownicy.

B. Spróbuj wstawić krotkę z nazwiskiem, które już znajduje się w relacji. Czy polecenie zakończyło się sukcesem?



Aby przedstawić ideę perspektyw rozważmy następujące zapytanie:

```
SELECT
 nazwisko,
 placa_pod AS placa
FROM pracownicy
WHERE etat='PROFESOR';
```

Wynikiem tego zapytania jest relacja, w której znajdują się dwa atrybuty: NAZWISKO i PLACA przechowujące dane dotyczące wszystkich pracowników pracujących na etacie PROFESOR. Ponieważ zapytanie zwraca relację, a polecenie SELECT służy do odczytywania danych z relacji, to teoretycznie istnieje możliwość przetwarzania za pomocą polecenia SELECT wyników innego polecenia SELECT. Jest to podstawowy pomysł stojący za koncepcją perspektyw. Perspektywą nazywamy obiekt bazy danych, który jest nazwanym zapytaniem, składowanym w bazie danych, „naśladującym" relację. Na rysunku na slajdzie przedstawiono schematycznie działanie perspektyw. W bazie danych znajduje się „relacja" PROFESOROWIE, którą można przeszukiwać za pomocą polecenia SELECT. W rzeczywistości nie jest to jednak relacja, a perspektywa, która zawiera wyniki zwracane przez inne zapytanie. Wyniki zapytania tworzącego perspektywę nie są fizycznie składowane na dysku, ale są generowane za każdym razem, gdy do perspektywy zostanie skierowane zapytanie.

W rezultacie, zamiast zapytania:

```
SELECT nazwisko
FROM profesorowie
WHERE placa>2500;
```

wykonywane jest zapytanie:

```
SELECT nazwisko
FROM (
 SELECT
  nazwisko,
  placa_pod AS placa
 FROM pracownicy
 WHERE etat='PROFESOR';
)
WHERE placa>2500;
```

Jak łatwo zauważyć, jest to zapytanie z podzapytaniem w klauzuli FROM.

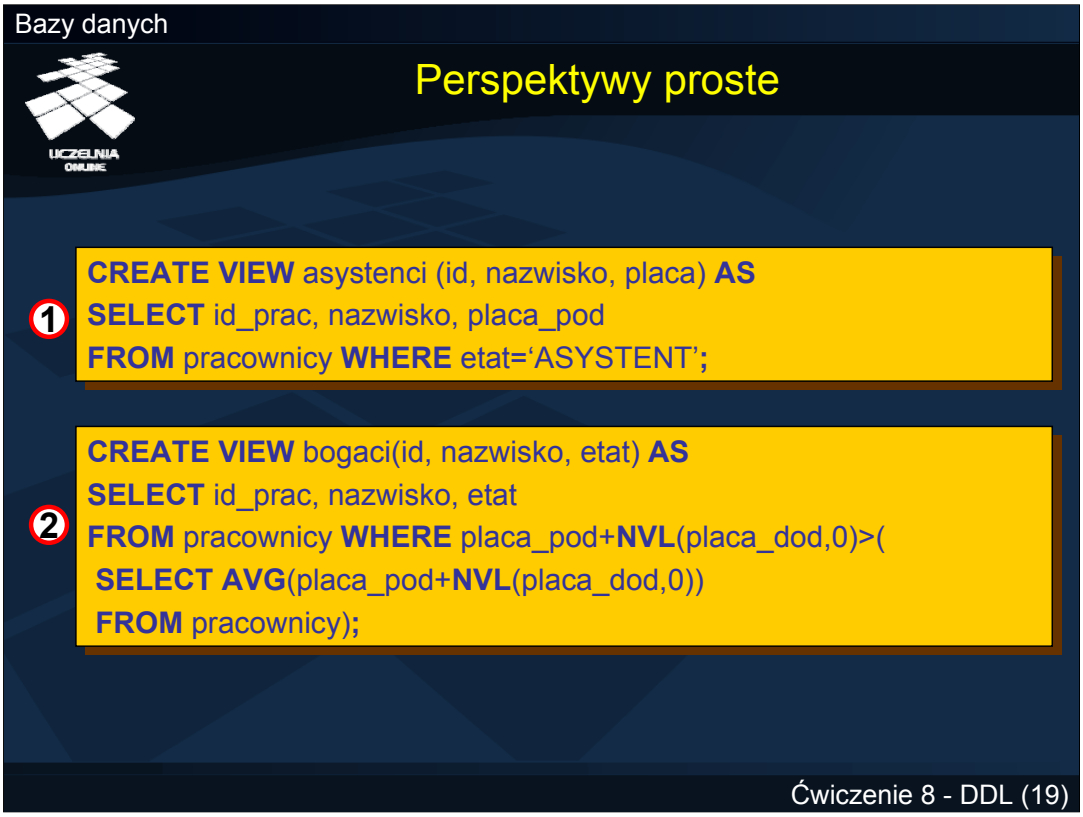
Poprzez perspektywy możliwe jest nie tylko wykonywanie zapytań, ale również danych w nich „składowanych" (modyfikacji ulegają dane w relacjach na których zdefiniowane są perspektywy). Niestety nie zawsze jest to możliwe. W ramach tych ćwiczeń zostaną omówione polecenia pozwalające na tworzenie perspektyw, oraz omówione zostaną warunki modyfikowalności danych w perspektywach.

Perspektywę można utworzyć za pomocą polecenia CREATE VIEW. Ogólną składnię polecenia tworzącego perspektywy przedstawiono na przykładzie (1). Polecenie rozpoczyna się od słów kluczowych CREATE VIEW, po którym podaje się nazwę tworzonej perspektywy, następnie (opcjonalnie) listę nazw atrybutów tworzonej perspektywy, słowo kluczowe AS, zapytanie definiujące perspektywę i opcjonalnie słowa kluczowe WITH CHECK OPTION. Należy tutaj zaznaczyć, że zapytanie definiujące perspektywę może się odwoływać do innych perspektyw utworzonych wcześniej.

Przykład (2) pokazuje polecenie tworzące perspektywę PROFESOROWIE, o atrybutach NAZWISKO i PLACA. W atrybucie NAZWISKO znajdują się nazwiska pracowników zatrudnionych na etacie PROFESOR, a w atrybucie PLACA znajduje się łączna, podstawowa i dodatkowa, płaca pracownika. W sytuacji, kiedy nie poda się listy nazw atrybutów perspektywy, nazwy te są generowane automatycznie na podstawie wyrażeń w klauzuli SELECT definiującej perspektywę. Jeżeli w klauzuli SELECT znajdują się wyrażenia, najlepiej jest nadać im aliasy. Wówczas atrybuty perspektywy będą miały nazwy identyczne z aliasami wyrażeń w klauzuli SELECT.

