Baza danych Oracle - programowanie

Wykładowca: *dr inż. Zbigniew Staszak*
pok. 122, bud. C3

Plan wykładu:

1. Informację wstępne.
2. Język SQL - dialekt Oracle'a.
3. Język PL/SQL.
4. Obiektowe rozszerzenia bazy danych Oracle.
5. Przechowywanie i ochrona danych.
6. Dostrajanie aplikacji i zapytań SQL.

1. Informacje wstępne

Nasza cywilizacja jest cywilizacją danych. Na podstawie zebranych danych fizycznych budowane są teorie naukowe, dane biznesowe służą jako podstawa podejmowania wszelkich decyzji, dane są wreszcie wartością samą w sobie – są gromadzone i przeszukiwane. Ze względu na rosnące „zasoby" danych naturalnym narzędziem do ich gromadzenia i przetwarzania stał się komputer. Systemy komputerowe, w ramach których dane są gromadzone i zarządzane nazywane są Systemami Zarządzania Bazą Danych (SZBD, ang. DBMS - Database Management Systems). Wśród tych systemów jednym z najważniejszych jest system oferowany przez firmę Oracle.

1.1. Obiekty bazy danych Oracle

Baza danych Oracle jest zbiorem danych pamiętanych w plikach. Posiada ona swoją strukturę logiczną (powiązania między zgromadzonymi danymi) i strukturę fizyczną (zestaw fizycznych obiektów, w ramach których dane są gromadzone). Konkretna realizacja bazy danych Oracle (instancja) składa się z obszaru pamięci nazywanego globalnym obszarem systemowym (ang. System Global Area - SGA) oraz z działających w tle procesów komunikujących się z obszarem SGA i plikami bazy danych.

Podstawowymi strukturami (obiektami) składowanymi w bazie danych Oracle są tabele. Można wyróżnić następujące ich typy:

- Tabele relacyjne,
- Tabele obiektowo-relacyjne,
- Tabele o strukturze indeksu,
- Tabele zewnętrzne,
- Tabele partycjonowane,
- Zmaterializowane perspektywy,
- Tabele tymczasowe,
- Klastry,
- Tabele usunięte.

Tabele relacyjne zawierają dane wprowadzane i przetwarzane przez użytkownika. Tabele obiektowo-relacyjne zawierają dane o typach definiowanych przez użytkownika, dla których można zastosować mechanizm dziedziczenia. Tabele o strukturze indeksu zawierają dane zapisywane w strukturze indeksu. Tabele zewnętrzne wykorzystywane są dla uzyskania dostępu do danych zewnętrznych o dużej objętości bez konieczności ładowania ich do bazy danych. W tabelach partycjonowanych następuje podział w ramach ich danych na partycje, którymi można oddzielnie zarządzać. Zmaterializowane perspektywy zawierają, uzyskaną w wyniku zapytania, replikę danych. W tabelach tymczasowych każdy użytkownik ma możliwość "widzenia" tylko wierszy wprowadzonych przez siebie. W strukturze klastra można zapisać dwie tabele, do których wspólnie często są kierowane zapytania. Tabele usunięte dają możliwość szybkiego odtworzenia tabel usuniętych za pomocą polecenia o specjalnej składni.

W celu uzyskania szybszego dostępu do danych zapisanych w tabelach, w systemie Oracle wykorzystywane są struktury (obiekty) składowanych w bazie danych nazywane indeksami. Można wyróżnić następujące rodzaje indeksów:

- Indeksy B*-tree,
- Indeksy bitmapowe,
- Indeksy z odwrotnym kluczem,
- Indeksy funkcyjne,
- Indeksy partycjonowane,
- Indeksy tekstowe.

Indeksy B*-tree budowane są w oparciu o drzewa binarne. Indeksy bitmapowe wykorzystywane są przede wszystkim podczas wsadowego ładownia danych (np. w hurtowniach danych). Indeksy z odwrotnym kluczem dają możliwość dynamicznego odwrócenia zaindeksowanych wartości przed ich zapisem. Indeksy funkcyjne pozwalają "oprzeć" indeks na funkcji od atrybutu. Indeksy partycjonowane wykorzystywane są do obsługi tabel partycjonowanych. Indeksy tekstowe są zbiorem tabel i indeksów utrzymywanych przez system Oracle w celu umożliwienia zaawansowanego wyszukiwania w ramach tekstu.

Oprócz tabel relacyjnych i indeksów system Oracle umożliwia wykorzystanie wielu innych struktur takich jak między innymi perspektywy (widoki), procedury, funkcje, wyzwalacze, pakiety, migawki, użytkownicy. Większość tych struktur zostanie opisanych w ramach niniejszego wykładu.

System Oracle posiada swój słownik, czyli bazę danych, w której pamiętane są dane o bazie (tzw. metadane). Mogą to być dane użytkowników, ich uprawnienia, definicje obiektów bazy danych, ograniczenia itp.

Obiekty bazy danych, które wymają fizycznego miejsca w pamięci stałej, w systemie Oracle uzyskują je w ramach tzw. przestrzeni tabel. Przestrzeń tabel składa się z jednego lub większej liczby plików. Każdy obiekt bazy danych może być zapisany w jednym takim pliku lub może być podzielony i zapisany w wielu plikach.

1.2. Metody dostępu do danych

W ramach systemu Oracle można wyróżnić następujące programowe metody dostępu do danych zapisanych w bazie:

- Język SQL,
- PL/SQL,
- Dynamiczny SQL,
- SQL*Plus,
- Java i JDBC,
- XML,
- Obiektowy SQL i PL/SQL,
- Data Pump,
- SQL*Loader,
- Zewnętrzne programy i procedury,
- UTL_MAIL.

Język SQL jest podstawowym językiem zapytań do relacyjnych baz danych. Język PL/SQL umożliwia tworzenie procedur i funkcji (podprogramów) składowanych w bazie danych a także wyzwalaczy.

W ramach jego składni można wykorzystywać plecenia SQL. Dynamiczny SQL jest wykorzystywany w ramach PL/SQL'a i pozwala definiować polecenia, uwzględniające w swojej składni w sposób dynamiczny aktualny stan bazy danych lub aktualne potrzeby użytkownika. SQL*Plus jest prostym interfejsem, w ramach którego można między innymi obsługiwać polecenia SQL i bloki PL/SQL. Dzięki możliwości wykorzystania w ramach systemu Oracle języka Java i JDBC, można definiować podprogramy składowane w bazie danych z wykorzystaniem tego języka. System Oracle pozwala na korzystanie z typów i interfejsów XML. Oferuje on także rozszerzenia SQL'a i PL/SQL'a o możliwości obiektowe (typy obiektowe, tabele obiektowe, metody). Operacje Data Pump Import i Data Pump Export umożliwiają sprawne wydobywanie danych i przenoszenie ich do innych baz danych. Do szybkiego ładowania plików do tabel Oracle można wykorzystać SQL*Loader'a. Do danych bazy danych Oracle można sięgać z zewnętrznych programów, w kodzie których można osadzać polecenia SQL. Istnieje także możliwość dołączania do Oracle bibliotek zewnętrznych podprogramów. Wykorzystywany w ramach PL/SQL'a pakiet UTL_MAIL umożliwia wysyłanie wiadomości e'mail bez konieczności obsługi protokołu SMTP.

1.3. Zarys opisu wykorzystywanej rzeczywistości.

Większość przykładów związanych z wykładem będzie oparta na bazie danych opisującej poniżej przedstawiony wycinek rzeczywistości.

