SQL ZAJĘCIA PRAKTYCZNE

Joanna Gościk

Politechnika Białostocka

SQL – ZAJĘCIA PRAKTYCZNE

Joanna Gościk

Politechnika Białostocka
OFICYNA WYDAWNICZA POLITECHNIKI BIAŁOSTOCKIEJ
BIAŁYSTOK 2026

Recenzent:
dr inż. Mariusz Rybnik

Redaktor naukowy dyscypliny informatyka techniczna i telekomunikacja:
dr hab. inż. Wojciech Kwedlo, prof. PB

Korekta językowa:
Katarzyna Duniewska

Skład i okładka:
Marcin Dominów

ISBN 978-83-68673-26-5 (e-Book)
DOI: 10.24427/978-83-68673-26-5

Oficyna Wydawnicza Politechniki Białostockiej
ul. Wiejska 45C, 15-351 Białystok
e-mail: oficyna.wydawnicza@pb.edu.pl
www.pb.edu.pl

Spis treści

Wstęp

Skrypt *SQL – zajęcia praktyczne* został opracowany z myślą o studentach studiów pierwszego stopnia kierunku informatyka, rozpoczynających systematyczną naukę pracy z relacyjnymi systemami baz danych oraz językiem zapytań SQL. Głównym celem niniejszego opracowania jest wsparcie procesu opanowania języka SQL na poziomie średniozaawansowanym, ze szczególnym uwzględnieniem rozwijania umiejętności praktycznych poprzez realizację uporządkowanych ćwiczeń oraz analizę przykładów odzwierciedlających rzeczywiste zastosowania. Podstawą prezentowanych treści jest **standardowy język SQL zgodny ze specyfikacją ANSI SQL**, stanowiący formalny i powszechnie akceptowany standard komunikacji z relacyjnymi bazami danych [1]. Przykłady oraz ćwiczenia zostały zrealizowane z wykorzystaniem środowiska Oracle, które stanowi jedną z implementacji tego standardu i zachowuje jego podstawowe założenia i mechanizmy.

Język SQL od wielu lat pozostaje podstawowym narzędziem umożliwiającym definiowanie struktur danych, ich przetwarzanie oraz kontrolę dostępu w relacyjnych systemach baz danych. Jego znaczenie wynika z deklaratywnego charakteru, wysokiego poziomu standaryzacji oraz szerokiego zastosowania w systemach informatycznych różnej skali – od prostych aplikacji po rozbudowane systemy korporacyjne. Jednocześnie należy podkreślić, że pomimo swojej ugruntowanej pozycji i powszechnego zastosowania SQL nie stanowi obecnie jedynego podejścia do przechowywania i przetwarzania danych. Wraz z rozwojem systemów rozproszonych oraz aplikacji przetwarzających bardzo duże zbiory danych pojawiły się alternatywne modele baz danych, określane zbiorczo jako NoSQL, obejmujące m.in. bazy dokumentowe, kolumnowe, grafowe oraz bazy klucz–wartość [2] [3]. W przypadku tych rozwiązań często odchodzi się od klasycznego modelu relacyjnego na rzecz większej elastyczności struktury danych oraz możliwości efektywnego działania w środowiskach rozproszonych. Istotną grupę stanowią również systemy określane jako NewSQL, których celem jest połączenie zalet relacyjnego modelu danych i języka SQL ze zwiększoną skalowalnością oraz wydajnością charakterystyczną dla systemów nowej generacji. Coraz większe znaczenie zyskują także bazy grafowe, umożliwiające efektywne modelowanie złożonych relacji pomiędzy obiektami, które znajdują zastosowanie m.in. w analizie sieci powiązań, systemach rekomendacyjnych oraz przetwarzaniu danych semantycznych. Pomimo rozwoju tych alternatywnych technologii standardowy SQL oraz relacyjne bazy danych pozostają fundamentem

współczesnych systemów informatycznych, a ich znajomość stanowi podstawę do zrozumienia zarówno klasycznych, jak i nowoczesnych metod zarządzania danymi.

Jak już stwierdzono, skrypt przeznaczony jest dla studentów studiów pierwszego stopnia na kierunku informatyka, którzy ukończyli podstawowy kurs programowania, posiadają elementarną wiedzę z zakresu algorytmiki i struktur danych oraz rozumieją podstawowe pojęcia związane z przetwarzaniem danych, takie jak typy danych, operacje logiczne oraz warunki. Nie jest wymagana wcześniejsza znajomość języka SQL, jednak przydatne jest ogólne rozumienie sposobu organizacji danych w postaci tabelarycznej oraz znajomość podstawowych pojęć związanych z relacyjnymi bazami danych, takich jak tabela, wiersz, kolumna czy klucz główny.

Celem skryptu jest umożliwienie opanowania praktycznych umiejętności formułowania zapytań w standardowym języku SQL, zgodnym ze specyfikacją ANSI SQL, oraz zrozumienia zasad działania relacyjnych baz danych. Po zapoznaniu się z materiałem możliwe będzie samodzielne pobieranie, filtrowanie i sortowanie danych, wykonywanie operacji agregujących i grupujących, stosowanie podzapytań w różnych kontekstach, łączenie wyników zapytań przy użyciu operatorów zbiorowych, a także definiowanie struktur baz danych oraz manipulowanie danymi. Istotnym elementem kształcenia jest również nabycie umiejętności tworzenia i wykorzystywania perspektyw, które stanowią ważny mechanizm abstrakcji oraz organizacji dostępu do danych.

Zakres skryptu obejmuje stopniowe wprowadzanie kolejnych elementów standardowego języka SQL. W pierwszej kolejności omówione zostały zapytania proste, umożliwiające pobieranie danych z pojedynczych tabel, stosowanie warunków selekcji oraz sortowanie wyników. Następnie przedstawiono mechanizmy grupowania danych oraz wykorzystanie funkcji agregujących, co pozwala na wykonywanie obliczeń na zbiorach danych. Kolejnym etapem było szczegółowe omówienie podzapytań, których zastosowania w niniejszym skrypcie skupiają się na klauzulach `WHERE`, `HAVING`, `FROM` oraz `SELECT`. Przedstawione zostały również operacje na zbiorach wynikowych z wykorzystaniem operatorów `UNION`, `MINUS` oraz `INTERSECT`, pozwalające na łączenie i porównywanie wyników wielu zapytań. Następnie omówione zostały elementy języka definicji danych (DDL), umożliwiające tworzenie i modyfikowanie struktur baz danych, oraz języka manipulacji danymi (DML) służącego do wstawiania, aktualizacji i usuwania danych. Ostatnią część skryptu poświęcono perspektywom (widokom), które umożliwiają definiowanie wirtualnych tabel w celu uproszczenia zapytań, poprawy czytelności oraz zwiększenia kontroli dostępu do danych.

W skrypcie przyjęto podejście zorientowane na rozwijanie umiejętności praktycznych. Poszczególne zagadnienia zostały przedstawione w sposób uporządkowany i uzupełnione przykładami, ćwiczeniami umożliwiającymi samodzielne utrwalenie materiału oraz komentarzami. Szczególny nacisk położono na zrozumienie logiki działania zapytań oraz świadome konstruowanie poprawnych i efektywnych rozwiązań. Wykorzystanie środowiska Oracle jako jednej z implementacji standardu ANSI SQL umożliwia zdobycie doświadczenia w pracy z profesjonalnym systemem zarządzania bazą danych, co stanowi istotny element przygotowania do dalszej edukacji

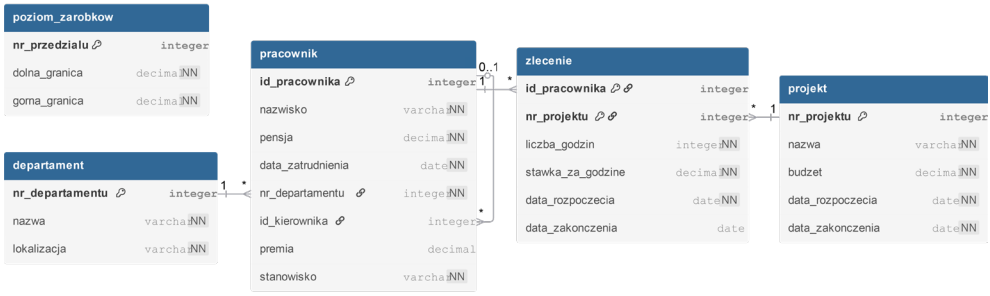
oraz przyszłej pracy zawodowej w obszarze informatyki, analizy danych czy inżynierii oprogramowania.

Skrypt *SQL – zajęcia praktyczne* ma na celu umożliwienie przejścia od podstawowej znajomości baz danych do praktycznego opanowania języka SQL na poziomie średniozaawansowanym. Zdobyta wiedza i umiejętności będą stanowiły istotny element wykształcenia informatycznego oraz solidną podstawę do dalszego poznawania zarówno relacyjnych, jak i nierelacyjnych systemów zarządzania danymi, w tym systemów NoSQL, NewSQL oraz baz grafowych.

Wstęp praktyczny

Należy zaznaczyć, że w skrypcie terminy „atrybut” i „kolumna” będą się przeplatać i oznaczać to samo.

Schemat relacyjnej bazy danych obowiązujący w skrypcie:



RYSUNEK 1. Schemat relacyjnej bazy danych

Tabela: poziom_zarobkow

`nr_przedziału` (klucz główny)

`dolna_granica`

`górna_granica`

Gdzie: dolna i górna granica określają przedział dla zarobków pracownika (pensja zsumowana z premią) z określonym poziomem zarobków (`nr_przedziału`).

Tabela: departament

`nr_departamentu` (klucz główny)

`nazwa`

`lokalizacja`

Gdzie: nazwa i lokalizacja to nazwa i lokalizacja departamentu o określonym numerze. Zgodnie z definicją klucza głównego jeden konkretny numer departamentu może pojawić się w tabeli departament tylko raz. Trzeba mieć więc świadomość tego, że zarówno nazwy, jak i lokalizacje departamentów mogą się w obrębie tej tabeli powtarzać.

Tabela: pracownik

id_pracownika (klucz główny)
nazwisko
pensja
data_zatrudnienia
nr_departamentu (klucz obcy; odnosi się do departament)
id_kierownika
premia
stanowisko

Wszystkie atrybuty/kolumny zależą bezpośrednio od klucza głównego tej relacji (id_pracownika), czyli jeden pracownik może mieć jedno nazwisko, jedną pensję czy mieć określony tylko jeden departament, w którym jest zatrudniony. Jeden pracownik może być kierownikiem wielu pracowników (jego identyfikator może znaleźć się w kolumnie id_kierownika wielokrotnie). Pracownik może mieć lub też może nie mieć kierownika – wtedy w kolumnie id_kierownika pojawi się pusta komórka (NULL) w wierszu danego pracownika. Podobnie w przypadku atrybutu premia: pracownik może otrzymywać lub nie otrzymywać premii.

Tabela: projekt

nr_projektu (klucz główny)
nazwa
budzet
data_rozpoczecia
data_zakonczenia

Gdzie: daty rozpoczęcia i zakończenia są planowanymi datami rozpoczęcia i zakończenia realizacji projektu. Podobnie budzet: jest to planowana kwota przeznaczona na realizację projektu – w rzeczywistości może jednak okazać się ona inna.

Tabela: zlecenie

`id_pracownika` (klucz obcy, odnosi się do pracownik)

`nr_projektu` (klucz obcy, odnosi się do projekt)

`liczba_godzin`

`stawka_za_godzinę`

`data_rozpoczęcia`

`data_zakończenia`

Kluczem głównym tabeli `zlecenie` jest kombinacja atrybutów (`id_pracownika`, `nr_projektu`), co oznacza, że kombinacja ich wartości jest unikalna w obrębie tej tabeli. Przy tak zdefiniowanym kluczu głównym: jeden pracownik mógł pracować nad wieloma projektami, jeden projekt mógł być realizowany przez wielu pracowników, a także jeden konkretny pracownik mógł pracować nad jednym konkretnym projektem tylko raz.

1. Zapytania proste

Zapytania proste SQL stanowią podstawowy mechanizm umożliwiający pobieranie danych z relacyjnych baz danych. Umiejętność konstruowania poprawnych i precyzyjnych zapytań jest kluczowa dla efektywnego dostępu do danych, ich analizy oraz interpretacji informacji przechowywanych w tabelach bazy danych. Wszystkie bardziej zaawansowane operacje języka SQL, takie jak grupowanie danych, złączenia, podzapytania czy operacje manipulacji danymi, opierają się na koncepcjach wprowadzanych na etapie zapytań prostych. Z tego względu dokładne zrozumienie ich składni, struktury oraz zasad działania stanowi niezbędny fundament dalszej nauki systemów baz danych.

W relacyjnych systemach baz danych informacje przechowywane są w postaci tabel składających się z wierszy i kolumn. Każda tabela reprezentuje relację, każdy wiersz odpowiada pojedynczemu rekordowi, natomiast każda kolumna reprezentuje określony atrybut tego rekordu. Język SQL ma charakter deklaratywny, co oznacza, że użytkownik określa, jakie dane mają zostać pobrane, natomiast system zarządzania bazą danych samodzielnie decyduje o sposobie ich uzyskania. Takie podejście pozwala skupić się na logicznym aspekcie przetwarzania danych, bez konieczności uwzględniania szczegółów implementacyjnych związanych z fizyczną organizacją danych.

Podstawowa składnia zapytania SQL obejmuje klauzule `SELECT` oraz `FROM`. Klauzula `SELECT` określa, które kolumny lub wyrażenia mają zostać uwzględnione w wyniku zapytania, natomiast klauzula `FROM` wskazuje tabelę lub tabele stanowiące źródło danych. W najprostszym przypadku zapytanie może zwracać wszystkie kolumny danej tabeli lub jedynie wybrane kolumny, w zależności od potrzeb użytkownika. Klauzula `SELECT` może również zawierać wyrażenia, wartości stałe oraz aliasy kolumn, co umożliwia poprawę czytelności wyników oraz wykonywanie obliczeń w trakcie pobierania danych.

Choć klauzule `SELECT` i `FROM` stanowią minimalną strukturę zapytania, język SQL udostępnia dodatkowe mechanizmy rozszerzające jego funkcjonalność i umożliwiające precyzyjną kontrolę nad zwracanym wynikiem. Jedną z najważniejszych jest klauzula `WHERE`, która służy do filtrowania wierszy na podstawie określonych warunków logicznych. Klauzula `WHERE` umożliwia ograniczenie zbioru wynikowego wyłącznie do tych rekordów, które spełniają wskazane kryteria selekcji. Jej zastosowanie ma szczególne znaczenie w przypadku pracy z dużymi zbiorami danych, ponieważ pozwala na wyodrębnienie informacji istotnych z punktu widzenia analizowanego problemu oraz eliminację danych zbędnych.

Warunki definiowane w klauzuli `WHERE` budowane są z wykorzystaniem operatorów porównania, takich jak równość, nierówność, większe od, mniejsze od, większe lub równe oraz mniejsze lub równe. Warunki te mogą być łączone i modyfikowane przy użyciu spójników logicznych, takich jak `AND`, `OR` oraz `NOT`. Spójnik `AND` wymaga, aby wszystkie połączone warunki były spełnione jednocześnie, natomiast spójnik `OR` oznacza, że spełniony musi być co najmniej jeden z warunków. Operator `NOT` pozwala na zanegowanie warunku i wykluczenie określonych rekordów ze zbioru wynikowego. Właściwe wykorzystanie spójników logicznych umożliwia konstruowanie złożonych warunków selekcji oraz precyzyjne określenie zakresu zwracanych danych.

Istotnym elementem zapytań prostych jest również możliwość pobierania danych z więcej niż jednej tabeli poprzez zastosowanie mechanizmu **złączeń** (`JOIN`). Złączenia umożliwiają powiązanie danych przechowywanych w różnych tabelach na podstawie określonych warunków zgodności wartości wybranych kolumn, zwykle powiązanych relacją klucza głównego i klucza obcego. Najczęściej stosowanym typem złączenia jest **złączenie wewnętrzne** (`INNER JOIN`), zwracające wyłącznie te rekordy, dla których istnieje zgodność wartości w obu łączonych tabelach. Oznacza to, że w wyniku zapytania uwzględniane są jedynie rekordy posiadające odpowiedniki w obu tabelach.

Oprócz złączeń wewnętrznych stosowane są również **złączenia zewnętrzne** (`OUTER JOIN`), umożliwiające uwzględnienie w wyniku zapytania także tych rekordów, które nie posiadają odpowiadających im rekordów w drugiej tabeli. W przypadku `LEFT OUTER JOIN` zwracane są wszystkie rekordy z tabeli znajdującej się po lewej stronie złączenia oraz odpowiadające im rekordy z tabeli po prawej stronie, natomiast brakujące wartości zastępowane są wartościami `NULL`. Analogicznie `RIGHT OUTER JOIN` zwraca wszystkie rekordy z tabeli po prawej stronie złączenia oraz odpowiadające im rekordy z tabeli po lewej stronie. Szczególnym przypadkiem złączenia zewnętrznego jest `FULL OUTER JOIN`, które zwraca wszystkie rekordy z obu tabel, niezależnie od tego, czy istnieją odpowiadające im rekordy w drugiej tabeli. W przypadku braku zgodności brakujące wartości również reprezentowane są przez wartości `NULL`. Zrozumienie zasad działania złączeń stanowi istotny element pracy z relacyjnymi bazami danych, ponieważ dane w rzeczywistych systemach informatycznych są zazwyczaj rozproszone w wielu powiązanych ze sobą tabelach.

Istotnym elementem zapytań prostych jest również klauzula `ORDER BY`, która umożliwia uporządkowanie wyników zapytania według wartości wybranych kolumn. Domyślnie kolejność rekordów zwracanych przez zapytanie nie jest określona, dlatego zastosowanie klauzuli `ORDER BY` pozwala na sortowanie wyników w kolejności rosnącej lub malejącej, co ma istotne znaczenie dla poprawy czytelności wyników oraz przygotowania danych do dalszej analizy.

Klauzula `SELECT` umożliwia również stosowanie wyrażeń, które mogą obejmować operacje arytmetyczne, operacje na łańcuchach znaków oraz wykorzystanie funkcji wbudowanych języka SQL. Pozwala to na realizację obliczeń oraz przekształceń danych bezpośrednio w trakcie wykonywania zapytania, bez konieczności ingerencji w dane przechowywane w bazie.

Istotnym aspektem pracy z zapytaniami SQL jest również zrozumienie logicznej kolejności ich przetwarzania. Pomimo że klauzula `SELECT` występuje na początku zapytania, przetwarzanie rozpoczyna się od wskazania źródła danych w klauzuli `FROM`, w której określane są również złączenia pomiędzy tabelami. Następnie stosowane są warunki filtrowania określone w klauzuli `WHERE`, które ograniczają zbiór rekordów do tych spełniających określone kryteria. Kolejnym etapem jest wyznaczenie wartości zwracanych przez klauzulę `SELECT`, a na końcu, jeżeli została ona zastosowana, wykonywane jest sortowanie wyników przy użyciu klauzuli `ORDER BY`. Zrozumienie tej logicznej kolejności stanowi podstawę poprawnego konstruowania zapytań oraz właściwej interpretacji ich wyników.

W wyniku zapoznania się z treścią niniejszego rozdziału student zdobędzie wiedzę i umiejętności umożliwiające samodzielne konstruowanie prostych zapytań SQL zgodnych ze standardem ANSI SQL oraz ich wykonywanie w środowisku Oracle. Student nabędzie umiejętności pobierania danych z jednej lub wielu tabel, stosowania różnych typów złączeń, w tym `INNER JOIN`, `LEFT OUTER JOIN`, `RIGHT OUTER JOIN` oraz `FULL OUTER JOIN`, oraz właściwego interpretowania wyników tych operacji. Opanuje również stosowanie klauzuli `WHERE` do filtrowania danych z wykorzystaniem operatorów porównania oraz spójników logicznych, co umożliwi precyzyjne definiowanie warunków selekcji. Ponadto student nauczy się wybierać określone kolumny, stosować aliasy, tworzyć wyrażenia obliczeniowe oraz porządkować wyniki zapytań przy użyciu klauzuli `ORDER BY`.

Podczas lektury rozdziału student rozwinie umiejętność analitycznego formułowania zapytań oraz zrozumie zasady logicznego przetwarzania zapytań. Zdobyta wiedza stanowić będzie podstawę do dalszego poznawania bardziej zaawansowanych mechanizmów języka SQL, takich jak zapytania grupujące, podzapytania, operacje na zbiorach wynikowych oraz inne elementy wykorzystywane w pracy z relacyjnymi bazami danych.

1.1. Zadania z rozwiązaniami i komentarzami

ZADANIE 1

Podać nazwisko i pensję każdego pracownika, dane posortować malejąco według daty zatrudnienia.

```
select id_pracownika, nazwisko, pensja, data_zatrudnienia
from pracownik
order by data_zatrudnienia DESC;
```

Komentarz 1

Jak widać, wyniki możemy porządkować po atrybucie, którego nie wybieramy – nie pojawia się wśród wynikowych kolumn. Wybieramy tak naprawdę całe wiersze (w tym przypadku z tabeli `pracownik`), a po słowie kluczowym `SELECT` wymieniamy atrybuty, które chcemy uwzględnić w wyniku.

Komentarz 2

Sytuacja ma się zgoła odmiennie, gdy w zapytaniu użyte zostanie słowo kluczowe `DISTINCT`, które eliminuje z wyników powtórzenia (wiersze z dokładnie takimi samymi wartościami).

Przykład:

```
select DISTINCT nazwisko, pensja
from pracownik
order by data_zatrudnienia DESC;
```

Komentarz 3

Uruchomienie powyższego zapytania spowoduje wywołanie błędu składniowego, czyli takiego, który uniemożliwi wykonanie zapytania. Polega on na tym, że zażądano wypisania bez powtórzeń nazwisk i pensji pracowników (unikalnych kombinacji wartości tych dwóch atrybutów). W związku z tym reszta atrybutów składających się na wiersz w tabeli `pracownik` zostaje „zapomniana" (sklejeniu ulegają kombinacje wartości nazwiska i pensji, np. data zatrudnienia jest całkowicie pomijana i nie ma wpływu na uzyskanie wspomnianych unikalnych kombinacji).

ZADANIE 2

Wyznaczyć dla każdego pracownika jego zarobki (pensja + premia). Nadać sumie etykietę. Wypisać nazwisko i wyznaczoną sumę.

```
select id_pracownika, nazwisko, pensja + nvl(premia,0) as zarobki
from pracownik;
```

Komentarz 1

Dla każdego pracownika w tabeli `pracownik` podane są jego pensja i premia. Premia dla dowolnego pracownika może być pusta (`NULL`, brak wartości w komórce) i w sytuacji, gdy wyznaczane są zarobki (`pensja + premia`) dla pracownika, który nie otrzymuje premii, suma ta będzie „wynosić" `NULL` (nie można dodać liczby do `NULL`). Konieczna, a także logiczna jest zamiana `NULL` na wartość, którą można by dodać do pensji. Oczywistym wyborem jest liczba 0 (drugi argument funkcji `nvl`).

ZADANIE 3

Sprawdzić działanie zapytania:

```
select id_pracownika, nazwisko, nr_przedzialu, pensja+nvl(premia,0),
      dolna_granica, gorna_granica
from pracownik, poziom_zarobkow
order by id_pracownika, nr_przedzialu;
```

Komentarz 1

Należy zastanowić się, jakie kolumny/atrybuty wchodzą w skład wynikowych wierszy. Trzeba zauważyć, że po słowie kluczowym `FROM` występują dwie tabele. Wynikiem tego jest powstanie iloczynu kartezjańskiego wierszy z obu tabel, tzn. każdy wiersz z tabeli `pracownik` sklejany jest z każdym wierszem z tabeli `poziom_zarobkow`. Jeśli tak, to w wyniku sklejenia dowolnego wiersza z tabeli `pracownik` (8 atrybutów/kolumn) z dowolnym wierszem z tabeli `poziom_zrobkow` (3 atrybuty/kolumny) powstaną wiersze 11-kolumnowe (8 + 3).

Komentarz 2

Należy zastanowić się, ile wierszy powstało w wyniku wykonania zapytania. Jak już wspomniano, powstał iloczyn kartezjański wierszy, czyli wynikiem będzie $x \times y$ wierszy, gdzie x to liczba wierszy w tabeli pracownik, a y liczba wierszy w tabeli `poziom_zrobkow`.