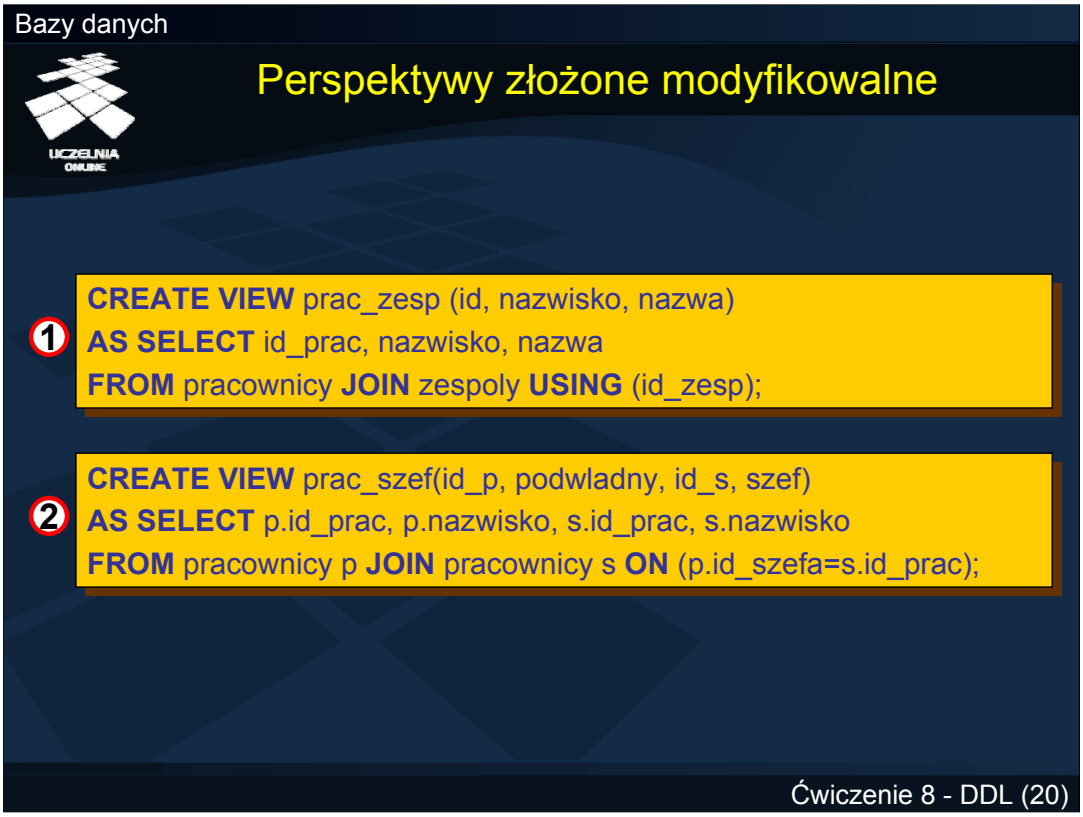
Przykład (3) pokazuje ogólną składnię polecenia usuwającego perspektywę. Polecenie rozpoczyna się od słów kluczowych DROP VIEW, po którym podaje się nazwę perspektywy do usunięcia i opcjonalnie słowa kluczowe RESTRICT albo CASCADE. Słowo kluczowe RESTRICT oznacza, że jeżeli istnieją inne perspektywy, korzystające z perspektywy, którą chcemy usunąć, to polecenie ma zakończyć się błędem. Takie zachowanie jest domyślne, a zatem użycie słowa RESTRICT jedynie poprawia czytelność poleceń. Słowo kluczowe CASCADE oznacza, że wszystkie perspektywy korzystające z usuwanej perspektywy są również usuwane. W SZBD Oracle 10g nie zaimplementowano jeszcze przewidzianej w standardzie funkcjonalności związanej z opcjami RESTRICT i CASCADE.

Próba dodania tych słów do polecenia DROP VIEW kończy się błędem informującym o błędnej składni polecenia. Użycie polecenia DROP VIEW zakończy się zawsze sukcesem, niezależnie, czy istnieją inne perspektywy korzystające z usuwanej perspektywy, czy nie. Perspektywy korzystające z usuwanej perspektywy nie są jednak usuwane, ale zaznaczane jako niepoprawne i nie jest możliwe korzystanie z nich. Przykład (4) pokazuje polecenie usuwające perspektywę PROFESOROWIE.



Wyróżnia się dwa rodzaje perspektyw: „perspektywy proste" i „perspektywy złożone". Perspektywy proste, to perspektywy, które są oparte na jednej relacji (tzw. relacji bazowej, brak połączeń) oraz nie zawierają: operatorów zbiorowych, operatora DISTINCT, funkcji grupowych, grupowania, sortowania i podzapytań w klauzuli SELECT. Perspektywy proste są modyfikowalne przez polecenia UPDATE i DELETE. Aby można było korzystać z poleceń INSERT, konieczne jest dodatkowo, aby perspektywy udostępniały wszystkie atrybuty klucza podstawowego, oraz wszystkie atrybuty obowiązkowe relacji bazowej. Jeżeli perspektywa zawiera atrybuty stanowiące wynik wyrażeń, to polecenia INSERT i UPDATE nie mogą dotyczyć tych atrybutów.

Przykład (1) zawiera polecenie tworzące perspektywę ASYSTENCI udostępniającą dane o pracownikach pracujących na etacie ASYSTENT. Ponieważ perspektywa nie korzysta z grupowania, sortowania, podzapytań w klauzuli SELECT i operatora DISTINCT jest to perspektywa prosta a zatem może być modyfikowana za pomocą poleceń UPDATE i DELETE. Perspektywa ta udostępnia również atrybut ID_PRAC, który jest kluczem podstawowym relacji PRACOWNICY, a zatem możliwe jest również wstawianie krotek poprzez tą relację. Przykład (2) pokazuje nieco bardziej skomplikowaną perspektywę. Jest to również perspektywa modyfikowalna, gdyż spełnia warunki perspektywy prostej (grupowanie jest wykonywane w podzapytaniu) i udostępnia klucz podstawowy.