Po wielu latach niezależności koty obu płci polujące na terenie wsi Wólka Mała postanowiły się zorganizować. Powstało więc stado dowodzone przez najwybitniejszego łowcę myszy o pseudonimie Tygrys. W ramach stada, pod przywództwem Tygrysa, w sposób naturalny, ukształtowała się nieformalna hierarchia kotów – każdy kot wiedział, jakiemu innemu kotu jest podporządkowany. Stado zostało dodatkowo, administracyjnie, podzielone na kilka posiadających unikalny numer i nazwę band, każda dowodzona przez wskazanego przez Tygrysa, wybitnego łowcę myszy. Każdej bandzie przydzielono

niezależny teren, na którym koty z bandy mogły organizować swoje polowania. Tygrysowi oraz członkom jego bandy zauszników, z racji pełnionych urzędów, przysługiwał przywilej polowania na całym obszarze kontrolowanym przez stado. Dla celów identyfikacji zobowiązano każdego kota do wybrania sobie unikalnego pseudonimu. Kot powinien też posiadać imię. Ustalono, że członek stada będzie co miesiąc wynagradzany przydziałem myszy za swój wkład w utrzymanie całego stada. Przydział ten będzie adekwatny do funkcji pełnionej w kociej społeczności. Funkcja ta będzie określała dolną i górną granicę przydziału myszy. Niezależnie od wielkości przydziału myszy przywódca stada, za szczególne zasługi, będzie mógł przyznać kotu, wedle własnego uznania, dodatkowy przydział myszy. Koty polowały szczęśliwie na przydzielonych sobie terenach, jednak od czasu do czasu dochodziło do incydentów z przedstawicielami innych ras. Uczestniczący w incydentach, identyfikowani przez imię, „poza rasowi" stawali się automatycznie osobistymi wrogami pokrzywdzonych kotów a ich stopień wrogości i gatunek były skrzętnie notowane. Opisywane były także, ku przestrodze kotom a niesławie „poza rasowym", wszystkie owe zdarzenia (obowiązkowo z ich datą). Zakładając jednak, że prawdziwy myśliwy potrafi unikać znanych wrogów, odnotowywano jedynie pierwszy incydent kota z konkretnym wrogiem. Z czasem koty zauważyły, że pewne „gratyfikacje" są w stanie zmniejszyć czujność wrogów. Notowano więc preferowaną przez każdego wroga "gratyfikację".

W wyniku analizy zarysowanego powyżej wycinka rzeczywistości powstał schemat bazy danych złożony z pięciu relacji o następujących, zapisanych w postaci predykatowej, schematach:

Kocury(pseudo, imie, plec, w_stadku_od, przydzial_myszy,
myszy_extra, #funkcja, #szef, #nr_bandy)
Bandy(nr_bandy, nazwa, teren, #szef_bandy)
Funkcje(funkcja, min_myszy, max_myszy)
Wrogowie(imie_wroga, stopien_wrogosci, gatunek, lapowka)
Wrogowie_Kocurow(#pseudo,#imie_wroga, data_incydentu,
opis_incydentu).

gdzie podkreślenie oznacza klucz główny a # klucz obcy. Klucz obcy `funkcja` w relacji Kocury wiąże ją z relacją Funkcje, klucz obcy `nr_bandy` z relacją Bandy a klucz obcy `szef` wiąże ją sama z sobą (wskazuje na przełożonego kota). Klucze obce `pseudo` i `imie_wroga` w relacji Wrogowie_kocurow wiążą ją odpowiednio z relacją Kocury i relacją Wrogowie. Klucz obcy `szef_bandy` w relacji Bandy wiąże ją z relacją Kocury.

Poniżej przedstawiono wyniki polecenia DESCRIBE (DESC) w (środowisko SQL*Plus) dla powyższego, zaimplementowanego w systemie Oracle, schematu.

```
SQL> DESC Kocury
 Nazwa                          Null?    Typ
 ------------------------------ -------- ----
 PSEUDO                         NOT NULL VARCHAR2(15)
 IMIE                           NOT NULL VARCHAR2(15)
 PLEC                                    VARCHAR2(1)
 W_STADKU_OD                             DATE
 PRZYDZIAL_MYSZY                         NUMBER(3)
 MYSZY_EXTRA                             NUMBER(3)
#FUNKCJA                                 VARCHAR2(10)
#SZEF                                    VARCHAR2(15)
#NR_BANDY                                NUMBER(2)

SQL> DESC Bandy
 Nazwa                          Null?    Typ
 ------------------------------ -------- ----
 NR_BANDY                       NOT NULL NUMBER(2)
 NAZWA                          NOT NULL VARCHAR2(20)
 TEREN                                   VARCHAR2(15)
#SZEF_BANDY                              VARCHAR2(15)
```

```
SQL> DESC Funkcje
 Nazwa                           Null?    Typ
 ------------------------------- -------- ----
 FUNKCJA                         NOT NULL VARCHAR2(10)
 MIN_MYSZY                                NUMBER(3)
 MAX_MYSZY                                NUMBER(3)

SQL> DESC Wrogowie
 Nazwa                           Null?    Typ
 ------------------------------- -------- ----
 IMIE_WROGA                      NOT NULL VARCHAR2(15)
 STOPIEN_WROGOSCI                         NUMBER(2)
 GATUNEK                                  VARCHAR2(15)
 LAPOWKA                                  VARCHAR2(20)

SQL> DESC Wrogowie_kocurow
 Nazwa                           Null?    Typ
 ------------------------------- -------- ----
#PSEUDO                          NOT NULL VARCHAR2(15)
#IMIE_WROGA                      NOT NULL VARCHAR2(15)
 DATA_INCYDENTU                  NOT NULL DATE
 OPIS_INCYDENTU                           VARCHAR2(50)
```

2. Język SQL - dialekt Oracle'a

SQL (ang. Structured Query Language) jest uznanym za międzynarodowy standard (SQL1 - ISO 1987, SQL2 – ISO 1992, SQL3 – ISO 1999, SQL 2003, SQL 2006, SQL 2008, SQL 2011) strukturalnym językiem zapytań do relacyjnych baz danych. W realizacjach komercyjnych standard ten bywa często rozszerzany. Każda taka implementacja SQL'a nazywana jest jego dialektem. W ramach niniejszego wykładu przedstawiony będzie dialekt SQL'a zaproponowany przez firmę Oracle. Baza danych opisująca rzeczywistość kotów zostanie zrealizowana w SZBD tej firmy a wszystkie przykłady wykonywane będą w oferowanym przez tą firmę środowisku SQL*Plus.

Ogólnie język SQL składa się z następujących trzech składowych:

- DDL (ang. Data Definition Language): język do definiowania obiektów bazy danych z podstawowymi poleceniami CREATE, ALTER i DROP,
- DML (ang. Data Manipulation Language): język do manipulowania danymi z podstawowymi poleceniami INSERT, UPDATE, DELETE i SELECT oraz do sterowania transakcjami z podstawowymi poleceniami COMMIT i ROLLBACK,
- DCL (ang. Data Control Language): język do nadawania uprawnień do obiektów i operacji bazodanowych z podstawowymi poleceniami GRANT i REWOKE.

Na potrzeby niniejszego wykładu ze składowej DML została wydzielona część, zawierająca polecenie SELECT, nazywana dalej składową DQL (ang. Data Query Language) SQL'a. W ramach wykładu poszczególne polecenia języka SQL, stanowiące części jego składowych, będą omawiane w kolejności wynikającej z potrzeb towarzyszącego wykładowi projektowi a nie w kolejności systematycznej przedstawionej w powyższym podziale.

2.1. Tabele relacyjne

Tabele relacyjne tworzone są, modyfikowane i usuwane za pomocą poleceń składowej DDL języka SQL. Struktura polecenia tworzącego tabelę wynika z modelu logicznego bazy danych a w ramach składni polecenia określane są także ograniczenia, którym podlegają atrybuty tabeli i cała struktura tworzonej bazy (więzy integralności). Do ograniczeń tych należy zaliczyć:

- obowiązkowość atrybutów,
- unikalność atrybutów,
- więzy dziedzinowe,
- integralność encji,
- integralność referencyjną,
- więzy propagacji,
- więzy ogólne.

W zależności od dialektu SQL'a większa lub mniejsza część tych więzów jest definiowana bezpośrednio za pomocą polecenia DDL tworzącego tabelę relacyjną. W bazie Oracle w ramach tego polecenia nie można jedynie zdefiniować więzów ogólnych.