Sprawdzić, na jakim poziom posiada wynagrodzenie każdy zatrudniony. Wyświetlić nazwisko i poziom wynagrodzenia.

```
select id_pracownika, nazwisko, nr_przedzialu,
      pensja+nvl(premia,0) as zarobki, dolna_granica, gorna_granica
from pracownik, poziom_zarobkow
where pensja+nvl(premia,0) between dolna_granica and gorna_granica
order by id_pracownika, nr_przedzialu;
```

Komentarz 3

Użyto klauzuli `WHERE`, która służy do filtrowania wierszy. W przypadku tego zadania należy uwzględnić tylko te wiersze (ze zbioru wierszy powstałych z iloczynu kartezjańskiego), w których zarobki pracownika mieszczą się pomiędzy dolną i górną granicą danego przedziału – tylko takie wiersze są poprawne.

ZADANIE 4

Sprawdzić działanie zapytania:

```
select id_pracownika, nazwisko, nr_departamentu, nazwa
from pracownik, departament;
```

Komentarz 1

Wybierane są dane z dwóch tabel: `pracownik` i `departament`. Przy tak skonstruowanym zapytaniu wywołany zostanie błąd składniowy polegający na tym, że w zapytaniu jest lub są niejednoznaczne odwołania do atrybutów/kolumn. Niejednoznaczność w tym przypadku polega na odwołaniu się do atrybutu `nr_departamentu` i jest spowodowana faktem, że atrybut o takiej nazwie pojawia się zarówno w tabeli `pracownik`, jak i `departament`.

```
select id_pracownika, nazwisko, departament.nr_departamentu, nazwa
from pracownik, departament;
```

Do każdego atrybutu można się odwoływać w formie `nazwa_tabeli.nazwa_atrybutu`. Nie zawsze jest to jednak konieczne, na przykład w podanym zapytaniu odwołujemy się do atrybutu `nazwisko` bez podania tabeli, z której pochodzi. Nie jest to w tym przypadku wymagane, ponieważ atrybut `nazwisko` występuje tylko w jednej z dwóch wymienionych po `FROM` tabel.

Komentarz 2

Należy zastanowić się, jakie kolumny/atrybuty wchodzą w skład wynikowych wierszy – analogicznie do poprzedniego zadania. Każdy wiersz z tabeli `pracownik` zostanie sklejony z każdym wierszem tabeli `departament`, a więc wynikowe wiersze będą zbudowane ze wszystkich atrybutów tabeli `pracownik` i wszystkich atrybutów tabeli `departament`, czyli będzie ich w sumie 11.

Wyznaczyć listę tych pracowników, których departament mieści się w Białymstoku lub Olsztynie. Wypisać nazwę departamentu i nazwisko.

Rozwiązanie 1:

```
select id_pracownika, nazwisko, pracownik.nr_departamentu,
      departament.nr_departamentu, nazwa, lokalizacja
from pracownik, departament
where pracownik.nr_departamentu = departament.nr_departamentu
      and lower(lokalizacja) = 'bialystok' or upper(lokalizacja) =
     'OLSZTYN';
```

Komentarz 1

Dzięki pierwszemu warunkowi określonemu w klauzuli `WHERE` zostawione zostały tylko te wiersze, które po sklejaniu mają zgodne departamenty.

Komentarz 2

Użyto funkcji `LOWER/UPPER`, ponieważ nie jest wiadome (nie można zakładać), jak pisane są teksty w bazie – wielkimi czy małymi literami. W związku z tym, przy wykorzystaniu funkcji `LOWER` to, co jest w bazie, zamieniane jest na małe litery (i wówczas porównywane z małymi literami). Analogicznie tłumaczy się wykorzystanie funkcji `UPPER`.

Komentarz 3

Zapytanie, mimo że się uruchomi, nie dostarczy poprawnych wyników. Przyczyną jest ta część kodu:

```
where pracownik.nr_departamentu = departament.nr_departamentu
      and lower(lokalizacja) = 'bialystok' or upper(lokalizacja) =
     'OLSZTYN';
```

Przykład:

W bazie są tylko jeden departament z Olsztyna i jeden z Białegostoku. W firmie zatrudnionych jest 350 osób, z czego 40 w Olsztynie i 100 w Białymstoku.

a) Ile powstanie wierszy?
   Powstanie 100 wierszy (z warunku `where pracownik.nr_departamentu = departament.nr_departamentu and lower(lokalizacja) = 'bialystok'`) + 350 wierszy (z warunku `or upper(lokalizacja) = 'OLSZTYN'`). Powodem dodania wartości 350 jest ponownie iloczyn kartezjański – wiersz z tabeli `departament` z lokalizacją `Olsztyn` zostanie dołączony do każdego pracownika.
b) Ile wierszy powinno powstać?
   40 + 100 = 140

Rozwiązanie 2:

```
select id_pracownika, nazwisko, pracownik.nr_departamentu,
      departament.nr_departamentu, nazwa, lokalizacja
from pracownik, departament
where pracownik.nr_departamentu = departament.nr_departamentu
      and ( lower(lokalizacja) = 'bialystok' or upper(lokalizacja) =
                                                    'OLSZTYN' )
order by id_pracownika, departament.nr_departamentu;
```

Komentarz 4

Dodanie nawiasów rozwiązuje problem uwzględnienia w wyniku niepoprawnie połączonych wierszy.

ZADANIE 5

Podać nazwę projektu wraz z planowanymi i faktycznymi datami rozpoczęcia realizacji projektu. Posortować wyniki tak, aby dla każdego projektu łatwo można było znaleźć datę rozpoczęcia realizacji (pierwszą faktyczną).

```
select p.nr_projektu, nazwa, p.data_rozpoczecia as planowana,
      z.data_rozpoczecia as faktyczna
from projekt p, zlecenie z
where p.nr_projektu = z.nr_projektu
order by p.nr_projektu, z.data_rozpoczecia;
```

Komentarz 1

Wprowadzono aliasy dla nazw tabel. Powodami są wygoda i skrócenie kodu: zamiast odwoływać się do atrybutu `nr_projektu` z tabeli `projekt` poprzez `projekt.nr_projektu`, można odwołać się do tabeli z wykorzystaniem nadanego mu aliasu `p`, czyli poprzez `p.nr_projektu`.

Komentarz 2

Planowane daty realizacji projektów znajdują się w tabeli `projekt`, natomiast faktyczne w tabeli `zlecenie`.

ZADANIE 6

Podać nazwiska tych pracowników, którzy nie mają kierowników.

```
select id_pracownika, nazwisko, id_kierownika
from pracownik
where id_kierownika is null;
```

Komentarz 1

Pracownik może mieć lub może nie mieć kierownika. Gdy go ma, w kolumnie `id_kierownika` znajdzie się identyfikator pracownika, który jest bezpośrednim przełożonym danej osoby. Gdy go nie ma, w kolumnie `id_kierownika` dla danej osoby będzie pusto/`NULL`.

ZADANIE 7

Wypisać nazwiska pracowników oraz nazwiska ich bezpośrednich przełożonych.

Komentarz 1

Mając dostępny dowolny wiersz tabeli `pracownik`, można wskazać dla dowolnej osoby identyfikator pracownika, który jest jego bezpośrednim przełożonym (kolumna `id_kierownika`). Nie ma się jednak bezpośredniego dostępu do jego nazwiska.

```
select p.id_pracownika || ' ' || p.nazwisko as „podwladny”,
       k.id_pracownika || ' ' || k.nazwisko as „kierownik”
from pracownik p, pracownik k
where p.id_kierownika = k.id_pracownika
order by p.id_pracownika;
```

Komentarz 2

W rozwiązaniu wykorzystano możliwość złączenia jednej tabeli z samą sobą (ang. *self-join*). Powstaje więc iloczyn kartezjański tych tabel, czyli każdy wiersz z tabeli `pracownik p` (podwładny) łączony jest z każdym wierszem z tabeli `pracownik k` (kierownik). Poprawne wiersze powstałe z tego połączenia zdefiniowano w klauzuli `WHERE`: aby znaleźć kierownika danej osoby, należy znaleźć takiego pracownika, którego identyfikator jest równy identyfikatorowi jego kierownika.

Komentarz 3

Aby wyniki były bardziej przejrzyste, wykorzystano operator konkatenacji ||.

ZADANIE 8

Wypisać nazwiska pracowników, nazwiska ich bezpośrednich przełożonych oraz ich zarobki pod warunkiem, że są na innych stanowiskach.

```
select p.id_pracownika || ' ' || p.nazwisko as „podwladny”,
       p.pensja+nvl(p.premia,0) as „zarobki podwladnego”,
       k.id_pracownika || ' ' || k.nazwisko as „kierownik”,
       k.pensja+nvl(k.premia,0) as „zarobki kierownika”
from pracownik p, pracownik k
where p.id_kierownika = k.id_pracownika and
       p.stanowisko != k.stanowisko
order by p.id_pracownika;
```

ZADANIE 9

Podać listę pracowników, których stanowisko rozpoczyna się literą „a”, a lokalizacja departamentu, w którym pracują, zawiera ciąg znaków „tok”. Wypisać nazwisko, stanowisko i lokalizację departamentu.

```
select id_pracownika, nazwisko, stanowisko, lokalizacja
from pracownik p, departament d
where p.nr_departamentu = d.nr_departamentu
      and upper(stanowisko) like 'A%'
      and lower(lokalizacja) like '%tok%';
```

Komentarz 1

Przypomnienie: nie można zakładać, że w kolumnach tekstowych bazy danych teksty będą miały określoną formę (będą pisane wielką/małą literą itp.). W związku z tym istnieje konieczność „dostosowania się” i użycia funkcji `LOWER`/`UPPER`.

ZADANIE 10

Wyznaczyć listę pracowników, których nazwiska rozpoczynają się od liter „m” lub „n” i którzy nie otrzymują premii.

```
select id_pracownika, nazwisko, premia
from pracownik
where ( lower(nazwisko) like 'n%' or lower(nazwisko) like 'm%' )
      and premia is null;
```

Komentarz 1

Należy pamiętać o logice. W tym przypadku trzeba wziąć pod uwagę odpowiednie umieszczenie nawiasów w klauzuli `WHERE`.

ZADANIE 11

Podać nazwiska i zarobki tych pracowników, których poziom wynagrodzenia (`nr_przedzialu` z tabeli `poziom_zarobkow`) jest równy 1 lub 2.

```
select id_pracownika, nazwisko, pensja+nvl(premia,0) as zarobki,
      nr_przedzialu
from pracownik, poziom_zarobkow
where pensja+nvl(premia,0) between dolna_granica and gorna_granica
      -- and ( nr_przedzialu = 1 or nr_przedzialu = 2 );
      and nr_przedzialu in (1, 2);
```

Komentarz 1

Dwa minusy umożliwiają wprowadzenie w skrypcie jednej linii komentarza. Komentarz wielolinijkowy: `/* … */`.

Komentarz 2

Przedstawiono dwie możliwości wyodrębnienia przedziałów 1 i 2. Słowo kluczowe `IN` oznacza „należy do zbioru" – po nim definiowany jest dopuszczalny zbiór wartości.

ZADANIE 12

W jakich departamentach prowadzone są prace nad poszczególnymi projektami? Lista powinna zawierać odpowiednio nazwy departamentów i projektów.

```
select distinct d.nr_departamentu, d.nazwa, p.nr_projektu, p.nazwa
from departament d, pracownik pr, zlecenie z, projekt p
where d.nr_departamentu = pr.nr_departamentu
      and pr.id_pracownika = z.id_pracownika
      and z.nr_projektu = p.nr_projektu
order by d.nr_departamentu, p.nr_projektu;
```

Komentarz 1

Konieczne jest wykorzystanie aż czterech tabel, co w wyniku daje `t1 * t2 * t3 *t4` wierszy, gdzie `ti` to odpowiednia tabela z klauzuli `FROM`.

Komentarz 2

Dodano słowo kluczowe `DISTINCT`. Ze względu na to, że wielu pracowników z jednego departamentu mogło realizować ten sam projekt, usunięto powtórzenia, ponieważ nie wnoszą one nic nowego do wyniku.

ZADANIE 13

Podać listę stanowisk w każdym departamencie (bez powtórzeń). Należy wypisać nazwę departamentu i stanowisko.

```
select distinct d.nr_departamentu, nazwa, stanowisko
from departament d, pracownik p
where d.nr_departamentu = p.nr_departamentu
order by d.nr_departamentu;
```

ZADANIE 14

Podać nazwiska osób pracujących na stanowisku sprzedawcy z departamentu, którego nazwa zawiera cyfrę 1.

```
select id_pracownika, nazwisko, stanowisko
from pracownik p, departament d
where p.nr_departamentu = d.nr_departamentu
      and lower(stanowisko) = 'sprzedawca'
      and nazwa like '%1%';
```

ZADANIE 15

Podać nazwy departamentów i wymienić stanowiska, jakie w tych departamentach występują (należy uwzględnić też departamenty, w których nikt nie pracuje). Wyniki uszeregować alfabetycznie po nazwach departamentów i stanowiskach.

```
select distinct d.nr_departamentu, nazwa, stanowisko
from pracownik p right join departament d
      on p.nr_departamentu = d.nr_departamentu
order by nazwa, stanowisko;
```

Komentarz 1

Wprowadzono złączenie zewnętrzne. W wynikach uwzględnione są wszystkie departamenty, nawet jeśli ich numer nie znajduje się w tabeli `pracownik p`. Po sklejeniu wszystkich wierszy tabeli `departament d` ze wszystkimi wierszami z tabeli `pracownik p` (następstwo wymienienia po `FROM` większej liczby tabel niż jedna) w wierszach dotyczących departamentów, dla których nie ma odpowiedników w tabeli `pracownik p` (nie ma pracowników w danym departamencie), wszystkie kolumny tabeli `pracownik p` wypełnione zostaną pustymi wartościami (`NULL`). Na przykład tak wyglądałby po połączeniu wiersz dla departamentu o numerze 5, w którym nie ma pracowników:

d.nr_departamentu	nazwa	lokalizacja	id_pracownika	nazwisko	...	p.nr_departamentu	...
5	Dep5	L5	NULL	NULL	...	NULL	...

ZADANIE 16

Podać nazwiska pracowników wraz z nazwami projektów, nad którymi pracowali. Należy uwzględnić w wynikach projekty, nad którymi nikt nie pracował oraz pracowników, którzy nie realizowali żadnego projektu.

```
select pr.id_pracownika, nazwisko, p.nr_projektu, nazwa
from pracownik pr left join zlecenie z
        on pr.id_pracownika = z.id_pracownika full join projekt p
        on z.nr_projektu = p.nr_projektu;
```

Komentarz 1

Wykorzystano dwa rodzaje złączeń zewnętrznych: `left join` (1) oraz `full join` (2). W wyniku złączenia (1) uwzględnieni zostali wszyscy pracownicy, nawet jeśli nie realizowali żadnego projektu. Wydawałoby się, że tabelę powstałą z (1) należy połączyć zewnętrznie, (`right join`), z tabelą `projekt`, tak aby uwzględnić projekty, które nie były dotychczas realizowane:

```
from pracownik pr left join zlecenie z
        on pr.id_pracownika = z.id_pracownika right join projekt p
        on z.nr_projektu = p.nr_projektu
```

Jest to jednak rozwiązanie nieprawidłowe. Wynikiem złączenia

```
from pracownik pr left join zlecenie z
        on pr.id_pracownika = z.id_pracownika
```

będzie tabela pracowników z dopisanymi im ich zleceniami. Uwzględnieni w niej będą także pracownicy, którzy nigdy nie realizowali projektów – dla nich we wszystkich dołączonych kolumnach tabeli `zlecenie` wystąpią „puste wartości” (`NULL`), także w kolumnie `nr_projektu`. Część łączenia

```
... right join projekt p
        on z.nr_projektu = p.nr_projektu
```

tłumaczy się jako: „dołącz wszystkie projekty, nawet te, które nie były realizowane”. Jednak co w takim razie z wierszami, w przypadku których w kolumnie `nr_projektu` tabeli `zlecenie` jest `NULL` (w wyniku łączenia (1))? Niestety nie ma podstaw, aby dołączyć je do (2), ponieważ łączone będą tylko te wiersze, w których zgadzają się numery projektów lub też `nr_projektu` z tabeli `projekt` nie pojawia się w tabeli `zlecenie`.

ZADANIE 17

Podać nazwiska pracowników wraz z nazwami projektów. Uwzględniamy tylko tych pracowników, którzy nie pracowali nad żadnym projektem i projekty, które nie były realizowane.

```
select pr.id_pracownika, nazwisko, p.nr_projektu, nazwa
from pracownik pr left join zlecenie z
      on pr.id_pracownika = z.id_pracownika full join projekt p
      on z.nr_projektu = p.nr_projektu
where z.id_pracownika is null;
```

1.2. Zasada nr 1

Nigdy nie tworzy się zapytań, dopasowując kod do zawartości bazy danych. Zapytania mają działać na każdej zawartości bazy danych określonego schematu.

1.3. Wskazówki dotyczące składni SQL

SELECT *nazwy kolumn (z nadanymi etykietami lub bez) po przecinkach*
FROM *nazwy tabel (z nadanymi etykietami, lub bez) po przecinkach*
[
WHERE *warunki*
GROUP BY *kolumny, po których odbywa się grupowanie (nazwy kolumn po przecinkach)*
HAVING *warunki dla nowo powstałych grup*
ORDER BY *nazwy kolumn po przecinkach, według których odbywać się będzie porządkowanie wyników (rosnąco ASC – domyślne, malejąco DESC)*
];

Nawiasy kwadratowe wskazują na klauzule, które można umieścić w zapytaniu, ale nie są one niezbędne do uruchomienia zapytania.

- Funkcja `nvl(war1, war2)`.
  Gdy odczytana wartość kolumny o nazwie `war1` jest wartością pustą (`NULL`), to zamieniana jest na wartość drugiego argumentu funkcji, czyli `war2`. W przeciwnym razie zwracana jest wartość `war1`.

- Funkcje `LOWER, UPPER`.
  Zgodnie z nazwą zamieniają one podany jako argument tekst na małe bądź wielkie litery.

- Słowo kluczowe `DISTINCT`.
  Służy ono do usuwania powtórzeń w wynikowych wierszach.

```
select distinct nazwisko
from pracownik;
```

Wypisane zostaną wszystkie nazwiska, bez powtarzania (jeśli jest pięciu Kowalskich, to nazwisko Kowalski zostanie wypisane tylko raz; bez słowa `DISTINCT` wypisanych zostałoby pięć wierszy z nazwiskiem Kowalski).

Wynikiem zapytania

```
select distinct nazwisko, pensja
from pracownik;
```

będą wiersze, w których nie powtórzy się kombinacja wartości atrybutów `nazwisko` oraz `pensja` – na przykład N1, 1000 (oznacza to, że jeśli byłoby parę osób o takim samym nazwisku i zarabiających tyle samo, wypisana zostałaby tylko jedna kombinacja tych wartości).

- Klauzula `WHERE`.
  Służy ona do filtrowania wierszy. W wynikach uwzględniane są tylko te wiersze, które spełniają warunki zdefiniowane w tej klauzuli.
  1. Warunki są łączone spójnikami `AND` lub `OR`.

```
WHERE pensja > 300 AND ( nr_departamentu = 30 OR nr_departamentu = 50 )
```

  2. `BETWEEN war1 AND war2`.

`WHERE pensja BETWEEN 300 AND 500` to samo, co:

`WHERE pensja >= 300 AND pensja <= 500`

  3. `IS NULL` (`IS NOT NULL`).
  `WHERE kolumna IS NULL` (wybierane są wiersze, w których w danej kolumnie występuje pusta wartość `NULL`, tzn. komórka jest pusta).

  4. `LIKE`.
  `WHERE kolumna LIKE 'A%'` (oznacza to, że wybierane są wiersze, w których wartość danej kolumny rozpoczyna się literą „a", po niej zaś może być wiele znaków).
  `%` → 0 lub więcej znaków
  `_` → jeden znak

Wybieranie danych z więcej niż jednej tabeli

Do każdej kolumny można odwoływać się w następujący sposób: `tabela.kolumna`.

Nie jest to jednak konieczne. Konieczność taka pojawia się w sytuacji, gdy wybieramy dane z większej liczby tabel i nazwa danej kolumny występuje w co najmniej dwóch tabelach.

Załóżmy, że w tabelach `tabela1` i `tabela2` jest kolumna o nazwie `kol`. Wtedy odwołanie w postaci

```
SELECT kol
FROM tabela1, tabela2;
```

jest niejednoznaczne i należy zastąpić `kol` jednym z następujących odwołań: `tabela1.kol` lub `tabela2.kol`.

Jeśli wybierane są dane z większej liczby tabel, powstaje **iloczyn kartezjański** wierszy, tzn. każdy wiersz z jednej tabeli łączony jest (sklejany) z każdym wierszem z drugiej tabeli. Jeśli istnieje jakaś reguła mówiąca, które wiersze po połączeniu należy pozostawić, a które odrzucić, to klauzula `WHERE` daje możliwość określenia warunków włączania wierszy do wyniku, na przykład:

```
WHERE tab1.kolumna1 = tab2.kolumna2
```

co oznacza, że wartości w wymienionych kolumnach mają sobie odpowiadać.

Preferowanym sposobem „eliminacji iloczynu kartezjańskiego" jest jednak zastosowanie złączeń tabel i pozostawienie filtrowania klauzuli `WHERE`.

Złączenia tabel

1. **LEFT [outer] JOIN**.

W tym przypadku zwracane są wszystkie wiersze tabeli po lewej stronie, nawet gdy nie ma odpowiedników w tabeli po prawej stronie.

```
SELECT kol_A, kol_B
FROM tab_A LEFT JOIN tab_B
ON tab_A.łącze_A = tab_B.łącze_B;
```

Wybrane zostaną wszystkie wiersze spełniające warunek `tab_A.łącze_A = tab_B.łącze_B` oraz wiersze z tabeli `tab_A`, dla których nie ma odpowiedników w tabeli `tab_B` (określona wartość `łącze_A` nie pojawia się w tabeli `tab_B`).

2. **RIGHT [outer] JOIN**.

W tym przypadku zwracane są wszystkie wiersze tabeli po prawej stronie, nawet gdy nie ma odpowiedników w tabeli po lewej stronie.

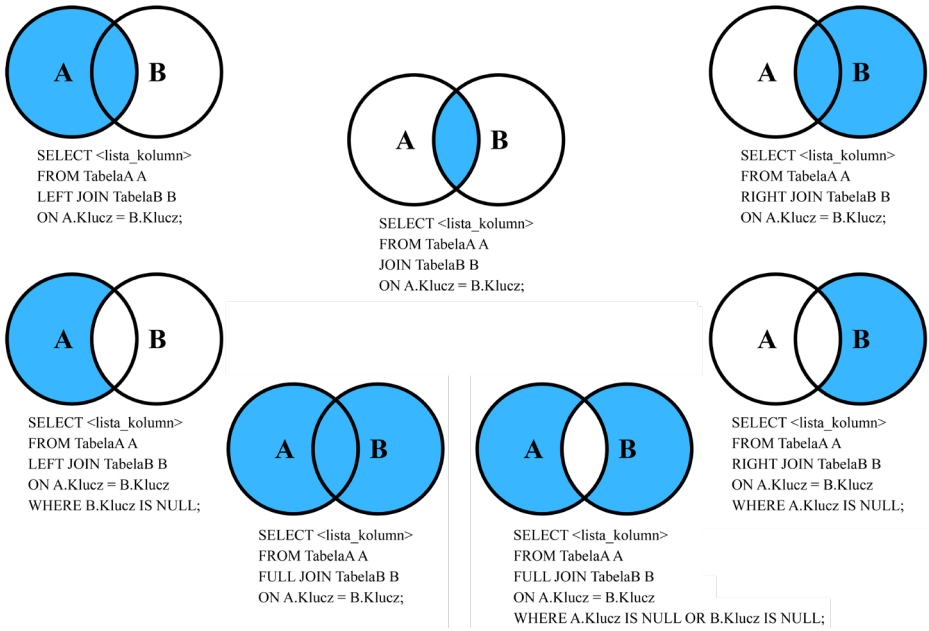
```
SELECT kol_A, kol_B
FROM tab_A RIGHT JOIN tab_B
ON tab_A.łącze_A = tab_B.łącze_B;
```

Wybrane zostaną wszystkie wiersze spełniające warunek `tab_A.łącze_A = tab_B.łącze_B` oraz wiersze z tabeli `tab_B`, dla których nie ma odpowiedników w tabeli `tab_A` (określona wartość `łącze_B` nie pojawia się w tabeli `tab_A`).