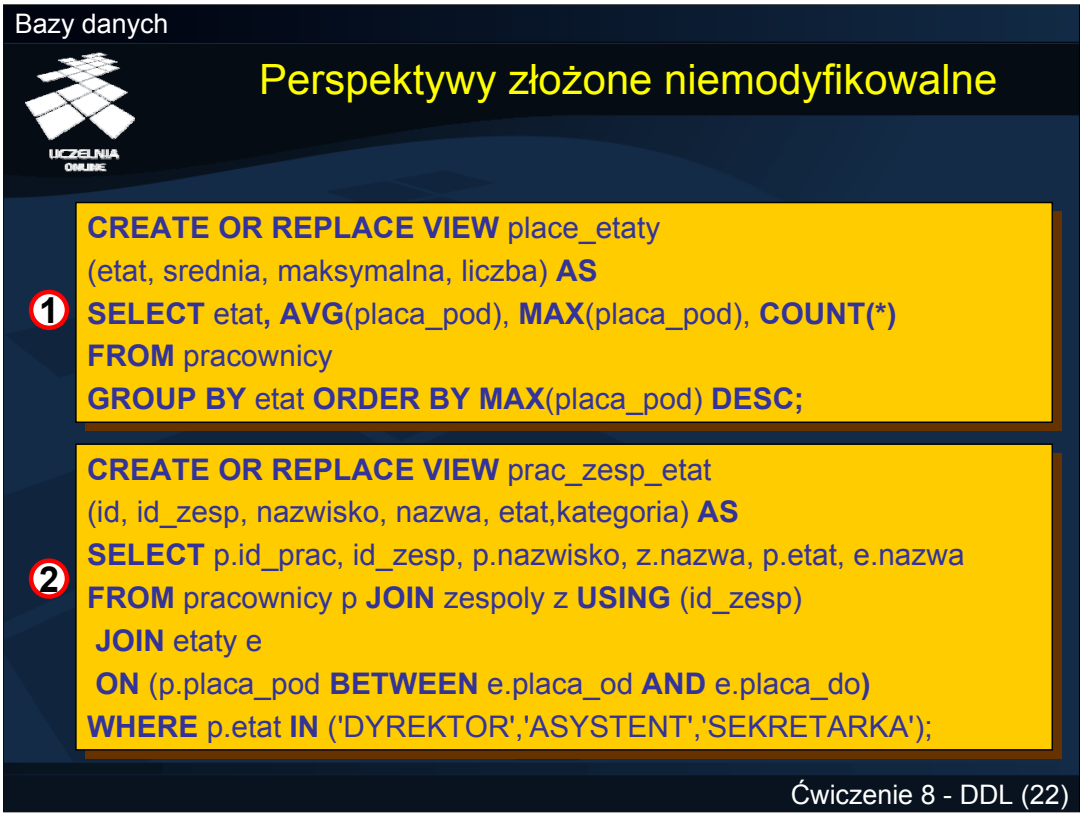


Perspektywami złożonymi nazywamy wszystkie perspektywy, które nie spełniają warunków perspektywy prostej. Niektóre z perspektyw złożonych są również modyfikowalne. Perspektywa złożona jest modyfikowalna jeśli nie zawiera: operatorów zbiorowych, operatora DISTINCT, funkcji grupowej, klauzul GROUP BY i ORDER BY. Połączenia mogą występować w tych perspektywach, jednak operacje modyfikujące dane poprzez perspektywę muszą dotyczyć jedynie atrybutów pochodzących z „relacji zachowującej klucz". Relacja zachowuje klucz, jeżeli każda unikalna wartość z tej relacji w perspektywie pozostaje unikalna. Aby móc korzystać z polecenia INSERT, perspektywa musi prezentować wszystkie atrybuty klucza podstawowego i wszystkie atrybuty wymagane relacji zachowującej klucz. Utworzona za pomocą polecenia INSERT krotka jest wstawiana jedynie do relacji zachowującej klucz. Polecenie UPDATE może dotyczyć tylko atrybutów pochodzących z relacji zachowującej klucz i nie stanowiących wyników wyrażeń. Polecenie DELETE może usunąć jedynie krotki pochodzące z relacji zachowującej klucz.

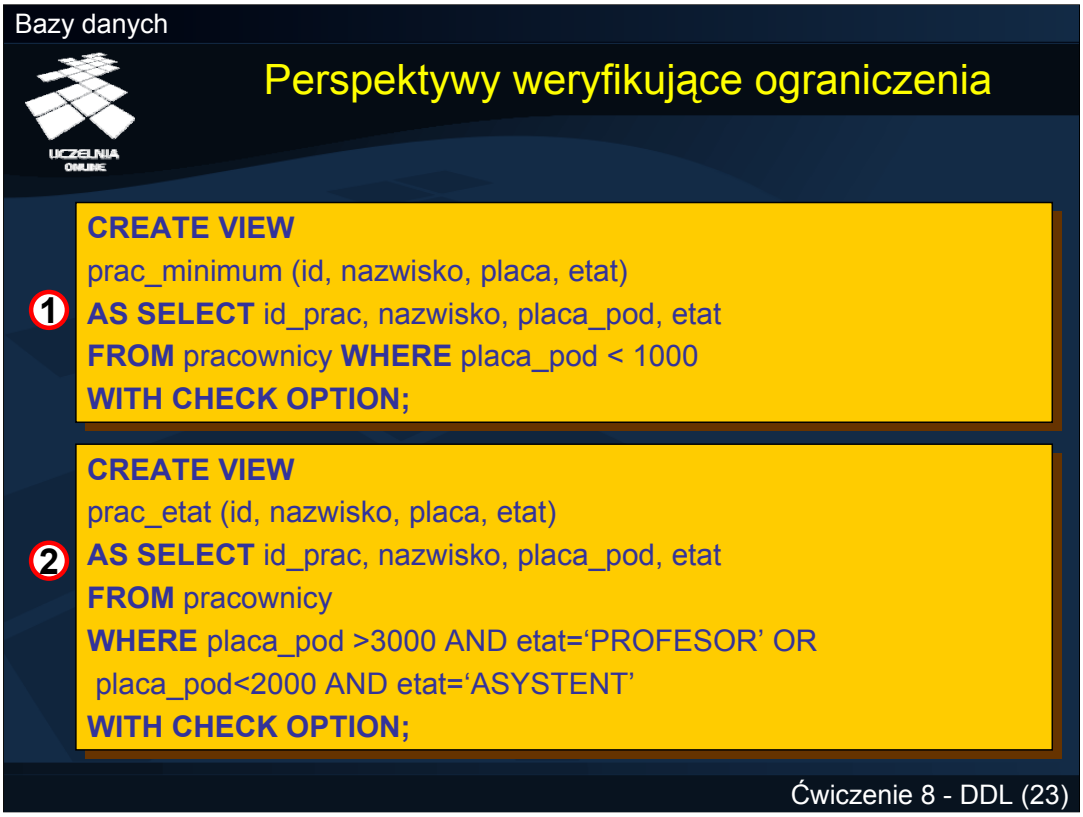
Przykład (1) pokazuje polecenie tworzące perspektywę PRAC_ZESP. Perspektywa ta nie zawiera operatorów zbiorowych, operatora DISTINCT, funkcji grupowej, klauzul GROUP BY i ORDER BY. Zawiera ona jednak połączenie. Zastanówmy się zatem, czy jedna z perspektyw bazowych zachowuje klucz. W perspektywie zastosowano połączenie naturalne z warunkiem połączeniowym zdefiniowanym na atrybucie ID_ZESP. Ponieważ ID_ZESP jest kluczem podstawowym relacji ZESPOLY, to wartości przechowywane w tym atrybucie są unikalne. W relacji PRACOWNICY atrybut ID_ZESP jest kluczem obcym i tutaj wartości atrybutu nie są unikalne. W związku z tym, do każdej krotki reprezentującej pracownika pasuje co najwyżej jedna krotka z relacji zespoły, a zatem żadna krotka z relacji pracownicy nie zostanie zwielokrotniona w wyniku połączenia, z czego wynika, że relacja pracownicy zachowuje klucz.