Tabela relacyjna (relacja) w bazie danych Oracle jest tworzona za pomocą polecenia CREATE TABLE. Najbardziej podstawowa składnia tego bardzo rozbudowanego polecenia jest następująca:

CREATE TABLE Nazwa_relacji
({nazwa_atrybutu typ_atrybutu [{**NOT NULL**} | **NULL**]
 [**DEFAULT** wartość_domyślna]
 [{ograniczenie_atrybutu [...]}] [, ...]}
 [,{ograniczenie_relacji [, ...]}]);

Relacja składa się z atrybutów o wartościach ograniczonych przez więzy dziedzinowe. Więzy te definiowane są za pomocą typu atrybutu i ograniczeń atrybutu. NOT NULL definiuje ograniczenie związane z obowiązkowością atrybutu (domyślnie atrybut traktowany jest jako nieobowiązkowy). Klauzula DEFAULT określa ewentualną jego

wartość domyślną. Ograniczenia atrybutu (wymienione po spacji) dotyczą realizacji integralności encji (prosty klucz główny), integralności referencyjnej (proste klucze obce) wraz z więzami propagacji, prostych kluczy unikalnych i dodatkowych ograniczeń dziedzinowych. Ograniczenia relacji dotyczą ograniczeń, których nie można zdefiniować w ramach ograniczeń atrybutu bo oparte są na więcej niż jednym atrybucie relacji. Dotyczą one integralności encji (złożony klucz główny), integralności referencyjnej (złożone klucze obce) wraz z więzami propagacji, złożonych kluczy unikalnych i innych ograniczeń opartych na co najmniej dwóch atrybutach relacji.

W systemie Oracle można wyróżnić następujące podstawowe typy atrybutów tabel relacyjnych (dopuszczane są także alternatywne nazwy tych typów wynikające ze standardu SQL):

Nazwa typu	Opis typu
`CHAR`	Łańcuchy znakowe o stałej długości (zwykle domyślnie długość ta wynosi 255 znaków).
`CHAR(w)`	Łańcuchy znakowe o stałej długości `w` znaków ($1 \le w \le 2000$).
`VARCHAR2`	Łańcuchy znakowe o zmiennej długości jednak nie większej niż 4000 znaków.
`VARCHAR2(w)`	Łańcuchy znakowe o zmiennej długości lecz nie większej niż `w` znaków (`w`≤4000).
`NCHAR`	Typ taki sam jak `CHAR` wykorzystujący jednak alternatywny w stosunku do reszty bazy danych zestaw znaków.
`NCHAR(w)`	Typ taki sam jak `CHAR(w)` wykorzystujący jednak alternatywny w stosunku do reszty bazy danych zestaw znaków.
`NVARCHAR2`	Typ taki sam jak `VARCHAR2` wykorzystujący jednak alternatywny w stosunku do reszty bazy danych zestaw znaków.
`NVARCHAR2(w)`	Typ taki jak jak `VARCHAR2(w)` wykorzystujący jednak alternatywny w stosunku do reszty bazy danych zestaw znaków.
`LONG`	Dane znakowe o zmiennej długości (do 2 GB).

	Oracle zaleca stosowanie `CLOB` lub `NCLOB`.
`CLOB`	Dane znakowe o dużej objętości (do 128 TB, w wersjach Oracle starszych niż 10g do 4 GB).
`NCLOB`	Typ taki sam jak `CLOB` wykorzystujący jednak alternatywny w stosunku do reszty bazy danych zestaw znaków.
`NUMBER`	Liczby całkowite o precyzji 38 (precyzja to liczba cyfr znaczących).
`NUMBER(w)`	Liczby całkowite o maksymalnej precyzji `w` ($w \leq 38$).
`NUMBER(w,d)`	Liczby rzeczywiste o precyzji `w` i `d` miejscach po przecinku ($w \leq 38$).
`DATE`	Data i czas.
`RAW`	Dane binarne o rozmiarze do 2 KB.
`LONG RAW`	Dane binarne o rozmiarze do 2 GB. Oracle zaleca stosowanie `BLOB`.
`ROWID`	Reprezentuje konkretny adres wiersza w tabeli
`ORA_ROWSCN`	Przechowuje numery zmian systemowych ostatnich transakcji modyfikujących wiersze.
`BLOB`	Dane binarne o dużej objętości (do 4 GB).
`BFILE`	Pliki binarne. Pozwala na przechowywanie danych binarnych tylko do odczytu w zewnętrznych plikach (poza bazą danych).
`XMLTYPE`	Przechowuje dokumenty XML w kolumnach `CLOB` (od wersji Oracle 9i). Definiowany poprzez moduł SYS (`SYS.XMLTYPE`).
Typy danych użytkownika	Typy złożone definiowane przez użytkownika (od Oracle 9i) wykorzystujące typy podstawowe określone powyżej i inne wcześniej zdefiniowane typy złożone.

Składnia ograniczenia atrybutu i ograniczenia relacji jest następująca:

[**CONSTRAINT** nazwa_ograniczenia] ograniczenie

Dostępne są następujące ograniczenia atrybutu i ograniczenia relacji:

Składnia ograniczenia	Opis ograniczenia
PRIMARY KEY [({nazwa_atrybut [, …]})]	Dla ograniczenia atrybutu definiuje atrybut pełniący rolę klucza głównego (integralność encji). W przypadku klucza złożonego (ograniczenie relacji) określana jest lista atrybutów kluczowych. Ograniczenie wyklucza się z ograniczeniem UNIQUE.
UNIQUE [({nazwa_atrybutu [, …]})]	Dla ograniczenia atrybutu definiuje atrybut, pełniący rolę klucza unikalnego (z ograniczeniem NOT NULL będzie to klucz alternatywny). W przypadku klucza złożonego (ograniczenie relacji) określana jest lista atrybutów kluczowych. Ograniczenie wyklucza się z ograniczeniem PRIMARY KEY.
[**FOREIGN KEY** ({nazwa_atrybutu [, …]})] **REFERENCES** nazwa_relacji [({nazwa_atrybutu [, …]})]	Definiuje klucz obcy (integralność referencyjna). Dla ograniczenia atrybutu (prosty klucz obcy) nie występuje klauzula FOREIGN KEY, w której wymienione są atrybuty tworzące klucz. Nazwa relacji oznacza relację powiązaną po nazwie której może wystąpić lista atrybutów tej relacji pełniących tam rolę klucza głównego, alternatywnego lub unikalnego. W przypadku klucza prostego jest to lista jednoelementowa.

ON DELETE {**CASCADE** | **SET NULL**}	Ograniczenie występujące po definicji klucza obcego. Określa ono obowiązujące więzy propagacji (CASCADE usuwanie kaskadowe a SET NULL usuwanie z wstawianiem wartości NULL). Jeśli ograniczenie nie wystąpi, obowiązuje usuwanie ograniczone.
CHECK (warunek)	Ograniczenie dotyczy więzów dziedzinowych. Określa ono warunek, który musi spełniać atrybut (ograniczenie atrybutu) lub kilka atrybutów (ograniczenie relacji) relacji. W niektórych dialektach SQL'a można w warunku wykorzystać podzapytanie (Oracle tego nie realizuje).
[NOT] DEFERRABLE [INITIALLY {IMMEDIATE|DEFFERED}]	Pozwala na odroczenie procesu sprawdzania czy dane spełniają ograniczenie. Domyślne ustawienie to NOT DEFERRABLE. Klauzula INITIALLY określa, czy sprawdzanie ograniczenia będzie się odbywało po zakończeniu każdego polecenia DML czy też dopiero po zaakceptowaniu transakcji (polecenie COMMIT). Wybranie INITIALLY DEFERRED wymaga aktywacji procesu odraczania poprzez zmianę parametru sesji poleceniem ALTER SESSION SET CONSTRAINTS = DEFERRED. Proces ten można podobnym poleceniem wyłączyć. ALTER SESSION nie działa dla opcji NOT DEFERRABLE.

Brak jawnego nadania nazwy ograniczeniu (klauzula CONSTRAINT) powoduje nadanie nazwy systemowej SYS_Cn, gdzie n jest numerem ograniczenia. Każde ograniczenie atrybutu można zdefiniować jako ograniczenie relacji ale nie odwrotnie.

Opisy wszystkich ograniczeń użytkownika można znaleźć w perspektywach systemowych USER_CONSTRAINTS i USER_CONS_COLUMNS natomiast opisy relacji użytkownika w perspektywie USER_TABLES.

Poniższe przykłady tworzenia tabel relacyjnych nie dotyczą bazy danych opisującej rzeczywistość kotów. Wynika to z tego, że schemat tej bazy zostanie zaimplementowany w trakcie laboratorium.