3. **FULL [outer] JOIN**.
W tym przypadku zwracane są wszystkie wiersze z obu tabel.

```
SELECT kol_A, kol_B
FROM tab_A FULL JOIN tab_B
ON tab_A.łącze_A = tab_B.łącze_B;
```

Wybrane zostaną wszystkie wiersze spełniające warunek `tab_A.łącze_A = tab_B.łącze_B` oraz wiersze z tabeli `tab_A`, dla których nie ma odpowiedników w tabeli `tab_B` (określona wartość `łącze_A` nie pojawia się w tabeli `tab_B`), oraz wiersze z tabeli `tab_B`, dla których nie ma odpowiedników w tabeli `tab_A` (określona wartość `łącze_B` nie pojawia się w tabeli `tab_A`).



RYSUNEK 2. Złączenia tabel

2. Zapytania grupujące

Zapytania grupujące stanowią istotny mechanizm języka SQL, umożliwiający analizę oraz agregację danych przechowywanych w relacyjnych bazach danych. Podczas gdy zapytania proste pozwalają na pobieranie pojedynczych rekordów oraz ich filtrowanie na podstawie określonych warunków, zapytania grupujące umożliwiają analizę zbiorów rekordów traktowanych jako logiczne całości oraz obliczanie wartości zbiorczych dla tych grup. Funkcjonalność ta ma fundamentalne znaczenie w praktycznych zastosowaniach baz danych, gdzie często zachodzi potrzeba obliczania sum, średnich, liczebności zbiorów lub innych miar statystycznych dla określonych kategorii danych.

W relacyjnych bazach danych informacje przechowywane są zazwyczaj w postaci szczegółowej, w której każdy wiersz reprezentuje pojedynczy obiekt, zdarzenie lub relację. W wielu przypadkach niezbędne jest jednak uzyskanie bardziej ogólnego obrazu danych, polegającego na ich podsumowaniu lub analizie według określonych kryteriów. Przykładowo może zachodzić potrzeba określenia liczby rekordów należących do danej kategorii, obliczenia średniej wartości określonego atrybutu w obrębie grupy lub wyznaczenia wartości minimalnej bądź maksymalnej dla określonych zbiorów danych. Język SQL umożliwia realizację takich operacji bezpośrednio w systemie zarządzania bazą danych, bez konieczności wykorzystywania zewnętrznych narzędzi przetwarzania danych.

Podstawowym mechanizmem służącym do grupowania danych w języku SQL jest klauzula `GROUP BY`, która umożliwia podział rekordów na grupy na podstawie wartości wybranych kolumn. Wszystkie rekordy posiadające identyczne wartości w kolumnach wskazanych w klauzuli `GROUP BY` są traktowane jako jedna grupa. Dla każdej z utworzonych grup możliwe jest zastosowanie **funkcji agregujących**, takich jak COUNT, SUM, AVG, `MIN` oraz MAX. Funkcje te umożliwiają obliczanie wartości zbiorczych dla całych grup rekordów, a nie dla pojedynczych wierszy. Dzięki temu możliwe jest uzyskanie informacji podsumowujących, takich jak liczba elementów w danej grupie, suma wartości określonego atrybutu lub średnia wartość w obrębie grupy.

Zastosowanie funkcji agregujących wprowadza istotne rozróżnienie pomiędzy operacjami wykonywanymi na poziomie pojedynczych rekordów a operacjami wykonywanymi na poziomie grup rekordów. W zapytaniach prostych przetwarzanie odbywa się na poziomie pojedynczych wierszy, natomiast w zapytaniach grupujących podstawową jednostką przetwarzania staje się grupa rekordów.

W języku SQL dostępne są dwie klauzule umożliwiające ograniczenie zbioru wynikowego: klauzula `WHERE` oraz klauzula `HAVING`. Pomimo podobnej funkcji klauzule

te działają na różnych etapach przetwarzania zapytania i odnoszą się do różnych poziomów przetwarzania danych.

Klauzula `WHERE` służy do filtrowania pojedynczych rekordów przed wykonaniem operacji grupowania. Oznacza to, że jedynie rekordy spełniające warunki określone w klauzuli `WHERE` są uwzględniane w dalszym procesie grupowania oraz w obliczeniach wykonywanych przez funkcje agregujące. Klauzula ta działa zatem na poziomie wierszy i pozwala ograniczyć zbiór danych wejściowych wykorzystywanych w operacji grupowania.

Klauzula `HAVING` służy natomiast do filtrowania grup rekordów po wykonaniu operacji grupowania oraz obliczeniu wartości agregujących. Klauzula `HAVING` umożliwia określenie warunków, które muszą spełniać całe grupy rekordów, aby zostały uwzględnione w wyniku zapytania. Dzięki temu możliwe jest na przykład ograniczenie wyników do grup zawierających określoną minimalną liczbę rekordów lub do grup, dla których wartość agregująca spełnia określone kryterium. Klauzula `HAVING` działa zatem na poziomie grup, a nie pojedynczych rekordów.

Zrozumienie różnicy pomiędzy klauzulami `WHERE` i `HAVING` ma kluczowe znaczenie dla poprawnego konstruowania zapytań grupujących. Klauzula `WHERE` określa, które rekordy zostaną uwzględnione w procesie grupowania, natomiast klauzula `HAVING`, które grupy zostaną uwzględnione w ostatecznym wyniku zapytania.

Przetwarzanie zapytania grupującego przebiega w określonej kolejności. W pierwszym etapie określane jest źródło danych w klauzuli `FROM`. Następnie klauzula `WHERE` filtruje rekordy na poziomie pojedynczych wierszy. Kolejnym etapem jest grupowanie rekordów przy użyciu klauzuli `GROUP BY`. Po utworzeniu grup wykonywane są obliczenia funkcji agregujących. Następnie klauzula `HAVING` filtruje grupy na podstawie określonych warunków. W dalszej kolejności klauzula `SELECT` określa strukturę wyniku zapytania, a klauzula `ORDER BY`, jeżeli została zastosowana, odpowiada za uporządkowanie wyników.

Możliwość wykonywania operacji grupowania i agregacji bezpośrednio w systemie zarządzania bazą danych zapewnia wysoką efektywność przetwarzania oraz umożliwia analizę dużych zbiorów danych w sposób wydajny i przejrzysty.

Po zapoznaniu się z treścią niniejszego rozdziału student będzie posiadał wiedzę i umiejętności umożliwiające samodzielne konstruowanie zapytań grupujących w języku SQL zgodnych ze standardem ANSI SQL oraz ich wykonywanie w środowisku Oracle. Student nauczy się stosować klauzulę `GROUP BY` do organizowania danych w logiczne grupy oraz wykorzystywać funkcje agregujące do obliczania wartości zbiorczych. Nabędzie również umiejętności rozróżniania oraz poprawnego stosowania klauzul `WHERE` i `HAVING` w zależności od tego, czy warunek selekcji dotyczy pojedynczych rekordów, czy całych grup.

Ponadto student zdobędzie umiejętności interpretowania wyników zapytań agregujących, konstruowania poprawnych i efektywnych zapytań grupujących oraz stosowania tych mechanizmów w praktycznych zadaniach z zakresu analizy danych. Student będzie również rozumiał logiczną kolejność przetwarzania zapytań grupujących oraz zależności pomiędzy poszczególnymi klauzulami języka SQL.

2.1. Zadania z rozwiązaniami i komentarzami

ZADANIE 1

Dla każdego stanowiska wyznaczyć średnie, minimalne i maksymalne zarobki. Wypisać stanowisko i wyznaczone wartości z odpowiednimi etykietami.

Rozwiązanie:

```
select stanowisko, min(pensja+nvl(premia,0)) as „minimalne zarobki”,
       round(avg(pensja+nvl(premia,0)), 2) as „srednie zarobki”,
       max(pensja+nvl(premia,0)) as „maksymalne zarobki”
from pracownik
group by stanowisko
order by stanowisko;
```

Poniżej pokazano błędne rozwiązanie:

```
select stanowisko, min(pensja+nvl(premia,0)) as „minimalne zarobki”,
       round(avg(pensja+nvl(premia,0)), 2) as „srednie zarobki”,
       max(pensja+nvl(premia,0)) as „maksymalne zarobki”
from pracownik;
```

Komentarz 1

Próba uruchomienia błędnego rozwiązania spowoduje wywołanie błędu składniowego, wskazującego, że powinno zostać zastosowane grupowanie. Należy zastanowić się, dlaczego.

Błędne rozwiązanie można rozbić na dwa zapytania:

[1]

```
select stanowisko
from pracownik;
```

Powyższe zapytanie zwróci tyle wierszy, ile jest wierszy w tabeli `pracownik`, czyli całą kolumnę `stanowisko`, na przykład:

```
Analityk
Księgowy
Analityk
Informatyk
Informatyk
```

[2]

```
select min(pensja+nvl(premia,0)) as „minimalne zarobki”,
       round(avg(pensja+nvl(premia,0)), 2) as „srednie zarobki”,
       max(pensja+nvl(premia,0)) as „maksymalne zarobki”
from pracownik
```

Powyższe zapytanie zwróci dokładnie jeden wiersz zawierający minimalne, średnie i maksymalne zarobki dla ogółu pracowników, na przykład:

```
3400  7300  9900
```

Gdyby możliwe było uruchomienie błędnego zapytania, wyniki wyglądałyby następująco:

```
Analityk      3400     7300     9900
Księgowy      3400     7300     9900
Analityk      3400     7300     9900
Informatyk    3400     7300     9900
Informatyk    3400     7300     9900
```

Na pierwszy rzut oka widać, że powyższe wyniki nie odpowiadają zadanej treści. Są błędne chociażby z tego powodu, że każdemu stanowisku przypisane zostały minimalne, średnie i maksymalne zarobki dla ogółu pracowników. Użytkownicy są więc chronieni przed popełnianiem takich błędów wywołaniem błędu składniowego.

Komentarz 2

Gdy początek zapytania ma formę: `select x, y, funkcja`, to jego interpretacja jest następująca: wybierz wartości atrybutów `x, y` oraz wartość funkcji `funkcja` dla każdej określonej kombinacji wartości atrybutów `x, y`. Aby to osiągnąć, trzeba wyodrębnić te określone kombinacje wartości, a można to osiągnąć z wykorzystaniem grupowania, czyli stosując `GROUP BY x,y` – powstanie tyle grup, ile jest różnych kombinacji wartości atrybutów `x, y`.

ZADANIE 2

Wypisać minimalną i maksymalną pensję w firmie.

```
select min(pensja) as minimalna, max(pensja) as maksymalna
from pracownik;
```

Komentarz 1

Grupowanie nie jest tutaj potrzebne, ponieważ wartości funkcji wyznaczane są dla całej tabeli `pracownik`.

ZADANIE 3

Wyznaczyć, ilu jest pracowników na stanowisku analityk.

```
select count(id_pracownika) as “liczba analitykow”
from pracownik
where upper(stanowisko) = 'ANALITYK';
```

ZADANIE 4

Wypisać nazwę departamentu oraz liczbę osób zatrudnionych w danym departamencie. Jeśli w jakimś departamencie nikt nie pracuje, jako liczba osób powinno pojawić się 0.

```
select nazwa, count(id_pracownika) as „liczba pracownikow”
from pracownik p right join departament d
      on p.nr_departamentu = d.nr_departamentu
group by nazwa, d.nr_departamentu;
```

Komentarz 1

Ze względu na konieczność uwzględnienia wszystkich departamentów użyto złączenia zewnętrznego `right join`.

Komentarz 2

W wyrażeniu `select` elementem nieagregującym jest nazwa departamentu. Zgodnie z zasadą nr 2 (2.2) grupowanie powinno mieć więc formę: `group by nazwa`. Do grupowania dodano jednak również `d.nr_departamentu`. Powodem jest nieunikalność nazw departamentów, co oznacza, że może być parę departamentów o takiej samej nazwie i oczywiście innych numerach (np. informatyka w Białymstoku, informatyka w Warszawie). Bez uwzględnienia `d.nr_departamentu` w grupowaniu możliwe byłoby „zlanie się” wielu departamentów o takiej samej nazwie w jedną grupę, co prowadziłoby do błędnych wyników/wniosków.

Komentarz 3

Należy zastanowić się, dlaczego w grupowaniu uwzględniono `d.nr_departamentu`, a nie `p.nr_departamentu`. W wyniku złączenia zewnętrznego w wynikach uwzględnione będą bowiem także te departamenty, w których nikt nie pracuje, co będzie skutkowało tym, że po złączeniu w kolumnie `p.nr_departamentu` pojawią się puste komórki/`NULL`. Grupowanie po `NULL`-ach jest dopuszczalne, ale sugeruje się grupowanie po atrybutach, ponieważ wtedy puste wartości się nie pojawiają.

Przykład 1:

```
select avg(pensja), id_kierownika
from pracownik
group by id_kierownika;
```

W kolumnie id_kierownika może wystąpić NULL – niektórzy pracownicy mogą nie mieć bezpośredniego kierownika. Wyniki mogą przedstawiać się następująco:

```
5000  (null)
1200  9
2500  6
1500  4
```

co sugeruje, że występuje pracownik o identyfikatorze NULL, a jego podwładni otrzymują średnią pensją równą 5000.

Komentarz 4

Argumentem funkcji count jest id_pracownika. Funkcja ta zwraca liczbę niepustych komórek w kolumnie zadanej jako parametr funkcji. Bezpiecznie jest więc podać jako argument kolumnę/atrybut, w której nie pojawiają się puste wartości, chyba że rozważane zagadnienie pozwala lub wręcz wymaga podania jako argumentu funkcji kolumny, w której mogą pojawić się puste wartości. Takim przykładem może być wyznaczenie liczby pracowników, którzy otrzymują premię:

```
select count(premia)
from pracownik;
```

ZADANIE 5

Wypisać nazwy departamentów wraz ze średnimi pensjami (nadać etykietę „srednia pensja”) w tych departamentach. Wyniki uporządkować od najwyższej średniej pensji do najniższej. Uwzględnić departamenty, w których nie ma pracowników.

```
select nazwa, round(avg(nvl(pensja, 0)),2) as „srednia pensja”
from departament d left join pracownik p
      on d.nr_departamentu = p.nr_departamentu
group by nazwa, d.nr_departamentu
order by round(avg(nvl(pensja, 0)),2) desc;
```

ZADANIE 6

Wyznaczyć nazwy departamentów, w których pracują więcej niż trzy osoby.

```
select nazwa
from departament d join pracownik p
      on d.nr_departamentu = p.nr_departamentu
```

```
group by nazwa, d.nr_departamentu
having count(id_pracownika) >= 3;
```

Komentarz 1

Klauzula `HAVING` jest odpowiednikiem klauzuli `WHERE`, tylko że stosowaną dla grup. Z wykorzystaniem klauzuli `WHERE` sprawdzane są warunki, które muszą być spełnione przez wiersze, aby zostały one uwzględnione w wyniku zapytania. Z wykorzystaniem klauzuli `HAVING` sprawdzane są natomiast warunki, które muszą być spełnione przez grupy, aby zostały one uwzględnione w wyniku zapytania.

Komentarz 2

W klauzuli `HAVING` można sprawdzać te wartości, które dla dowolnie wybranej grupy są wspólne – przyjmują jedną konkretną wartość. Sprawdzane więc mogą być wartości funkcji agregujących oraz atrybuty, po których grupowano. W zapytaniu grupowano po nazwie departamentu i numerze departamentu z tabeli `departament` – powstanie tyle grup, ile jest różnych kombinacji wartości tych atrybutów. Pewne więc jest, że wewnątrz jednej grupy każdy z tych atrybutów może przyjąć tylko jedną wartość.

ZADANIE 7

Wyznaczyć nazwy departamentów, w których średnia pensja jest wyższa niż 1500 i które mają lokalizację w Białymstoku albo Warszawie.

```
select nazwa
from departament d join pracownik p
      on p.nr_departamentu = d.nr_departamentu
where lower(lokalizacja) = 'bialystok'
      or lower(lokalizacja) = 'warszawa'
group by nazwa, d.nr_departamentu
having avg(pensja) > 1500
order by d.nr_departamentu;
```

ZADANIE 8

Wypisać nazwy departamentów, stanowiska oraz liczbę osób pracujących na danym stanowisku w danym departamencie.

```
select nazwa, stanowisko, count(id_pracownika) as „liczba pracownikow”
from departament d join pracownik p
      on d.nr_departamentu = p.nr_departamentu
```

```
group by nazwa, d.nr_departamentu, stanowisko
order by d.nr_departamentu;
```

ZADANIE 9

Wypisać identyfikatory i nazwiska pracowników oraz to, ilu mają podwładnych:

a) z uwzględnieniem tylko tych, którzy mają podwładnych:

```
select k.id_pracownika, k.nazwisko, count(p.id_pracownika) as
                                                    „liczba podwladnych”
from pracownik k join pracownik p
      on k.id_pracownika = p.id_kierownika
group by k.id_pracownika, k.nazwisko;
```

b) z uwzględnieniem tylko tych, którzy podwładnych nie mają (0 jest liczbą ich podwładnych):

```
select k.id_pracownika, k.nazwisko, count(p.id_pracownika) as
                                                    „liczba podwladnych”
from pracownik k left join pracownik p
      on k.id_pracownika = p.id_kierownika
group by k.id_pracownika, k.nazwisko
having count(p.id_pracownika) = 0;
```

c) z uwzględnieniem wszystkich pracowników:

```
select k.id_pracownika, k.nazwisko, count(p.id_pracownika) as
                                                    „liczba podwladnych”
from pracownik k left join pracownik p
      on k.id_pracownika = p.id_kierownika
group by k.id_pracownika, k.nazwisko;
```

ZADANIE 10

Dla każdego kierownika wyznaczyć średnią pensję jego podwładnych na określonym stanowisku. Wypisać nazwisko kierownika, stanowisko podwładnych oraz średnią pensję podwładnych danego kierownika na danym stanowisku.

```
select k.nazwisko, p.stanowisko, avg(p.pensja) as
                                                „srednia pensja podwladnych”
```

```
from pracownik k join pracownik p
        on k.id_pracownika = p.id_kierownika
group by k.id_pracownika, k.nazwisko, p.stanowisko;
```

ZADANIE 11

Wypisać poziomy zarobków oraz liczbę osób, które mają zarobki na danym poziomie.

```
select nr_przedzialu, count(id_pracownika)
from poziom_zarobkow left join pracownik
        on pensja+nvl(premia,0) between dolna_granica and gorna_granica
group by nr_przedzialu;
```

Komentarz 1

Użyto złączenia zewnętrznego `left join`, aby uwzględnić wszystkie wysokości zarobków – nawet te, których nikt nie otrzymuje.

ZADANIE 12

Wyznaczyć liczbę pracowników pracujących na stanowisku `'sprzedawca'`, którzy mają kierowników i którzy zostali zatrudnieni tego samego dnia. Wypisać datę zatrudnienia oraz liczbę pracowników tego dnia zatrudnionych. Wyniki uporządkować malejąco ze względu na datę zatrudnienia.

```
select data_zatrudnienia, count(id_pracownika)
from pracownik
where lower(stanowisko) = 'sprzedawca'
        and id_kierownika is not null
group by data_zatrudnienia
order by data_zatrudnienia desc;
```

Komentarz 1

Po pogrupowaniu dostępne są tylko te atrybuty, po których grupowano. W związku z tym wyniki można porządkować tylko po tych atrybutach.

ZADANIE 13

Wypisać nazwy projektów, liczbę osób, które pracowały nad danym projektem oraz to, ile sumarycznie wydano na realizację danego projektu.

```
select nazwa, count(id_pracownika) as „liczba realizujacych”,
      sum(nvl(liczba_godzin*stawka_za_godzine, 0)) as „wydano
      na realizacje”
from zlecenie z right join projekt p
      on z.nr_projektu = p.nr_projektu
group by nazwa, p.nr_projektu
order by p.nr_projektu;
```

ZADANIE 14

Wyznaczyć liczbę różnych stanowisk w każdym departamencie (podać też nazwę departamentu).

```
select nazwa, count(distinct stanowisko) as „liczba roznych stanowisk”
from departament d left join pracownik p
      on p.nr_departamentu = d.nr_departamentu
group by nazwa, d.nr_departamentu
order by d.nr_departamentu;
```

2.2. Zasada nr 2

Masz w `SELECT` ⇒ masz po `GROUP BY`.

```
select x, y, funkcja
from tabela
group by x, y;
```

Odwrotna implikacja nie jest prawdziwa: można grupować po czymś, co nie jest wymienione w `SELECT` [4].

Zasada ta jest wynikiem ograniczeń składniowych SQL, które to chronią przed popełnianiem poważnych błędów. Została ona wyjaśniona w części praktycznej tego rozdziału.

2.3. Wskazówki dotyczące składni SQL

1. Tworzenie grup.

Klauzula **GROUP BY** (nazwa lub nazwy kolumn po przecinkach).

Grupa – zbiór wierszy, dla których wartość jednej cechy (lub wielu cech – kombinacja wartości tych cech) jest taka sama. Wartości jakiego atrybutu (lub zbioru atrybutów) mają być wspólne dla całej grupy, specyfikuje się właśnie po `GROUP BY`.

2. Funkcje grupujące.

COUNT(nazwa kolumny lub *)

Liczy wystąpienia jakiejkolwiek wartości w danej kolumnie (tzn. liczbę wierszy, w których w danej kolumnie występuje jakaś wartość: nie `NULL`).

SUM(nazwa kolumny)

Wynikiem jest suma wartości w danej kolumnie.

AVG(nazwa kolumny)

Wynikiem jest średnia wartość wartości danej kolumny.

MIN(nazwa kolumny)

Wynikiem jest minimalna wartość wartości danej kolumny.

MAX(nazwa kolumny)

Wynikiem jest maksymalna wartość wartości danej kolumny.

3. Sprawdzanie warunków dla grup.

Klauzula `HAVING`

Przy użyciu klauzuli `HAVING` sprawdzane są warunki dla grup, tzn. warunki, jakie grupa wierszy musi spełnić, aby zostać uwzględniona w wynikach zapytania (np. liczność grupy >3). Warunki (podobnie jak w przypadku klauzuli `WHERE`) można łączyć spójnikami `AND` i `OR`.

4. **Aliasy tabel**

Aliasy skracają zapytanie, dzięki czemu staje się ono bardziej przejrzyste. Konieczne jest ich użycie, gdy w klauzuli `FROM` pojawiają się parokrotne odwołania do tej samej tabeli.

```
SELECT a.nr_przedzialu, b.nr_przedzialu
FROM poziom_zarobkow a, poziom_zarobkow b
WHERE a.dolna_granica < b.dolna_granica;
```

Zostaną stworzone tymczasowe nazwy tabeli `poziom_zarobkow`: `a` i `b`. Wypisane zostaną pary numerów przedziałów, z tym że pierwsza wartość będzie odpowiadała numerowi przedziału z mniejszą dolną granicą (wypisane zostaną wszystkie możliwe pary).

3. Zapytania zagnieżdżone/podzapytania

Podzapytania stanowią zaawansowany i niezwykle istotny mechanizm języka SQL, umożliwiający zagnieżdżanie jednego zapytania wewnątrz innego zapytania. Podzapytanie, określane również jako zapytanie zagnieżdżone lub zapytanie wewnętrzne, to pełne zapytanie SQL, którego wynik wykorzystywany jest przez zapytanie zewnętrzne do wyznaczenia ostatecznego rezultatu. Mechanizm ten znacząco rozszerza możliwości języka SQL, umożliwiając formułowanie złożonych warunków selekcji oraz wyrażanie zależności logicznych bez konieczności stosowania wielu oddzielnych zapytań lub technik programowania proceduralnego.

W niniejszym rozdziale podzapytania wykorzystywane będą przede wszystkim w klauzulach `WHERE` oraz `HAVING`, w których pełnią funkcję warunków selekcji. Klauzula `WHERE` działa na poziomie pojedynczych rekordów i służy do filtrowania danych przed wykonaniem operacji grupowania. W przypadku zastosowania podzapytania w klauzuli `WHERE` jego wynik wykorzystywany jest do określenia, czy dany rekord spełnia warunki selekcji. Pozwala to na dynamiczne określanie kryteriów filtrowania na podstawie danych przechowywanych w bazie.

Klauzula `HAVING` działa natomiast na poziomie grup rekordów i stosowana jest po wykonaniu operacji grupowania oraz obliczeniu wartości funkcji agregujących. Podzapytania wykorzystywane w klauzuli `HAVING` umożliwiają formułowanie warunków selekcji odnoszących się do całych grup, na przykład poprzez porównywanie wartości agregujących z wartościami wyznaczonymi przez inne zapytania. Rozróżnienie pomiędzy działaniem klauzul `WHERE` i `HAVING` ma kluczowe znaczenie dla poprawnego stosowania podzapytań, ponieważ klauzula `WHERE` odnosi się do pojedynczych rekordów, natomiast klauzula `HAVING` do całych grup rekordów.