Ponieważ perspektywa udostępnia atrybut ID_PRAC tworzący klucz podstawowy relacji PRACOWNICY, jest to perspektywa modyfikowalna za pomocą wszystkich trzech poleceń: INSERT, UPDATE i DELETE.

Przykład (2) pokazuje polecenie tworzące perspektywę PRAC_SZEF, która prezentuje identyfikatory i nazwiska pracowników, i ich szefów. Perspektywa ta nie zawiera operatorów zbiorowych, operatora DISTINCT, funkcji grupowej, klauzul GROUP BY i ORDER BY. Zawiera ona jednak połączenie zwrotne. W połączeniu zwrotnym wykorzystywany jest równościowy warunek połączeniowy wykorzystujący atrybuty ID_PRAC i ID_SZEFA. Atrybut ID_PRAC jest kluczem podstawowym relacji pracownicy, a atrybut ID_SZEFA jest kluczem obcym wskazującym na relację PRACOWNICY. Poprzez analogię do przykładu (1) można łatwo zauważyć, że klucz zachowany jest w relacji PRACOWNICY reprezentującej podwładnych, z czego wynika, że poprzez perspektywę PRAC_SZEF można modyfikować atrybuty dotyczące podwładnych. Ponieważ perspektywa udostępnia atrybut tworzący klucz podstawowy relacji PRACOWNICY możliwe jest używanie polecenia INSERT to wstawiania nowych krotek reprezentujących podwładnych.



Perspektywy, które nie spełniają warunków omówionych na dwóch poprzednich slajdach, są perspektywami złożonymi, niemodyfikowalnymi. Na slajdzie przedstawiono dwie przykładowe perspektywy, które są niemodyfikowalne. Przykład (1) pokazuje perspektywę, która przedstawia średnią i maksymalną płacę pracowników na danym etacie, oraz ich liczbę, całość posortowaną według najwyższej płacy w etacie. Ponieważ w perspektywie wykorzystano grupowanie, funkcje grupowe i sortowanie, jest to perspektywa niemodyfikowalna. Przykład drugi pokazuje perspektywę, która nie zawiera operatorów zbiorowych, operatora DISTINCT, funkcji grupowej oraz klauzul GROUP BY i ORDER BY. Rozważmy zatem połączenia wykonywane w zapytaniu tej perspektywy i zastanówmy się, czy któraś z relacji zachowuje klucz. Pierwszym połączeniem jest połączenie naturalne relacji PRACOWNICY z relacją ZESPOLY z warunkiem równościowym zdefiniowanym na atrybucie ID_ZESP. Identyczne połączenie rozważane było już na poprzednim slajdzie (przykład (1)). Ustalono wówczas, że w wyniku tego połączenia klucza nie zachowuje relacja ZESPOLY. Tutaj jest tak samo. Kolejnym połączeniem jest połączenie wyniku poprzedniego połączenia z relacją etaty. Jest to połączenie nierównościowe, w którym placa podstawowa pracownika musi się mieścić w widełkach płacowych skojarzonych z konkretnym etatem. Ponieważ wielu pracowników może mieć identyczną płacę, a dodatkowo jedna płaca może pasować do więcej niż jednej kategorii, to krotki z obu łączonych relacji zostaną zwielokrotnione. W rezultacie żadna z relacji w tej perspektywie nie zachowuje klucza i perspektywa jest niemodyfikowalna.



Rozważmy następującą perspektywę:

CREATE VIEW

prac_minimum (id, nazwisko, placa, etat)

AS SELECT id_prac, nazwisko, placa_pod, etat

FROM pracownicy WHERE placa_pod < 1000;

Perspektywa udostępnia wszystkich pracowników zarabiających poniżej 1000 złotych. Ponieważ jest to perspektywa prosta, która udostępnia klucz podstawowy relacji bazowej, jest to perspektywa modyfikowalna. Zastanówmy się co się stanie, jeżeli zostanie wykonane następujące polecenie:

INSERT INTO prac_minimum (id, nazwisko, placa, etat) VALUES (300,'Zielinski',2000,'PROFESOR');

Ponieważ perspektywa PRAC_MINIMUM jest modyfikowalna wstawienie krotki uda się, jednak późniejsze wykonanie zapytania do tej perspektywy wykaże, że nowa krotka się w niej nie pojawiła. Nie ma w tym nic dziwnego, ponieważ krotka ta nie spełnia warunku selekcji zapytania definiującego perspektywę. Podobny efekt można uzyskać modyfikując płacę pracowników widocznych poprzez perspektywę. Jeżeli podniesiemy pracownikom płacę powyżej 1000 zł to ci pracownicy znikną z perspektywy. Oczywiście, ich krotki nie są naprawdę usuwane z bazy danych, ale nie spełniają już warunku selekcji zapytania w perspektywie. Aby zapobiec takim sytuacjom, możliwe jest nakazanie SZBD, aby nie pozwalał na wykonywanie operacji INSERT i UPDATE, które spowodują, że jakaś krotka znika z perspektywy. Można to zrobić dopisując słowa kluczowe WITH CHECK OPTION na końcu definicji perspektywy tak, jak pokazano to na przykładzie (1).