Zad. *Zdefiniować tabelę relacyjną* `Osoby` *o atrybutach* `pesel`, `nazwisko`, `imie`, `plec` *wraz ze stosownymi ograniczeniami.*

```
SQL> CREATE TABLE Osoby
  2  (pesel NUMBER(11) CONSTRAINT os_pk PRIMARY KEY
  3                    CONCTRAINT os_pe_ch CHECK(pesel>10000000000),
  4   nazwisko VARCHAR2(20) CONSTRAINT os_naz_nn NOT NULL,
  5   imie VARCHAR2(15) CONSTRAINT os_im_nn NOT NULL,
  6   plec CHAR(1) CONSTRAINT os_plec_ch CHECK(plec IN ('K','M'))
  7  );
Tabela została utworzona.
SQL>
```

Zad. *Zdefiniować tabelę relacyjną* `Pozycje_w_dok` *o atrybutach* `nr_dokumentu`, `nr_pozycji`, `tresc_pozycji` *wraz ze stosownymi ograniczeniami.*

```
SQL> CREATE TABLE Pozycje_w_dok
  2  (nr_dokumentu VARCHAR2(20),
  3   nr_pozycji NUMBER(5),
  4   tresc_pozycji LONG CONSTRAINT poz_w_dok_tp_nn NOT NULL,
  5   CONSTRAINT poz_w_dok_pk
  6              PRIMARY KEY(nr_dokumentu, nr_pozycji)
  7  );
Tabela została utworzona.
SQL>
```

Klucz główny relacji `Pozycje_w_dok` jest ograniczeniem tabelowym (klucz złożony) stąd jego definicja po definicji wszystkich atrybutów.

Zad. *Niech dane honorowych dawców krwi gromadzone będą w relacji* `Dawcy`*, dane pobranych od nich donacji w relacji* `Donacje` *a dane badań wirusologicznych donacji w relacji* `Badania`*. Zdefiniować wymienione relacje wraz z ich atrybutami, ograniczeniami na atrybuty oraz odpowiednimi powiązaniami.*

```
SQL> CREATE TABLE Dawcy_honorowi
  2  (nr_dawcy NUMBER(5) CONSTRAINT dah_pk PRIMARY KEY,
  3   nazwisko VARCHAR2(20) CONSTRAINT dah_naz_nn NOT NULL,
  4   imie VARCHAR2(15) CONSTRAINT dah_im_nn NOT NULL,
  5   plec CHAR(1) CONSTRAINT dah_plec_nn NOT NULL
  6           CONSTRAINT dah_plec_ch CHECK(plec IN ('K','M')),
  7   grupa_krwi CHAR(2) CONSTRAINT dah_grk_ch
  8              CHECK (grupa_krwi IN ('0','A','B','AB')),
  9   adres VARCHAR2(50) CONSTRAINT dah_adr_nn NOT NULL
 10  );

Tabela została utworzona.

SQL> CREATE TABLE Oddania
  2  (nr_donacji VARCHAR2(20) CONSTRAINT odd_pk PRIMARY KEY,
  3   objetosc NUMBER(3) CONSTRAINT odd_ob_nn NOT NULL
  4                    CONSTRAINT odd_ob_ch CHECK (objetosc>0),
  5   data_pobrania DATE CONSTRAINT odd_dap_nn NOT NULL,
  6   nr_dawcy NUMBER(5) CONSTRAINT odd_nrda_nn NOT NULL
  7                        CONSTRAINT odd_nrda_fk
  8                        REFERENCES Dawcy_honorowi(nr_dawcy)
  9  );

Tabela została utworzona.
SQL> CREATE TABLE Badania
  2  (id_badania VARCHAR2(15) CONSTRAINT ba_pk PRIMARY KEY,
  3   wynik_wr CHAR(1)   CONSTRAINT ba_wwr_ch
  4                      CHECK (wynik_wr IN ('-', '+','?')),
  5   wynik_hiv CHAR(1)  CONSTRAINT ba_whiv_ch
  6                      CHECK (wynik_hiv IN ('-','+','?')),
  7   wynik_hbs CHAR(1)  CONSTRAINT ba_whbs_ch
  8                      CHECK (wynik_hbs IN ('-','+','?')),
  9   wynik_ahcv CHAR(1) CONSTRAINT ba_wahcv_ch
 10                      CHECK (wynik_ahcv IN ('-','+','?')),
 11   data_badania DATE,
 12   nr_donacji VARCHAR2(20) CONSTRAINT ba_nrdo_nn NOT NULL
 13                           CONSTRAINT ba_nrdo_fk
 14                             REFERENCES Oddania(nr_donacji)
 15                             ON DELETE CASCADE
 16  );
Tabela została utworzona.
SQL>
```

Należy w tym miejscu zwrócić uwagę na kolejność w jakiej budowane są relacje przez kolejne polecenia CREATE TABLE. Najpierw konstruowane są relacje nie odwołujące się do żadnych innych relacji a dopiero potem relacje od nich zależne (poprzez klucze obce). Jak już wcześniej wspomniano, każde ograniczenie kolumnowe może być zdefiniowane jako tabelowe. Przykładowo równoważna definicja relacji `Donacje` przy definicji wszystkich ograniczeń w postaci tabelowej miałaby następujący kształt:

```
SQL> CREATE TABLE Oddania
  2  (nr_donacji VARCHAR2(20),
  3   objetosc NUMBER(3) CONSTRAINT odd_ob_nn NOT NULL,
  4   data_pobrania DATE CONSTRAINT odd_dap_nn NOT NULL,
  5   nr_dawcy NUMBER(5) CONSTRAINT odd_nrda_nn NOT NULL,
  6   CONSTRAINT odd_pk PRIMARY KEY (nr_donacji),
  7   CONSTRAINT odd_ob_ch CHECK (objetosc>0),
  8   CONSTRAINT odd_nrda_fk FOREIGN KEY (nr_dawcy)
  9                        REFERENCES Dawcy_honorowi(nr_dawcy)
 10  );
Tabela została utworzona.
SQL>
```

Należy tu zwrócić uwagę na wspominany już fakt, że po klauzurze PRIMARY KEY wymieniona jest nazwa atrybutu będącego kluczem głównym a klucz obcy definiowany jako ograniczenie tabelowe posiada dodatkowy element składni w postaci klauzuli FOREIGN KEY.

Zdefiniowane poleceniem CREATE TABLE relacje bazy danych można modyfikować poleceniem ALTER TABLE. Potrzeba takiej modyfikacji najczęściej związana jest ze zmianą sytuacji w modelowanej rzeczywistości. Najbardziej podstawowa składnia polecenia ALTER TABLE w dialekcie Oracle SQL'a jest następująca:

ALTER TABLE Nazwa_relacji
{**ADD** nazwa_atrybutu typ_atrybutu
[{**NOT NULL**} | **NULL**]
[**DEFAULT** wartość_domyślna] ograniczenie_atrybutu}
| {**DROP COLUMN** nazwa_atrybutu
[**CASCADE CONSTRAINTS**}
| {**ADD** ograniczenie_relacji}
| {**DROP {**rodzaj_ograniczenia |
CONSTRAINT nazwa_ograniczenia}}
| {**DISABLE** | **ENABLE**} {rodzaj_ograniczenia |
CONSTRAINT nazwa_ograniczenia}
| {**MODIFY** nazwa_atrybutu typ [**DEFAULT** wartość_domyślna]
[{**NULL** | **NOT NULL**}]}

Dzięki powyższemu poleceniu można:

- dodać do relacji nowy atrybut,
- usunąć z relacji atrybut. przy czym w niektórych. Klauzula CASCADE CONSTRAINTS powodująca usunięcie atrybutu także ze wszystkich obiektów odwołujących się do usuwanego atrybutu,
- dodać ograniczenie relacji,
- usunąć ograniczenie poprzez jego rodzaj (dostępne rodzaje to UNIQUE i PRIMARY KEY) lub jego nazwę,
- czasowo wyłączyć lub włączyć ograniczenie poprzez wymienienie jego rodzaju (dostępne rodzaje to UNIQUE, PRIMARY KEY i ALL TRIGGERS) lub nazwy,
- zmodyfikować atrybut określając jego wartość domyślną lub/oraz obowiązkowość.

Modyfikacja tabeli relacyjnej wymaga spełnienia następujących warunków:

- nie można modyfikować atrybutu, dla którego dopuszczono wartości puste,
- nie można rozszerzyć relacji o nowy atrybut niepusty,
- zmniejszenie rozmiaru typu atrybutu jest możliwe, gdy wszystkie krotki relacji mają wartości puste dla tego atrybutu.

