KRYSTYNA CZAPLA

BAZY DANYCH

PODSTAWY PROJEKTOWANIA I JĘZYKA SQL

Idealna baza danych — szyta na Twoją miarę!

- Faza projektu, czyli jak opracować tabele i określić zależności między nimi
- Faza implementacji, czyli jak przejść od projektu do tworzenia fizycznej bazy
- Faza trzecia, czyli jak czerpać informacje z bazy danych w nowoczesnych aplikacjach

Helion

Spis treści

Wstęp

Książka zawiera podstawową wiedzę na temat projektowania i eksploatacji baz danych. Powstała jako owoc wieloletnich doświadczeń w prowadzeniu zajęć dydaktycznych z przedmiotu „Bazy danych" na studiach licencjackich i inżynierskich. Jest przeznaczona dla osób chcących uporządkować swoją wiedzę z tego zakresu. W książce położono nacisk przede wszystkim na aspekty praktyczne. Ważnym celem tego opracowania jest dostarczenie studentom przykładowych zadań i ich rozwiązań, służących do wykorzystania w trakcie nauki.

Zostały tu omówione dwie tradycyjne metody projektowania — modelowanie i normalizacja. Prześledzenie ich obu uzmysłowi Czytelnikowi, że niezależnie od wybranej metodologii, proces tworzenia bazy przebiega w podobnych etapach. Pierwszy z nich — faza projektu — polega na dopracowaniu koncepcji tabel, ich pól, kluczy, określeniu zależności miedzy tabelami i integralności danych. Drugi — implementacja — to etap przejścia od projektu do tworzenia fizycznej bazy z wykorzystaniem dostępnych narzędzi. Faza trzecia to tworzenie aplikacji, która ma umożliwić użytkownikowi przyjazne korzystanie z informacji pozyskanych z bazy i zachowanie poprawności danych na każdym etapie użytkowania bazy. Przyjmuje się, że dane to wartości przechowywane w bazie, zaś informacje to dane przetwarzane i udostępniane użytkownikowi (te są dynamiczne). Informacje mogą być prezentowane na wiele sposobów — jako wynik zastosowania polecenia, wyświetlane w odpowiednio zaprojektowanej formatce na ekranie komputera lub wydrukowane w postaci odpowiednio zredagowanego raportu.

Czas poświęcony na projektowanie struktury bazy to czas dobrze zainwestowany. Dobry projekt ma kolosalne znaczenie dla funkcjonowania, integralności i dokładności danych w bazie. Struktury logiczne bazy są projektowane niezależnie od jakiegokolwiek Systemu Zarządzania Bazą Danych (SZBD). Metody projektowania dostarczają rozwiązań umożliwiających zdefiniowanie bazy w sposób poprawny i skuteczny. Na etapie gotowego projektu jest określany sposób implementacji (czyli czy będzie to aplikacja jednostanowiskowa, architektura klient-serwer, webowa itd.) oraz jest dokonywany wybór środowiska (tu ma miejsce ustalenie, w jakim SZBD będzie uruchamiana aplikacja). Konkretny system SZBD nie może rzutować na sposób projektowania bazy, dostarcza on tylko określonych narzędzi potrzebnych do implementacji projektu.

Obszerną część tego opracowania stanowią podstawy języka SQL, ze szczególnym uwzględnieniem poleceń z grupy DQL, DML i DDL. Znajdują się tutaj wyjaśnienia

pojęć i składni języka SQL oraz bogaty zestaw poleceń ilustrujących ich użycie w bazie Oracle. Są tu omówione polecenia składające się na język manipulowania danymi i definiowania danych, są również zaprezentowane operacje relacyjne i działania na zbiorach. Dziesiątki przykładów dają możliwość wykonania ćwiczeń utrwalających zdobyte wiadomości. Do przykładów są dołączone komentarze, które pozwalają przeanalizować wykonywane polecenia. Daje to możliwość gruntownego opanowania podstaw języka SQL, a przyswojenie przykładów wykorzystujących jedną z najpopularniejszych platform ułatwi uchwycenie niewielkich różnic i niuansów składni stosowanych w implementacjach poszczególnych dialektów standardu SQL, czyli dialektu ANSI implementowanego w bazie Oracle, InterBase i innych oraz dialektu Sybase zaimplementowanego m.in. w bazach Sybase i MS SQL.

Na końcu został dołączony wykaz literatury z tego zakresu — w nadziei, że wniknięcie w zagadnienia baz danych stanie się dla wielu początkiem wspaniałej przygody zawodowej informatyka.

Wprowadzenie

Zacznijmy od odpowiedzi na pytanie: czym jest baza danych? Baza danych jest tematycznie wyodrębnionym, logicznie zintegrowanym i odpowiednio uporządkowanym oraz utrwalonym zbiorem danych. Nie ma znaczenia, czy to kartoteka papierowa, czy dane w aplikacji komputerowej. Jeśli informacje są uporządkowane i gromadzone w określonym celu, już stają się bazą danych. My zajmiemy się bazami wykorzystywanymi w systemach informatycznych. Taka baza zawiera struktury, w których są przechowywane dane, oraz opis zawartości — zwany katalogiem systemowym lub metadanymi. Katalog systemowy zawiera zbiór tabel i perspektyw, w których jest opisany schemat bazy oraz wszystkie jej obiekty. Przechowywanie metadanych zapewnia niezależność danych i aplikacji działających w środowisku bazy. Zadania związane z przechowywaniem danych i sprawowaniem kontroli nad nimi realizuje specjalistyczne oprogramowanie — system zarządzania bazą danych (ang. *database management system* — DBMS). SZBD izoluje dane fizycznie przechowywane w bazie od programów użytkowych, zapewnia niezależność danych od technologii czy narzędzi, w których jest wykonana aplikacja użytkownika. Aplikacja zaś stanowi interfejs pomiędzy bazą (a dokładniej SZBD) a użytkownikiem.

System zarządzania bazą danych realizuje następujące funkcjonalności [9]:

- za pomocą *języka definiowania danych* (DDL) umożliwia tworzenie, modyfikowanie i usuwanie struktur danych oraz tworzenie nowych baz danych,
- za pomocą *języka manipulowania danymi* (DML) umożliwia realizację żądań użytkownika obejmujących wstawianie, modyfikowanie, usuwanie i wyszukiwanie danych,
- za pomocą *języka kontroli danych* (DCL) umożliwia autoryzację dostępu do danych,
- gwarantuje spójność bazy danych na każdym etapie jej przetwarzania oraz zapewnia bezpieczeństwo danych w wypadku awarii sprzętowo-programowej, daje także możliwość odtworzenia stanu bazy sprzed awarii,
- umożliwia synchronizowanie jednoczesnego dostępu do danych dla wielu użytkowników poprzez system kontroli transakcji,

- pozwala na przechowywanie bardzo dużych ilości danych — aktualnie to wiele terabajtów (10^{12} bajtów) czy nawet petabajtów (10^{15} bajtów) — przez długi czas; dotyczy to danych z wewnętrznymi mechanizmami efektywnego i optymalnego dostępu,
- zapewnia obsługę wielu interfejsów dostępu do bazy.

Pierwsze komercyjne SZBD pojawiły się w latach sześćdziesiątych ubiegłego stulecia i zastąpiły wówczas stosowane systemy oddzielnych plików, w których były przechowywane dane. W tamtych systemach wykorzystywano modele: sieciowy (grafowy) oraz hierarchiczny (drzewiasty) — dominowały one na rynku aż do lat osiemdziesiątych. Dziś bardzo rzadko można spotkać nieliczne implementacje systemów firmy IBM opartych na modelu hierarchicznym.

Druga generacja to systemy relacyjne, oparte na matematycznym pojęciu relacji. Za twórcę tego modelu baz jest uznawany E.F. Codd, matematyk z wykształcenia, który — używając terminologii z teorii mnogości i rachunku zdań — zaproponował solidne podstawy teoretyczne, pozwalające na uporanie się z problemami spójności i redundancji danych. Zaproponował także normalizację jako metodę projektowania struktur danych w bazie oraz zdefiniował algebrę relacji umożliwiającą rozwój języków przetwarzania danych opartych na przetwarzaniu zbiorów, a w szczególności deklaratywnego języka SQL. Pierwsza publikacja E.F. Codda na temat relacyjnego modelu danych pt. *A Relational Model of Data for Large Shared Data Banks* została opublikowana w 1970 roku.