Próba wstawienia krotki do perspektywy tworzonej przez polecenie na przykładzie (1), która nie spełnia warunku placa_pod<1000 zakończy się błędem. Na przykładzie (2) pokazano polecenie tworzące perspektywę, która udostępnia dane dotyczące pracowników, którzy zarabiają powyżej 3000 i są profesorami, oraz pracowników, którzy zarabiają poniżej 2000 i są asystentami. Próba wstawienia do perspektywy pracownika, który nie spełnia tego ograniczenia zakończy się błędem.

Na ćwiczeniu omawiającym podzapytania, dowiedzieliście się, że można umieszczać podzapytania w klauzuli FROM. Po tym ćwiczeniu wiecie już, że takie podzapytania bardzo przypominają perspektywy. Dlatego też, podzapytania w klauzuli FROM nazywa się perspektywami wbudowanymi. Przykłady (1) i (2) na slajdzie przedstawiają polecenie tworzące przykładową perspektywę z danymi o profesorach i polecenie wykonujące zapytanie do tej perspektywy. Na przykładzie (3) pokazano zapytanie (2), które wykorzystuje perspektywę wbudowaną analogiczną do perspektywy PROFESOROWIE.

Zadanie (3)

- Stwórz perspektywę o nazwie ZESPOLY_SKLAD, która przedstawia nazwy wszystkich zespołów, a dla każdego zespołu liczbę zatrudnionych w nim pracowników. Zespoły, które nie posiadają pracowników również mają zostać uwzględnione w perspektywie.

Slajd pokazuje rozwiązanie zadania (3), którego treść przytoczono poniżej.

Stwórz perspektywę o nazwie ZESPOLY_SKLAD, która przedstawia nazwy wszystkich zespołów, a dla każdego zespołu liczbę zatrudnionych w nim pracowników. Zespoły, które nie posiadają pracowników również mają zostać uwzględnione w perspektywie.

Zadania

4. Utwórz sekwencję MYSEQ rozpoczynającą się od 300 i zwiększającą się w każdym kroku o 10.
5. Wykorzystaj utworzoną sekwencję do wstawienia nowego asystenta o nazwisku Trąbczyński do relacji PRACOWNICY.
6. Zmodyfikuj pracownikowi Trąbczyńskiemu płacę dodatkową na wartość wskazywaną przez sekwencję MYSEQ.
7. Stwórz nową sekwencję o niskiej wartości maksymalnej. Zaobserwuj, co się dzieje, gdy następuje „przepełnienie” sekwencji.

Zadania

8. Zdefiniuj perspektywę ASYSTENCI, udostępniającą identyfikator, nazwisko, płacę i liczbę przepracowanych lat (do obliczenia liczby przepracowanych lat użyj funkcji MONTHS_BETWEEN i SYSDATE) wszystkich asystentów.
9. Zdefiniuj perspektywę PLACE udostępniającą następujące dane: numer zespołu, średnią, minimalną i maksymalną płacę w zespole (miesięczna płaca wraz z dodatkami), fundusz płac (suma pieniędzy wypłacanych miesięcznie pracownikom) oraz liczbę wypłacanych pensji i dodatków.

Zadania

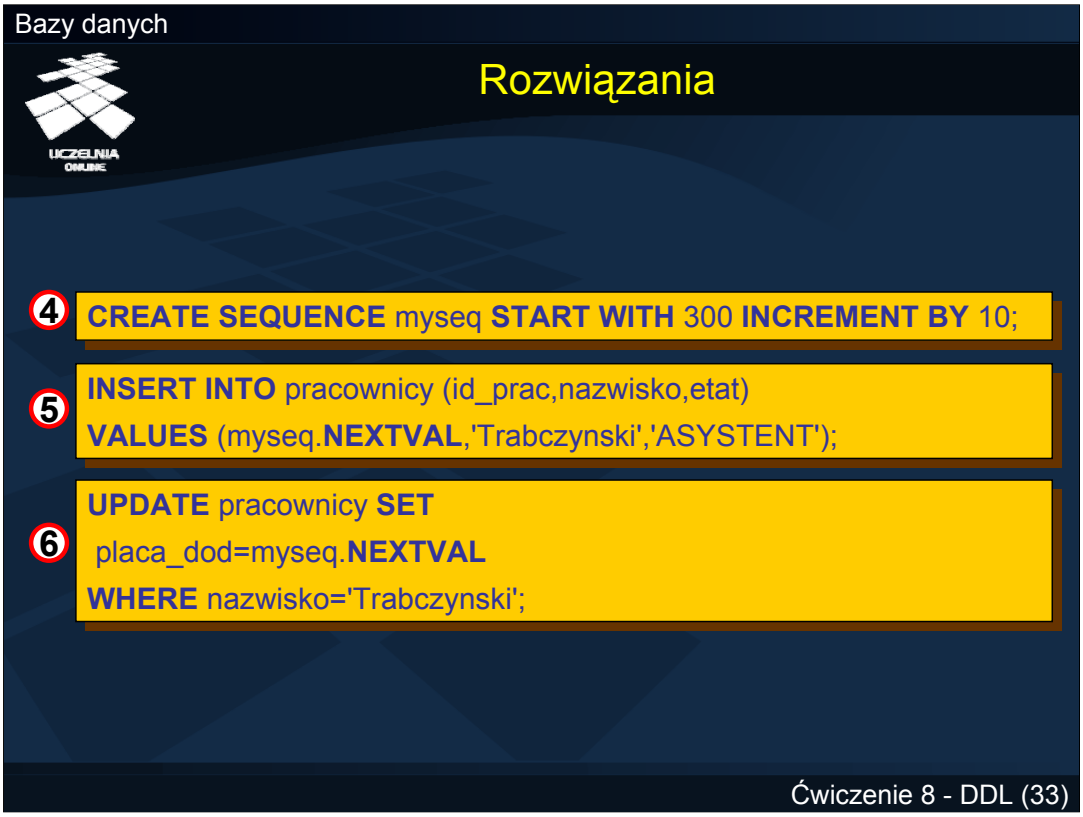
10. Korzystając z perspektywy PLACE wyświetl nazwiska i płace tych pracowników, którzy zarabiają mniej niż średnia w ich zespole.
11. Zdefiniuj perspektywę PLACE_MINIMALNE wyświetlającą pracowników zarabiających poniżej 1500 złotych. Perspektywa musi zapewniać weryfikację danych, w taki sposób, aby za jej pomocą nie można było podnieść pensji pracownika powyżej pułapu 1500 złotych.