Zad. *Rozszerzyć relację* `Dawcy` *o nowy atrybut o nazwie* `status_dawcy` *określający czy dawca jest aktywny (status* `'A'`*), czy też jest emerytem (status* `'E'`*), czy też już nie żyje (status* `'N'`*).*

```
SQL> ALTER TABLE Dawcy ADD status_dawcy CHAR(1) DEFAULT 'A'
  2                    CONSTRAINT da_sta_ch
  3                    CHECK (status_dawcy IN ('A','E','N'));

Tabela została zmieniona.

SQL>
```

Zad. *Dla atrybutu* `objetosc` *w relacji* `Donacje` *dodać ograniczenie niedopuszczające do wpisania donacji o objętości większej niż 999 ml.*

```
SQL> ALTER TABLE Donacje
  2        ADD CONSTRAINT do_obj_ch CHECK (objetosc<=999);

Tabela została zmieniona.

SQL>
```

Powyżej zdefiniowane ograniczenie jest nadmiarowe (typ atrybutu `objetosc` określony przez `NUMBER(3)` ogranicza objętość donacji od góry na żądanej wartości), zostanie więc usunięte.

Zad. *Usunąć ograniczenie o nazwie* `do_obj_ch` *dla atrybutu* `objetosc` *z relacji* `Donacje`*.*

```
SQL> ALTER TABLE Donacje DROP CONSTRAINT do_obj_ch;

Tabela została zmieniona.

SQL>
```

Relacja usuwana jest ze schematu bazy danych poleceniem DROP TABLE o składni:

DROP TABLE Nazwa_relacji [**CASCADE CONSTRAINT**]

Opcjonalna klauzula CASCADE CONSTRAINT powoduje usunięcie także wszystkich obiektów, które są od usuwanej relacji zależne.

2.2. Zapytanie SELECT

Zapytanie SELECT jest głównym i jedynym poleceniem składowej DQL języka SQL. Pozwala ono na realizację, znanych z języka algebry relacji, operacji selekcji, rzutowania, iloczynu kartezjańskiego i złączenia. Podstawowa składnia polecenia SELECT w systemie Oracle jest następująca:

SELECT [**DISTINCT** | **ALL**] {wyrażenie [alias] [, ...]} | *
FROM {NazwaRelacjiPerspektywy [alias]
[operator_złączenia NazwaRelacjiPerspektywy [alias]
[ON warunek_złączenia]] [, ...]}
[**WHERE** warunek_selekcji_krotek]
[**GROUP BY** {wyrażenie [, ...]}
[**HAVING** warunek_selekcji_grup]]
[**ORDER BY** {wyrażenie [**DESC** | **ASC**] [, ...]}]

Wyrażenia w powyższej składni zwykle oparte są na atrybutach relacji (atrybut lub stała jest szczególnym przypadkiem wyrażenia). Alias oznacza nazwę alternatywną (zastępczą). W przypadku aliasu relacji ew. pespektywy (widoku) zastępuje on ich nazwę co oznacza, że do ich atrybutów nie można się odnosić poprzez tą nazwę. Klauzula SELECT omawianego polecenia realizuje operację rzutowania (projekcji). Po klauzurze FROM określana jest relacja/perspektywa, z której pobierane są dane. Jeśli ramach tej klauzuli, po przecinku, wymienione jest kilka relacji, dane pobierane są z iloczynu kartezjańskiego tych relacji. Jeśli wykorzystywany jest tam operator złączenia to relacją, z której pobierane są dane jest wynik złączenia zrealizowany z warunkiem złączenia wymienionym po ON. Klauzula WHERE realizuje operację selekcji krotek. Klauzula GROUP BY tworzy grupy krotek o tej samej wartości wyrażenia (wyrażeń) wymienionego po klauzurze. Klauzula HAVING realizuje operację selekcji grup. Występująca jako ostatnia klauzula ORDER BY powoduje uporządkowanie krotek relacji wynikowej wg. wartości wyrażenia (wyrażeń) wymienionego po klauzurze. Pozostałe elementy składni będą wyjaśnione przy okazji odpowiednich przykładów.

Wynikiem zapytania SELECT jest zawsze relacja więc operacja ta jest domknięta. W systemie Oracle większość klauzul tego zapytania może zawierać zagnieżdżone polecenia SELECT (tzw. podzapytania). Dotyczy to klauzul WHERE, HAVING, SELECT i FROM. Ponadto zawsze zagnieżdżone polecenie SELECT występuje w przypadku złączeń pionowych realizowanych przez operatory UNION, INTERSECT i MINUS.

Kolejność wykonywania klauzul w poleceniu SELECT jest inna niż ta występująca w powyższej składni Znajomość tej kolejności w niektórych przypadkach może ułatwić lub nawet umożliwić konstruowanie zapytań.

Klauzule polecenia SELECT wykonywane są w następującej kolejności:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

2.2.1. Proste zapytania

Klauzula SELECT i FROM

Najprostsza postać zapytania SELECT zawiera tylko klauzule SELECT i FROM. Po klauzurze FROM określana jest relacja/widok, z której pobierane są dane (może nią być wynik złączenia wielu relacji/widoków) a po klauzurze SELECT specyfikowane są wyrażenia, których wartości należy wyświetlić.

Zad. *Wylistować wartości wszystkich atrybutów relacji Funkcje.*

```
SQL> SELECT * FROM Funkcje;

FUNKCJA    MIN_MYSZY MAX_MYSZY
---------- --------- ---------
SZEFUNIO           90       110
BANDZIOR           70        90
LOWCZY             60        70
LAPACZ             50        60
KOT                40        50
MILUSIA            20        30
DZIELCZY           45        55
7 wierszy zostało wybranych.
SQL>
```

Znak * w powyższym poleceniu oznacza wszystkie atrybuty.

Zad. *Określić funkcje, które pełnią koty w każdej z band.*

```
SQL> SELECT DISTINCT nr_bandy,funkcja FROM Kocury;

 NR_BANDY FUNKCJA
--------- ----------
        1 DZIELCZY
        1 MILUSIA
        1 SZEFUNIO
        2 BANDZIOR
        2 LAPACZ
        2 LOWCZY
        2 MILUSIA
        3 BANDZIOR
        3 KOT
        3 LOWCZY
        3 MILUSIA
        4 KOT
        4 LAPACZ
        4 LOWCZY
14 wierszy zostało wybranych.
SQL>
```

Polecenie powyższe realizuje operację projekcji (rzutowania) relacji `Kocury` względem atrybutów `nr_bandy` i `funkcja`. Kwalifikator DISTINCT w powyższym zapytaniu powoduje, że nie są wyświetlane powtórzenia krotek w relacji wynikowej. Domyślnie przyjmowany jest kwalifikator ALL (wyświetlanie z powtórzeniami) i nie musi być on specyfikowany.

Najprostszą postać polecenia SELECT można rozszerzyć o:

a - literał: jest to stała typu łańcuch znaków, data lub liczba (data lub
b łańcuch znaków w pojedynczych apostrofach) zwana pseudokolumną.
c

Zad. *Określić minimalny i maksymalny przydział myszy związany z każdą funkcją.*

```
SQL> SELECT funkcja,'moze zjadac od ',min_myszy,
            ' do ',max_myszy
  2  FROM Funkcje;

FUNKCJA    'MOZEZJADACOD'   MIN_MYSZY 'DO'  MAX_MYSZY
---------- -------------- ---------- ---- ----------
SZEFUNIO   moze zjadac od          90  do         110
BANDZIOR   moze zjadac od          70  do          90
LOWCZY     moze zjadac od          60  do          70
LAPACZ     moze zjadac od          50  do          60
KOT        moze zjadac od          40  do          50
MILUSIA    moze zjadac od          20  do          30
DZIELCZY   moze zjadac od          45  do          55
HONOROWA   moze zjadac od           6  do          25

8 wierszy zostało wybranych.

SQL>
```

- alias atrybutu/wyrażenia: jest to nazwa alternatywna atrybutu lub wyrażenia wymienianego w klauzurze SELECT wyświetlana w nagłówku kolumny i ew. wykorzystywana w klauzurze ORDER BY.