Trzecia generacja jest rozszerzeniem czysto relacyjnych systemów, które — z uwagi na popularyzację obiektowych języków programowania i kolejne obiektowe rozszerzenia standardu SQL — stały się zbyt ubogie. W latach dziewięćdziesiątych pojawiła się koncepcja obiektowych baz danych. Obecnie najczęściej stosuje się model relacyjny wraz z rozszerzeniem obiektowo-relacyjnym oraz model semistrukturalny, oparty na języku XML i powiązanych z nim standardach. Ciągle jednak model relacyjny pozostaje modelem fundamentalnym.

Warto poświęcić trochę uwagi wymogom formalnym, jakim musi sprostać SZBD, aby mógł być traktowany jako system relacyjny. Twórca tego systemu E.F. Codd zapisał z matematyczną precyzją i zwięzłością 12 reguł definiujących wizjonerskie na owe czasy wymagania dla systemu relacyjnego. Wizjonerskie, bo wówczas na rynku funkcjonowały systemy konstruowane w oparciu o model hierarchiczny uzupełnione kilkoma cechami modelu relacyjnego (dwuczęściowy artykuł na ten temat ukazał się w „ComputerWorld” w 1985 r.). Oto krótkie podsumowanie tego zagadnienia [20]:

- Dane w bazie muszą być reprezentowane wyłącznie na poziomie logicznym — jako wartości w tabelach (reguła 1. — Reguła informacji).
- Każda tabela musi mieć zdefiniowany klucz główny, który funkcjonuje jako unikalny identyfikator wiersza w tabeli. Dostęp do każdej jednostki informacji

(wartości atomowej) musi być możliwy poprzez nazwę tablicy, nazwę kolumny i wartości klucza głównego (reguła 2. — Reguła gwarantowanego dostępu).

- Baza musi wykorzystywać katalog systemowy, w którym w tablicach są przechowywane metadane (reguła 4. — Metadane).

- W katalogu systemowym, a nie w aplikacjach klienckich, muszą być również przechowywane reguły integralności danych (reguła 10. — Reguła niezależności reguł integralności).
- Musi istnieć jeden język pozwalający na manipulowanie danymi w bazie (reguła 5. — Reguła uniwersalnego języka).
- Muszą być możliwe operacje na wielu rekordach przy użyciu pojedynczych poleceń języka (reguła 7. — Operacje dodawania, modyfikacji i usuwania rekordów na wysokim poziomie).
- Niedopuszczalne są jakiekolwiek zmiany danych w tabeli, które naruszałyby reguły integralności zapisane w katalogu systemowym (reguła 12. — Reguła zachowania integralności).
- Zmiany logiczne czy fizyczne dokonywane na danych w bazie (np. zdefiniowanie indeksu, podział tablicy itp.) nie powinny mieć znaczenia dla użytkownika (reguła 8. — Fizyczna niezależność danych, reguła 9. — Logiczna niezależność danych oraz reguła 11. — Niezależność dystrybucyjna).
- Muszą być możliwe modyfikacje danych wynikowych, udostępnianych poprzez perspektywy (reguła 6. — Reguła wprowadzania modyfikacji w perspektywach).
- Wartości puste (NULL) są reprezentowane jako brak informacji, a obsługa tych wartości odbywa się w sposób jednolity, niezależny od typu danych (reguła 3. — Systematyczne traktowanie wartości pustych).
- I jeszcze uzupełnienie zwane regułą zero: Każdy system uważany za relacyjny musi mieć możliwość zarządzania danymi tylko za pomocą mechanizmów

relacyjnych.

Wtedy, kiedy te podstawy dla modelu relacyjnego były formułowane, chyba nawet sam autor nie przewidywał, że wkrótce nastaną prawdziwe złote czasy i okres prosperity dla tego systemu. Z bardzo kosztownego i skomplikowanego stał się przyjazny dla użytkownika, rozpowszechniony masowo i dostępny nawet dla indywidualnych, prywatnych zastosowań. Stało się to dzięki powstaniu wielu różnych systemów zarządzania relacyjnymi bazami danych, takich jak: Informix, Progress, Sybase, Oracle, FoxPro, Microsoft SQL Serwer, DB2, MySQL, Microsoft Access i wiele, wiele innych. Na rozwój technologii ma wpływ również Internet. Obecnie lawinowo narastająca ilość danych gromadzonych w bazach zmusza informatyków do

poszukiwania nowych rozwiązań, tworzenia i przechowywania danych w chmurze, poza siedzibą klienta i udostępniania tych danych poprzez sieć. Niezwykle interesujący będzie rozwój systemów zarządzania bazami danych w takim środowisku w najbliższym czasie.

Systemy z bazą danych są rozpowszechnione w bardzo różnych zastosowaniach, wymuszających określoną specjalizację ze względu na specyfikę realizowanych problemów. Najpopularniejsze są dwa główne rodzaje baz danych — operacyjne i analityczne.

Operacyjne bazy danych (OLTP) to takie, które podlegają częstym modyfikacjom, są one wykorzystywane głównie do przetwarzania transakcji, np. w bankach, sklepach czy wytwórniach. Dane w takiej bazie muszą być zawsze aktualne i poprawne, niezależnie od dynamiki zmian i operacji w nich rejestrowanych.

Analityczne bazy danych (OLAP) są wykorzystywane w sytuacjach, kiedy jest potrzeba gromadzenia i śledzenia danych zmieniających się w czasie, np. bazy danych statystycznych, bazy w laboratoriach chemicznych, agencjach marketingowych itp. Ten typ bazy pomaga prześledzić pewne trendy, przygotować strategie biznesowe itd.

Uwzględniając różnorodne zastosowania systemów informacyjnych z bazami danych, można jeszcze wyróżnić [17]:

- systemy czasu rzeczywistego (ang. *real-time*), w których jest realizowane sterowanie procesami odbywającymi się w czasie rzeczywistym,
- systemy projektowe, np. *systemy CAD*, wspomagające projektowanie, w których głównym problemem są prace grupowe, wersjonowanie projektów,
- systemy CASE (ang. *Computer Aided Software Engineering*) zajmujące się modelowaniem systemów, projektowaniem struktur danych i aplikacji, tworzeniem dokumentacji czy generowaniem fizycznych struktur,
- systemy zarządzania obiegiem dokumentów (ang. *Workflow*) mają za zadanie wspomaganie pracy nad dokumentami, śledzenie obiegu dokumentów i wykonywanych w związku z nimi czynności,
- systemy informacji przestrzennych (ang. *Geographical Information System* — GIS) zajmują się projektowaniem map, analizą przestrzenną, planowaniem przestrzennym itp.,
- systemy wspomagania decyzji (ang. *Decision Support System*) mają zadania doradcze — za pomocą różnorodnych narzędzi dają możliwość przeprowadzania analiz i obrazowania skutków podejmowanych decyzji,
- systemy zarządzania ERP (ang. *Enterprise Resorce Planning*) — realizują zintegrowane zarządzanie oraz planowanie zasobów przedsiębiorstwa; jest to grupa systemów obejmujących różne klasy tego wyspecjalizowanego oprogramowania rozróżnianego symbolami MRP oraz MRPII i MRPIII,
- systemy informowania kierownictwa (ang. *Executive Information Systems)*,

- systemy przeznaczone dla portali internetowych, serwerów WWW i wiele innych.

Rozdział 1.

Modelowanie logiczne

Analizując cykl życia systemu informatycznego, można wyróżnić kolejne etapy, poprzez które system ewoluuje — od momentu zdefiniowania problemu projektowego aż po wdrożenie go u użytkownika. Do najważniejszych należą: analiza, projektowanie, implementacja i wdrażanie.