Zadania

12. Spróbuj za pomocą perspektywy PLACE_MINIMALNE zwiększyć pensję pracownika o nazwisku Kotlarczyk do 1700 złotych. Co zauważyłeś/łaś?
13. Stwórz perspektywę PRAC_SZEF prezentującą informacje o pracownikach (identyfikator, nazwisko i etat) oraz ich przełożonych (identyfikator i nazwisko). Zwróć uwagę na to, aby można było przez perspektywę PRAC_SZEF wstawiać nowych pracowników oraz modyfikować i usuwać istniejących pracowników. Sprawdź, czy rzeczywiście jest to możliwe.

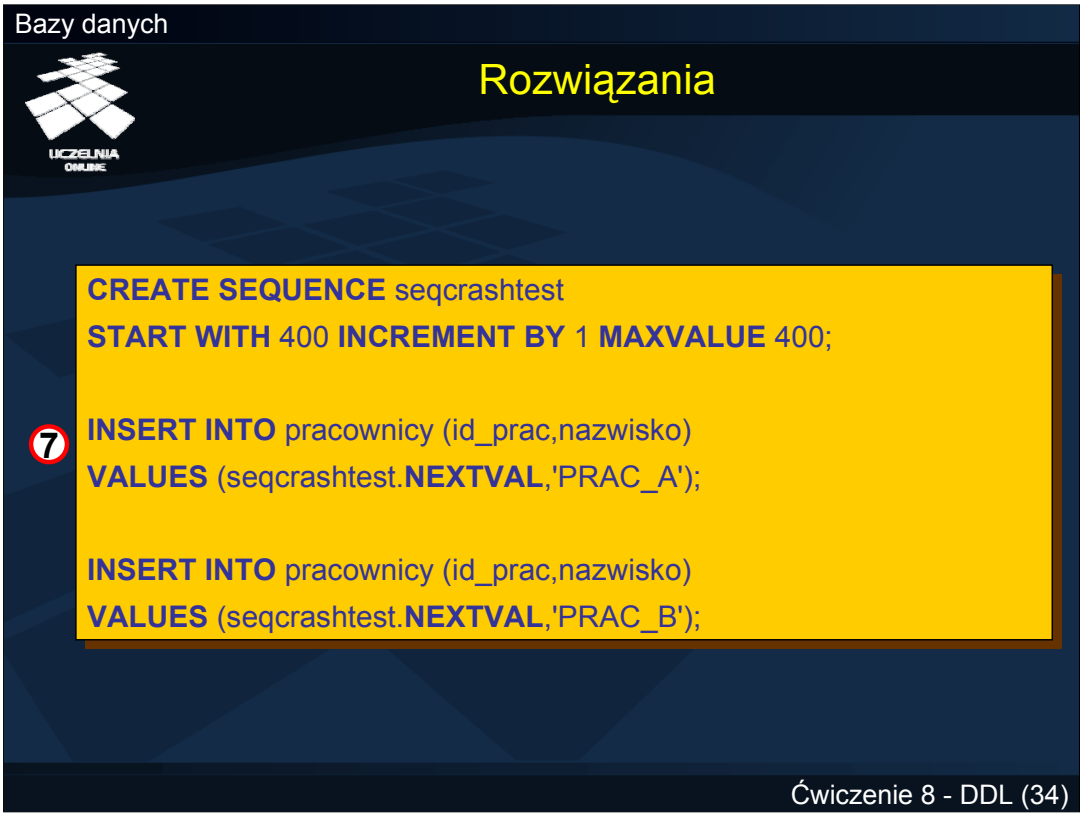
Zadania

14. Stwórz perspektywę ZAROBKI wyświetlającą identyfikator, nazwisko, etat i płacę podstawową pracownika. Perspektywa musi zapewniać kontrolę pensji pracownika (pensja pracownika nie może być wyższa niż pensja jego szefa).



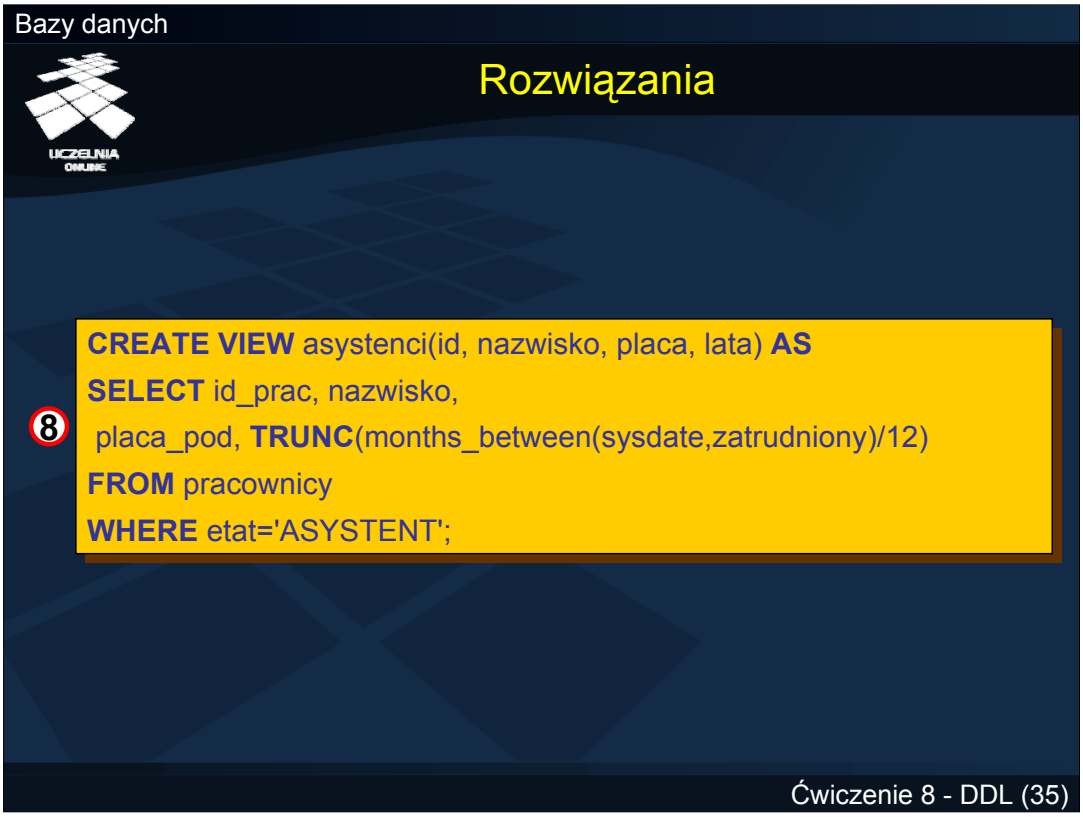
Slajd pokazuje rozwiązanie zadań (4), (5) i (6) których treść przytoczono poniżej.