Zad. *Określić minimalny i maksymalny przydział myszy związany z każdą funkcją (wykorzystać aliasy kolumn).*

```
SQL> SELECT funkcja Funkcyjka,'moze zjadac od ' " ",
       min_myszy "Min myszy",' do ' " ",max_myszy "Max myszy"
  2  FROM Funkcje;
```

```
FUNKCYJKA                  Min myszy       Max myszy
---------- -------------- ---------- ---- ----------
SZEFUNIO   moze zjadac od          90  do         110
BANDZIOR   moze zjadac od          70  do          90
LOWCZY     moze zjadac od          60  do          70
LAPACZ     moze zjadac od          50  do          60
KOT        moze zjadac od          40  do          50
MILUSIA    moze zjadac od          20  do          30
DZIELCZY   moze zjadac od          45  do          55
HONOROWA   moze zjadac od           6  do          25

8 wierszy zostało wybranych.

SQL>
```

- wyrażenie arytmetyczne.

Zad. *Dla każdego kota określić jego roczne spożycie myszy.*

```
SQL> SELECT imie,
  2         (NVL(przydzial_myszy,0)+NVL(myszy_extra,0))*12
  3         "Zjada rocznie"
  4  FROM Kocury;

IMIE            Zjada rocznie
--------------- -------------
MRUCZEK                  1632
CHYTRY                    600
MICKA                     864
RUDA                      768
BOLEK                    1116
JACEK                     804
ZUZIA                     780
BARI                      672
BELA                      624
KOREK                    1056
PUNIA                     732
SONIA                     660
LUCEK                     516
PUCEK                     780
MELA                      612
KSAWERY                   612
LATKA                     480
DUDEK                     480

18 wierszy zostało wybranych.

SQL>
```

Funkcja NVL zwraca wartość pierwszego argumentu jeśli jest ona różna od NULL, w przeciwnym przypadku zwraca wartość drugiego argumentu.

- operator konkatenacji: operator o symbolu || umożliwiający łączenie wyświetlanych wartości atrybutów w pojedynczy łańcuch znaków.

Zad. *Określić minimalny i maksymalny przydział myszy związany z każdą funkcją.*

```
SQL> SELECT funkcja||' moze zjadac od '||min_myszy||
            ' do '||max_myszy||' myszy miesiecznie'
            "Mozliwosci funkcyjne"
  2  FROM Funkcje;

Mozliwosci funkcyjne
------------------------------------------------------------
SZEFUNIO moze zjadac od 90 do 110 myszy miesiecznie
BANDZIOR moze zjadac od 70 do 90 myszy miesiecznie
LOWCZY moze zjadac od 60 do 70 myszy miesiecznie
LAPACZ moze zjadac od 50 do 60 myszy miesiecznie
KOT moze zjadac od 40 do 50 myszy miesiecznie
MILUSIA moze zjadac od 20 do 30 myszy miesiecznie
DZIELCZY moze zjadac od 45 do 55 myszy miesiecznie
HONOROWA moze zjadac od 6 do 25 myszy miesiecznie

8 wierszy zostało wybranych.

SQL>
```

Klauzula WHERE

Klauzula WHERE umożliwia operację selekcji na relacji określonej przez klauzulę FROM. Selekcja ta odbywa się zgodnie z warunkiem (wartością wyrażenia logicznego) umieszczonym po klauzurze WHERE.

Zad. *Znaleźć imiona wszystkich kotów pełniących funkcję MILUSIA.*

```
SQL> SELECT imie
  2  FROM Kocury
  3  WHERE funkcja='MILUSIA';
```

```
IMIE
--------------
RUDA
BELA
MICKA
SONIA

SQL>
```

Podczas konstrukcji warunku selekcji można wykorzystać następujące operatory:

=, !=, >, >=, <, <=, IS NULL, BETWEEN ... AND ..., IN (zbiór), LIKE, NOT, AND, OR

Wymienione powyżej operatory, w ramach każdego wyrażenia SQL'a, wykonywane są według następującego priorytetu (od najwyższego do najniższego) :

1. =, !=, <, >,<=, >=, BETWEEN ... AND ... , IN, LIKE, IS NULL
2. NOT
3. AND
4. OR

W systemach realizujących model relacyjny, a więc i w systemie Oracle, wprowadzona jest niezgodna z tym modelem dodatkowa specjalna wartość reprezentująca brak informacji o wartości atrybutu. Jest to wartość NULL (w rzeczywistości jest to brak wartości!) różna od zera dla typów liczbowych lub np. od znaku pustego dla typów znakowych (jej obsługa wykonywana jest w Oracle np. za pomocą wykorzystywanej już wcześniej funkcji NVL). Wprowadza ona *de facto* logikę trójwartościową (TRUE, FALSE, NULL) zamiast tradycyjnej logiki dwuwartościowej (TRUE, FALSE). W przypadku wyrażeń logicznych z argumentami o wartości NULL obowiązują więc następujące reguły dotyczące wartości wyniku wyrażenia:

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Wartość NULL mogą posiadać atrybuty, dla których dopuszczalna jest ich nieobowiązkowość. W takiej sytuacji wartością wyrażenia arytmetycznego, w którym występuje argument o wartości NULL jest NULL. Wyrażenie z operatorem relacyjnym (np. <), w którego skład wchodzi wyrażenie arytmetyczne o wartości NULL posiada także wartość NULL. Wartość NULL w wyrażeniach arytmetycznych wymaga więc specjalnej obsługi (np. poprzez funkcję NVL).

Zad. *Znaleźć koty nie posiadające dodatkowych przydziałów myszy.*

```
SQL> SELECT pseudo,plec||'   '
  2  FROM Kocury
  3  WHERE myszy_extra IS NULL;

PSEUDO          PLEC
--------------- ----
BOLEK           M
RAFA            M
KURKA           D
MAN             M
DAMA            D
PLACEK          M
RURA            M
ZERO            M
UCHO            D
MALY            M
SZYBKA          D

11 wierszy zostało wybranych.

SQL>
```

Zad. *Znaleźć koty o przydziale myszy między 50 a 60.*

```
SQL> SELECT pseudo
  2  FROM Kocury
  3  WHERE przydzial_myszy BETWEEN 50 AND 60;

PSEUDO
---------------
BOLEK
MAN
DAMA
RURA

SQL>
```

Zad. *Znaleźć imiona kotów pełniących funkcję BANDZIOR lub LOWCZY, których bezpośrednim szefem jest TYGRYS.*

```
SQL> SELECT imie
  2  FROM Kocury
  3  WHERE funkcja IN ('BANDZIOR','LOWCZY') AND szef='TYGRYS';

IMIE
---------------
KOREK
BOLEK
PUCEK

SQL>
```

Zad. *Znaleźć koty, w których imieniu występuje jako druga litera O.*

```
SQL> SELECT imie
  2  FROM Kocury
  3  WHERE imie LIKE '_O%';

IMIE
---------------
KOREK
BOLEK
SONIA

SQL>
```

Znak '_' we wzorcu oznacza dowolny znak a znak '%' oznacza dowolną resztę łańcucha. W przypadku konieczności sprawdzenia obecności w łańcuchu znaków '_' lub '%' należy je umieścić po znaku cytowania zdefiniowanym w klauzurze ESCAPE np.:

```
WHERE imie LIKE '_&_U_&%U%' ESCAPE '&'
```

Powyższy zapis oznacza, że pierwszy znak imienia jest dowolny, drugim znakiem jest _ , trzecim U, czwarty znak jest dowolny, piątym znakiem jest %, szóstym U a pozostałe znaki są dowolne.