Etap analizy wymaga dogłębnego rozpoznania określonego wycinka rzeczywistości, dla którego jest projektowany system, oraz dokładnego określenia wymagań przyszłych użytkowników w stosunku do struktury i funkcjonalności systemu. Na tym etapie zapada decyzja o wyborze metody i środków wykorzystanych do projektowania oraz dokonuje się wyboru modelu bazy. Wynikiem prac tej fazy są modele konceptualne, które opisują rodzaje i struktury informacji przetwarzanej w systemie oraz funkcjonalności budowanej aplikacji.

Etap projektowania to etap tworzenia modeli logicznych, w których są definiowane typy danych i ich opis, a następnie jest generowany projekt fizyczny w związku z wybraną platformą systemową dla bazy danych i aplikacji.

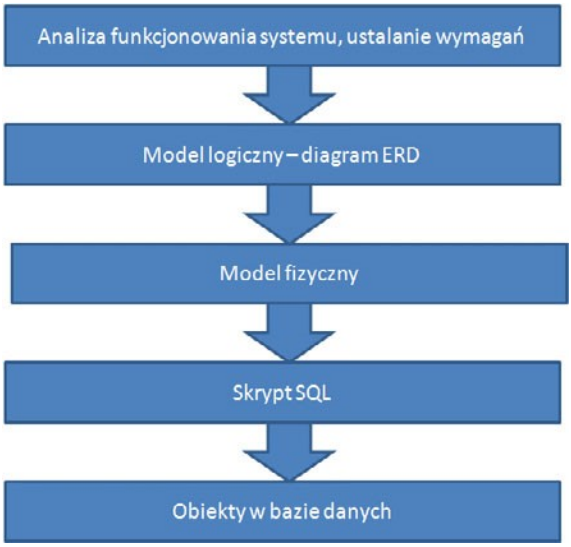
Etap implementacji to programowanie bazy i aplikacji oraz testy sytemu przed wdrożeniem u użytkownika.

Wdrożenie obejmuje instalację gotowego systemu u użytkownika, załadowanie aktualnych danych do bazy oraz wprowadzenie ewentualnych poprawek i uaktualnianie wersji.

Modelowanie danych jest metodą projektowania nowej bazy danych zwaną metodą projektowania zstępującego, czyli postępującej od ogółu do szczegółu (ang. *top-down design*). Takie podejście oznacza tworzenie systemu przez stopniowe wyodrębnianie jego składników od poziomu znacznej ogólności aż do wymuszonej metodologią postępowania koniecznej szczegółowości.

W modelowaniu definiuje się dwa rodzaje modeli — logiczny (konceptualny) i fizyczny (implementacyjny) (rysunek 1.1).

Projekt logiczny (zwany również wysokopoziomowym modelem bazy danych — por. Ullman, Widom, s. 93) prezentuje obiekty niezależnie od modelu implementacyjnego. Może on być zaprezentowany jako diagram związków encji (najstarsza metoda) lub za pomocą języka UML (ang. *Unified Modeling Language* — zunifikowany język modelowania) albo też języka ODL (ang. *Object Description Language* — język opisywania



Rysunek 1.1. *Etapy procesu projektowania w modelowaniu logicznym*

obiektów). W tym opracowaniu zostanie omówiona fundamentalna metoda definiowania modeli logicznych w projektowaniu relacyjnych baz danych — model związków encji (ang. *Entity Relationship Model*). W tym modelu są uwzględnione obiekty reprezentujące określone kategorie informacji, których opis ma być rejestrowany w projektowanym systemie. Obiekty powiązane są ze sobą za pomocą określonych związków. Całość jest zapisana w ustalony graficznie sposób w postaci diagramu (ang. *Entity Relationship Diagram*).

Projekt fizyczny wykorzystywany na etapie implementacji systemu zawiera konkretne typy struktur danych, zdefiniowanych dla modelu konceptualnego. Mogą to być struktury danych dla modeli: relacyjnego, obiektowego oraz obiektowo-relacyjnego.

W pierwszym etapie modelowania otrzymujemy projekt logiczny bazy zawierający encje i związki między encjami.

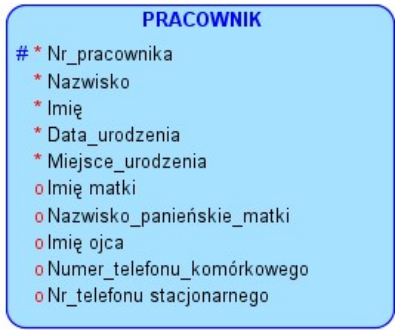
1.1. Encje i atrybuty

Encja (ang. *entity*) w projekcie logicznym to typ obiektu świata rzeczywistego: materialny (np. czytelnik, książka, samochód) lub abstrakcyjny (np. zdarzenie, pojęcie), którego opis właściwości będzie przechowywany w projektowanym systemie. Każdy element analizowanej rzeczywistości musi być reprezentowany przez jedną encję — podczas wprowadzania opisu takiego obiektu do gotowego już systemu nie może być wątpliwości, gdzie zaklasyfikować daną informację. Stopień uszczegółowienia analizowanej rzeczywistości, a tym samym liczba encji pozyskanych w trakcie analizy, zależy od przeznaczenia projektowanego systemu.

Encja posiada unikalną nazwę — jest to rzeczownik liczby pojedynczej. Encja reprezentuje pewien zbiór wystąpień obiektów tego samego typu (minimum dwóch), mających takie same właściwości (atrybuty). Podczas modelowania danych dowolny obiekt ze świata rzeczywistego jest reprezentowany jako wystąpienie encji. Każde wystąpienie encji (instancja encji) musi być wyraźnie odróżnialne od wszystkich

innych instancji tego typu encji. Prawidłowa identyfikacja encji jest jednym z najważniejszych zadań w trakcie projektowania bazy danych.

Właściwości encji są opisywane za pomocą atrybutów (rysunek 1.2). Liczba atrybutów w opisie encji może być różna — w wypadku encji zwanych słownikami mogą to być tylko dwa atrybuty. Mogą jednak wystąpić encje mające bardzo rozbudowany opis sięgający kilkuset atrybutów, np. encje odpowiadające rozbudowanym raportom.



Rysunek 1.2. *Przykład encji Pracownik*

W bazach relacyjnych atrybuty powinny być prostych typów, co oznacza, że jeśli w bazie mielibyśmy zapisać np. okres urlopu pracownika, to własność okres_urlopu musi być reprezentowana przez dwa atrybuty mające prosty typ data, tzn. data_początku_urlopu i data_końca_urlopu. Z podobnym problemem rozbijania atrybutów na typy proste musimy się zmierzyć, wprowadzając do opisu encji Pracownik np. atrybut Adres. Jeśli ten atrybut ma przechowywać dane traktowane jako pomocnicze, które nie będą podlegać wyszukiwaniu, to można pozostawić Adres jako atrybut typu znakowego. Jednak wyróżnienie w adresie poszczególnych elementów (m.in. Miejscowości, Kodu pocztowego, Ulicy, Nr domu, Numeru mieszkania) i utworzenie dla każdego takiego elementu oddzielnego atrybutu pozwoli swobodnie korzystać z tych danych w wypadku koniecznej modyfikacji wartości wybranego atrybutu czy też w celu wyszukania np. pracowników z danej miejscowości.

Każdy atrybut musi mieć:

- unikalną nazwę w ramach jednej encji,
- określoną dziedzinę definiującą typ danych, maksymalny rozmiar, zbiór dozwolonych wartości lub zakres wartości,
- informację NULL/NOT NULL określającą, czy dozwolone lub niedozwolone są wartości puste, tzn. czy atrybut musi mieć podane wartości, czy może być pominięty podczas wprowadzania danych do bazy, ♦ atrybut może też mieć unikalne wartości.

Atrybut musi spełniać określone zadanie, tzn. identyfikować, opisywać, klasyfikować, określać ilość lub wyrażać stan encji.

Atrybuty można podzielić na atrybuty identyfikacyjne i opisowe.

Identyfikator to atrybut lub zbiór atrybutów jednoznacznie identyfikujący wystąpienie encji. Mogą być identyfikatory naturalne, pochodzące z rzeczywistości, zweryfikowane już w innych systemach — np. w opisie pracownika będą to PESEL, NIP, REGON czy nr dowodu osobistego. Identyfikatorem sztucznym jest atrybut numeryczny dodany do opisu w celu numerowania kolejnego wystąpienia encji, np. Nr pracownika, Nr katalogowy itp.