Jednym z podstawowych operatorów wykorzystywanych w połączeniu z podzapytaniami jest słowo kluczowe `IN`, które umożliwia sprawdzenie, czy określona wartość należy do zbioru wartości zwróconych przez podzapytanie. Operator `IN` przyjmuje wartość logiczną „prawda", jeżeli porównywana wartość jest równa dowolnej wartości zwróconej przez podzapytanie. Operator `NOT IN` stanowi jego negację i umożliwia wykluczenie wartości należących do zbioru wynikowego podzapytania. Ponadto język SQL udostępnia operatory `ANY` oraz `ALL`, które umożliwiają porównanie wartości z dowolnym elementem zbioru wynikowego podzapytania lub ze wszystkimi elementami tego zbioru. Operatory te zapewniają dużą elastyczność w konstruowaniu warunków selekcji opartych na zbiorach wartości.

Istotnym mechanizmem związanym z podzapytaniami jest również zastosowanie słowa kluczowego `EXISTS` umożliwiającego sprawdzenie, czy podzapytanie zwraca jakiekolwiek rekordy. Operator `EXISTS` przyjmuje wartość logiczną „prawda", jeżeli podzapytanie zwróci co najmniej jeden rekord, niezależnie od jego zawartości. Operator `NOT EXISTS` umożliwia natomiast wybór rekordów, dla których podzapytanie nie zwraca żadnych wyników. W przeciwieństwie do operatora `IN`, który porównuje konkretne wartości, operator `EXISTS` sprawdza jedynie istnienie rekordów spełniających określone warunki.

Z punktu widzenia wydajności istnieją istotne różnice pomiędzy operatorami `IN` i `EXISTS`. Operator `IN` jest zazwyczaj stosowany w przypadkach, gdy podzapytanie zwraca stosunkowo niewielki zbiór wartości i gdy zachodzi potrzeba bezpośredniego porównania wartości. Operator `EXISTS` jest natomiast często bardziej efektywny w przypadku dużych zbiorów danych lub w sytuacjach, gdy konieczne jest jedynie sprawdzenie istnienia rekordów spełniających określony warunek. Wynika to z faktu, że system zarządzania bazą danych może zakończyć przetwarzanie podzapytania natychmiast po znalezieniu pierwszego pasującego rekordu. Współczesne systemy zarządzania bazami danych, w tym system Oracle, stosują mechanizmy optymalizacji, które w wielu przypadkach umożliwiają efektywne przetwarzanie obu konstrukcji, jednak zrozumienie różnic pomiędzy nimi ma istotne znaczenie dla projektowania wydajnych zapytań.

Podzapytania można podzielić na dwie podstawowe kategorie: **podzapytania nieskorelowane** oraz **podzapytania skorelowane** [5]. Podzapytanie nieskorelowane jest niezależne od zapytania zewnętrznego i może zostać wykonane jednokrotnie, a jego wynik wykorzystywany jest przez zapytanie zewnętrzne. Podzapytanie skorelowane odwołuje się natomiast do wartości z zapytania zewnętrznego i jest wykonywane wielokrotnie, osobno dla każdego rekordu przetwarzanego przez zapytanie zewnętrzne. Podzapytania skorelowane umożliwiają formułowanie bardziej złożonych i dynamicznych warunków selekcji, jednak mogą wiązać się z większym kosztem obliczeniowym.

Należy wiedzieć, że podzapytania nieskorelowane są wykonywane przed zapytaniem zewnętrznym, natomiast podzapytania skorelowane są wykonywane wielokrotnie w trakcie przetwarzania zapytania zewnętrznego. Zrozumienie tej różnicy ma istotne znaczenie dla poprawnej interpretacji działania zapytań oraz oceny ich wydajności.

Po zapoznaniu się z treścią niniejszego rozdziału student zdobędzie wiedzę i umiejętności umożliwiające samodzielne konstruowanie podzapytań oraz ich wykorzystywanie w klauzulach `WHERE` i `HAVING`. Student nauczy się stosować operatory `IN`, `NOT IN`, `ANY` oraz `ALL` do porównywania wartości ze zbiorami wynikowymi podzapytań, a także operatory `EXISTS` i `NOT EXISTS` do sprawdzania istnienia określonych rekordów.

Ponadto student nabędzie umiejętność rozróżniania podzapytań skorelowanych i nieskorelowanych, zrozumie ich sposób działania oraz wpływ na wydajność zapytań. Student nauczy się integrować podzapytania z zapytaniami głównymi, interpretować ich wyniki oraz wykorzystywać je w rozwiązywaniu złożonych problemów związanych z wyszukiwaniem i analizą danych.

Przykłady dwóch głównych typów podzapytań

1. **Podzapytania skorelowane:** podzapytania, które odwołują się do kolumn z zewnętrznego zapytania. Działają one w kontekście każdego wiersza zewnętrznego zapytania, co oznacza, że są wykonywane wielokrotnie, raz dla każdego wiersza. Umożliwiają dynamiczne porównanie wartości w obrębie danego kontekstu.

Przykład:

```
SELECT nazwisko
FROM pracownik p
WHERE pensja > (SELECT avg(pensja)
                FROM pracownik
                WHERE nr_departamentu = p.nr_departamentu);
```

Zwrócone zostaną nazwiska pracowników, których pensja jest wyższa od średniej pensji w departamencie, w którym zatrudniony jest dany pracownik.

2. **Podzapytania nieskorelowane:** podzapytania, które mogą być wykonywane niezależnie od głównego zapytania.

Przykład:

```
SELECT nazwisko
FROM pracownik
WHERE nr_departamentu IN (SELECT nr_departamentu
                          FROM departament
                          WHERE nazwa = 'informatyka');
```

Zwrócone zostaną nazwiska pracowników, którzy pracują w departamencie o nazwie `'informatyka'`.

3.1. Zadania z rozwiązaniami i komentarzami

ZADANIE 1

Wypisać nazwiska osób pracujących w tym samym departamencie, co pracownik o nazwisku `'Nazwisko2'`. Podać też nazwę departamentu.

Błędne rozwiązanie:

```
select id_pracownika, nazwisko
from pracownik
where nr_departamentu = ( select nr_departamentu
                          from pracownik
                          where lower(nazwisko) = 'nazwisko2' );
```

Komentarz 1

Należy rozważyć zasadność użycia podzapytania. W każdym zapytaniu wykorzystującym klauzulę `WHERE` analiza warunków zapisanych w `WHERE` odbywa się sekwencyjnie, wiersz po wierszu. W `WHERE` zakres widzenia to tylko jeden wiersz – ten, który aktualnie jest analizowany. Niewidoczny jest więc departament pracownika `'Nazwisko2'` – potrzebny byłby zakres widzenia całej tabeli `pracownik`, co zapewnia podzapytanie.

Komentarz 2

Przy próbie wywołania powyższego zapytania wywoływany jest błąd składniowy: jednowierszowe zapytanie zwraca więcej niż jeden wiersz. Problemem jest tutaj podzapytanie, które zwróci tyle wierszy, ilu pracowników ma nazwisko `'Nazwisko2'`. Stosowanie znaków <, >, =, != przed podzapytaniem jest dozwolone wyłącznie w sytuacji, gdy podzapytanie po nim następujące zwraca jeden wiersz.

Rozwiązanie:

```
select id_pracownika, nazwisko
from pracownik
where nr_departamentu in ( select nr_departamentu
                           from pracownik
                           where lower(nazwisko) = 'nazwisko2' );
```

Komentarz 3

Zamiast `IN` można zastosować na przykład `=ANY` (równe co najmniej 1).

ZADANIE 2

Podać listę pracowników, którzy zarabiają więcej niż pracownik o nazwisku `‘Nazwisko3’` i mniej niż pracownik o nazwisku `‘Nazwisko7’`.

```
select id_pracownika, nazwisko, pensja
from pracownik
```

```
where pensja+nvl(premia,0) > ( select max(pensja+nvl(premia,0))
                          from pracownik
                          where lower(nazwisko) = 'nazwisko3')
and pensja+nvl(premia,0) < ( select min(pensja+nvl(premia,0))
                          from pracownik
                          where lower(nazwisko) = 'nazwisko7');
```

Komentarz 1

… > (select max … znaczy: zarobki są większe od zarobków wszystkich pracowników o nazwisku Nazwisko3. Taki sam efekt ma zastosowanie >all (select pensja+nvl(premia,0)) …).

Komentarz 2

… < (select min … znaczy: zarobki są mniejsze od zarobków wszystkich pracowników o nazwisku Nazwisko7. Taki sam efekt ma zastosowanie <all (select pensja+nvl(premia,0)) …).

ZADANIE 3

Podać nazwisko i identyfikator najdłużej pracującego pracownika.

Rozwiązanie błędne:

```
select id_pracownika, nazwisko, data_zatrudnienia
from pracownik
where data_zatrudnienia < all ( select data_zatrudnienia
                          from pracownik );
```

Komentarz 1

W powyższym rozwiązaniu występuje błąd logiczny polegający na szukaniu daty zatrudnienia, która jest mniejsza od wszystkich dat zatrudnienia występujących w tabeli pracownik. Sprawdzane jest więc też, czy zadana data jest mniejsza od samej siebie, co nigdy nie zostanie spełnione.

Rozwiązanie:

```
select id_pracownika, nazwisko, data_zatrudnienia
from pracownik
where data_zatrudnienia = ( select min(data_zatrudnienia)
                          from pracownik );
```

ZADANIE 4

Wypisać nazwiska osób zarabiających więcej niż wynosi średnia pensja w firmie.

```
select id_pracownika, nazwisko, pensja+nvl(premia,0) as zarobki
from pracownik
where pensja+nvl(premia,0) > ( select avg(pensja)
                               from pracownik );
```

ZADANIE 5

Wypisać nazwy departamentów, w których zatrudniony jest co najmniej jeden analityk, a w sumie pracują w nim co najmniej trzy osoby.

Rozwiązanie błędne:

```
select nazwa
from departament d join pracownik p
      on d.nr_departamentu = p.nr_departamentu
where lower(stanowisko) = 'analityk'
group by nazwa, d.nr_departamentu
having count(id_pracownika) >= 3;
```

Komentarz 1

Rozwiązanie jest jak najbardziej poprawne składniowo. Jest jednak błędne, bo nie realizuje zadanego polecenia. W rozwiązaniu tym, w klauzuli `WHERE`, ograniczono pracowników tylko do tych, którzy pracują na stanowisku analityk. W konsekwencji po pogrupowaniu liczeni są tylko analitycy. Rozwiązanie to byłoby odpowiedzią na zadanie: „podaj nazwy departamentów, w których pracuje co najmniej trzech analityków". Trzeba więc sprawdzać cały departament, a nie jego pojedynczych pracowników.

Rozwiązanie:

```
select nazwa
from departament d join pracownik p
      on d.nr_departamentu = p.nr_departamentu
where d.nr_departamentu in ( select nr_departamentu
                             from pracownik
                             where lower(stanowisko) = 'analityk' )
group by nazwa, d.nr_departamentu
having count(id_pracownika) >= 3;
```

ZADANIE 6

Wypisać nazwy projektów, nad którymi pracował co najmniej jeden kierownik (pracownik mający podwładnych).

```
select nazwa
from projekt
where nr_projektu in ( select nr_projektu
                       from zlecenie
                       where id_pracownika in ( select id_kierownika
                                                from pracownik ) );
```

Komentarz 1

Kierownik to pracownik, który ma podwładnych, co oznacza, że jego identyfikator pojawia się dla jakiegoś pracownika w kolumnie `id_kierownika`.

ZADANIE 7

Wypisać nazwy projektów, na realizację których przeznaczono więcej, niż faktycznie wydano w tym celu.

Rozwiązanie 1:

```
select nazwa
from projekt p left join zlecenie z
      on p.nr_projektu = z.nr_projektu
group by p.nr_projektu, nazwa, budzet
having budzet > sum(nvl(liczba_godzin*stawka_za_godzine, 0));
```

Komentarz 1

Użyto złączenia zewnętrznego `left join`, aby w wynikach uwzględnić wszystkie projekty, nawet te nierealizowane.

Komentarz 2

Grupowanie po atrybucie `budzet` jest konieczne, jeśli jego wartość będzie sprawdzana w klauzuli `HAVING`. Jest to grupowanie „nadmiarowe", wiadomo bowiem, że po pogrupowaniu po numerze projektu z tabeli `projekt` każda dowolnie wybrana grupa będzie skupiać w sobie wiersze związane z jednym konkretnym projektem, jednym konkretnym numerem projektu tabeli `p`.

Komentarz 3

Wykorzystano funkcję `nvl` w celu zamiany potencjalnego `NULL`-a na 0. Inaczej nie byłoby możliwe porównanie budżetu z wydatkami, jeśli byłyby one `NULL`-em. Taka sytuacja wystąpi, gdy porównywany będzie budżet projektu, który nie był realizowany.

Rozwiązanie 2:

```
select nazwa
from projekt p
where budzet > ( select sum(nvl(liczba_godzin*stawka_za_godzine, 0))
            from zlecenie
            where nr_projektu = p.nr_projektu );
```

ZADANIE 8

Podać listę tych stanowisk, dla których średnia wartość zarobków jest wyższa niż średnia ogólna.

```
select stanowisko
from pracownik
group by stanowisko
having avg(pensja+nvl(premia,0)) > ( select avg(pensja+nvl(premia,0))
                                    from pracownik );
```

ZADANIE 9

Podać identyfikatory, nazwiska i zarobki pracowników, którzy zarabiają w przedziale `<1.1*MIN; 0.9*MAX>`, gdzie `MIN` i `MAX` to odpowiednio minimalne i maksymalne zarobki w firmie.

```
select id_pracownika, nazwisko, pensja+nvl(premia,0) as zarobki
from pracownik
where pensja+nvl(premia,0) between 1.1*( select min(pensja+nvl(premia,0))
                                        from pracownik )
      and 0.9*( select max(pensja+nvl(premia,0))
               from pracownik );
```

ZADANIE 10

Podać listę pracowników, których pensja jest wyższa niż pensja jakiegokolwiek (każdego) analityka.

```
select id_pracownika, nazwisko, stanowisko, pensja
from pracownik
where pensja >all ( select pensja
                    from pracownik
                    where upper(stanowisko) = 'ANALITYK' );
```

ZADANIE 11

Podać nazwę departamentu, w którym pracuje największa liczba pracowników.

Rozwiązanie 1:

```
select nazwa
from departament d join pracownik p
      on d.nr_departamentu = p.nr_departamentu
group by nazwa, d.nr_departamentu
having count(id_pracownika) = ( select max(count(id_pracownika))
                                from pracownik
                                group by nr_departamentu );
```

Komentarz 1

W podzapytaniu wykorzystano zagnieżdżanie funkcji grupujących. W pokazanym przypadku dla każdego departamentu najpierw zostanie wyznaczona wartość funkcji wewnętrznej `count(id_pracownika)`. Wynikiem będą liczby reprezentujące liczbę pracowników w departamentach, na przykład {4, 2, 6, 5, 4}. W bazie jest zatem pięć departamentów, w których ktoś pracuje (w bazie mogą istnieć departamenty z zerową obsadą). Następnie z wyznaczonego zbioru liczb wybrana zostanie ta największa.

Rozwiązanie 2:

```
select nazwa
from departament d join  pracownik p
      on d.nr_departamentu = p.nr_departamentu
group by nazwa, d.nr_departamentu
having count(id_pracownika) >= all ( select count(id_pracownika)
```

```
                    from pracownik
                    group by nr_departamentu );
```

ZADANIE 12

Podać nazwiska i pensje pracowników zarabiających najwięcej w każdym departamencie. Podać też nazwę departamentu.

```
select id_pracownika, nazwisko, pensja+nvl(premia,0), nazwa
from pracownik p join departament d
      on p.nr_departamentu = d.nr_departamentu
where pensja+nvl(premia,0) = ( select max(pensja+nvl(premia,0))
                    from pracownik
                    where nr_departamentu = p.nr_departamentu );
```

Komentarz 1

W rozwiązaniu wykorzystano korelację zapytań poprzez porównanie: `where nr_departamentu = p.nr_departamentu`. Bez tego warunku w podzapytaniu wyznaczone zostałyby maksymalne zarobki spośród pracowników. Wprowadzając ten warunek ograniczono w podzapytaniu wiersze z tabeli `pracownik` do tych pracowników, którzy pracują w departamencie aktualnie analizowanym w głównym `WHERE`. Pamiętając, że w klauzuli `WHERE` analizowany jest sekwencyjnie każdy wiersz z tabel(i) określonej(ych) po `FROM` oraz o tym, że w `WHERE` definiowane są warunki, które te wiersze muszą spełniać, aby mogły wejść w zbiór wynikowych wierszy, a także o tym, że w klauzuli `WHERE` dostępne są całe wiersze (wszystkie kolumny/atrybuty), w podzapytaniu można odwołać się na zewnątrz, właśnie do głównego `WHERE`, bo na tym poziomie skonstruowane zostało podzapytanie. Reasumując, `p.nr_departamentu` w podzapytaniu wskazuje na aktualnie analizowany w głównym `WHERE` numer departamentu z tabeli `p`.

Komentarz 2

Podzapytanie będzie wyznaczało maksymalne zarobki dla każdego wiersza wynikającego z połączenia tabel `pracownik` i `departament`, czyli dla każdego pracownika.

Komentarz 3

W podzapytaniu warunek korelujący podzapytanie z zapytaniem głównym ma postać: `where nr_departamentu = p.nr_departamentu`. Człon `nr_departamentu` odnosi się do najbliższej tabeli, z której może pobrać ten numer. Jest nią, jak wiadomo, tabela `pracownik` z podzapytania. Człon `p.nr_departamentu` odnosi się do najbliższej tabeli o nazwie `p`, z której można pobrać numer departamentu. Jest nią oczywiście tabela `pracownik p` z zapytania głównego.

ZADANIE 13

Wypisać nazwy departamentów, w których nikt nie otrzymuje ani najniższej, ani najwyższej pensji (ogólnej).

Rozwiązanie błędne:

```
select nazwa
from pracownik p join departament d
      on p.nr_departamentu = d.nr_departamentu
where pensja > ( select min(pensja) from pracownik )
      and pensja < ( select max(pensja) from pracownik );
```

Komentarz 1

Nie uwzględniono departamentów, w których nie ma pracowników, chociaż w tych departamentach faktycznie żaden pracownik nie otrzymuje ani minimalnej, ani maksymalnej pensji.

Komentarz 2

Konsekwencją tak określonego `FROM` jest to, że w klauzuli `WHERE` sprawdzonych zostanie tyle wierszy, ilu jest pracowników w bazie, więc dla każdego pracownika sprawdzone zostaną zdefiniowane w `WHERE` warunki. Dowolnie wybrany departament zatrudniający na przykład trzech pracowników, z których dwóch nie ma ani minimalnej, ani maksymalnej pensji, a trzeci pracownik ma pensję minimalną, nie powinien zostać uwzględniony w wynikach. Przy tak skonstruowanym zapytaniu uwzględniona będzie jednak sytuacja, gdy: dla dwóch pracowników warunki są spełnione ⇒ departament „zatwierdzony". Sprawdzać należy warunki dla jednostki „obejmującej pracowników" – w tym przypadku dla departamentu. Oznacza to, że cały departament powinien spełniać warunki, a nie tylko pojedyncze wiersze z tego departamentu.

Rozwiązanie 1:

```
select nazwa
from pracownik p right join departament d
      on p.nr_departamentu = d.nr_departamentu
group by d.nr_departamentu, nazwa
having min(pensja) > ( select min(pensja) from pracownik )
      and max(pensja) < ( select max(pensja) from pracownik );
```

Rozwiązanie 2:

```
select nazwa
from departament
where nr_departament not in
      ( select nr_departamentu
      from pracownik
      where pensja = ( select min(pensja) from pracownik )
            or pensja = ( select max(pensja) from pracownik ) );
```

Rozwiązanie 3:

```
select nazwa
from departament
where not exists
      ( select 1
      from pracownik
      where nr_departamentu = departament.nr_departamentu
            and ( pensja = ( select min(pensja) from pracownik )
                  or pensja = ( select max(pensja) from pracownik ) ) );
```

Komentarz 3

Wykorzystano składnię `not exists ( select 1 … )`. Po słowach kluczowych `EXISTS/NOT EXISTS` następuje podzapytanie. W przypadku pierwszego słowa kluczowego warunek jest spełniony, gdy podzapytanie zwróci co najmniej jeden wiersz. W przypadku drugiego słowa kluczowego warunek jest natomiast spełniony, gdy podzapytanie nie zwróci ani jednego wiersza. Nie ma tak naprawdę znaczenia, co podzapytania będą zwracać: liczby, tekst itp. Wykorzystano więc składnię `select 1`.

Komentarz 4

W rozwiązaniu użyto `NOT EXISTS`, a nie `NOT IN`. Jest to spowodowane tym, że wykorzystanie `EXISTS/NOT EXISTS` zamiast `IN/NOT IN` jest bardziej wydajne. Wynika to z faktu, że w przypadku `IN/NOT IN` podzapytanie musi wykonać się w całości, tzn. przejść co najmniej tyle razy, ile wierszy w tabeli(ach) jest użytych w podzapytaniu. W przypadku `EXISTS/NOT EXISTS` nie ma to miejsca – podzapytanie kończy swoje działanie w momencie znalezienia potwierdzenia istnienia co najmniej jednego wiersza wynikowego dla podzapytania.

Przykład 1:

```
select 3, 'abc', sqrt(25)
from pracownik;
```

Zwróconych zostanie tyle wierszy postaci `3 abc 5`, ile jest wierszy w tabeli `pracownik`.

ZADANIE 14

Wypisać nazwiska osób zarabiających więcej niż co najmniej jeden kierownik.

Rozwiązanie 1:

```
select id_pracownika, nazwisko, stanowisko, pensja+nvl(premia,0) as zarobki
from pracownik
where pensja+nvl(premia,0) > ( select min(k.pensja+nvl(k.premia,0))
                               from pracownik p join pracownik k
                               on p.id_kierownika = k.id_pracownika );
```

Rozwiązanie 2:

```
select id_pracownika, nazwisko, stanowisko, pensja+nvl(premia,0) as zarobki
from pracownik
where pensja+nvl(premia,0) > ( select min(pensja+nvl(premia,0))
                               from pracownik
                               where id_pracownika in ( select id_kierownika
                                                        from pracownik ) );
```

ZADANIE 15

Wypisać nazwiska podwładnych oraz kierowników pod warunkiem, że zarówno podwładny, jak i kierownik pracowali nad jakimś projektem, ale nigdy nad tym samym.

```
select p.nazwisko as podwladny, k.nazwisko as kierownik
from pracownik p join pracownik k
      on p.id_kierownika = k.id_pracownika
where exists ( select 1
               from zlecenie
               where id_pracownika = p.id_pracownika )
```

```
    and exists ( select id_pracownika
          from zlecenie
          where id_pracownika = k.id_pracownika )
    and not exists ( select 1
          from zlecenie z
          where id_pracownika = p.id_pracownika
                and exists ( select 1
                      from zlecenie
                      where id_pracownika = k.id_pracownika
                            and nr_projektu = z.nr_projektu ) );
```

ZADANIE 16

Wypisać nazwiska pracowników, którzy za realizację żadnego projektu (dowolnego realizowanego przez daną osobę) nie dostali najwyższego wynagrodzenia.

```
select id_pracownika, nazwisko
from pracownik
where exists ( select 1
          from zlecenie
          where id_pracownika = pracownik.id_pracownika )
    and not exists ( select 1
                from zlecenie z1
                where id_pracownika = pracownik.id_pracownika
                      and liczba_godzin*stawka_za_godzine =
                            ( select max(liczba_godzin*stawka_za_godzine)
                            from zlecenie
                            where nr_projektu = z1.nr_projektu ) );
```

ZADANIE 17

Wypisać nazwiska oraz pensje tych osób, które nie są kierownikami (nie mają podwładnych).

Rozwiązanie 1:

```
select nazwisko, pensja
from pracownik p
```

```
where not exists ( select 1
                   from pracownik
                   where p.id_pracownika = id_kierownika );
```

Rozwiązanie 2 („trochę błędne”):

```
select nazwisko, pensja
from pracownik
where id_pracownika not in ( select id_kierownika
                             from pracownik );
```

Komentarz 1

Rozwiązanie nr 2 jest nie do końca poprawne: bez względu na zawartość tabeli `pracownik`, jeśli w kolumnie `id_kierownika` pojawi się co najmniej jedna pusta komórka, zapytanie to nie zwróci ani jednego wiersza. Powodem jest niemożność porównania `id_pracownika` (z klauzuli `WHERE`) ze zwracaną wartością `NULL` – nie jest możliwe stwierdzenie, czy `id_pracownika` jest równe, czy różne od `NULL`. W przypadku użycia `NOT IN`, gdy w wynikach podzapytania po nim następującego może pojawić się `NULL`, należy w podzapytaniu możliwość tę wyeliminować.