Zad. *Określić pseudonim, funkcję, przydział myszy i przydział dodatkowy dla kotów posiadających dodatkowy przydział myszy, których podstawowy przydział przewyższa 70 lub które pełnią funkcję MILUSIA.*

```
SQL> SELECT funkcja,NVL(przydzial_myszy,0),myszy_extra
  2  FROM Kocury
  3  WHERE myszy_extra IS NOT NULL
  4        AND
  5       (NVL(przydzial_myszy,0)>70 OR funkcja='MILUSIA');

FUNKCJA    PRZYDZIAL_MYSZY MYSZY_EXTRA
---------- --------------- -----------
SZEFUNIO               103          33
BANDZIOR                75          13
MILUSIA                 22          42
BANDZIOR                72          21
MILUSIA                 24          28
MILUSIA                 25          47
MILUSIA                 20          35

7 wierszy zostało wybranych.

SQL>
```

Klauzula ORBER BY

Krotki relacji wynikowej zapytania SELECT są zwykle nieuporządkowane. Do ich jawnego uporządkowania względem wartości atrybutu/wyrażenia (listy atrybutów i/lub wyrażeń) relacji wynikowej służy klauzula ORDER BY. W klauzuli tej może pojawić się: identyfikator atrybutu, wyrażenie, alias wyrażenia lub atrybutu z

klauzuli SELECT, numer wyrażenia/atrybutu z listy wyświetlanych wartości przez klauzulę SELECT. Domyślnym kierunkiem porządkowania jest kierunek rosnący (ASC). Kierunek malejący definiowany jest poprzez wstawienie słowa DESC za identyfikatorem atrybutu/wyrażenia porządkowania. Porządkowanie realizowane jest także niejawnie. Wchodzi ono w skład następujących operacji: CREATE INDEX, DISTINCT, GROUP BY, ORDER BY, INTERSECT, MINUS, UNION, złączenie nieindeksowanych relacji.

Zad. *Wyświetl dane wrogów zgodnie z malejącym stopniem wrogości.*

```
SQL> SELECT stopien_wrogosci "Jak grozny",
     imie_wroga "Imie wroga"
  2  FROM Wrogowie
  3  ORDER BY stopien_wrogosci DESC;

Jak grozny Imie wroga
---------- ---------------
        10 KAZIO
        10 DZIKI BILL
         7 SWAWOLNY DYZIO
         5 CHYTRUSEK
         4 BUREK
         3 BAZYLI
         2 REKSIO
         1 GLUPIA ZOSKA
         1 SMUKLA
         1 BETHOVEN

10 wierszy zostało wybranych.

SQL>
```

Zad. *Wyświetlić dane kotów, dla których przydział myszy przekracza 60. Dane uporządkować najpierw rosnąco według płci i nazwy bandy a następnie malejąco według daty wstąpienia do stada i dalej rosnąco według nazwy funkcji.*

```
SQL> SELECT pseudo "Pseudonim",plec||'   ' "Plec",
  2         nr_bandy "Banda",w_stadku_od "Wstapil",
  3         przydzial_myszy "Zjada"
  4  FROM Kocury WHERE przydzial_myszy>60
  4  ORDER BY 2,"Banda",w_stadku_od DESC,funkcja;
```

```
Pseudonim       Plec      Banda Wstapil         Zjada
--------------- ---- ---------- ---------- ----------
SZYBKA          D             2 2006-07-21         65
KURKA           D             3 2008-01-01         61
TYGRYS          M             1 2002-01-01        103
PLACEK          M             2 2008-12-01         67
LYSY            M             2 2006-08-15         72
ZOMBI           M             3 2004-03-16         75
RAFA            M             4 2006-10-15         65

7 wierszy zostało wybranych.

SQL>
```

Należy zwrócić uwagę na to, że porządkowanie jest operacją czasochłonną stąd nie należy jej nadużywać.

2.2.2. Zapytania z grupowaniem

Dla użytkownika często interesująca jest informacja dotycząca nie pojedynczej krotki ale całej grupy krotek. Aby uzyskać taką informację należy wykonać zapytanie z grupowaniem.

Klauzula GROUP BY

Klauzula GROUP BY pozwala ona na pogrupowanie krotek relacji ze względu wartość atrybutu/wyrażenia (listy atrybutów i/lub wyrażeń) wymienionego po klauzurze (do grupy wchodzą krotki o tej samej wartości atrybutu/wyrażenia) i wyświetlenie jednej krotki wynikowej reprezentującej każdą grupę. Klauzula SELECT, w przypadku wykorzystania grupowania, może jedynie zawierać: atrybuty/wyrażenia wymienione po klauzurze GROUP BY, funkcje agregujące, pseudokolumny, ew. wyrażenia, w skład których wchodzą wymienione elementy. Jeśli grupowanie wykonywane jest ze względu na atrybut/wyrażenie przyjmujący wartości puste (NULL), są one traktowane jako dodatkowa wartość atrybutu/wyrażenia (powstaje związana z nimi grupa).

Zad. *Znaleźć pseudonimy kotów posiadających podwładnych.*

```
SQL> SELECT szef
  2  FROM Kocury
  3  GROUP BY szef;

SZEF
---------------
KURKA
LYSY
RAFA
TYGRYS
ZOMBI

6 wierszy zostało wybranych.

SQL>
```

Szósty wybrany wiersz dotyczy grupy, w której atrybut `szef` nie posiada wartości (NULL).

Zad. *Znaleźć liczbę kotek (pleć żeńska) pełniących w każdej z band określoną funkcję.*

```
SQL> SELECT COUNT(*)||
           ' - liczba kotek pelniacych w bandzie '||nr_bandy||
           ' funkcje '||funkcja "Statystyka funkcyjna"
  2  FROM Kocury
  3  WHERE plec='D'
  4  GROUP BY nr_bandy,funkcja;

Statystyka funkcyjna
-------------------------------------------------------------2
- liczba kotek pelniacych w bandzie 1 funkcje MILUSIA
1 - liczba kotek pelniacych w bandzie 2 funkcje LOWCZY
1 - liczba kotek pelniacych w bandzie 2 funkcje MILUSIA
1 - liczba kotek pelniacych w bandzie 3 funkcje LOWCZY
1 - liczba kotek pelniacych w bandzie 3 funkcje MILUSIA
1 - liczba kotek pelniacych w bandzie 4 funkcje KOT
1 - liczba kotek pelniacych w bandzie 4 funkcje LAPACZ

7 wierszy zostało wybranych.

SQL>
```

Do znalezienia liczby kotek zastosowano funkcję agregującą COUNT określającą liczbę krotek w grupie. Funkcja ta ma składnię:

COUNT(* | {[**DISTINCT** | **ALL**] wyrażenie})

Funkcja COUNT zawsze zlicza krotki w grupie. Jeśli argumentem funkcji jest *, zliczane są wszystkie krotki, jeśli argumentem jest wyrażenie (szczególnym jego przypadkiem jest np. atrybut), zliczane są tylko te krotki, dla których wyrażenie jest różne od NULL'a (brane są pod uwagę także powtórzenia wartości wyrażenia – domyślne ALL), jeśli dodatkowo przed wyrażeniem wystąpi kwalifikator DISTINCT, podobnie zliczane są tylko te krotki, dla których wyrażenie jest różne od NULL'a jednak w zliczaniu pomijane są krotki z powtarzającą się wartością wyrażenia. Oprócz funkcji COUNT standardowo w systemach bazodanowych implementowane są zwykle następujące funkcje agregujące:

SUM([**DISTINCT | ALL]** wyrażenie) – zwraca sumę wartości wyrażeń różnych od NULL, pobranych z każdej krotki grupy; DISTINCT powoduje pomijanie w sumie powtarzającej się wartości wyrażenia (domyślne ALL),
AVG([**DISTINCT | ALL]** wyrażenie) – zwraca średnią arytmetyczną z wartości wyrażeń różnych od NULL, pobranych z każdej krotki grupy; DISTINCT powoduje pomijanie w sumie powtarzającej się wartości wyrażenia (domyślne ALL),
MAX(wyrażenie) – zwraca maksymalną wartość spośród wyrażeń różnych od NULL, pobranych z każdej krotki grupy,
MIN(wyrażenie) – zwraca minimalną wartość spośród wyrażeń różnych od NULL, pobranych z każdej krotki grupy.

Dialekt SQL'a implementowany w systemie Oracle uzupełnia ten zestaw funkcji agregujących o wiele innych funkcji, najczęściej o charakterze statystycznym. Funkcje agregujące mogą ogólnie być wykorzystywane jedynie w klauzulach SELECT i HAVING zapytania SELECT. Do użycia ich w klauzurze SELECT nie jest wymagane grupowanie. Funkcje agregujące działają wtedy na całej relacji wynikowej i tylko one (ew. pseudokolumny) mogą wystąpić w ramach tej klauzury.