Atrybut opisowy (deskrypcyjny) to każdy atrybut poza identyfikatorem. Reprezentuje podstawowe własności encji przechowywane w bazie. Wartości deskryptorów mogą być opcjonalne lub obowiązkowe.

W notacji Barkera stosuje się następujące znaki dla oznaczenia rodzaju atrybutu:

- # — oznacza identyfikator, atrybut taki ma unikalne i obligatoryjne wartości,
- * — oznacza atrybut z wartościami obligatoryjnymi,
- o — oznacza atrybut z wartościami opcjonalnymi.

1.2. Typy związków

Związek, zwany asocjacją, reprezentuje powiązania pomiędzy encjami i modeluje zależności występujące między obiektami świata rzeczywistego. Wnosi do projektu określone informacje, np. *Klient posiada Rachunek*, *Rachunek należy do Klienta*. Związek jest przedstawiony graficznie za pomocą linii łączącej encje — i dodatkowo zawierającej oznaczenia ułatwiające interpretację związku. Graficzny sposób przedstawienia związku jest różny w zależności od przyjętej notacji, musi jednak jednoznacznie określać trzy cechy związku: stopień związku, typ związku i jego istnienie.

Stopień związku określa liczbę encji połączonych związkiem.

Wyróżnia się związki:

- unarne (łączące encję samą ze sobą),
- binarne (łączące dwie encje),
- ternarne (łączące trzy encje), *n*-arne (łączące *n* encji).

Typ związku, zwany licznością związku, określa, ile wystąpień jednej encji może wchodzić w powiązania z różnymi wystąpieniami innej encji.

Wyróżnia się następujące typy związków:

- jeden do jeden, oznaczane jako 1:1, jeden do wielu, oznaczane jako 1:N,
- wiele do wielu, oznaczane jako N:M.

Dwa pierwsze typy związków (1:1 i 1:N) to związki proste, natomiast związek N:M to związek złożony lub wieloznaczny, traktowany w informatyce jako nieimplementowalny. Związek 1:1 to związek jedno-jednoznaczny, związek 1:N to związek jednoznaczny.

Typ „wiele" w notacji Barkera oznaczamy w diagramie linią rozgałęzioną (tzw. „kurzą stopką"), typ „jeden" — linią pojedynczą.

Istnienie (znane również jako klasa przynależności lub uczestnictwo) określa, czy związek jest opcjonalny, czy obligatoryjny. Jeśli jest chociaż jedno wystąpienie encji danego typu nie biorące udziału w powiązaniu, to ta encja jest w związku opcjonalnym. Jeśli wszystkie wystąpienia encji danego typu muszą brać udział w powiązaniu, to

związek jest obligatoryjny. W notacji Barkera uczestnictwo opcjonalne jest oznaczone linią przerywaną, uczestnictwo obligatoryjne — linią ciągłą. Wszystkie przykłady poniższych związków są prezentowane w notacji Barkera.

1.3. Transformacja modelu logicznego do fizycznego

Na etapie analizy wyróżniamy i opisujemy encje, uwzględniamy też wszystkie bezpośrednie powiązania między encjami i zapisujemy wyniki tych prac w postaci diagramu. Otrzymujemy w ten sposób model logiczny danych, niezależny od implementacji. Projekt logiczny jest następnie transformowany do modelu fizycznego, odpowiednio dla wybranego systemu zarządzania bazą danych (tabela 1.1).

Tabela 1.1. *Przyporządkowanie obiektów modelu logicznego i fizycznego*

Obiekty modelu logicznego	Obiekty modelu fizycznego
Encja	Tabela
Atrybut encji	Kolumna tabeli
Identyfikator encji	Klucz główny tabeli
Związek	Klucz obcy i reguły integralności

Zasady transformacji encji są następujące:

1. Dla każdej encji jest definiowana tablica o tej samej nazwie. Dobrym zwyczajem jest stosowanie rzeczowników liczby mnogiej jako nazw tablic.
2. Każdy atrybut encji odpowiada kolumnie tablicy.
3. Typ danych atrybutu encji jest odwzorowany w odpowiadający mu typ danych atrybutu relacji.
4. Unikalny identyfikator encji staje się kluczem głównym tablicy.
5. Obligatoryjność atrybutów encji jest reprezentowana przez ograniczenie NOT NULL dla danego atrybutu w tablicy.
6. Opcjonalność atrybutów encji jest reprezentowana przez ograniczenie NULL dla danego atrybutu w tablicy.
7. Ograniczenia integralnościowe dla atrybutów encji są również przenoszone jako ograniczenia integralnościowe atrybutów w tablicy.

Transformacja związków odbywa się według następujących reguł:

- Związek unarny (typu 1:1 i 1:N) jest implementowany poprzez klucz obcy w tej samej tabeli.
- Związek binarny typu 1:1 jest implementowany poprzez klucz obcy we wskazanej tabeli. Jeśli encje powiązane takim związkiem były równorzędne, tzn.

uczestnictwo związku było takie samo na obu stronach związku, to w trakcie transformacji następuje wymiana kluczy obcych. Jeśli klucze obce występują w obu tablicach, projekt wymaga optymalizacji i usunięcia jednego klucza w wybranej tabeli. Taka decyzja musi być poprzedzona analizą spodziewanych wartości w kolumnach kluczy obcych. Jeśli uczestnictwo związku jest różne, to klucz obcy występuje w tablicy definiowanej dla encji, która ma powiązanie obligatoryjne.

- Związek binarny typu 1:N jest implementowany poprzez klucz obcy w tabeli po stronie „wiele".
- Związek binarny typu N:M wymaga zdefiniowania encji asocjacyjnej, która pozwoli zastąpić ten związek dwoma związkami typu 1:N (strona „wiele" związku oraz uczestnictwo obligatoryjne zawsze występuje przy encji wtrąconej). Encja asocjacyjna może być encją wirtualną w projekcie logicznym, będzie wtedy traktowana jako encja słaba i zostanie dla niej zdefiniowana tablica w projekcie fizycznym. Encja słaba nie posiada swojego identyfikatora, natomiast odziedziczy ona identyfikator ze związków, w które wchodzi. Związek N:M będzie więc reprezentowany w modelu relacyjnym przez dodatkową tablicę, w której najczęściej klucze obce są włączone do klucza głównego.

Ograniczenia integralnościowe, zwane więzami integralności, to reguły, które zapewniają, że dane wprowadzane do bazy są poprawne. Więzy mogą dotyczyć pojedynczego atrybutu, wybranych atrybutów lub całej tablicy. Nad zapewnieniem integralności danych czuwa System Zarządzania Bazą Danych. Wyróżniamy następujące rodzaje ograniczeń

integralnościowych:

- więzy klucza głównego,
- więzy klucza obcego,
- więzy dotyczące dziedzin,
- więzy dotyczące dopuszczalności lub zakazu wprowadzania dla wskazanego atrybutu wartości NULL,
- więzy zwane regułami biznesowymi.

Pierwsza reguła wymusza, aby klucz główny miał obligatoryjne i unikalne wartości.

Zasada dotycząca klucza obcego wynika z jego definicji. Klucz obcy musi mieć wartość z kolumny klucza głównego tablicy nadrzędnej, może również dopuszczać NULL (w wypadku uczestnictwa opcjonalnego od strony encji podrzędnej). Ograniczenie to wymusza, aby w tablicy nadrzędnej istniały wartości, do których odwołuje się klucz obcy.

Więzy dotyczące dziedziny mogą ograniczać zbiór wartości domeny do określonego podzbioru: przedziału lub wyliczeniowej listy wartości, np.: 0<WIEK<100 albo OCENA IN (2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0).

Więzy dotyczące dopuszczalności lub zakazu wprowadzania wartości NULL dla danego atrybutu są ustalane w trakcie projektowania opisu encji. Jeśli projektowany atrybut ma wartości obligatoryjne, to odpowiadająca mu kolumna tabeli jest definiowana z opcją

NOT NULL, co skutkuje wymuszaniem wprowadzania wartości dla tego atrybutu podczas wstawiania wierszy do tabeli.