Rozwiązanie 2 (poprawne):

```
select nazwisko, pensja
from pracownik
where id_pracownika not in ( select id_kierownika
                             from pracownik
where id_kierownika is not null );
```

ZADANIE 18

Wypisać nazwiska osób zarabiających więcej niż co najmniej trzech pracowników, z założeniem, że każdy pracownik zarabiający mniej od danego realizował jakiś projekt.

```
select nazwisko
from pracownik p1
where 3 <= ( select count(id_pracownika)
             from pracownik
             where pensja+nvl(premia,0) < p1.pensja+nvl(p1.premia,0) )
      and not exists ( select 4
```

```
from pracownik p2
where pensja+nvl(premia,0) < p1.pensja+nvl(p1.premia,0)
      and not exists ( select 3
            from zlecenie
            where id_pracownika = p2.id_pracownika ) );
```

ZADANIE 19

Wypisać nazwy departamentów, w których pracuje tylu kierowników, ile różnych poziomów zarobków występuje w tym departamencie.

```
select nr_departamentu, nazwa
from departament d
where ( select count(id_pracownika)
      from pracownik p
      where nr_departamentu = d.nr_departamentu
            and exists ( select 3
                  from pracownik
                  where id_pracownika = p.id_kierownika ) ) =
      ( select count(distinct nr_przedzialu)
      from pracownik join poziom_zarobkow
            on pensja+nvl(premia,0) between dolna_granica and gorna_granica
      where nr_departamentu = d.nr_departamentu );
```

ZADANIE 20

Wypisać nazwy departamentów, w których nie pracuje żaden kierownik (osoba mająca podwładnych).

Rozwiązanie 1:

```
select nazwa
from departament
where nr_departamentu not in ( select nr_departamentu
                  from pracownik
                  where id_pracownika in ( select id_kierownika
                        from pracownik ) );
```

Rozwiązanie 2:

```
select nr_departamentu, nazwa
from departament d
where not exists ( select 3
                   from pracownik p
                   where nr_departamentu = d.nr_departamentu
                         and exists ( select 3
                                      from pracownik
                                      where id_kierownika = p.id_pracownika ) );
```

ZADANIE 21

Wypisać nazwy departamentów, w których pracuje najwięcej osób o najwyższym poziomie zarobków (najwyższym poziomie pojawiającym się u pracowników, a nie najwyższym poziomie „teoretycznym" – występującym w tabeli `poziom_zarobkow`).

```
select nazwa
from departament d
where ( select count(id_pracownika)
        from pracownik join poziom_zarobkow
             on pensja+nvl(premia,0) between dolna_granica and gorna_granica
        where nr_departamentu = d.nr_departamentu
             and dolna_granica = ( select max(dolna_granica)
                                   from pracownik join poziom_zarobkow
             on pensja+nvl(premia,0) between dolna_granica and gorna_granica ) )
        >= all ( select count(id_pracownika)
                 from pracownik join poziom_zarobkow
             on pensja+nvl(premia,0) between dolna_granica and gorna_granica
        where dolna_granica = ( select max(dolna_granica)
                                from pracownik join poziom_zarobkow
             on pensja+nvl(premia,0) between dolna_granica and gorna_granica )
        group by nr_departamentu );
```

Komentarz 1

W celu wyszukania najwyższego poziomu zarobków użyto atrybutu `dolna_granica`. Intuicyjne wydawałoby się użycie `max(nr_przedzialu)`, nie wiadomo jednak, jak numery przedziałów są reprezentowane w bazie: czy są to kolejne numery, czy też losowo wygenerowane identyfikatory.

ZADANIE 22

Wypisać średnie zarobki ludzi zajmujących to samo stanowisko. Podać liczbę osób oraz stanowisko. Wypisywać tylko wtedy, gdy w obrębie jednej grupy znajdują się:

a) co najmniej jeden pracownik i jego szef (są na tym samym stanowisku):

```
select stanowisko, avg(pensja+nvl(premia,0)) as „srednie zarobki”
from pracownik p1
where exists ( select 2
      from pracownik p2
      where stanowisko = p1.stanowisko
            and exists ( select 3
                  from pracownik
                  where stanowisko = p1.stanowisko
                        and id_pracownika = p2.id_kierownika ) );
```

b) dokładnie dwie takie pary:

```
select stanowisko, avg(pensja+nvl(premia,0)) as „srednie zarobki”
from pracownik p1
where ( select count(id_pracownika)
      from pracownik p2
      where stanowisko = p1.stanowisko
            and exists ( select 3
                  from pracownik
                  where stanowisko = p1.stanowisko
                        and id_pracownika = p2.id_kierownika ) ) = 2;
```

ZADANIE 23

Wypisać nazwiska osób zarabiających najmniej w najmniej licznym departamencie.

```
select nazwisko
from pracownik p
where pensja+nvl(premia,0) = ( select min(pensja+nvl(premia,0))
                        from pracownik
                        where nr_departamentu = p.nr_departamentu )
      and ( select count(id_pracownika)
            from pracownik
```

```
        where nr_departamentu = p.nr_departamentu ) =
    ( select min(count(id_pracownika))
    from pracownik
    group by nr_departamentu );
```

ZADANIE 24

Dla każdego departamentu podać jego nazwę oraz stanowiska, na których pracuje najwięcej ludzi w tym departamencie.

```
select d.nr_departamentu, max(nazwa) as nazwa, stanowisko
from departament d join pracownik p
    on d.nr_departamentu = p.nr_departamentu
group by d.nr_departamentu, stanowisko
having count(id_pracownika ) = ( select max(count(id_pracownika))
                                 from pracownik
                                 where nr_departamentu = d.nr_departamentu
                                 group by stanowisko );
```

Komentarz 1

Zapis `max(nazwa)` w klauzuli `SELECT` pozwala na uniknięcie grupowania po nazwie departamentu. Jest to możliwe dzięki wykorzystaniu właściwości grupowania: grupując po numerze departamentu użytkownik jest pewien, że wewnątrz dowolnie wybranej grupy znajdują się wiersze z dokładnie jednego departamentu, a w związku z tym wewnątrz tej grupy wszystkie wiersze mają taką samą nazwę.

Komentarz 2

Zamiana wewnątrz podzapytania warunku `where nr_departamentu = d.nr_departamentu` na warunek postaci `where nr_departamentu = p.nr_departamentu` spowodowałaby wywołanie błędu składniowego wskazującego na błędne grupowanie. Będąc „wewnątrz" podzapytania, użytkownik jest na poziomie `HAVING` zapytania głównego, a więc po grupowaniu. Grupowanie odbywa się po dwóch atrybutach, `d.nr_departamentu` i `stanowisko`, i tylko one są dostępne w `HAVING`. Odwołanie się do `p.nr_departamentu` w podzapytaniu jest więc niemożliwe.

ZADANIE 25

Wypisać nazwy projektów, nad którymi wszystkie osoby pracowały tyle samo godzin (np. trzy osoby, z których każda pracowała piętnaście godzin).

```
select nazwa
from projekt p
where ( select count(distinct liczba_godzin)
      from zlecenie
      where nr_projektu = p.nr_projektu ) = 1;
```

ZADANIE 26

Wypisać nazwiska osób mających co najmniej czterech podwładnych, przy czym każdy z nich pracuje na innym stanowisku.

```
select nazwisko
from pracownik p
where ( select count(id_pracownika)
      from pracownik
      where id_kierownika = p.id_pracownika ) >= 4
and ( select count(distinct stanowisko)
      from pracownik
      where id_kierownika = p.id_pracownika ) =
      ( select count(id_pracownika)
      from pracownik
      where id_kierownika = p.id_pracownika );
```

ZADANIE 27

Wypisać nazwiska osób, które pracowały nad realizacją każdego projektu i przynajmniej przy jednym projekcie dana osoba pracowała jako jedyna.

```
select nazwisko
from pracownik p
where ( select count(nr_projektu)
      from zlecenie
      where id_pracownika = p.id_pracownika ) =
      ( select count(distinct nr_projektu)
      from zlecenie )
```

```
and exists ( select 2
      from zlecenie z1
      where id_pracownika = p.id_pracownika
            and not exists ( select 3
                  from zlecenie
                  where nr_projektu = z1.nr_projektu
                        and id_pracownika != z1.id_pracownika ) );
```

ZADANIE 28

Wypisać nazwy departamentów, których pracownicy mają zarobki na co najmniej trzech różnych poziomach, ale żaden nie na najniższym.

```
select nazwa
from departament d
where ( select count(distinct nr_przedzialu)
      from pracownik join poziom_zarobkow
            on pensja+nvl(premia,0) between dolna_granica and gorna_granica
      where nr_departamentu = d.nr_departamentu ) >= 3
and not exists ( select 1
      from pracownik join poziom_zarobkow
            on pensja+nvl(premia,0) between dolna_granica and gorna_granica
      where nr_departamentu = d.nr_departamentu
            and dolna_granica = ( select min(dolna_granica)
                  from poziom_zarobkow ) );
```

ZADANIE 29

Wypisać nazwiska osób wraz z nazwami projektów, nad którymi ostatnio poszczególne osoby pracowały, pod warunkiem, że dana osoba pracowała kiedyś nad jednym projektem wraz ze swoim przełożonym (oprócz nich nikt inny nie pracował nad tym projektem).

```
select nazwisko, nazwa
from pracownik p1, projekt pr1
where nr_projektu in ( select nr_projektu
            from zlecenie
            where id_pracownika = p1.id_pracownika
                  and data_rozpoczecia = ( select max(data_rozpoczecia)
```

```
                        from zlecenie
                        where id_pracownika = p1.id_pracownika ) )
and exists ( select 2
      from zlecenie z1
      where id_pracownika = p1.id_pracownika
            and exists ( select 3
                  from zlecenie
                  where id_pracownika = p1.id_kierownika
                        and nr_projektu = z1.nr_projektu )
            and not exists ( select 3
      from zlecenie
      where nr_projektu = z1.nr_projektu
            and id_pracownika != p1.id_pracownika
            and id_pracownika != p1.id_kierownika ) );
```

ZADANIE 30

Wypisać nazwy departamentów, w których pracują osoby na stanowiskach występujących tylko w danym departamencie – stanowiska występujące w danym departamencie występują tylko w nim.

```
select nazwa
from departament d
where exists ( select 2
            from pracownik
            where nr_departamentu = d.nr_departamentu )
and not exists ( select 2
            from pracownik p1
            where nr_departamentu = d.nr_departamentu
                  and exists ( select 3
                        from pracownik
                        where stanowisko = p1.stanowisko
                              and nr_departamentu != d.nr_departamentu ) );
```

ZADANIE 31

Wypisać stanowiska oraz nazwiska osób, które zostały najwcześniej zatrudnione na danym stanowisku, pod warunkiem, że dla danego stanowiska istnieje tylko jedna taka osoba (w obrębie departamentu).

```
select stanowisko, nr_departamentu, nazwisko
from pracownik p1
where data_zatrudnienia = ( select min(data_zatrudnienia)
                            from pracownik
                            where nr_departamentu = p1.nr_departamentu
                                  and stanowisko = p1.stanowisko )
and not exists ( select 2
                 from pracownik
                 where nr_departamentu = p1.nr_departamentu
                       and stanowisko = p1.stanowisko
                       and data_zatrudnienia = p1.data_zatrudnienia
                       and id_pracownika != p1.id_pracownika );
```

ZADANIE 32

Wypisać nazwy projektów, podczas realizacji których stawka za godzinę była dla wszystkich jednakowa, nie wszyscy pracowali nad danym projektem tyle samo godzin, a maksymalny czas pracy nad danym projektem (jeśli chodzi o czas pracy na jedną osobę) był mniejszy od czasu realizacji co najmniej jednego projektu.

```
select p.nr_projektu, nazwa
from projekt p join zlecenie z
      on p.nr_projektu = z.nr_projektu
group by p.nr_projektu, nazwa
having min(stawka_za_godzine) = max(stawka_za_godzine)
      and min(liczba_godzin) != max(liczba_godzin)
      and max(liczba_godzin) < any ( select sum(liczba_godzin)
                                     from zlecenie
                                     group by nr_projektu );
```

ZADANIE 33

Wypisać nazwy departamentów, w których wszyscy pracownicy otrzymują pensje na najczęściej pojawiającym się poziomie.

```
select nazwa
from departament d
where exists ( select 3
            from pracownik
            where nr_departamentu = d.nr_departamentu )
   and not exists ( select 1
      from pracownik p1 join poziom_zarobkow pz1
          on pensja+nvl(premia,0) between dolna_granica and gorna_granica
      where nr_departamentu = d.nr_departamentu
           and nr_przedzialu not in ( select nr_przedzialu
            from pracownik p1 join poziom_zarobkow pz1
                  on pensja+nvl(premia,0) between
                                        dolna_granica and gorna_granica
            group by nr_przedzialu
            having count(id_pracownika) >= all ( select count(id_pracownika)
                  from pracownik p1 join poziom_zarobkow pz1
                  on pensja+nvl(premia,0) between
                                        dolna_granica and gorna_granica
                  group by nr_przedzialu ) ) );
```

ZADANIE 34

Wypisać nazwisko kierownika (osoba mająca podwładnych), datę jego zatrudnienia oraz liczbę jego podwładnych zatrudnionych później niż dany kierownik, pod warunkiem, że wszystkie średnie pensje jego podwładnych wyznaczone ze względu na stanowisko (podwładnych) są niższe niż pensja kierownika.

```
select k.id_pracownika, max(k.nazwisko) as nazwisko,
      max(k.data_zatrudnienia) as data_zatrudnienia,
      count(p.id_pracownika) as „liczba podwladnych”
from pracownik k join pracownik p
      on k.id_pracownika = p.id_kierownika
where k.data_zatrudnienia < p.data_zatrudnienia
      and not exists ( select 3
```

```
                    from pracownik
                    where id_kierownika = k.id_pracownika
                    group by stanowisko
                    having avg(pensja) > k.pensja )
group by k.id_pracownika, k.nazwisko, k.data_zatrudnienia;
```

ZADANIE 35

Wypisać nazwisko kierownika (osoba mająca podwładnych) oraz nazwisko jego podwładnego, pod warunkiem, że podwładny jest także kierownikiem i że ma tyle samo podwładnych co jego bezpośredni przełożony.

```
select k.id_pracownika, k.nazwisko, p.id_pracownika, p.nazwisko
from pracownik k join pracownik p
      on k.id_pracownika = p.id_kierownika
where p.id_pracownika in ( select id_kierownika from pracownik )
      and ( select count(id_pracownika)
            from pracownik
            where id_kierownika = p.id_pracownika ) =
      ( select count(id_pracownika)
      from pracownik
      where id_kierownika = k.id_pracownika );
```

ZADANIE 36

Wypisać nazwy departamentów, w których pracuje tyle samo podwładnych (ludzi niemających podwładnych) co kierowników (ludzi mających podwładnych).

```
select nazwa
from departament d
where ( select count(id_pracownika)
      from pracownik p
      where nr_departamentu = d.nr_departamentu
            and not exists ( select 3
                  from pracownik
                  where id_kierownika = p.id_pracownika ) ) =
      ( select count(id_pracownika)
      from pracownik p
```

```
        where nr_departamentu = d.nr_departamentu
              and exists ( select 3
              from pracownik
              where id_kierownika = p.id_pracownika ) );
```

ZADANIE 37

Wypisać stanowiska, na których pracują osoby z co najmniej dwóch różnych departamentów, pod warunkiem, że w każdym z tych departamentów na danym stanowisku pracuje jakiś kierownik (osoba mająca podwładnych).

```
select stanowisko
from pracownik p1
where ( select count(distinct nr_departamentu)
      from pracownik
      where stanowisko = p1.stanowisko ) >= 2
      and ( select count(distinct nr_departamentu)
            from pracownik
            where stanowisko = p1.stanowisko ) =
      ( select count(distinct nr_departamentu)
      from pracownik p2
      where stanowisko = p1.stanowisko
            and exists ( select 4 from pracownik
                  where id_kierownika = p2.id_pracownika ) );
```

ZADANIE 38

Wypisać nazwy projektów, przy realizacji których zdarzyło się tak, że w jednym czasie pracowało nad nim co najmniej dwóch pracowników.

```
select nazwa
from projekt p
where exists ( select 3
      from zlecenie z
      where nr_projektu = p.nr_projektu
            and exists ( select 2
                  from zlecenie
                  where nr_projektu = p.nr_projektu
```

```
        and id_pracownika != z.id_pracownika
        and data_zakonczenia >= z.data_rozpoczecia
        and data_rozpoczecia <= z.data_zakonczenia ) );
```

ZADANIE 39

Wypisać nazwiska pracowników, którzy musieli pracować najdłużej zanim dostali do realizacji jakiś projekt. Należy wziąć pod uwagę tylko te osoby, które pracowały przy realizacji jakiegoś projektu.

```
select nazwisko
from pracownik p1
where ( select min(data_rozpoczecia) - data_zatrudnienia
      from zlecenie z join pracownik p2
            on z.id_pracownika = p2.id_pracownika
      where p2.id_pracownika = p1.id_pracownika
            and data_rozpoczecia > data_zatrudnienia
      group by data_zatrudnienia
      ) >= all
      (
      select min(data_rozpoczecia) - data_zatrudnienia
      from zlecenie z join pracownik p2
            on z.id_pracownika = p2.id_pracownika
      where data_rozpoczecia > data_zatrudnienia
      group by p2.id_pracownika, data_zatrudnienia
      )
   and exists ( select 4
            from zlecenie
            where id_pracownika = p1.id_pracownika );
```

ZADANIE 40

Wypisać nazwy projektów wraz z nazwiskami osób, które zostały zatrudnione w czasie realizacji danego projektu (pomiędzy datami początku i końca realizacji), pod warunkiem, że dane osoby pracowały nad realizacją kolejnego projektu (tego, którego rozpoczęcie realizacji było najbliższe dacie końca realizacji projektu wypisywanego).

```
select nazwa, nazwisko
from projekt pr, pracownik p
```

```
where data_zatrudnienia between ( select min(data_rozpoczecia)
            from zlecenie
            where nr_projektu = pr.nr_projektu )
      and ( select max(data_rozpoczecia)
            from zlecenie
            where nr_projektu = pr.nr_projektu )
      and id_pracownika in ( select id_pracownika
            from zlecenie z1
            where data_rozpoczecia = ( select min(min(data_rozpoczecia))
                  from zlecenie
                  where data_rozpoczecia > ( select min(data_rozpoczecia)
                        from zlecenie
                        where nr_projektu = pr.nr_projektu )
                  group by nr_projektu ) );
```

ZADANIE 41

Wypisać nazwy projektów, których planowane daty rozpoczęcia oraz końca realizacji nie przekraczają rzeczywistych dat i przy realizacji których pracowała co najmniej dwa razy (tyle że w innych okresach) ta sama osoba.

```
select nazwa
from projekt pr
where ( select min(data_rozpoczecia)
      from zlecenie
      where nr_projektu = pr.nr_projektu ) >= data_rozpoczecia
   and ( select max(data_zakonczenia)
      from zlecenie
      where nr_projektu = pr.nr_projektu ) <= data_rozpoczecia
   and exists ( select 4
            from zlecenie z1
            where nr_projektu = pr.nr_projektu
                  and exists ( select 3
                        from zlecenie
                        where nr_projektu = pr.nr_projektu
                        and id_pracownika = z1.id_pracownika
                        and data_rozpoczecia != z1.data_rozpoczecia ) );
```

ZADANIE 42

Wypisać nazwiska osób wraz z poziomami ich zarobków oraz nazwami departamentów, w których pracują, pod warunkiem, że zarobki tych osób znajdują się najbliżej górnej granicy określającej dany poziom zarobków (biorąc pod uwagę ogół pracowników), a w danym departamencie nie ma drugiej takiej osoby (spełniającej dane założenie).

```
select nazwisko, nr_przedzialu, nazwa
from pracownik p1 join poziom_zarobkow pz1
      on pensja+nvl(premia,0) between dolna_granica and gorna_granica
      join departament d
      on p1.nr_departamentu = d.nr_departamentu
where gorna_granica - pensja+nvl(premia,0) =
      ( select min(gorna_granica - pensja+nvl(premia,0))
      from pracownik p2 join poziom_zarobkow pz2
            on pensja+nvl(premia,0) between dolna_granica and gorna_granica
      where nr_przedzialu = pz1.nr_przedzialu )
   and not exists ( select 4
      from pracownik p2 join poziom_zarobkow pz2
            on pensja+nvl(premia,0) between dolna_granica and gorna_granica
      where nr_departamentu = p1.nr_departamentu
            and gorna_granica - pensja+nvl(premia,0) = pz1.gorna_granica –
                                    p1.pensja+nvl(p1.premia,0)
            and nr_przedzialu = pz1.nr_przedzialu );
```

ZADANIE 43

Wypisać nazwiska kierowników, którzy jeśli pracowali nad realizacją jakiegoś projektu, to pracowali wyłącznie z kierownikami.

```
select nazwisko
from pracownik p1
where exists ( select 3
      from pracownik
      where id_kierownika = p1.id_pracownika )
   and not exists ( select 3
            from zlecenie z1
            where id_pracownika = p1.id_pracownika
                  and exists ( select 1
```

```
from zlecenie z2
where nr_projektu = z1.nr_projektu
      and not exists ( select 3
            from pracownik
            where id_kierownika = z2.id_pracownika ) ) );
```

ZADANIE 44

Wypisać datę zatrudnienia, pod warunkiem, że danego dnia zostało zatrudnionych co najmniej dwóch pracowników (wypisać liczbę osób zatrudnionych danego dnia), z których każdy pracował nad jakimś projektem, ale nie nad takim, którego realizacja rozpoczęła się „najbliżej" danej daty.

```
select data_zatrudnienia, count(id_pracownika)
from pracownik p1
where ( select count(id_pracownika)
      from pracownik
      where data_zatrudnienia = p1.data_zatrudnienia ) >= 2
    and not exists ( select 2
      from pracownik p2
      where data_zatrudnienia = p1.data_zatrudnienia
            and not exists ( select 4
                  from zlecenie
                  where id_pracownika = p2.id_pracownika ) )
    and not exists ( select 4
      from pracownik p2
      where data_zatrudnienia = p1.data_zatrudnienia
            and exists
      ( select 2
      from zlecenie z1
      where ( select min(data_rozpoczecia)
            from zlecenie z2
            where nr_projektu = z1.nr_projektu ) > p1.data_zatrudnienia
      and id_pracownika = p2.id_pracownika
      and ( select min(data_rozpoczecia)
            from zlecenie z2
            where nr_projektu = z1.nr_projektu ) - p1.data_zatrudnienia =
```

```
                ( select min(data_rozpoczecia - p1.data_zatrudnienia)
        from zlecenie z2
        where ( select min(data_rozpoczecia)
                from zlecenie z3
                where nr_projektu = z2.nr_projektu ) > p1.data_zatrudnienia
        and data_rozpoczecia = ( select min(data_rozpoczecia)
                from zlecenie z3
                where nr_projektu = z2.nr_projektu ) ) ) )
group by data_zatrudnienia;
```

ZADANIE 45

Wypisać nazwy departamentów, a także lokalizację oraz liczbę pracowników danego departamentu, w których wszystkie osoby pracujące na tym samym stanowisku mają takie same warunki dotyczące premii (chodzi o dostawanie premii bądź nie). Dodatkowy warunek: w danym departamencie wszystkie stanowiska muszą być „równo obsadzone".

```
select max(nazwa) as nazwa, max(lokalizacja) as lokalizacja,
count(id_pracownika)
from departament d1 join pracownik p1
        on d1.nr_departamentu = p1.nr_departamentu
where not exists ( select 2
                from pracownik p2 join pracownik p3
                        on p2.stanowisko = p3.stanowisko
                where nr_departamentu = d1.nr_departamentu
                    and p2.premia is null and p3.premia is not null )
                    and not exists ( select 2
                        from pracownik p2
                        where nr_departamentu = d1.nr_departamentu
                                and ( select count(id_pracownika)
                                        from pracownik p3
                                        where stanowisko = p2.stanowisko
                                            and nr_departamentu = d1.nr_departamentu ) !=
                                        any ( select count(id_pracownika)
                                            from pracownik p3
                                            where stanowisko != p2.stanowisko
```

```
                                        and nr_departamentu = d1.nr_departamentu ) )
group by d1.nr_departamentu;
```

ZADANIE 46

Podać nazwę dla każdego departamentu, w którym nie pracują kierownicy, którzy nie realizowali projektów wraz z przynajmniej jednym innym kierownikiem.