- (4) Utwórz sekwencję MYSEQ rozpoczynającą się od 300 i zwiększającą się w każdym kroku o 10.
- (5) Wykorzystaj utworzoną sekwencję do wstawienia nowego asystenta o nazwisku Trąbczyński do relacji PRACOWNICY.
- (6). Zmodyfikuj pracownikowi Trąbczyńskiemu płacę dodatkową na wartość wskazywaną przez sekwencję MYSEQ.



Slajd pokazuje rozwiązanie zadania (7), którego treść przytoczono poniżej.

Stwórz nową sekwencję o niskiej wartości maksymalnej. Zaobserwuj, co się dzieje, gdy następuje „przepełnienie” sekwencji.



Slajd pokazuje rozwiązanie zadania (8), którego treść przytoczono poniżej.

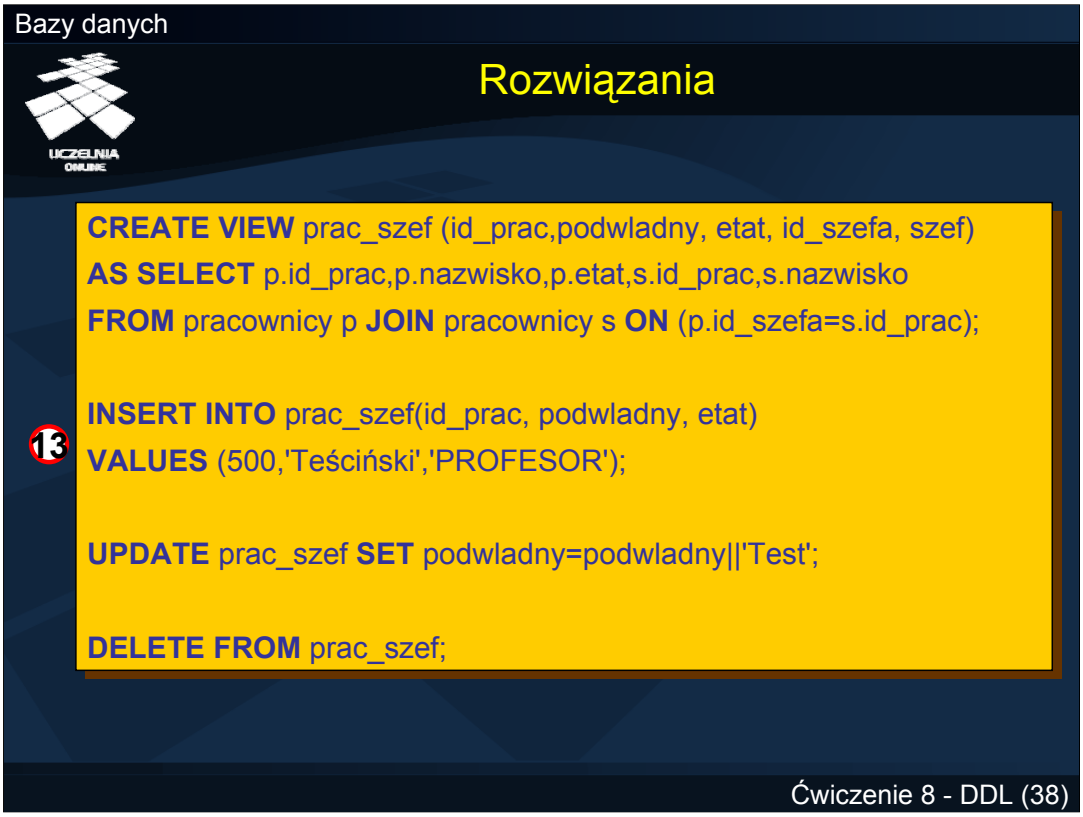
Zdefiniuj perspektywę ASYSTENCI, udostępniającą identyfikator, nazwisko, płacę i liczbę przepracowanych lat (do obliczenia liczby przepracowanych lat użyj funkcji MONTHS_BETWEEN i SYSDATE) wszystkich asystentów.

Slajd pokazuje rozwiązanie zadania (9), którego treść przytoczono poniżej.

Zdefiniuj perspektywę PLACE udostępniającą następujące dane: numer zespołu, średnią, minimalną i maksymalną płacę w zespole (miesięczna płaca wraz z dodatkami), fundusz płac (suma pieniędzy wypłacanych miesięcznie pracownikom) oraz liczbę wypłacanych pensji i dodatków.

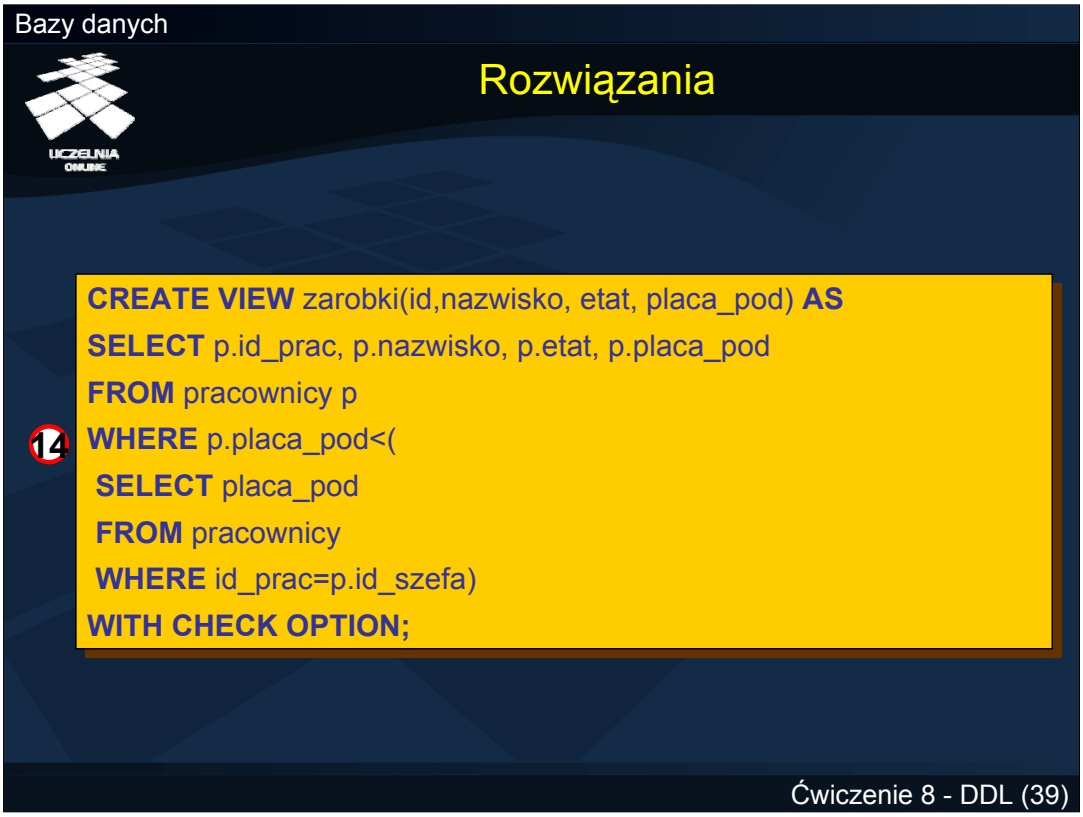
Slajd pokazuje rozwiązanie zadań (10), (11) i (12), których treść przytoczono poniżej.