Reguły biznesowe muszą być jawnie zapisane w projekcie, w repozytorium. Nie istnieje żadna notacja graficzna dla zapisania tych reguł. Przykładem tego typu więzów mogą być różne zasady stosowane w określonej organizacji, np. w bibliotece można wypożyczyć maks. 10 książek jednocześnie na okres 2 miesięcy itp.

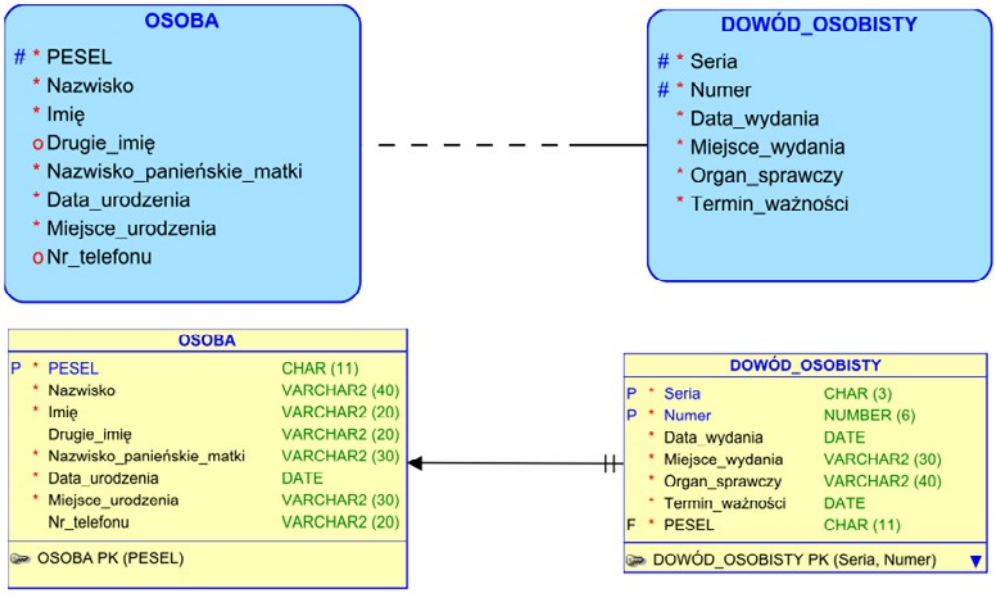
Przykłady poszczególnych związków i ich fizyczna implementacja zostaną omówione poniżej.

1.4. Przykłady implementacji związków

Związek binarny typu 1:1

Z takim związkiem mamy do czynienia wówczas, gdy z jednym wystąpieniem encji pierwszego typu może wejść w powiązanie co najwyżej jedno wystąpienie encji drugiego typu — i na odwrót (na związek należy patrzeć z obu stron).

Jako przykład takiego związku rozważmy związek pomiędzy encjami OSOBA i DOWÓD OSOBISTY (rysunek 1.3). Możemy go opisać słownie następująco: każdy dowód osobisty musi należeć do co najwyżej jednej osoby. Osoba może posiadać jeden dowód osobisty (jeśli jest pełnoletnia i otrzymanego dowodu nie utraciła).

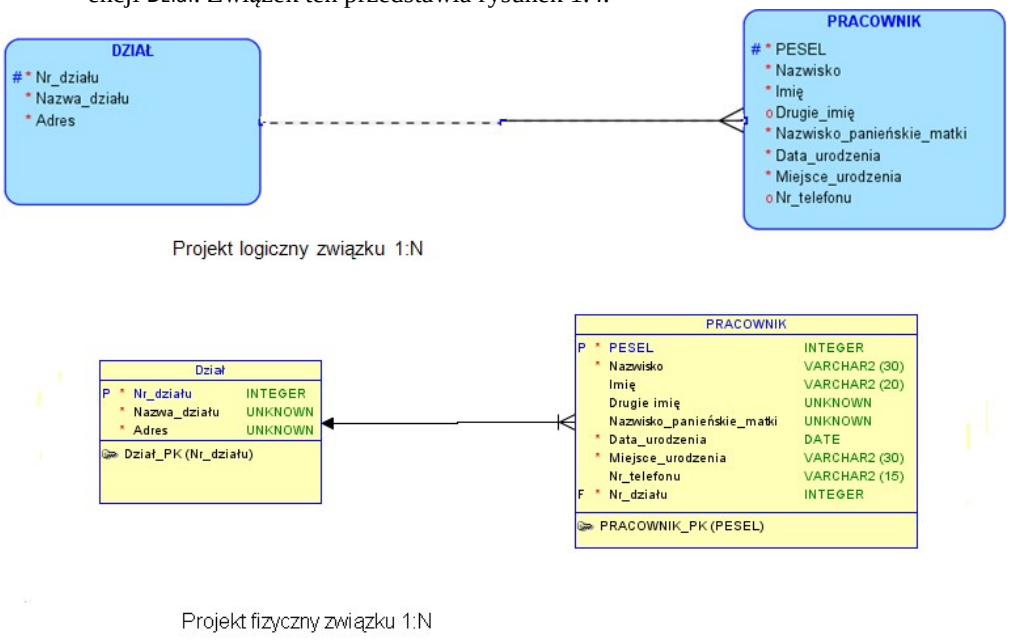


Rysunek 1.3. *Projekt i implementacja związku 1:1*

Rysunek 1.3 przedstawia omówiony związek binarny typu 1:1, opcjonalny od strony encji Osoba, obligatoryjny od strony encji Dowód osobisty — oraz fizyczną implementację tego związku.

Związek binarny typu 1:N

Jako przykład związku 1:N rozważmy związek pomiędzy encjami DZIAŁ i PRACOWNIK prezentujący informacje o zatrudnieniu. Możemy go opisać słownie: każdy pracownik pracuje w jednym dziale, posiada jedno miejsce pracy i musi być zatrudniony, ponieważ informacji o osobach niezwiązanych z działami firmy nie przechowujemy w bazie. W dziale może pracować wielu pracowników, ale też mogą występować działy w fazie organizacji lub likwidacji, które nie mają obsady kadrowej, czyli w których po prostu nikt nie pracuje. Opisaną wyżej sytuację można modelować za pomocą związku binarnego typu 1:N, obligatoryjnego od strony encji Pracownik, opcjonalnego od strony encji Dział. Związek ten przedstawia rysunek 1.4.



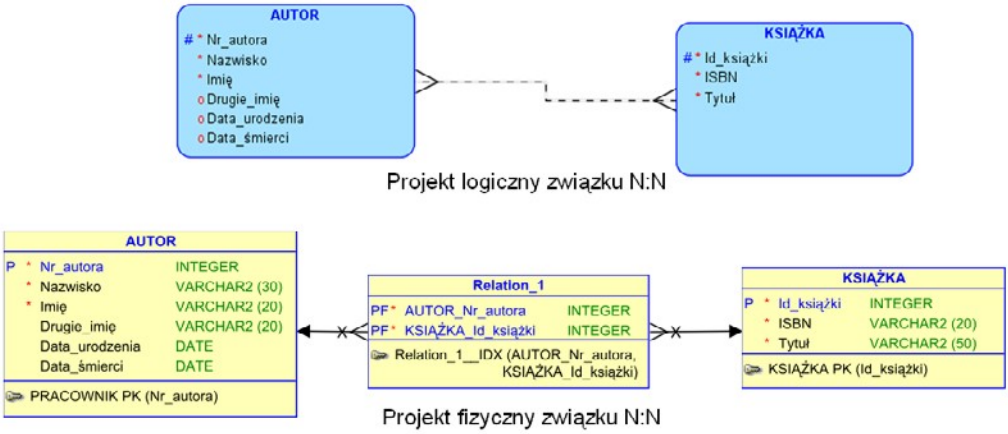
Rysunek 1.4. *Projekt i implementacja związku 1:N*

Związek binarny typu N:M

Związek N:M jest w informatyce traktowany jako nieimplementowalny, tzn. nie może być zapisany w bazie fizycznej za pomocą dwóch tablic, bo w każdej z nich klucze obce miałyby wiele wartości, co jest niezgodne z wymaganiami stawianymi bazom relacyjnym. Związek wieloznaczny (N:M) wymaga dopracowania poprzez włączenie dodatkowej encji asocjacyjnej, w której będą przechowywane informacje wynikające ze związku.