Dodatkowo: dla dowolnie wybranego pracownika* z tego departamentu można wybrać innego pracownika** z tego departamentu, którego kierownik realizował tyle samo projektów co kierownik pracownika*.

```
select nazwa
from departament d
where not exists ( select 3
      from pracownik p1
      where nr_departamentu = d.nr_departamentu
            and exists ( select 9
                  from pracownik
                  where id_kierownika = p1.id_pracownika )
      and exists ( select 3
            from zlecenie z1
            where id_pracownika = p1.id_pracownika
                  and ( select count(id_pracownika)
                        from zlecenie z2
                        where nr_projektu = z1.nr_projektu
                              and exists ( select 9
                                    from pracownik
                                    where id_kierownika = z2.id_pracownika ) ) = 1 ) )
      and not exists ( select p2.id_pracownika, p3.id_pracownika
            from pracownik p2 join pracownik p3
                  on p2.nr_departamentu = p3.nr_departamentu
            where p2.id_pracownika != p3.id_pracownika
                  and ( select count(nr_projeku)
                  from zlecenie
                  where id_pracownika = p2.id_kierownika ) !=
                  ( select count(nr_projeku)
                  from zlecenie
                  where id_pracownika = p3.id_kierownika ) );
```

ZADANIE 47

Dla każdej pary podwładny-kierownik wypisać ich identyfikatory oraz nazwiska, pod warunkiem, że w przypadku każdego zlecenia danego kierownika ten projekt realizował także dany podwładny i jeszcze inny jego podwładny.

```
select p.id_pracownika, p.nazwisko, k.id_pracownika, k.nazwisko
from pracownik p join pracownik k
      on p.id_kierownika = k.id_pracownika
where not exists ( select 2
            from zlecenie z1
            where id_pracownika = k.id_pracownika
                  and not exists (select 2
                        from zlecenie z2 join zlecenie z3
                              on z2.id_pracownika != z3.id_pracownika
                        where z2.id_pracownika = p.id_pracownika
                              and z2.nr_projektu = z3.nr_projektu
                              and z2.nr_projektu = z1.nr_projektu
                              and z2.id_pracownika = p.id_pracownika
                              and exists ( select 1
                                    from pracownik
                                    where id_kierownika = k.id_pracownika
                                     and id_pracownika = z3.id_pracownika ) ) );
```

ZADANIE 48

Wypisać identyfikatory i nazwiska pracowników, którzy pracowali wyłącznie nad projektami z przekroczonym budżetem, a ich kierownicy zrealizowali więcej projektów niż ci pracownicy.

```
select id_pracownika, nazwisko
from pracownik p1
where exists ( select 2
      from zlecenie
      where id_pracownika = p1.id_pracownika )
    and not exists ( select 3
      from zlecenie z1
      where id_pracownika = p1.id_pracownika
            and ( select sum(stawka_za_godzine*ilosc_godzin)
```

```
                from zlecenie
                where nr_projektu = z1.nr_projektu ) < ( select budzet
                                from projekt
                                where nr_projektu = z1.nr_projektu ) )
    and ( select count(nr_projektu)
      from zlecenie
      where id_pracownika = p1.id_pracownika ) <
      ( select count(nr_projektu)
      from zlecenie
      where id_pracownika = p1.id_kierownika );
```

ZADANIE 49

Dla każdego departamentu, w przypadku którego panuje największe zróżnicowanie w stanowiskach kierowników (w porównaniu do reszty departamentów), podać jego nazwę oraz numer przedziału zarobków dominującego w danym departamencie.

```
select nazwa, nr_przedzialu
from departament d, poziom_zarobkow pz
where ( select count(distinct stanowisko)
      from pracownik p
      where nr_departamentu = d.nr_departamentu
            and exists ( select 3
                  from pracownik
                  where id_kierownika = p.id_pracownika ) ) >= all
                  ( select count(distinct stanowisko)
                  from pracownik p
                  where exists ( select 3
                        from pracownik
                        where id_kierownika = p.id_pracownika )
                  group by nr_departamentu )
    and ( select count(id_pracownika)
      from pracownik p join poziom_zarobkow
            on pensja+nvl(premia,0) between dolna_granica and gorna_granica
      where nr_departamentu = d.nr_departamentu
            and nr_przedzialu = pz.nr_przedzialu ) >= all
                  ( select count(id_pracownika)
```

```
from pracownik p join poziom_zarobkow
   on pensja+nvl(premia,0) between dolna_granica
   and gorna_granica
where nr_departamentu = d.nr_departamentu
group by nr_przedzialu );
```

ZADANIE 50

Wypisać nazwy departamentów, w przypadku których każdy kierownik w nich zatrudniony ma co najmniej dwóch podwładnych z innego departamentu, a średnie zarobki w tych departamentach są większe od średnich zarobków w co najmniej dwóch innych departamentach.

```
select nazwa
from departament d join pracownik p
      on d.nr_departamentu = p.nr_departamentu
where not exists ( select 3
            from pracownik p1
            where nr_departamentu = d.nr_departamentu
                  and exists ( select 3
                        from pracownik
                        where id_kierownika = p1.id_pracownika )
                  and ( select count(id_pracownika)
                        from pracownik
                        here nr_departamentu != d.nr_departamentu
                              and id_kierownika = p1.id_pracownika ) < 2 )
      and ( select count(nr_departamentu)
            from pracownik p1
            where nr_departamentu != d.nr_departamentu
                  and ( select avg(pensja+nvl(premia,0))
                        from pracownik
                        where nr_departamentu = p1.nr_departamentu ) <
                        ( select avg(pensja+nvl(premia,0))
                        from pracownik
                        where nr_departamentu = d.nr_departamentu )  ) >= 2;
```

3.2. Zasada nr 3

W przypadku, gdy zadanie opiera się na wyszukiwaniu danych, gdzie sens zagadnienia można streścić, używając słów kategorycznych/silnych, na przykład: zawsze, nigdy, każdy, żaden, wszyscy itp., najczęściej głównym trzonem rozwiązania staje się przeczenie (`NOT IN, NOT EXISTS`).

3.3. Wskazówki dotyczące składni SQL

Przykład:

```
SELECT nazwisko
FROM pracownik
WHERE pensja = ( SELECT pensja
                 FROM pracownik
                 WHERE id_pacownika = 1 );
```

Wybrane zostaną nazwiska tych osób, które mają taką samą pensję co pracownik o identyfikatorze 1.

Przy sprawdzaniu warunków (klauzule `WHERE` i `HAVING`) można używać następujących słów kluczowych:

1. `ANY`.
   Na przykład `WHERE x > ANY (SELECT...)` ← podzapytanie,
   czyli `x` ma być większe od co najmniej jednej wartości wybranej w podzapytaniu.

2. `ALL`.
   Na przykład `WHERE x > ALL (SELECT...)` ← podzapytanie,
   czyli `x` ma być większe od wszystkich wartości wybranych w podzapytaniu.

3. `IN`.
   Na przykład `WHERE x IN (SELECT...)` ← podzapytanie,
   czyli `x` ma należeć do zbioru wierszy zwracanych przez podzapytanie.
   Istnieje przeczenie: `NOT IN` (`x` nie należy do zbioru).

4. `EXISTS`.
   Na przykład `WHERE EXISTS (SELECT...)` ← podzapytanie,
   czyli istnieje co najmniej jeden wiersz spełniający warunki określone w podzapytaniu (podzapytanie zwraca co najmniej jeden wiersz).

Przykład:

```
SELECT a.nazwisko
FROM pracownik a
WHERE EXISTS ( SELECT b.nazwisko
            FROM pracownik b
            WHERE a.id_pracownika != b.id_pracownika AND b.pensja=a.pensja );
```

Wybierane są nazwiska osób pod warunkiem, że istnieją też inne osoby mające taką samą pensję co dana osoba.

Istnieje przeczenie: `NOT EXISTS` (nie istnieje).

4. Złączenia zbiorów

Relacyjne systemy baz danych opierają się na zasadach wynikających z algebry relacji oraz teorii zbiorów. Jedną z podstawowych funkcjonalności języka SQL jest możliwość łączenia wyników wielu zapytań przy użyciu operacji zbiorowych. Operacje te umożliwiają scalanie, porównywanie oraz filtrowanie zbiorów wynikowych w sposób zgodny z formalnymi zasadami teorii zbiorów. Mechanizmy te znajdują szerokie zastosowanie w sytuacjach, gdy konieczne jest pobranie danych z wielu źródeł, porównanie wyników różnych zapytań lub utworzenie jednolitego zbioru wynikowego na podstawie wielu warunków selekcji.

Operacje zbiorowe w języku SQL umożliwiają połączenie wyników dwóch lub więcej instrukcji `SELECT` w jeden zbiór wynikowy. Do najczęściej stosowanych operatorów zbiorowych należą `UNION`, `UNION ALL`, `INTERSECT` oraz `MINUS`. Każdy z nich realizuje określoną operację na zbiorach wynikowych, umożliwiając wyrażenie złożonych zależności logicznych pomiędzy wynikami zapytań.

Operator `UNION` służy do połączenia wyników dwóch lub więcej zapytań `SELECT` w jeden zbiór wynikowy, przy jednoczesnym automatycznym usunięciu duplikatów. Oznacza to, że wynik końcowy zawiera wyłącznie unikalne rekordy, zgodnie z definicją zbioru w ujęciu matematycznym. Operator ten znajduje zastosowanie w sytuacjach, gdy konieczne jest utworzenie jednolitego zestawu danych pochodzących z różnych zapytań, bez powtarzających się wartości.

Operator `UNION ALL` działa analogicznie do operatora `UNION`, jednak w przeciwieństwie do niego nie eliminuje duplikatów. Ze względu na brak konieczności eliminowania duplikatów operator `UNION ALL` jest zazwyczaj bardziej wydajny niż `UNION`, szczególnie w przypadku przetwarzania dużych zbiorów danych. Operator ten powinien być stosowany w sytuacjach, gdy duplikaty mają znaczenie semantyczne lub gdy istotna jest wysoka wydajność zapytania.

Operator `INTERSECT` umożliwia wyznaczenie części wspólnej wyników dwóch zapytań `SELECT`. Wynik tej operacji zawiera wyłącznie te rekordy, które występują jednocześnie w obu zbiorach wynikowych. Operator ten jest szczególnie użyteczny w przypadku identyfikowania wspólnych elementów zbiorów danych, takich jak rekordy spełniające wiele warunków lub występujące w różnych kontekstach analitycznych.

Operator `MINUS`, dostępny m.in. w systemie Oracle, umożliwia wyznaczenie różnicy zbiorów, zwracając rekordy występujące w wyniku pierwszego zapytania, które nie występują w wyniku drugiego zapytania. Operator ten pozwala na identyfikację

rekordów nieposiadających odpowiedników w innym zbiorze danych. W niektórych systemach zarządzania bazami danych operator ten występuje pod nazwą EXCEPT. Operator MINUS znajduje zastosowanie m.in. w analizie różnic pomiędzy zbiorami danych oraz w identyfikowaniu brakujących lub wykluczonych rekordów.

Zastosowanie operatorów zbiorowych wymaga spełnienia określonych warunków strukturalnych. Każde z zapytań SELECT uczestniczących w operacji musi zwracać taką samą liczbę kolumn, a odpowiadające sobie kolumny muszą posiadać zgodne typy danych. Ponadto kolejność kolumn w zapytaniach musi być zgodna, ponieważ operacje zbiorowe wykonywane są na podstawie porównania odpowiadających sobie kolumn w zbiorach wynikowych.

Każde zapytanie SELECT wykonywane jest niezależnie, a następnie system zarządzania bazą danych wykonuje odpowiednią operację zbiorową w celu utworzenia wyniku końcowego. W przypadku operatorów eliminujących duplikaty, takich jak UNION oraz INTERSECT, system musi wykonać dodatkowe operacje porównywania rekordów, co może wpływać na wydajność zapytania. Z tego względu operator UNION ALL, który nie eliminuje duplikatów, jest zazwyczaj bardziej wydajny.

Po zapoznaniu się z treścią niniejszego rozdziału student zdobędzie wiedzę i umiejętności umożliwiające samodzielne stosowanie operatorów UNION, UNION ALL, INTERSECT oraz MINUS w zapytaniach SQL. Student nauczy się rozróżniać działanie poszczególnych operatorów, w szczególności w zakresie obsługi duplikatów oraz wpływu na wydajność zapytań.

Ponadto student nabędzie umiejętności konstruowania poprawnych zapytań wykorzystujących operacje zbiorowe, zapewniania zgodności strukturalnej zapytań oraz interpretowania otrzymanych wyników. Zdobyta wiedza umożliwi mu efektywne wykonywanie operacji porównywania, łączenia oraz analizy danych przechowywanych w relacyjnych bazach danych.

4.1. UNION

ZADANIE 1

Wypisać nazwy projektów wraz z nazwiskami pracowników, którzy nad danym projektem pracowali. W wynikach powinni zostać uwzględnieni pracownicy, którzy nie pracowali nad żadnym projektem oraz projekty do tej pory nierealizowane.

```
select p.nr_projektu, nazwa, pr.id_pracownika, nazwisko
from projekt p, zlecenie z, pracownik pr
where p.nr_projektu = z.nr_projektu
      and z.id_pracownika = pr.id_pracownika
union
select nr_projektu, nazwa, null, null
```

```
from projekt p
where not exists ( select 4
            from zlecenie
            where nr_projektu = p.nr_projektu )
union
select null, null, id_pracownika, nazwisko
from pracownik p
where not exists ( select 2
            from zlecenie
            where id_pracownika = p.id_pracownika );
```

Komentarz 1

Do każdego wyrażenia `SELECT` dodano numery projektów i identyfikatory pracowników ze względu na usuwanie duplikatów przez `UNION`: może być parę projektów o takiej samej nazwie (ale różnych numerach) i paru pracowników o takich samych nazwiskach (ale różnych identyfikatorach). Wiersze z takimi samymi wartościami nazw projektów i nazwisk (gdyby wyrażenie `SELECT` miało postać: `select nazwa, nazwisko`) zostałyby potraktowane jako jeden i tylko ten jeden wiersz znalazłby się w wynikach zapytania. Aby tego uniknąć, uzupełniono wyrażenie `SELECT` wspomnianymi dwoma atrybutami.

Komentarz 2

Wyrażenia `SELECT` w drugim i trzecim zapytaniu uzupełnione zostały `NULL`-ami. Powody to składnia `UNION` i przejrzystość wyników. Pierwsze zapytanie zwraca cztery wartości, w związku z czym każde kolejne także powinno zwracać cztery wartości. Drugie zapytanie dotyczy nierealizowanych projektów, czyli pola odpowiadające identyfikatorowi pracownika oraz jego nazwisku powinny zostać puste, ponieważ nad danym projektem nikt nie pracował. Podobnie zapytanie trzecie dotyczy pracowników, którzy nie realizowali projektów, czyli pola odpowiadające numerowi projektu i jego nazwie powinny być puste, ponieważ dany pracownik nie realizował żadnych projektów.

ZADANIE 2

Dla każdego stanowiska podać liczbę kierowników zatrudnionych na danym stanowisku oraz liczbę pracowników, którzy nie są kierownikami, zatrudnionych na danym stanowisku.

```
select stanowisko, 'Liczba kierownikow: ' || count(id_pracownika)
from pracownik p1
```

```
where exists ( select 2
      from pracownik
      where id_kierownika = p1.id_pracownika )
group by stanowisko
union
select stanowisko, 'Liczba podwladnych: ' || count(id_pracownika)
from pracownik p1
where not exists ( select 2
      from pracownik
      where id_kierownika = p1.id_pracownika )
group by stanowisko;
```

ZADANIE 3

Dla każdego departamentu podać: jego nazwę, średnie zarobki kierowników z tego departamentu, którzy realizowali projekty, oraz średnie zarobki kierowników z tego departamentu, którzy nie realizowali projektów.

```
select d1.nr_departamentu, nazwa,
'Srednie zarobki kierownikow realizujacych projekty: ' ||
avg(pensja+nvl(premia,0))
from pracownik p1 join departament d1
      on p1.nr_departamentu = d1.nr_departamentu
where exists ( select 2
      from pracownik
      where id_kierownika = p1.id_pracownika )
and exists ( select 2
      from zlecenie
      where id_pracownika = p1.id_pracownika )
group by d1.nr_departamentu, nazwa
union
select d1.nr_departamentu, nazwa,
'Srednie zarobki kierownikow nierealizujacych projekty: ' ||
avg(pensja+nvl(premia,0))
from pracownik p1 join departament d1
      on p1.nr_departamentu = d1.nr_departamentu
where exists ( select 2
```

```
        from pracownik
        where id_kierownika = p1.id_pracownika )
and not exists ( select 2
        from zlecenie
        where id_pracownika = p1.id_pracownika )
group by d1.nr_departamentu, nazwa;
```

4.2. MINUS

```
SELECT lista_kolumn
...
MINUS
(
SELECT lista_kolumn
...
UNION
SELECT lista_kolumn
...
);
```

ZADANIE 5

Wypisać nazwy departamentów, w których nikt nie pracuje.

```
select nr_departamentu, nazwa
from departament
minus
select nr_departamentu, nazwa
from departament d
where exists ( select 4
        from pracownik
        where nr_departamentu = d.nr_departamentu );
```

ZADANIE 6

Wypisać nazwy departamentów, w których nikt nie pracuje na stanowiskach analityk oraz księgowy.

```
select nr_departamentu, nazwa
from departament
minus
select nr_departamentu, nazwa
from departament d
where exists ( select 4
      from pracownik
      where nr_departamentu = d.nr_departamentu
            and lower(stanowisko) in ('analityk', 'ksiegowy') );
```

ZADANIE 7

Wypisać stanowiska, w przypadku których żaden z pracowników na nich zatrudniony nie otrzymuje ani minimalnej, ani maksymalnej pensji.

```
select stanowisko
from pracownik
minus
select stanowisko
from pracownik
where pensja = ( select min(pensja) from pracownik )
      or pensja = ( select min(pensja) from pracownik );
```

4.3. INTERSECT

ZADANIE 8

Wypisać nazwiska pracowników, którzy pracowali nad pierwszym i ostatnim projektem (bierzemy pod uwagę realizowane projekty, czyli pierwszy projekt to projekt z najwcześniejszą datą rozpoczęcia realizacji – tabela `Zlecenie`; analogicznie w przypadku projektu ostatniego).

```
select id_pracownika, nazwisko
from pracownik
where id_pracownika in ( select id_pracownika
      from zlecenie
```

```
      where nr_projektu in ( select nr_projektu
            from zlecenie
            where data_rozpoczecia = ( select min(data_rozpoczecia)
                                       from zlecenie ) ) )
intersect
select id_pracownika, nazwisko
from pracownik
where id_pracownika in ( select id_pracownika
      from zlecenie
      where nr_projektu in ( select nr_projektu
            from zlecenie
            group by nr_projektu
            having min(data_rozpoczecia) = ( select
            max(min(data_rozpoczecia))
                                       from zlecenie
                                       group by nr_projektu ) ) );
```

ZADANIE 9

Wypisać nazwy departamentów, w których pracują co najmniej trzy osoby, a średnia pensja otrzymywana przez osoby w nich zatrudnione jest większa od ogólnej średniej pensji.

```
select d.nr_departamentu, nazwa
from departament d join pracownik p
      on d.nr_departamentu = p.nr_departamentu
group by d.nr_departamentu, nazwa
having count(id_pracownika) >= 3
intersect
select d.nr_departamentu, nazwa
from departament d join pracownik p
      on d.nr_departamentu = p.nr_departamentu
group by d.nr_departamentu, nazwa
having avg(pensja) > ( select avg(pensja)
                  from pracownik );
```

Uwaga 1

Można łączyć `UNION`, `MINUS` oraz `INTERSECT`.

5. Podzapytania w klauzuli FROM

Zapytania zagnieżdżone umieszczane w klauzuli FROM stanowią zaawansowany mechanizm języka SQL, umożliwiający konstruowanie zapytań wieloetapowych. Tego rodzaju podzapytania, określane najczęściej jako **tabele pochodne** (ang. *derived tables*) lub **widoki inline**, pozwalają traktować wynik zapytania wewnętrznego jako tymczasowe źródło danych w zapytaniu zewnętrznym.

Z teoretycznego punktu widzenia tabela pochodna stanowi relację wyznaczaną dynamicznie w trakcie wykonywania zapytania. Język SQL implementuje tę zasadę, umożliwiając umieszczanie zapytań w klauzuli FROM, których wynik przyjmuje postać tymczasowej relacji uczestniczącej w dalszych operacjach, takich jak złączenia, filtrowanie czy grupowanie. W przeciwieństwie do podzapytań w klauzulach WHERE lub SELECT, które zazwyczaj zwracają pojedyncze wartości lub służą jako warunki logiczne, podzapytania w klauzuli FROM zwracają pełne zbiory wierszy i kolumn.

Podstawowym celem stosowania zapytań zagnieżdżonych w klauzuli FROM jest dekompozycja złożonych problemów na logicznie uporządkowane etapy przetwarzania. W wielu przypadkach analitycznych konieczne jest najpierw wyznaczenie wyniku pośredniego – na przykład agregacji danych w podziale na grupy – a następnie wykorzystanie tego wyniku w dalszych operacjach, takich jak złączenie z inną tabelą lub dodatkowe filtrowanie. Tabele pochodne umożliwiają wyrażenie takiej sekwencji operacji w sposób przejrzysty i logicznie spójny w ramach jednej instrukcji SQL.

Konstruowanie zapytań zagnieżdżonych w klauzuli FROM podlega określonym ograniczeniom składniowym. Przede wszystkim każda tabela pochodna musi zostać ujęta w nawiasy oraz posiadać alias. Nadanie aliasu jest obowiązkowe, ponieważ zapytanie zewnętrzne musi posiadać możliwość jednoznacznego odwołania się do tabeli pochodnej. Kolumny tabeli pochodnej muszą mieć poprawne identyfikatory wynikające z zapytania wewnętrznego lub jawnie zdefiniowane poprzez aliasy kolumnowe. Kolejnym, nie mniej ważnym, ograniczeniem jest brak możliwości komunikacji między tabelami pochodnymi podczas ich definiowania (definiowania bazowego zapytania SQL).

Istotne znaczenie mają również aspekty wydajnościowe. Choć zastosowanie tabel pochodnych zwiększa przejrzystość i modularność zapytań, ich użycie może wpływać na sposób optymalizacji zapytania przez system zarządzania bazą danych. W niektórych przypadkach optymalizator może przekształcić lub scalić zapytania wewnętrzne,

w innych natomiast złożona struktura zagnieżdżeń może prowadzić do zwiększenia kosztu obliczeniowego. Konieczna jest zatem umiejętność oceny zasadności zastosowania tabel pochodnych w konkretnym kontekście.

Po zapoznaniu się z treścią niniejszego rozdziału student zdobędzie zarówno wiedzę teoretyczną, jak i umiejętności praktyczne w zakresie konstruowania zapytań zagnieżdżonych w klauzuli `FROM`. Nauczy się projektować tabele pochodne jako etapy pośrednie przetwarzania danych, poprawnie stosować aliasowanie oraz zapewniać zgodność składniową zapytań wielopoziomowych.

Student rozwinie także umiejętności dekompozycji złożonych problemów analitycznych na logicznie uporządkowane etapy, interpretowania wyników pośrednich oraz integrowania przekształconych lub zagregowanych danych z zapytaniami zewnętrznymi. Ponadto będzie miał świadomość ograniczeń składniowych i zasad przetwarzania zapytań, co umożliwi mu konstruowanie zapytań poprawnych semantycznie i efektywnych wydajnościowo.

5.1. Zadania z rozwiązaniami i komentarzami

ZADANIE 1

Wypisać nazwę departamentu i liczbę różnych stanowisk w tym departamencie.

```
select d.nr_departamentu, nazwa, count(liczba) as „liczba roznych stanowisk”
from departament d, ( select nr_departamentu, count(id_pracownika) as liczba
                      from pracownik
                      group by nr_departamentu, stanowisko ) x
where d.nr_departamentu = x.nr_departamentu
group by d.nr_departamentu, nazwa;
```

Komentarz 1

W zapytaniu, w klauzuli `FROM`, zdefiniowano podzapytanie, które można traktować jak dynamiczną tabelę. Nazwa tej tabeli to `x`.

Komentarz 2

Zapytanie x oprócz numeru departamentu zwraca liczbę pracowników – wynik działania funkcji `count`. Gdy zaistnieje potrzeba odwołania się do tej funkcji – już poza zapytaniem `x` – składnia nie pozwoli na odwołanie typu `x.count(id_pracownika)`, dlatego każda funkcja wybierana w podzapytaniu w klauzuli `FROM` musi mieć nadaną nazwę (w tym przypadku jest to liczba).