- (10) Korzystając z perspektywy PLACE wyświetl nazwiska i płace tych pracowników, którzy zarabiają mniej niż średnia w ich zespole.
- (11) Zdefiniuj perspektywę PLACE_MINIMALNE wyświetlającą pracowników zarabiających poniżej 1500 złotych. Perspektywa musi zapewniać weryfikację danych, w taki sposób, aby za jej pomocą nie można było podnieść pensji pracownika powyżej pułapu 1500 złotych.
- (12) Spróbuj za pomocą perspektywy PLACE_MINIMALNE zwiększyć pensję pracownika o nazwisku Kotlarczyk do 1700 złotych. Co zauważyłeś/łaś?



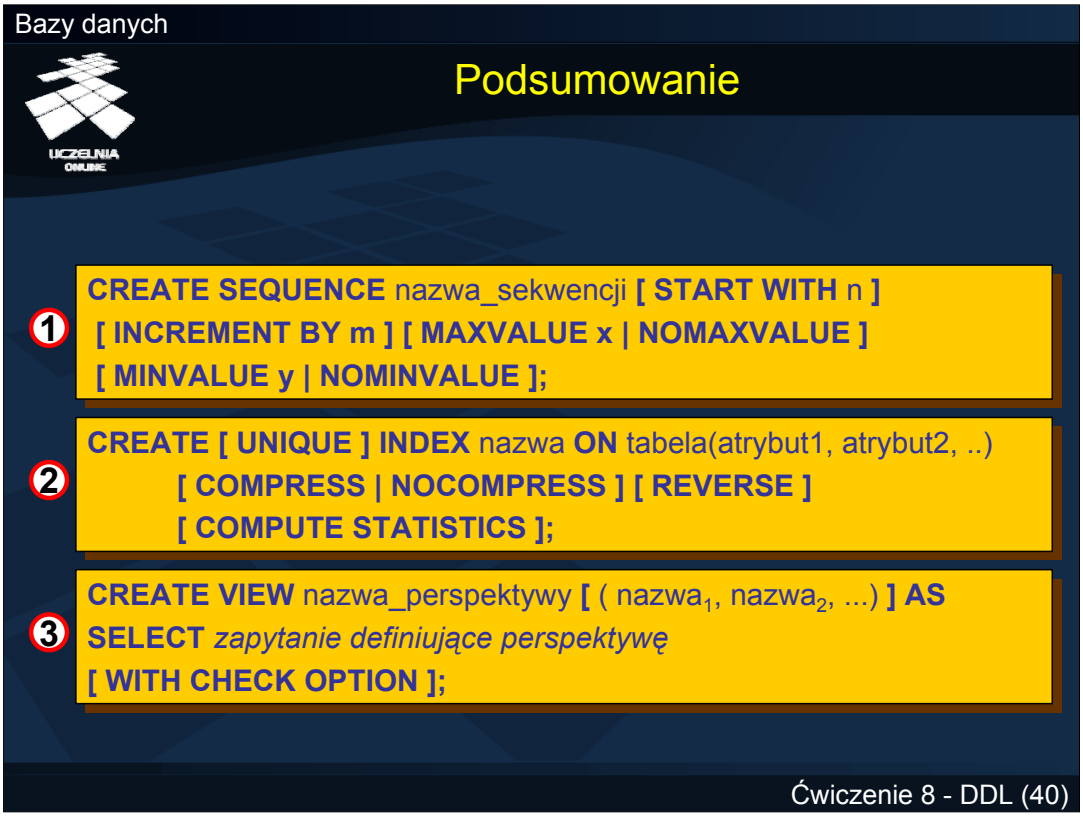
Slajd pokazuje rozwiązanie zadania (13), którego treść przytoczono poniżej.

Stwórz perspektywę PRAC_SZEF prezentującą informacje o pracownikach (identyfikator, nazwisko i etat) oraz ich przełożonych (identyfikator i nazwisko). Zwróć uwagę na to, aby można było przez perspektywę PRAC_SZEF wstawiać nowych pracowników oraz modyfikować i usuwać istniejących pracowników. Sprawdź, czy rzeczywiście jest to możliwe.



Slajd pokazuje rozwiązanie zadania (14), którego treść przytoczono poniżej.

Stwórz perspektywę ZAROBKI wyświetlającą identyfikator, nazwisko, etat i płacę podstawową pracownika. Perspektywa musi zapewniać kontrolę pensji pracownika (pensja pracownika nie może być wyższa niż pensja jego szefa).



Na ćwiczeniu poznaliście Państwo sekwencje pozwalające na wygodne i automatyczne generowanie wartości kluczy, dowiedzieliście się co to są indeksy i jak się je tworzy, oraz w jaki sposób przyspieszają one realizację zapytań. Poznaliście również perspektywy. Dowiedzieliście się do czego służą, jak się je tworzy i usuwa, oraz jak można je wykorzystać do wstawiania, usuwania i modyfikacji danych w relacjach. Każdy z wyżej wymienionych tematów utrwaliliście państwo za pomocą zadań do samodzielnego wykonania.