Wprowadzenie dodatkowej encji przekształca związek typu wiele do wielu w dwa związki typu jeden do wielu. Strona „wiele" związku oraz uczestnictwo obligatoryjne zawsze jest po stronie encji asocjacyjnej.

Należy rozróżnić dwa typy związków wieloznacznych. Pierwszy typ takiego powiązania to związek, w którym pary wystąpień poszczególnych encji reprezentujących powiązanie występują jednokrotnie. W takim związku encja asocjacyjna będzie encją słabą (ang.*weak entity*) i odziedziczy identyfikator ze związku, zaś w projekcie fizycznym będzie ona reprezentowana przez tablicę ze złożonym kluczem głównym składającym się z dwóch kluczy obcych (w wypadku związku binarnego). Przykładem takiego związku może być ten pomiędzy encjami AUTOR i KSIĄŻKA. Jeden autor może napisać wiele książek i jedna książka może być napisana przez wielu autorów. Przy modelowaniu tego związku przy użyciu jakiegokolwiek narzędzia typu CASE zostanie zdefiniowana encja asocjacyjna słaba. Rozwiązanie to będzie poprawne, bo w tablicy dla encji asocjacyjnej z kluczem głównym <Nr_Autora,Id_Książki> zapiszemy informację o autorstwie książki, respektując wymagania unikalności wartości dla klucza głównego — przecież żaden autor nie pisze wielokrotnie tej samej książki. Związek ten i jego fizyczny obraz przedstawia rysunek 1.5.



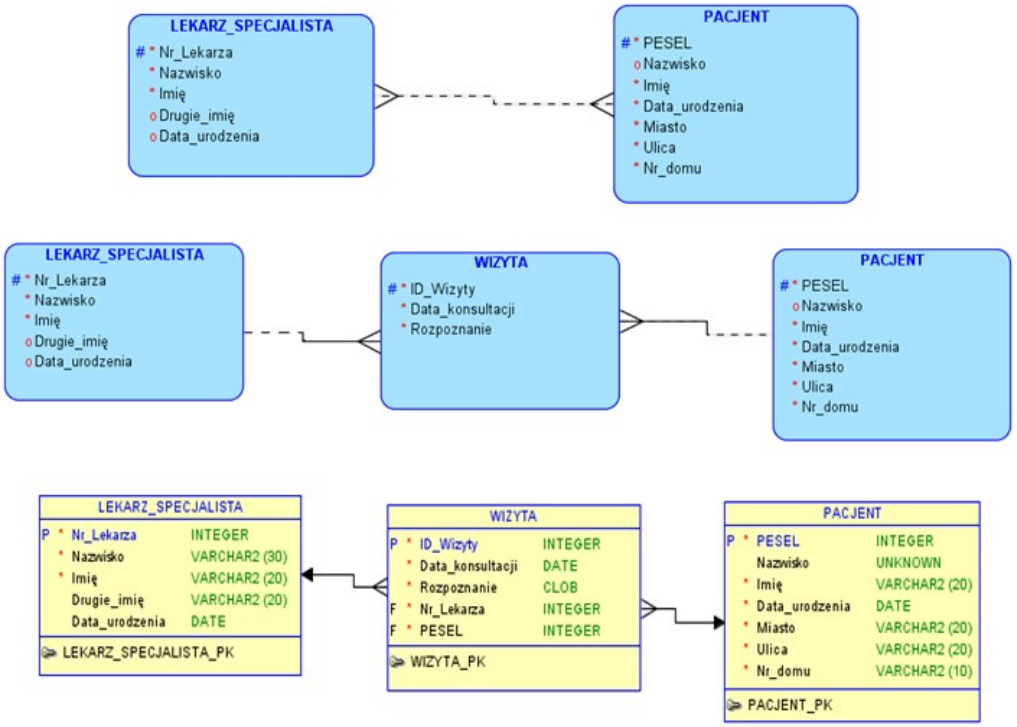
Rysunek 1.5. *Związek Autor-Książka N:M, implementowany z encją słabą i projekt tablic*

Inaczej jest implementowany związek typu N:M, jeśli pary wystąpień poszczególnych encji biorące udział w powiązaniu mogą wystąpić wielokrotnie, np. w związku Lekarz_Specjalista-Pacjent. W takim związku jeden Lekarz_Specjalista przyjmuje wielu Pacjentów i jeden Pacjent może leczyć się u wielu Lekarzy_Specjalistów, a dodatkowo ten sam Lekarz_Specjalista może wielokrotnie przyjmować tego samego Pacjenta. Wprowadzenie encji słabej w celu uszczegółowienia związku wieloznacznego powodowałoby istotne ograniczenia dla informacji rejestrowanych w projektowanym systemie. W tablicy dla encji słabej moglibyśmy zapisać tylko unikalne wartości dla klucza głównego <Nr_Lekarza_Specjalisty,Nr_Pacjenta>, nie moglibyśmy natomiast wprowadzać informacji o kolejnych konsultacjach danego pacjenta u tego samego specjalisty. Jeśli w bazie miałyby być zapisane informacje o poszczególnych wizytach pacjenta u lekarza specjalisty, w projekcie logicznym należy samodzielnie zdefiniować encję asocjacyjną dla dopracowania związku N:M, np. encję Wizyta. Taka encja będzie miała zdefiniowany własny identyfikator. Przy encji Wizyta będzie strona „wiele”

związków i uczestnictwo obligatoryjne. Interpretacja uzyskanych związków łączących encję Wizyta jest następująca:

- Jeśli istnieje wystąpienie encji Wizyta, to musi ono dotyczyć konkretnego Pacjenta i Lekarza_Specjalisty (liczność „jeden”).
- Nie może istnieć wystąpienie encji Wizyta bez wystąpienia encji Pacjent i Lekarz_Specjalista (uczestnictwo obligatoryjne).
- Zarówno każdy Pacjent, jak i Lekarz_Specjalista mogą wielokrotnie realizować Wizyty (liczność „wiele”).

Taki związek może mieć własne atrybuty. Przykładami atrybutów dotyczących związku są: data_konsultacji, rozpoznanie itp. Przykład omawianego związku prezentuje rysunek 1.6.



Rysunek 1.6. *Związek N:M pomiędzy encjami Lekarz_Specjalista i Pacjent oraz encja Wizyta na dopracowanie związku*

Związek unarny rekursywny

Związek unarny rekursywny oznacza powiązanie pomiędzy różnymi wystąpieniami encji tego samego typu. Na przykład w encji Pracownik (w której poszczególne wystąpienia reprezentują kolejnych pracowników firmy) określmy związek Kierownik, wskazujący osoby pełniące funkcje kierownicze i podległych im pracowników. Wśród pracowników możemy wskazać kogoś, kto jest kierownikiem (nie wszyscy są

kierownikami), komu podlegają inni pracownicy (może być ich wielu). Z drugiej strony, jeden pracownik może mieć jednego kierownika, ale również są wśród zatrudnionych osób tacy pracownicy, którzy nie mają swojego szefa (np. dyrektor). Związek Kierownik jest więc związkiem typu 1:N wskazującym kierownika i N-podległych pracowników. Ponieważ zarówno kierownicy, jak i podlegli im pracownicy są wystąpieniami encji Pracownik, to związek jest określony na jednym typie encji (stopień unarny). Jest to związek rekursywny, bo reprezentuje wielopoziomową strukturę zależności służbowych między pracownikami. Jest związkiem opcjonalnym obustronnie (to jedyna poprawna wersja), ponieważ wśród pracowników nie wszyscy są kierownikami i nie wszyscy są podległymi pracownikami. Zbiór wystąpień każdej encji jest zbiorem skończonym.

Przykład takiego związku reprezentuje rysunek 1.7.