Komentarz 3

W zapytaniu mamy dostępne dwie tabele, z których można wybierać dane. Łączenie tabel `where d.nr_departamentu = x.nr_departamentu` ma miejsce poza definicją tabeli `x` – wewnątrz tej definicji **nie ma możliwości** odwołania do innych tabel wymienionych w klauzuli `FROM`.

ZADANIE 2

Wypisać nazwy departamentów, w których pensje są najmniej zróżnicowane (występuje najmniejszy zakres zmienności).

```
select nazwa
from ( select nazwa, count(distinct pensja) as liczba_roznych_departament
      from departament d left join pracownik p
            on d.nr_departamentu = p.nr_departamentu
      group by d.nr_departamentu, nazwa ),
      ( select min(count(distinct pensja)) as liczba_roznych_ogolnie
      from departament d left join pracownik p
            on d.nr_departamentu = p.nr_departamentu
      group by d.nr_departamentu )
where liczba_roznych_departament = liczba_roznych_ogolnie;
```

ZADANIE 3

Wypisać stanowiska, dla których minimalna pensja jest równa ogólnej minimalnej pensji.

```
select stanowisko
from ( select stanowisko, min(pensja) as min_stanowisko
      from pracownik
      group by stanowisko ), ( select min(pensja) as min_ogolnie
                              from pracownik )
where min_stanowisko = min_ogolnie;
```

ZADANIE 4

Wypisać nazwiska pracowników, którzy pracowali nad największą liczbą projektów.

```
select nazwisko
from ( select nazwisko, count(nr_projektu) as liczba_projektow_pracownik
```

```
        from pracownik p left join zlecenie z
                on p.id_pracownika = z.id_pracownika
        group by nazwisko, p.id_pracownika ),
        ( select max(count(nr_projektu)) as liczba_projektow_max
        from zlecenie
        group by id_pracownika )
where liczba_projektow_pracownik = liczba_projektow_max;
```

ZADANIE 5

Wypisać parami nazwiska dwóch kierowników, pod warunkiem, że pracowali nad taką samą liczbą projektów (należy pamiętać, że 0 także jest brane pod uwagę).

Rozwiązanie 1 (niepoprawne logicznie):

```
select k1.nazwisko, k2.nazwisko
from ( select p.id_pracownika, nazwisko, count(nr_projektu) as liczba1
        from pracownik p left join zlecenie z
                on p.id_pracownika = z.id_pracownika
        where exists ( select 3
                        from pracownik
                        where id_kierownika = p.id_pracownika )
        group by nazwisko, p.id_pracownika ) k1,
        ( select p.id_pracownika, nazwisko, count(nr_projektu) as liczba2
        from pracownik p left join zlecenie z
                on p.id_pracownika = z.id_pracownika
        where exists ( select 3
                        from pracownik
                        where id_kierownika = p.id_pracownika )
        group by nazwisko, p.id_pracownika ) k2
where liczba1 = liczba2;
```

Komentarz 1

W wynikach uwzględnione są pary: `kierownik.x – kierownik.x` (ponieważ dla tej samej osoby zgadzają się liczby projektów).

Rozwiązanie 2 (niepoprawne logicznie):

```
select k1.nazwisko, k2.nazwisko
from ( select p.id_pracownika, nazwisko, count(nr_projektu) as liczba1
      from pracownik p left join zlecenie z
            on p.id_pracownika = z.id_pracownika
      where exists ( select 3
                  from pracownik
                  where id_kierownika = p.id_pracownika )
      group by nazwisko, p.id_pracownika ) k1,
      ( select p.id_pracownika, nazwisko, count(nr_projektu) as liczba2
      from pracownik p left join zlecenie z
            on p.id_pracownika = z.id_pracownika
      where exists ( select 3
                  from pracownik
                  where id_kierownika = p.id_pracownika )
      group by nazwisko, p.id_pracownika ) k2
where liczba1 = liczba2 and k1.id_pracownika != k2.id_pracownika;
```

Komentarz 2

W wynikach uwzględnione są powtórzenia: `kierownik.x – kierownik.y`, `kierownik.y – kierownik.x`, bowiem po złączeniu tabel `k1` i `k2` (a zawierają one takie same wiersze) powstaje iloczyn kartezjański wierszy obu tabel.

Rozwiązanie:

```
select k1.nazwisko, k2.nazwisko
from ( select p.id_pracownika, nazwisko, count(nr_projektu) as liczba1
      from pracownik p left join zlecenie z
            on p.id_pracownika = z.id_pracownika
      where exists ( select 3
                  from pracownik
                  where id_kierownika = p.id_pracownika )
      group by nazwisko, p.id_pracownika ) k1,
      ( select p.id_pracownika, nazwisko, count(nr_projektu) as liczba2
      from pracownik p left join zlecenie z
            on p.id_pracownika = z.id_pracownika
```

```
      where exists ( select 3
                     from pracownik
                     where id_kierownika = p.id_pracownika )
      group by nazwisko, p.id_pracownika ) k2
where liczba1 = liczba2 and k1.id_pracownika > k2.id_pracownika;
```

Komentarz 3

Porównanie typu `k1.id_pracownika > k2.id_pracownika` rozwiązuje problem powtórzeń, zadziała bowiem dla identyfikatorów zarówno typu tekstowego, jak i liczbowego czy dat.

6. Podzapytania w klauzuli SELECT

Zapytania zagnieżdżone umieszczane w klauzuli SELECT stanowią zaawansowany mechanizm języka SQL, umożliwiający dynamiczne wyznaczanie wartości w trakcie pobierania danych. W odróżnieniu od podzapytań wykorzystywanych w klauzulach WHERE lub HAVING, które odpowiadają przede wszystkim za filtrowanie danych, podzapytania w klauzuli SELECT działają jako wyrażenia obliczeniowe. Umożliwiają one rozszerzenie każdego wiersza zbioru wynikowego o dodatkowe wartości wyznaczane na podstawie innych zapytań, często z wykorzystaniem funkcji agregujących lub danych pochodzących z powiązanych tabel.

Po określeniu źródła danych w klauzuli FROM oraz zastosowaniu ewentualnych operacji filtrowania i grupowania klauzula SELECT definiuje ostateczną strukturę wyniku zapytania. Jeżeli w jej obrębie występuje zapytanie zagnieżdżone, generuje ono wartość, która staje się częścią każdego wiersza wyniku. Tego rodzaju podzapytania określane są najczęściej jako **podzapytania skalarne**, ponieważ muszą zwracać dokładnie jedną wartość (jedną kolumnę i jeden wiersz) dla każdego przetwarzanego wiersza zapytania zewnętrznego.

Typowym zastosowaniem zapytań zagnieżdżonych w klauzuli SELECT jest łączenie danych szczegółowych z informacjami o charakterze agregującym. Przykładowo możliwe jest wyświetlenie danych dotyczących pojedynczych rekordów wraz ze średnią wartością określonego atrybutu, łączną liczbą powiązanych rekordów lub wartością minimalną bądź maksymalną obliczoną na podstawie innej tabeli. Mechanizm ten umożliwia prezentację informacji szczegółowych i podsumowujących w jednym zbiorze wynikowym, bez konieczności wykonywania wielu odrębnych zapytań.

W wielu przypadkach podzapytania w klauzuli SELECT mają charakter **skorelowany**, co oznacza, że odwołują się do kolumn zapytania zewnętrznego. W rezultacie takie podzapytanie wykonywane jest wielokrotnie – osobno dla każdego wiersza przetwarzanego przez zapytanie główne. Choć podzapytania skorelowane zapewniają dużą elastyczność i siłę wyrazu, ich stosowanie może wiązać się z konsekwencjami wydajnościowymi, zwłaszcza w przypadku przetwarzania dużych zbiorów danych. Zrozumienie mechanizmu korelacji oraz jej wpływu na sposób wykonania zapytania ma zatem istotne znaczenie.

Zapytania zagnieżdżone w klauzuli SELECT podlegają określonym ograniczeniom składniowym i semantycznym. Przede wszystkim podzapytanie użyte jako wyrażenie kolumnowe musi zwracać pojedynczą wartość. Zwrócenie więcej niż jednego wiersza skutkuje błędem wykonania. Podzapytanie musi być ujęte w nawiasy oraz – w celu

zwiększenia czytelności – może wymagać nadania aliasu. W przypadku zapytań grupujących należy również pamiętać, że każda kolumna występująca w klauzuli SELECT, która nie jest objęta funkcją agregującą, musi zostać uwzględniona w klauzuli GROUP BY, niezależnie od obecności zapytań zagnieżdżonych.

Należy również uwzględnić logiczną kolejność przetwarzania zapytania SQL. Ze względu na to, że klauzula SELECT jest przetwarzana po klauzulach FROM, WHERE oraz GROUP BY, podzapytania umieszczone w SELECT nie wpływają bezpośrednio na wybór wierszy, lecz jedynie na sposób prezentacji danych w wyniku końcowym. Zrozumienie tej zależności jest kluczowe dla poprawnego konstruowania zapytań.

Z punktu widzenia wydajności nadmierne stosowanie podzapytań skorelowanych w klauzuli SELECT może prowadzić do wielokrotnego wykonywania zapytania wewnętrznego, co może negatywnie wpływać na czas realizacji zapytania. W niektórych sytuacjach równoważny rezultat można uzyskać poprzez zastosowanie złączeń lub tabel pochodnych, które mogą być przetwarzane efektywniej przez optymalizator zapytań. Niezbędna jest zatem umiejętność oceny alternatywnych konstrukcji zapytań.

Po zapoznaniu się z treścią niniejszego rozdziału student zdobędzie pogłębioną wiedzę na temat roli i zastosowań zapytań zagnieżdżonych w klauzuli SELECT. Nauczy się konstruować podzapytania skalarne, rozróżniać ich formy skorelowane i nieskorelowane oraz integrować dane agregujące i powiązane w jednym zbiorze wynikowym.

Student nabędzie także umiejętności respektowania ograniczeń składniowych, zapewniania poprawności semantycznej zapytań oraz interpretowania logicznej kolejności ich przetwarzania. Ponadto rozwinie kompetencje w zakresie oceny wpływu zastosowanych konstrukcji na wydajność zapytań oraz doboru odpowiednich rozwiązań w zależności od charakteru analizowanego problemu.

6.1. Zadania z rozwiązaniami i komentarzami

ZADANIE 1

Wypisać nazwy projektów oraz różnicę: budzet – ile_na_projekt_wydano.

```
select nazwa, budzet - ( select nvl(sum(liczba_godzin*stawka_za_godzine), 0)
                         from zlecenie
                         where nr_projektu = p.nr_projektu )
from projekt p;
```

Komentarz 1

Podzapytanie SELECT umieszczone w głównej klauzuli SELECT może zwracać tylko jedną wartość (jedna liczba, jeden wiersz itd.). Wynika to z tego, że pozostałe atrybuty z głównej klauzuli SELECT (nazwa, budzet) przyjmują konkretne wartości (jeden wiersz). Do tego wiersza można więc „dokleić" jedną wartość/jeden wiersz.

Komentarz 2

W podzapytaniu SELECT jest odwołanie do tabeli „zewnętrznej" projekt p. W odróżnieniu od podzapytań w klauzuli FROM z podzapytań w klauzuli SELECT można odwoływać się na zewnątrz. Wynika to z tego, że pozostałe atrybuty z głównej klauzuli SELECT (nazwa, budzet) pobierane są z tabeli określonej w klauzuli FROM (projekt p). Skoro takie odwołanie jest możliwe, to możliwe jest też odwołanie do tabeli projekt p z poziomu podzapytania w klauzuli SELECT, ponieważ podzapytanie to również znajduje się w głównej klauzuli SELECT.

ZADANIE 2

Wypisać nazwy departamentów i to, jaki procent ogółu pracowników stanowią osoby zatrudnione w danym departamencie.

```
select nazwa, round( ( select count(id_pracownika)
                       from pracownik
                       where nr_departamentu = d.nr_departamentu ) /
                ( select count(id_pracownika)
                from pracownik ) * 100, 0) as „procent ogolu zatrudnionych"
from departament d;
```

ZADANIE 3

Wypisać stanowiska i średnie wartości pensji dla kierowników na danym stanowisku oraz podwładnych na danym stanowisku.

```
select stanowisko, ( select avg(pensja)
      from pracownik p1
      where stanowisko = p.stanowisko
            and exists ( select 1
                  from pracownik
                  where id_kierownika = p1.id_pracownika ) ) as „Sr. pensja
                  kierownikow",
      ( select avg(pensja)
      from pracownik p1
      where stanowisko = p.stanowisko
            and not exists ( select 1
                  from pracownik
```

```
                where id_kierownika = p1.id_pracownika ) ) as „Sr. pensja
                podwladnych”
from ( select distinct stanowisko
      from pracownik ) p;
```

ZADANIE 4

Wypisać nazwy i numery departamentów oraz to, ile wynosi następujący iloraz: (liczba kierowników w danym departamencie)/(ogólna liczba kierowników). Zadanie należy rozwiązać dwoma sposobami:

a) podzapytanie w klauzuli `FROM`:

```
select nr_departamentu, nazwa, liczba1/liczba2 as iloraz
from ( select d.nr_departamentu, nazwa, count(id_pracownika) as liczba1
      from departament d left join pracownik p
            on d.nr_departamentu = p.nr_departamentu
      where exists ( select 3
            from pracownik
            where id_kierownika = p.id_pracownika )
      group by d.nr_departamentu, nazwa ),
      ( select count(id_pracownika) as liczba2
      from pracownik p
      where exists ( select 3
            from pracownik
            where id_kierownika = p.id_pracownika ) )
where liczba2 != 0
union
select nr_departamentu, nazwa, 0
from departament d
where not exists ( select 3
            from pracownik
            where nr_departamentu = d.nr_departamentu );
```

Komentarz 1

W pierwszym zapytaniu nie zostaną wybrane departamenty, w których nie ma pracowników. Użyto więc UNION, aby uwzględnić te departamenty.

b) podzapytanie w klauzuli SELECT:

```
select nr_departamentu, nazwa,
      ( select count(id_pracownika)
      from pracownik p
      where exists ( select 3
            from pracownik
            where id_kierownika = p.id_pracownika )
                  and nr_departamentu = d.nr_departamentu ) /
      ( select count(id_pracownika)
      from pracownik p
      where exists ( select 3
            from pracownik
            where id_kierownika = p.id_pracownika  )
            having count(id_pracownika) != 0 ) as iloraz
from departament d;
```

Komentarz 2

W drugim podzapytaniu w głównej klauzuli SELECT użyto HAVING bez GROUP BY. Jest to sensowne, gdy analizowana jest cała zawartość tabel wymienionych w klauzuli FROM (wynik ich połączenia), natomiast nie w kontekście grup.

ZADANIE 5

Dla każdego departamentu, w którym pracuje co najmniej dwóch kierowników, podać najbardziej obsadzone stanowiska.

```
select t1.nr_departamentu, nazwa, stanowisko
from ( select nr_departamentu, nazwa, count(id_pracownika) as
                                          liczba_departament_kierownik
      from pracownik p join departament d
            on p.nr_departamentu = d.nr_departamentu
      here exists ( select 3
            from pracownik
```

```
            where id_kierownika = p.id_pracownika )
      group by nr_departamentu ) t1 join
      ( select nr_departamentu, stanowisko, count(id_pracownika) as
                                          liczba_departament_stanowisko
      from pracownik p
      group by nr_departamentu, stanowisko ) t2
            on t1.nr_departamentu = t2.nr_departamentu
where liczba_departament_kierownik >= 2
      and liczba_departament_stanowisko = ( select
                                    max(liczba_departament_stanowisko)
                                    from t2
                                    where nr_departamentu = t1.nr_departamentu );
```

ZADANIE 6

Dla każdego departamentu, w przypadku którego najwcześniejsza data zatrudnienia nie jest ogólną najwcześniejszą datą, podać nazwę departamentu oraz:

a) numer przedziału zarobków, który dominuje w danym departamencie;
b) jaki procent ogółu przedziałów zarobków w danym departamencie nie występuje.

```
select min(nazwa) as nazwa, nr_przedzialu,
      100 - ( select count(distinct nr_przedzialu)
      from pracownik join poziom_zarobkow
            on pensja+nvl(premia,0) between dolna_granica
      and gorna_granica
      where nr_departamentu = d.nr_departamentu )/
      ( select count(nr_przedzialu)
      from poziom_zarobkow ) * 100 as „procent niewystepujacych
      przedzialow”
from departament d join pracownik p
      on d.nr_departamentu = p.nr_departamentu
      join poziom_zarobkow
      on pensja+nvl(premia,0) between dolna_granica and gorna_granica
where ( select min(data_zatrudnienia)
      from pracownik
      where nr_departamentu = d.nr_departamentu ) !=
      ( select min(data_zatrudnienia)
      from pracownik )
```

```
group by d.nr_departamentu, nr_przedzialu
having count(id_pracownika) = ( select max(count(id_pracownika))
            from pracownik join poziom_zarobkow
                  on pensja+nvl(premia,0) between dolna_granica
                  and gorna_granica
      where nr_departamentu = d.nr_departamentu
      group by nr_przedzialu );
```

7. DDL, DML

Pełne zrozumienie języka SQL nie ogranicza się wyłącznie do konstruowania zapytań służących do pobierania danych. W relacyjnych systemach baz danych równie istotne jest zrozumienie zasad tworzenia i modyfikowania struktur bazy danych oraz kontrolowanego wprowadzania zmian w jej zawartości. Zadania te realizowane są przez dwie podstawowe części języka SQL: **język definicji danych (ang. *Data Definition Language* – DDL)** oraz **język manipulacji danymi (ang. *Data Manipulation Language* – DML)**. Oba te komponenty stanowią fundament zarządzania relacyjną bazą danych na poziomie strukturalnym i operacyjnym.

Język definicji danych odpowiada za tworzenie oraz modyfikowanie obiektów schematu bazy danych. Przy użyciu instrukcji DDL definiowana jest logiczna struktura bazy danych, w tym tabele, ograniczenia integralności oraz relacje pomiędzy obiektami. Do podstawowych poleceń DDL należą `CREATE`, `ALTER` oraz `DROP`, które umożliwiają odpowiednio tworzenie nowych obiektów, modyfikowanie istniejących struktur oraz ich usuwanie. W procesie definiowania tabel szczególne znaczenie mają właściwy dobór typów danych, określenie kluczy głównych i obcych oraz zdefiniowanie ograniczeń integralności, takich jak `NOT NULL` czy `UNIQUE`. Decyzje te mają bezpośredni wpływ na spójność, poprawność oraz długoterminową utrzymywalność systemu bazodanowego.

Język manipulacji danymi koncentruje się natomiast na operacjach wykonywanych na danych przechowywanych w strukturach zdefiniowanych przez DDL. Do podstawowych instrukcji DML należą `INSERT`, `UPDATE`, `DELETE` oraz `SELECT`, z których każda pełni określoną funkcję w zarządzaniu zawartością bazy danych. Instrukcja `INSERT` umożliwia wprowadzanie nowych rekordów, `UPDATE` pozwala na modyfikację istniejących danych, `DELETE` służy do usuwania wybranych rekordów, natomiast `SELECT` umożliwia ich pobieranie w celach analitycznych lub raportowych. Instrukcje DML działają w ramach ograniczeń zdefiniowanych w schemacie bazy danych i muszą respektować reguły integralności oraz zgodności typów danych.

W niniejszym rozdziale szczególny nacisk położony zostanie na praktyczne zastosowanie instrukcji DDL i DML w środowisku relacyjnej bazy danych. Studenci poznają zasady projektowania struktur tabel odpowiadających modelowi logicznemu danych, definiowania kluczy i relacji, modyfikowania schematu bazy danych oraz kontrolowanego wprowadzania zmian w jej zawartości.

Po zapoznaniu się z treścią rozdziału student zdobędzie umiejętności projektowania i implementowania struktur relacyjnej bazy danych z wykorzystaniem instrukcji DDL, a także definiowania kluczy głównych i obcych oraz innych ograniczeń integralności. Nabędzie również kompetencje w zakresie wstawiania, aktualizowania i usuwania danych przy użyciu instrukcji DML, z zachowaniem zasad spójności oraz kontroli transakcji. Student będzie potrafił identyfikować i unikać typowych błędów składniowych oraz logicznych związanych z operacjami na strukturze i zawartości bazy danych.

7.1. Zadania z rozwiązaniami i komentarzami

ZADANIE 1

Stworzyć tabele (bez tworzenia kluczy):

a) `Autor`
`id_autora`
`imie`
`nazwisko`

b) `Ksiazka`
`id_ksiazki`
`tytul`
`cena`
`data_wydania`

c) `Autor_ksiazka`
`id_autora`
`id_ksiazki`

Podkreślone atrybuty mają definiować klucze główne relacji.

```
create table autor (
id_autora int,
imie varchar2(50),
nazwisko varchar2(150) );

create table ksiazka (
id_ksiazki int,
tytul varchar2(200),
cena int,
data_wydania date );

create table autor_ksiazka (
id_autora int,
id_ksiazki int);
```

ZADANIE 2

Nałożyć ograniczenia na nowo powstałe tabele (nadać wybranym ograniczeniom własne nazwy):

1. Klucze główne (kolumny podkreślone wchodzą w skład klucza głównego danej tabeli).

Komentarz 1

Klucz główny to jedno lub więcej pól (kolumn, atrybutów) w tabeli jednoznacznie identyfikujących każdy wiersz (rekord) w tej tabeli. Klucz główny musi spełniać następujące warunki:

a) **unikalność**: żaden rekord w tabeli nie może mieć tej samej wartości klucza głównego co inny rekord – gwarantuje to, że każdy wiersz jest unikalny;
b) **niepustość**: wartości klucza głównego nie mogą być puste (`NULL`), każdy rekord musi mieć przypisaną unikalną wartość klucza głównego;
c) **stabilność**: wartości klucza głównego powinny być stałe i niezmienne przez cały czas życia rekordu, aby uniknąć problemów z integracją danych.

Klucz główny odgrywa kluczową rolę w relacyjnych bazach danych, ponieważ umożliwia tworzenie relacji między tabelami oraz zapewnia integralność danych.

Komentarz 2

Definicja klucza wraz z definicją tabeli:

```
create table x (
 id_ksiazki int primary key,
      ..... );
```

Tutaj kluczem głównym tabeli `x` stanie się `id_ksiazki`.

Każde ograniczenie, które jest nakładane, ma swoją nazwę – w tym przypadku będzie to nazwa systemowa w stylu `SYS_028649163234` – nic niemówiąca użytkownikowi.

W celu efektywnego zarządzania bazą trzeba mieć kontrolę nad nałożonymi ograniczeniami. Z użyciem słowa kluczowego `CONSTRAINT` nadaje się ograniczeniu nazwę:

```
create table x (
 id_ksiazki int constraint kl_glowny_x primary key,
      ..... );
```

Klucz podwójny (wielokrotny):

```
create table y (
( id_ksiazki int primary key,
      id_autora int primary key );
```

Próba stworzenia tabeli z dwoma kluczami głównymi stanowi poważny błąd. Jedna tabela może mieć zdefiniowany tylko jeden klucz główny:

```
create table y (
( id_ksiazki int,
      id_autora int,
      constraint kl_glowny_y primary key(id_ksiazki, id_autora));
```

Rozwiązanie:

```
alter table ksiazka add constraint kl_glowny_ksiazka primary key
      (id_ksiazki);


alter table autor add constraint kl_glowny_autor primary key (id_autora);


alter table autor_ksiazka add constraint kl_glowny_autor_ksiazka primary
      key
(id_autora, id_ksiazki);
```

2. Klucze obce (z opcją `ON DELETE CASCADE`).

```
alter table autor_ksiazka add constraint kl_obcy_autor_ksiazka__autor
foreign key (id_autora) references autor(id_autora) on delete cascade;


alter table autor_ksiazka add constraint kl_obcy_autor_ksiazka__ksiazka
foreign key (id_ksiazki) references ksiazka(id_ksiazki) on delete cascade;
```

3. `Data_wydania`, przy czym domyślną datą ma być data dzisiejsza, bez nazwy ograniczenia.

```
alter table ksiazka modify data_wydania default sysdate;
```

Wraz z definicją tabeli:

```
create table ksiazka (
 ...
data_wydania date default sysdate,
 ... );
```

4. Cena książki powinna być większa od 0.

```
alter table ksiazka add constraint ksiazka_check_cena check ( cena > 0 );
```

Wraz z definicją tabeli:

```
create table ksiazka (
 ...
cena number constraint ksiazka_check_cena check ( cena > 0 ),
 ... ;
```

5. Imię i nazwisko autora oraz tytuł książki nie mogą być puste.

```
alter table autor add constraint autor_check_imie check ( imie is not null );

alter table autor add constraint autor_check_nazwisko check ( nazwisko
      is not null );

alter table ksiazka add constraint ksiazka_check_tytul check ( tytul is not
      null );
```

Komentarz 3

Wszystkie ograniczenia można definiować po zdefiniowaniu wszystkich kolumn:

```
create table TABELA (
      kolumna1 ...,
      kolumna2 ...,
      ...
      kolumnaN ...,
      constraint tabela_kl_glowny primary key (kolumna1, kolumna2),
      constraint tabela_check_kolumna3 check (kolumna3 is not null),
      constraint tabela_check_kolumna4 check (kolumna4 > 0) );
```

ZADANIE 3

Wstawić do każdej z tabel przykładowe wiersze.