Zad. *Znaleźć średnie spożycie myszy dla każdej płci (z uwzględnieniem przydziałów dodatkowych).*

```
SQL> SELECT DECODE(plec,'D','Kotka','Kocor')"Plec",
  2         AVG(NVL(przydzial_myszy,0)+NVL(myszy_extra,0))
  3         "Sredni przydzial"
  4  FROM Kocury
  5  GROUP BY plec;

Plec  Sredni przydzial
----- ----------------
Kotka             57,5
Kocor             68,9

SQL>
```

Wykorzystana w powyższym kodzie, charakterystyczna dla Oracle, funkcja DECODE realizuje instrukcję warunkową CASE. Wartość pierwszego argumentu funkcji porównywana jest z wartością pierwszego argumentu kolejnych par argumentów. Jeśli wystąpi równość, zwracana jest wartość drugiego argumentu odpowiedniej pary, jeśli nie to sprawdzana jest kolejna para. W przypadku, gdy dla żadnej pary nie będzie równości, zwracana jest wartość ostatniego argumentu funkcji, jeśli on wystąpi (jeśli nie wystąpi, zwracana jest wartość NULL). Funkcję DECODE można dowolnie zagnieżdżać. W powyższym kodzie wartość atrybutu `plec` porównywana jest ze stałą znakową `'D'` (pierwszy element pierwszej pary). Jeśli wystąpi równość, zwracana jest stała łańcuchowa `'Kotka'` (drugi element pierwszej pary), w przeciwnym wypadku, ze względu na brak kolejnych par, zwracana jest stała łańcuchowa `'Kocor'` (ostatni argument funkcji). Zamiast funkcji DECODE można użyć, zgodnej ze standardem ANSI SQL’a, funkcję CASE. Poniżej przedstawiono rozwiązanie powyższego zadania z wykorzystaniem tej funkcji.

```
SQL> SELECT CASE plec
  2           WHEN 'D' THEN 'Kotka'
  3           ELSE 'Kocur'
  4         END "Plec",
  5         AVG(NVL(przydzial_myszy,0)+NVL(myszy_extra,0))
  6         "Sredni przydzial"
  7  FROM Kocury
  8  GROUP BY plec;

SQL>
```

Wartość wyrażenia po CASE odpowiada pierwszemu argumentowi funkcji DECODE, wartości po WHEN i THEN definiują kolejne pary (elementów WHEN … THEN może być dowolna liczba), wartość po ELSE odpowiada ostatniemu argumentowi funkcji DECODE (element ELSE może nie wystąpić).

Klauzula HAVING

Klauzula HAVING służy do selekcji grup powstałych w wyniku działania klauzuli GROUP BY i bez tej ostatniej klauzuli nie może wystąpić. O tym jakie grupy są wybierane decyduje warunek umieszczony po HAVING. Warunek ten może być zbudowany jedynie na bazie atrybutu/wyrażenia (atrybutów/wyrażeń) grupowania lub/i na bazie funkcji agregujących.

Zad. *Znaleźć bandy, w których występują „kominy myszowe” (przydział myszy niewielkiej liczby kotów przekracza zdecydowanie przydziały pozostałych kotów).*

```
SQL> SELECT nr_bandy "Banda kominowa",
  2         AVG(przydzial_myszy) "Sredni przydzial",
  3         (MAX(NVL(przydzial_myszy,0))+
  4          MIN(NVL(przydzial_myszy,0)))/2
  5         "(MAX+MIN)/2"
  6  FROM Kocury
  7  GROUP BY nr_bandy
  8  HAVING (MAX(NVL(przydzial_myszy,0))+
  9          MIN(NVL(przydzial_myszy,0)))/2>
 10          AVG(NVL(przydzial_myszy,0));

Banda kominowa Sredni przydzial (MAX+MIN)/2
-------------- ---------------- -----------
             1               50        62,5
             4             49,4        52,5

SQL>
```

Klauzule CONNECT BY i START WITH

Zaprezentowana wcześniej podstawowa składnia polecenia SELECT uzupełniona jest w systemie Oracle o dodatkowe klauzule CONNECT BY i START WITH. Są one wykorzystywane najczęściej w przypadku istnienia, w ramach modelu konceptualnego bazy, związku encji (klasy) samej ze sobą (związek hierarchiczny), który to związek określa hierarchię np. przełożony - podwładny. Realizacją takiego związku jest klucz obcy powiązany z kluczem głównym tej samej relacji. System Oracle wykorzystuje taki związek do budowy, za pomocą wspomnianych klauzul, drzewa krotek odzwierciedlającego tą hierarchię. W klauzuli START WITH określany jest warunek wskazujący krotkę, która ma być korzeniem drzewa (jeśli n krotek spełnia warunek, budowanych jest n drzew) a w klauzuli CONNECT BY warunek definiujący sposób budowy drzewa, tzn. określający, jaka krotka ma być dołączona jako liść do aktualnego węzła. W ramach warunku po klauzuli CONNECT BY występuje słowo kluczowe PRIOR wskazujące tzw. atrybut nadrzędny (w krotce aktualnego węzła), z którego pobierana jest wartość do porównania z wartością drugiego atrybutu warunku nazywanego atrybutem podrzędnym (z krotki, która może się stać liściem). Drzewo budowane jest poprzez wybieranie jako liści krotek spełniających warunek po CONNECT BY. Najczęściej warunek ten oparty jest na równości klucza obcego, reprezentującego związek hierarchiczny, z kluczem głównym tej samej relacji. W ramach zapytania budującego drzewo można wykorzystywać:

- pseudoatrybut `level` - określa „głębokość” obsługiwanej krotki w drzewie,
- operator CONNECT_BY_ROOT atrybut - zwraca, dla wskazanego atrybutu z aktualnie obsługiwanej krotki w drzewie, wartość tego atrybutu w krotce korzenia,
- funkcję SYS_CONNECT_BY_PATH(atrybut, separator) - zwraca, dla obsługiwanej krotki, ścieżkę w drzewie od krotki korzenia do tej krotki, w postaci łańcucha znaków zbudowanego z kolejnych wartości wskazanego atrybutu w każdym węźle pośrednim, oddzielonych łańcuchem separatora.

Zad. *Określić hierarchię w stadzie kotów od przywódcy stada począwszy. W zbudowanym drzewie pominąć kota o imieniu KOREK wraz z wszystkimi jego podwładnymi oraz koty o funkcji MILUSIA.*

```
SQL> SELECT imie "Imie",level "Pozycja",
  2         nr_bandy "Banda",NVL(przydzial_myszy,0) "Zjada"
  3  FROM Kocury WHERE funkcja!='MILUSIA'
  4  CONNECT BY PRIOR pseudo=szef AND imie!='KOREK'
  5  START WITH szef IS NULL
  6  ORDER BY nr_bandy,level;
```

Imie	Pozycja	Banda	Zjada
MRUCZEK	1	1	103
CHYTRY	2	1	50
BOLEK	2	2	72
JACEK	3	2	67
ZUZIA	3	2	65
BARI	3	2	56
PUCEK	2	4	65
MELA	3	4	51
KSAWERY	3	4	51
LATKA	3	4	40
DUDEK	3	4	40

Zad. *Dla każdego kota należącego do poddrzew o korzeniach ZOMBI i RAFA (pseudonimy kotów) przedstawić pseudonim kota z korzenia poddrzewa oraz w postaci kolejnych pseudonimów, ścieżki od pseudonimu kota z korzenia począwszy na pseudonimie obsługiwanego kota skończywszy.*

```
SQL> SELECT pseudo "Kot",
  2         DECODE(CONNECT_BY_ROOT pseudo,
  3                pseudo,NULL,CONNECT_BY_ROOT pseudo) "Szef",
  4         SYS_CONNECT_BY_PATH(pseudo,'/')
  5                                      "Sciezka pseudonimow"
  6  FROM Kocury
  7  CONNECT BY PRIOR pseudo=szef
  8  START WITH pseudo IN ('ZOMBI','RAFA');
```

Kot	Szef	Sciezka pseudonimow
RAFA		/RAFA
DAMA	RAFA	/RAFA/DAMA
MALY	RAFA	/RAFA/MALY
MAN	RAFA	/RAFA/MAN
UCHO	RAFA	/RAFA/UCHO
ZOMBI		/ZOMBI
KURKA	ZOMBI	/ZOMBI/KURKA
ZERO	ZOMBI	/ZOMBI/KURKA/ZERO
PUSZYSTA	ZOMBI	/ZOMBI/PUSZYSTA