Rysunek 1.7. *Związek unarny Pracownik*

Związek ternarny

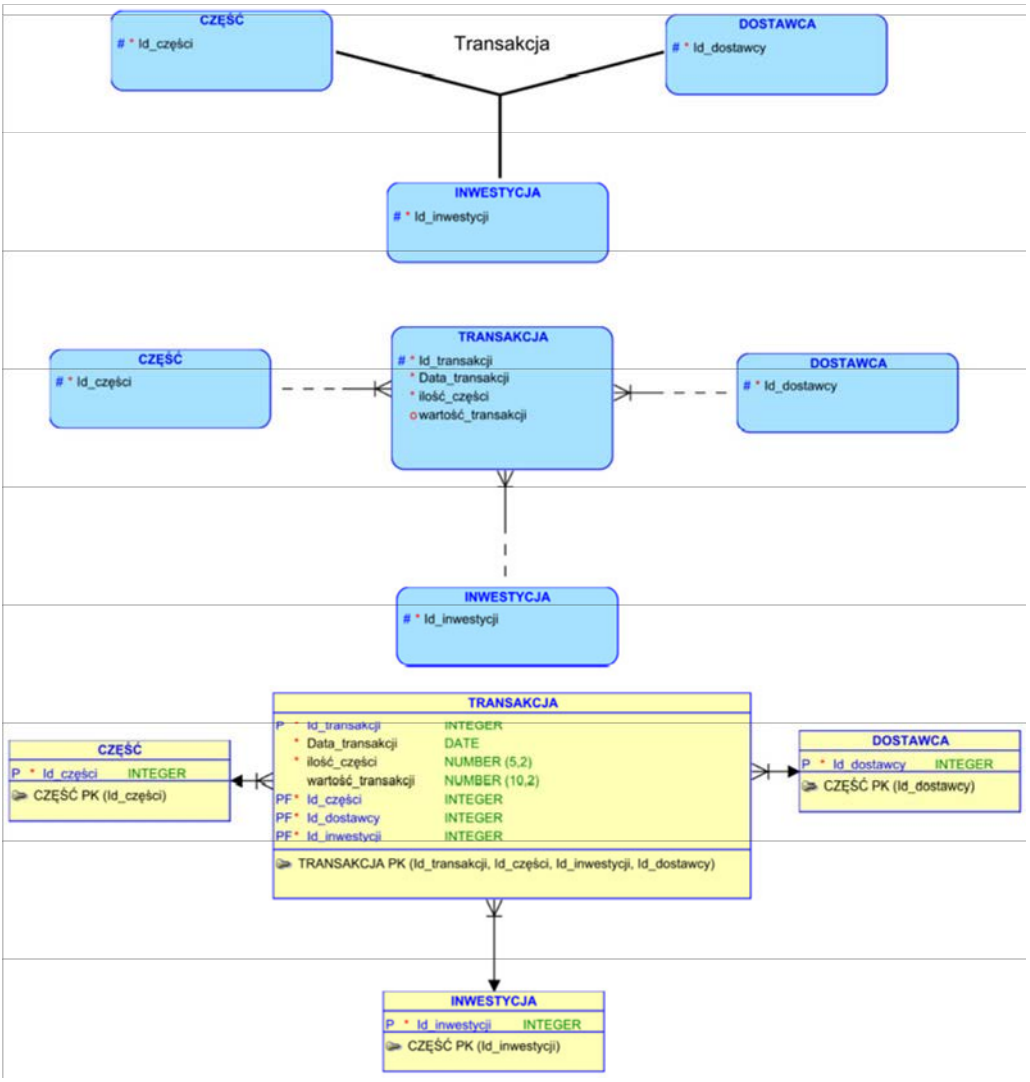
Tego typu związek łączy jednocześnie trzy encje. Jest to związek nieupraszczalny, tzn. nie można go wyrazić za pomocą związków binarnych, bo w tym wypadku każdy związek pomiędzy parą encji jest związkiem wieloznacznym, zaś wyróżnianie związków binarnych wprowadza dodatkową informację do modelowanej rzeczywistości i powoduje utratę finalnej informacji, którą zawiera związek ternarny.

Rozważmy związek pomiędzy trzema encjami CZĘŚĆ-DOSTAWCA-INWESTYCJA (rysunek 1.8). Taki związek uwzględniamy w celu wyliczenia kosztów realizowanych inwestycji. Próba uproszczenia takiego związku poprzez uwzględnianie związków binarnych nie jest do zaakceptowania. Każdy związek binarny jest typu N:M i wyraża informację inną niż docelowa, np.:

- Jedna część może pochodzić od różnych dostawców, każdy dostawca może dostarczać wielu części — to informacja o ofercie dostawców,
- Jedna część może być dostarczona na różne inwestycje, każda inwestycja może wykorzystywać wiele części — to informacja o zapotrzebowaniu na materiały na poszczególnych inwestycjach,

- Jeden dostawca może dostarczać części na wiele inwestycji, z jedną inwestycją współpracuje wielu dostawców — to informacja o współpracy dostawców z inwestorami realizującymi poszczególne inwestycje.

Żaden ze związków binarnych nie może posłużyć do wyliczenia kosztów poszczególnych inwestycji. Dopiero jednoczesny związek trzech encji umożliwi realizację takiego zadania. Koszt inwestycji obliczymy, uwzględniając konkretną inwestycję, wykorzystane części i dostawcę — każdy dostawca może przecież dyktować własne ceny na poszczególne części. Związek będzie posiadał również własne atrybuty, np.: Ilość(części), datę itp.



Rysunek 1.8. *Związek ternarny CZĘŚĆ-DOSTAWCA-INWESTYCJA*

1.5. Pragmatyczne aspekty modelowania

Wszystko to, co omówiliśmy w poprzednim rozdziale, znacznie przybliża problematykę modelowania danych. Warto uzupełnić te rozważania ważnymi praktycznymi aspektami tej metody projektowania.

1.5.1. Problem pułapek połączeń

Zdefiniowany projekt musi być poddany wnikliwej ocenie w celu wychwycenia możliwych błędów. Należy przeanalizować wszystkie możliwe zapytania do bazy kierowane przez przyszłego użytkownika, aby zauważyć niepoprawne połączenia między encjami. W projekcie mogą bowiem wystąpić problemy — zwane pułapkami — które są trudne do zlokalizowania na pierwszy rzut oka.

Przeanalizujmy fragment projektu, w którym modelowana jest określona rzeczywistość. W projekcie uwzględnione są firmy ubezpieczeniowe, które prowadzą działalność. Każda firma posiada wiele oddziałów (biur) i zatrudnia wielu pracowników. Dla przejrzystości, pominięte zostaną inne kategorie informacji m.in. produkty ubezpieczeniowe, klienci oraz zawierane umowy.

W projekcie rozpatrzymy trzy encje Firma_ubezpieczeniowa, Pracownik i Biuro oraz określone związki. Fragment diagramu związków encji prezentują przykłady.

Przykład 1.1. _____

Encja Firma_ubezpieczeniowa jest encją nadrzędną dla dwóch encji Pracownik i Biuro (rysunek 1.9a). W diagramie występują dwa związki proste typu 1:N. Pierwszy związek dotyczy zatrudnienia pracownika przez firmę, drugi przyporządkowuje biura do poszczególnych firm ubezpieczeniowych.

W takiej bazie nie będzie można jednoznacznie ustalić, w jakim biurze jest zatrudniony określony pracownik. Jeśli informacja o miejscu zatrudnienia pracownika jest istotna, należy zmienić związki w projekcie, co przedstawia rysunek 1.9b, eliminując tym samym problem zwany wiatrakiem. Dla łatwiejszego zobrazowania problemu na rysunku wykorzystano diagramy Venna, zwane diagramami wystąpień lub instancji. Ułatwiają one zilustrowanie, jak konkretne wystąpienia encji są połączone. Owal na diagramie reprezentuje konkretny typ encji — zbiór instancji, zaś linia wskazuje instancję związku.