Komentarz 1

W związku z tym, że data wydania ma ustawioną wartość domyślną, nie trzeba jej wstawiać:

```
insert into ksiazka (id_ksiazki, tytul, cena) values ......
```

W nawiasie podaje się atrybuty/kolumny, które będą uzupełniane.

Rozwiązanie:

```
insert into ksiazka values
(1, ‘Tytul 1’, 34.50, to_date(‘12/05/2019’, ‘DD/MM/YYYY’));
insert into ksiazka values
(2, 'Tytul 2', 23.99, to_date('24/05/2022', 'DD/MM/YYYY'));
insert into ksiazka values
(3, 'Tytul 3', 41.20, to_date('09/05/2018', 'DD/MM/YYYY'));
insert into ksiazka values
(4, 'Tytul 4', 37.50, to_date('22/05/2020', 'DD/MM/YYYY'));
insert into ksiazka values
(5, 'Tytul 5', 30.50, to_date('15/05/2022', 'DD/MM/YYYY'));
```

Powyżej wykorzystano funkcję `to_date` – powodem jest nieznajomość formatu daty na serwerze. Aby uniknąć błędów składniowych, zamienia się tekst, na przykład: `‘15/05/2022’` tłumaczy się na datę `‘DD/MM/YYYY’` (pierwsze dwie litery to dzień itd.).

```
insert into autor values
(1, ‘IMIE 1’, ‘NAZWISKO 1’);
insert into autor values
(2, ‘IMIE 2’, ‘NAZWISKO 2’);
insert into autor values
(3, ‘IMIE 2’, ‘NAZWISKO 3’);
insert into autor values
(4, ‘IMIE 4’, ‘NAZWISKO 4’);

insert into autor_ksiazka values (2, 1);
insert into autor_ksiazka values (4, 1);

insert into autor_ksiazka values (1, 2);

insert into autor_ksiazka values (2, 3);
insert into autor_ksiazka values (3, 3);
```

```
insert into autor_ksiazka values (4, 4);


insert into autor_ksiazka values (2, 5);
insert into autor_ksiazka values (1, 5);
```

ZADANIE 4

Do tabeli `Ksiazka` dodać kolumnę `wydawca` (typ danych: tekst).

```
alter table ksiazka add wydawca varchar2(250);
```

ZADANIE 5

Nałożyć na kolumnę `wydawca` ograniczenie o nazwie `Wyd`, które uniemożliwi wstawienie wydawcy o nazwie będącej ciągiem znaków krótszym niż 10.

```
alter table ksiazka add constraint wyd check (length(wydawca)>=10);
```

ZADANIE 6

Usunąć ograniczenie nałożone na kolumnę `wydawca`.

```
alter table ksiazka drop constraint wyd;
```

ZADANIE 7

Przy pomocy jednego polecenia stworzyć tabelę `Autor1` i wstawić wszystkie wiersze z tabeli `Autor`. Sprawdzić, czy na tabelę `Autor1` są nałożone jakieś ograniczenia.

```
create table autor1 as select * from autor;


select * from user_constraints where table_name = 'AUTOR1';
```

Dla porównania:

```
select * from user_constraints where table_name = ‘AUTOR’;
```

ZADANIE 8

Usunąć z bazy wszystkich tych autorów, którzy zawsze byli tylko współautorami.

```
delete from autor a
where not exists ( select 3
```

```
        from autor_ksiazka b
        where id_autora = a.id_autora
              and ( select count(id_autora)
                    from autor_ksiazka
                    where id_ksiazki = b.id_ksiazki ) = 1 );
```

ZADANIE 9

Podnieść o 10% cenę wszystkich książek, które zostały napisane przez co najmniej dwóch autorów.

```
update ksiazka a
set cena = 1.1 * cena
where ( select count(id_autora)
        from autor_ksiazka
        where id_ksiazki = a.id_ksiazki ) >= 2;
```

ZADANIE 10

Dodać do tabeli `Ksiazka` kolumnę `ile_wydan`. Wstawić do tej kolumny wartości (ilość wydań – ile razy dany tytuł pojawia się w tabeli).

```
alter table ksiazka add ile_wydan int;
```

```
update ksiazka a
set ile_wydan = ( select count(id_ksiazki)
                  from ksiazka
                  where tytul = a.tytul );
```

ZADANIE 11

Dodać do tabeli `Autor` kolumny: `ile_pojedynczo`, `ile_wspolautor`. Wstawić do tych kolumn następujące wartości: ile książek dany autor napisał samodzielnie, a ile jako współautor.

```
alter table autor add ile_pojedynczo int;
alter table autor add ile_wspolautor int;
```

```
update autor a
```

```
set ile_pojedynczo = ( select count(id_ksiazki)
                       from autor_ksiazka b
                       where id_autora = a.id_autora
                             and ( select count(id_autora)
                                   from autor_ksiazka
                                   where id_ksiazki = b.id_ksiazki ) = 1 );

update autor a
set ile_wspolautor = ( select count(id_ksiazki)
                       from autor_ksiazka b
                       where id_autora = a.id_autora
                             and ( select count(id_autora)
                                   from autor_ksiazka
                                   where id_ksiazki = b.id_ksiazki ) > 1 );
```

ZADANIE 12

Usunąć utworzone tabele.

```
drop table ksiazka cascade constraints;
drop table autor cascade constraints;
drop table autor_ksiazka;
```

7.2. Wskazówki dotyczące składni SQL

Tworzenie tabel

```
CREATE TABLE nazwa_tabeli
(
n_kolumny_1 typ_danych_kolumny_1 [domyślna wartość kolumny_1]
      [ograniczenia kolumny_1],
...
n_kolumny_n typ_danych_kolumny_n [domyślna wartość kolumny_n]
      [ograniczenia kolumny_n],
ograniczenia_tabeli (po przecinkach)
);
CREATE TABLE nazwa_tabeli(nazwy kolumn po przecinkach) AS (podzapytanie);
```

Domyślna wartość kolumny

Na przykład `numer NUMBER(4) DEFAULT 0`.

Ograniczenia kolumny

`CONSTRAINT nazwa_ograniczenia ograniczenie`

Forma `CONSTRAINT nazwa_ograniczenia` nie jest konieczna, ale pozwala nazwać ograniczenie, co czyni schemat bardziej przejrzystym.

CHECK (ograniczenia)

Na przykład `check (n_kolumny > 0)`, `check (n_kolumny LIKE ...)` itd.

Jest to ograniczenie statyczne – nie można używać podzapytań, natomiast można `OR`, `AND`.

NOT NULL (nie może być `NULL`) lub NULL (może być `NULL`)

Na przykład `nazwa kolumny NOT NULL`.

Jeśli definiowany jest klucz główny/obcy złożony z jednej kolumny, to można ten klucz „ustanowić" przy definiowaniu kolumny:

```
[CONSTRAINT nazwa_ograniczenia] PRIMARY KEY
[CONSTRAINT nazwa_ograniczenia] FOREIGN KEY REFERENCES tabela (nazwa_
      kolumny) [ON DELETE CASCADE]
```

Jeśli definiowany jest klucz główny złożony z paru kolumn, to można ten klucz „ustanowić" tylko po definicjach wszystkich kolumn:

```
[CONSTRAINT nazwa_ograniczenia] PRIMARY KEY (kolumna_1,...,kolumna_n)
[CONSTRAINT nazwa_ograniczenia] FOREIGN KEY (nazwa kolumny w tworzonej
      tabeli) REFERENCES tabela (nazwa_kolumny) [ON DELETE CASCADE]
```

ON DELETE CASCADE

Przy usuwaniu wierszy z tabeli nadrzędnej wszystkie wiersze w tabelach podrzędnych zawierające odpowiadające wartości w kolumnie łączącej te tabele zostaną usunięte.

Na przykład przy usunięciu wiersza z tabeli `departament` z numerem departamentu równym 10 wszystkie wiersze z tabeli `pracownik`, gdzie numer departamentu jest równy 10, także zostaną usunięte.

Wstawianie danych

```
INSERT INTO nazwa_tabeli [(nazwa_kolumny_1,...,nazwa_kolumny_n)] VALUES
      (war_1,...,war_n);
```

```
INSERT INTO nazwa_tabeli [(nazwa_kolumny_1,...,nazwa_kolumny_n)]
      ZAPYTANIE;

ZAPYTANIE: SELECT...
```

Dodawanie kolumn, modyfikowanie

```
ALTER TABLE nazwa_tabeli
ADD [nazwa_kolumny typ_danych] [domyślne] [ograniczenia_kolumny];

ALTER TABLE nazwa_tabeli
MODIFY [nazwa_kolumny typ_danych] [domyślne] [ograniczenia_kolumny];

ALTER TABLE nazwa_tabeli
MODIFY [ograniczenia_tabeli] ENABLE/DISABLE;

ALTER TABLE nazwa_tabeli
DROP CONSTRAINT nazwa_ograniczenia;

ALTER TABLE nazwa_tabeli
DROP PRIMARY KEY [CASCADE];
```

Usuwanie wierszy

```
DELETE FROM nazwa_tabeli
WHERE warunki;
```

W klauzuli `WHERE` określane są warunki, które muszą być spełnione przez wiersze, aby zostać usunięte.

Zmiana wartości w wierszach

```
UPDATE nazwa_tabeli
SET nowe_wartosci
WHERE warunki;
```

W klauzuli `WHERE` określane są warunki, które muszą być spełnione przez wiersze, aby zostać zmienione.

`nowe_wartosci`: `nazwa_kolumny = wartość` lub `nazwa_kolumny = (podzapytanie)`

Usuwanie tabel

```
DROP TABLE nazwa_tabeli [CASCADE CONSTRAINTS];
```

`CASCADE CONSTRAINTS` – potrzebne, gdy w tabelach podrzędnych klucze obce nie są zdefiniowane z opcją `ON DELETE CASCADE`.

Informacje o tabelach, kolumnach

```
SELECT CONSTRAINTNAME, SEARCHCONDITION (lub *)
FROM USERCONSTRAINTS
WHERE TABLENAME='&TABELA';
```

```
SELECT CONSTRAINTNAME, SEARCHCONDITION, CONSTRAINTTYPE
FROM USERCONSTRAINTS
WHERE TABLENAME='PRACOWNIK';
```

```
SELECT columnname, DATADEFAULT from usertabcolumns where
      tablename='NAZWATABELI';
SELECT columnname, DATADEFAULT from userconscolumns where
      tablename='NAZWATABELI';
```

8. Perspektywy/widoki

Perspektywy/widoki wywodzą się z modelu relacyjnego zaproponowanego w [6], w którym relacje definiowane są jako zbiory wierszy powstałych w wyniku zastosowania wyrażeń algebry relacji. Widok można interpretować jako relację pochodną (ang. *derived relation*), definiowaną na podstawie wyrażenia relacyjnego (zapytania), która nie jest przechowywana fizycznie jako odrębny zbiór danych (chyba że ma postać materializowaną), lecz wyznaczana dynamicznie w momencie jej użycia. W implementacji SQL widok definiowany jest przy użyciu instrukcji `CREATE VIEW` na podstawie zapytania `SELECT`. Użytkownik może odwoływać się do widoku w sposób analogiczny jak w przypadku odwołania do zwykłej tabeli.

Po zapoznaniu się z treścią niniejszego rozdziału student nabędzie umiejętności definiowania widoków przy użyciu instrukcji `CREATE VIEW`, modyfikowania i usuwania widoków oraz wykorzystywania ich w zapytaniach `SELECT`.

8.1. Zadania z rozwiązaniami

ZADANIE 1

Stworzyć perspektywę zawierającą informacje o nazwach departamentów, ich numerach i liczbach osób w nich zatrudnionych.

```
create or replace view zad1(nr_departamentu, nazwa, liczba_pracownikow) as
select  d.nr_departamentu, nazwa, count(id_pracownika)
      from departament d left join pracownik p
            on  d.nr_departamentu = p.nr_departamentu
      group by  d.nr_departamentu, nazwa;
```

ZADANIE 2

Wykorzystując perspektywę utworzoną w zadaniu 1, wskazać nazwy departamentów, w których pracuje najwięcej osób.

```
select nr_departamentu, nazwa
```

```
from zad1
where liczba_pracownikow = ( select max(liczba_pracownikow)
                             from zad1 );
```

ZADANIE 3

Stworzyć perspektywę zawierającą informacje o nazwach departamentów, ich numerach, a także stanowiskach i liczbach osób na danym stanowisku w danym departamencie.

```
create or replace view zad3 (nr_departamentu, nazwa, stanowisko, liczba_
      pracownikow) as
      select  d.nr_departamentu, nazwa, stanowisko, count(id_pracownika)
      from departament d join pracownik p
            on  d.nr_departamentu =  p.nr_departamentu
      group by  d.nr_departamentu, nazwa, stanowisko;
```

ZADANIE 4

Wykorzystując perspektywy utworzone w zadaniach 1 i 3, wyznaczyć, jaki procent stanowią osoby na określonych stanowiskach w określonych departamentach.

```
select  x.nr_departamentu, x.nazwa, y.stanowisko,
      round( y.liczba_pracownikow/x.liczba_pracownikow * 100, 0) as procent
from zad1 x, zad3 y
where  x.nr_departamentu =  y.nr_departamentu
      and x.liczba_pracownikow != 0
order by  x.nr_departamentu;
```

ZADANIE 5

Stworzyć perspektywę zawierającą informacje o poziomach zarobków, nazwiskach pracowników oraz różnicach pomiędzy pensjami a górnymi granicami ich poziomów.

```
create or replace view zad5(nr_przedzialu, id_pracownika, nazwisko,
      roznica) as
      select nr_przedzialu, id_pracownika, nazwisko, gorna_granica
     - pensja
      from poziom_zarobkow, pracownik
      where pensja+nvl(premia,0) between dolna_granica and gorna_granica;
```

ZADANIE 6

Wykorzystując perspektywę utworzoną w zadaniu 5, wyznaczyć nazwiska osób, których pensje znajdują się najbliżej górnej granicy odpowiedniego poziomu zarobków.

```
select id_pracownika, nazwisko, nr_przedzialu, roznica
from zad5 x
where roznica = ( select min(roznica)
                  from zad5
                  where nr_przedzialu =  x.nr_przedzialu );
```

ZADANIE 7

Stworzyć perspektywę zawierającą informacje o nazwiskach i identyfikatorach osób najwcześniej zatrudnionych.

```
create or replace view zad7(id_pracownika, nazwisko) as
      select id_pracownika, nazwisko
      from pracownik
      where data_zatrudnienia = ( select min(data_zatrudnienia)
                                  from pracownik );
```

ZADANIE 8

Stworzyć perspektywę zawierającą informacje o nazwach projektów i ich numerach, pod warunkiem, że nad tymi projektami pracowały osoby najwcześniej zatrudnione.

```
create or replace view zad8(nr_projektu, nazwa) as
      select nr_projektu, nazwa
      from projekt p
      where exists ( select 3
            from zlecenie
            where id_pracownika in ( select id_pracownika
                                     from zad7 )
                  and nr_projektu =  p.nr_projektu );
```

ZADANIE 9

Wykorzystując perspektywy utworzone w zadaniach 7 i 8, wypisać nazwiska pracowników, którzy pracowali nad projektem, nad którym pracowały osoby najwcześniej zatrudnione, ale same najwcześniej zatrudnionymi nie są.

```
select id_pracownika, nazwisko
from pracownik
where id_pracownika not in ( select id_pracownika
                             from zad7 )
      and id_pracownika in ( select id_pracownika
                             from zlecenie
                             where nr_projektu in ( select nr_projektu
                                                    from zad8 ) );
```

ZADANIE 10

Wypisać nazwiska osób (używając do tego celu dowolnych utworzonych perspektyw), które pracowały największą liczbę godzin nad dokładnie jednym projektem (dla poszczególnych osób istnieje tylko jeden projekt, nad którym dana osoba pracowała największą liczbę godzin spośród wszystkich pracujących nad danym projektem) i jako jedyne najwcześniej zaczęły nad nim pracę.

```
create or replace view zad10a (id_pracownika, nazwisko, nr_projektu ) as
select  p.id_pracownika, nazwisko, nr_projektu
      from pracownik p join zlecenie z
            on  p.id_pracownika =  z.id_pracownika
      where liczba_godzin = ( select max(liczba_godzin)
                              from zlecenie
                              where nr_projektu =  z.nr_projektu );

create or replace view zad10b (id_pracownika, nazwisko, nr_projektu ) as
select id_pracownika, nazwisko, nr_projektu
      from zad10a x
      where 1 = ( select count(nr_projektu)
                  from zad10a
                  where id_pracownika =  x.id_pracownika );

create or replace view zad10c (id_pracownika, nazwisko, nr_projektu) as
```

```
      select  p.id_pracownika, nazwisko, nr_projektu
      from pracownik p join zlecenie z
            on  p.id_pracownika =  z.id_pracownika
      where data_rozpoczecia = ( select min(data_rozpoczecia)
                              from zlecenie
                              where nr_projektu = z.nr_projektu );


create or replace view zad10d (id_pracownika, nazwisko, nr_projektu ) as
      select id_pracownika, nazwisko, nr_projektu
      from zad10c x
      where 1 = ( select count(id_pracownika)
                  from zad10a
                  where nr_projektu =  x.nr_projektu );


select *
from zad10a x, zad10c y
where  x.nr_projektu =  y.nr_projektu
      and  x.id_pracownika =  y.id_pracownika;
```

8.2. Wskazówki dotyczące składni SQL

```
CREATE [or replace] VIEW nazwa_perspektywy AS SELECT ... ;
```

Opcja `or replace` – gdy istnieje już perspektywa o podanej nazwie (`nazwa_perspektywy`), zostanie ona nadpisana nowo tworzoną perspektywą.

Nadanie nowych nazw wybieranym kolumnom (konieczne, gdy wynikiem zapytania jest działanie jakiejś funkcji, np. `COUNT`):

```
CREATE VIEW nazwa_perspektywy (nazwa1, nazwa2, ...) AS SELECT ... ;


SELECT * FROM nazwa_perspektywy;
```

9. Podsumowanie

Niniejszy skrypt dostarcza kompleksowego przeglądu kluczowych elementów języka SQL, niezbędnych do efektywnego zarządzania i analizy danych w relacyjnych bazach danych. Omówione tematy obejmują zarówno podstawowe, jak i zaawansowane aspekty pracy z SQL, co pozwala na lepsze zrozumienie oraz praktyczne zastosowanie tego języka.

W pierwszej części skryptu zaprezentowano zapytania proste, które stanowią fundament interakcji z bazą danych. Zrozumienie składni zapytań, a także funkcji filtrowania i sortowania danych przy użyciu klauzul `SELECT`, `WHERE` oraz `ORDER BY` jest kluczowe dla skutecznego wydobywania informacji. Przedstawiono także metody łączenia tabel za pomocą złączeń (`JOIN`). Ponadto opisano różne typy złączeń, takie jak `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN` oraz `FULL JOIN`, które umożliwiają łączenie danych z wielu tabel w celu uzyskania bardziej kompleksowych zestawów wyników.

Kolejna sekcja była skoncentrowana na zapytaniach grupujących, które umożliwiają agregację danych. Opisano rolę klauzuli `GROUP BY` oraz funkcji agregujących, takich jak `SUM()`, `COUNT()`, `AVG()`, `MAX()`, i `MIN()`. Wprowadzono również klauzulę `HAVING`, która pozwala na filtrację wyników po przeprowadzeniu agregacji.

Podzapytania zostały omówione jako zaawansowane narzędzie do dynamicznego pobierania danych. Przedstawiono różne rodzaje podzapytań, w tym podzapytania skorelowane i nieskorelowane. Zrozumienie podzapytań jest niezbędne do tworzenia bardziej złożonych i funkcjonalnych zapytań.

W sekcji dotyczącej łączeń zbiorów opisano operatory `UNION`, `MINUS` oraz `INTERSECT`, pozwalające na porównywanie i łączenie wyników z różnych zapytań. Operacje te są niezwykle przydatne w analizie danych, umożliwiając elastyczne zarządzanie zestawami wyników.

W części dotyczącej zapytań w klauzuli `FROM` omówiono zaś znaczenie korzystania z wyników podzapytań jako z tabel. Umożliwia to dalsze operacje na uzyskanych danych, co zwiększa możliwości analityczne. Przykłady ilustrują, jak podzapytania w klauzuli `FROM` mogą być używane do tworzenia wirtualnych tabel, na podstawie których można przeprowadzać agregacje i inne operacje.

W kontekście zapytań w klauzuli `SELECT` omówiono znaczenie podzapytań, które mogą być osadzone wewnątrz głównego zapytania. Umożliwiają one uzyskanie dodatkowych wartości w ramach wyników, co zwiększa elastyczność w analizie danych. Przykłady pokazują, jak podzapytania mogą dostarczać agregaty lub inne wartości na podstawie danych w tabelach, dzięki czemu możliwe są bardziej złożone operacje.

Scharakteryzowano również języki DDL (ang. *Data Definition Language*) oraz DML (ang. *Data Manipulation Language*), które odgrywają kluczowe role w definiowaniu struktury bazy danych oraz manipulacji danymi. Zrozumienie tych języków jest niezbędne do skutecznego zarządzania bazami danych.

Na zakończenie skryptu omówiono perspektywy pozwalające na tworzenie wirtualnych tabel na podstawie zapytań, stanowiące wartościowe narzędzie służące do uproszczenia dostępu do danych oraz umożliwiające ochronę wrażliwych informacji.

Bibliografia

1. Mushingairi, D. (2025). *What Is ANSI SQL and Why You Should Use It*. https://blog.devart.com/what-is-ansi-sql.html
2. Kumar, S. (2022). *SQL vs NoSQL vs NewSQL: An In-depth Literature Review*. https://blog.reachsumit.com/posts/2022/06/sql-nosql-newsql
3. Putra, A. (2025). *What Is a Graph Database? (Animated + Practice)*. https://www.youtube.com/watch?v=-6Xc2_IOh-0
4. Beaulieu, A. (2020). *Learning SQL: Generate, Manipulate, and Retrieve Data* (3rd ed.). O'Reilly Media.
5. Toolbox, D. E. (2025). *Correlated and non-correlated subqueries*. https://www.youtube.com/watch?v=HwUPurpgs2k
6. Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, *13*(6), 377–387.

Abstract

The academic textbook *SQL – Practical Classes* is designed for undergraduate students of Computer Science and provides a structured, practice-oriented introduction to the SQL language at an intermediate level. The textbook is based on the ANSI SQL standard and implemented in the Oracle database environment. Its primary objective is to develop both conceptual understanding and practical competence in constructing correct, efficient, and logically coherent SQL queries.

The material is organized progressively, beginning with simple `SELECT` queries, including filtering with the `WHERE` clause, logical operators, ordering of results, and column expressions. Subsequent chapters address grouping queries with `GROUP BY` and `HAVING`, emphasizing the distinction between row-level and group-level filtering. The textbook then introduces subqueries in the `WHERE` and `HAVING` clauses, including the use of `IN`, `NOT IN`, `ANY`, `ALL`, `EXISTS`, and `NOT EXISTS`, as well as correlated and non-correlated subqueries.

Further chapters examine set operations such as `UNION`, `UNION ALL`, `INTERSECT`, and `MINUS`, highlighting their theoretical foundations in relational algebra and their performance implications. The construction of queries in the `FROM` and `SELECT` clauses is discussed in detail, including joins (`INNER`, `LEFT`, `RIGHT`, `FULL`), derived tables, aliasing, syntactic constraints, and logical query processing order.

The textbook also covers Data Definition Language (DDL) and Data Manipulation Language (DML), focusing on schema creation, modification, integrity constraints, transactional behavior, and controlled data operations. The final chapter introduces views, including their theoretical basis, practical applications in abstraction and security, and distinctions between standard and materialized views.

Upon completion of the textbook, students acquire the ability to design structured SQL queries, manage relational database schemas, manipulate data consistently, apply set operations and subqueries effectively, and understand both syntactic rules and logical execution principles. The textbook provides a comprehensive foundation for advanced topics in database systems, data analysis, and database administration.

Spis rysunków

Politechnika
Białostocka