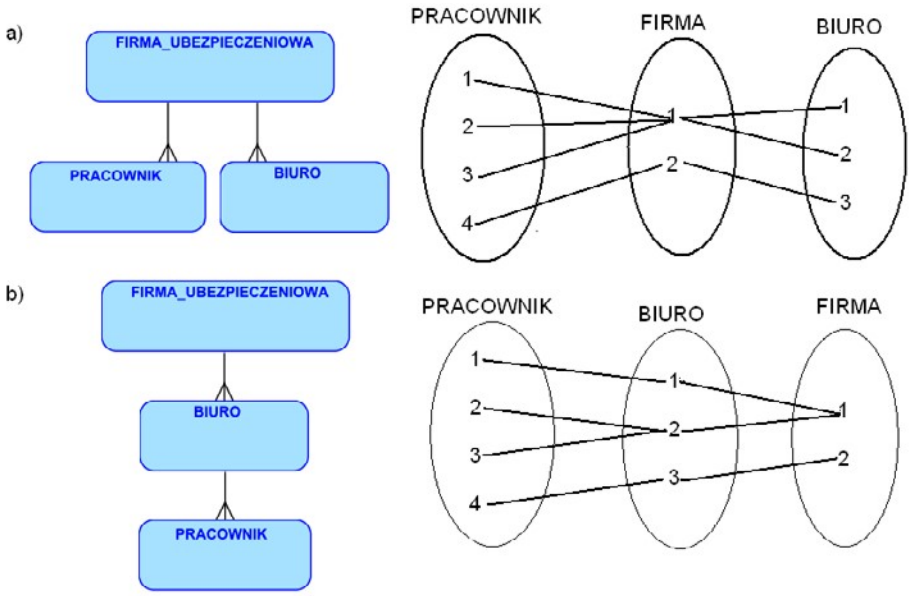
Pułapka znana jako wiatrak albo wachlarz może wystąpić, gdy co najmniej dwa związki typu 1:N wychodzą z jednej encji. Wówczas ścieżki połączeń pomiędzy poszczególnymi wystąpieniami encji nie są jednoznaczne.

Przykład 1.2. _____

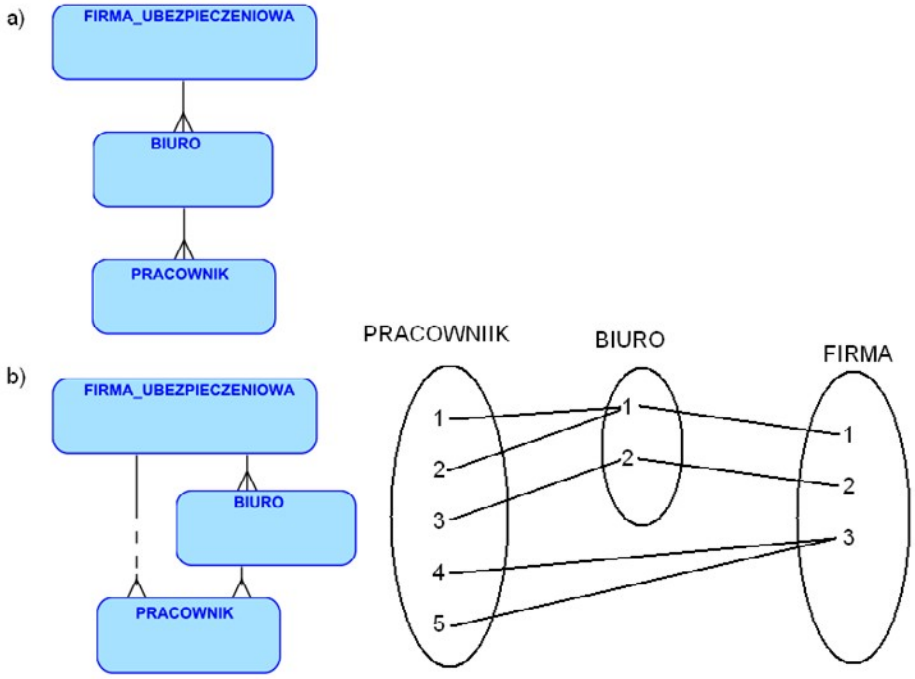
Rozwiązanie, do jakiego doszliśmy w poprzednim przykładzie, prezentuje szeregowo połączone trzy encje Firma_ubezpieczeniowa, Pracownik i Biuro. Rozwiązanie to może

okazać się niewystarczające, jeśli wśród pracowników znajdują się osoby, które nie są zatrudnione w oddziałach lub biurach firmy ubezpieczeniowej, tylko bezpośrednio w centrali. Uczestnictwo związku dotyczącego zatrudnienia pomiędzy encją Biuro a Pracownik będzie opcjonalne, przedstawia to rysunek 1.10a. Model sugeruje istnienie związku pomiędzy typami encji Firma_ubezpieczeniowa i Pracownik, ale nie ma ścieżek łączących określone wystąpienia encji, ponieważ związek opcjonalny może powodować przerwanie takiej ścieżki.

Tę pułapkę zwykle usuwa się poprzez dodanie do modelu dodatkowego związku bezpośredniego, przedstawia to rysunek 1.10b.



Rysunek 1.9. *Pułapka połączeń zwana wiatrakiem i sposób jej eliminowania*



Rysunek 1.10. *Pułapka połączeń zwana próżnią i sposób jej eliminowania*

Pułapka zwana szczeliną albo próżnią może wystąpić wówczas, gdy związki opcjonalne znajdą się w ścieżce powiązań pomiędzy encjami.

1.5.2. Upraszczanie związków wiele do wielu

Problem związków wieloznacznych został szeroko omówiony w rozdziale 1.4 tego opracowania. Należy jednak zwrócić uwagę na fakt, że często skrupulatne dopracowywanie każdego związku N:M niezależnie, poprzez zastępowanie go związkami prostymi, wprowadza problem pułapek połączeń. W takich sytuacjach jedynym racjonalnym rozwiązaniem jest umiejętne zdefiniowanie związku N-arnego reprezentującego informacje z wielu związków wieloznacznych. Np. dla opisu: *Sprawa karna może dotyczyć więcej niż jednego przestępstwa, może też dotyczyć jednego lub więcej oskarżonych*… dopracowywanie każdego binarnego związku wieloznacznego oddzielnie wprowadza kolejne problemy — pułapki połączeń. Traktując każdy związek binarny (sprawa-przestępstwo, sprawa-oskarżony) oddzielnie, można ustalić, jakie przestępstwa były sądzone w ramach określonej sprawy, jacy przestępcy byli oskarżani, ale nie da się uzyskać jednoznacznej informacji, za jakie przestępstwa był sądzony określony przestępca. W tym wypadku jest konieczne zdefiniowanie związku ternarnego o nazwie Wątek_sprawy, który będzie łączył jednoznacznie każdą sprawę z określonym przestępcą sądzonym za każde przestępstwo

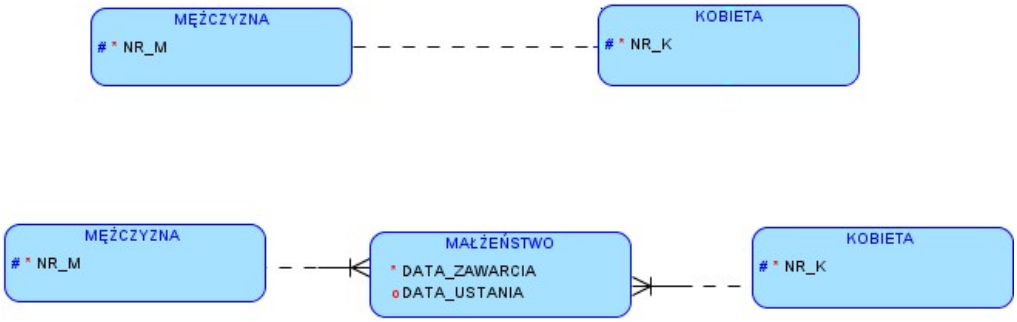
oddzielnie.

1.5.3. Modelowanie czasu

Uwzględnienie w modelu danych zmian, jakim podlegają one w czasie, jest ważnym problemem w projektowaniu. Wiele zdarzeń, informacji rejestrowanych w bazie jest związanych z określonym znacznikiem czasu. Na przykład, jeśli w projekcie mamy encję Pracownik z listą atrybutów reprezentujących dane osobowe, a wśród nich np. pensja, to ten atrybut przechowuje bieżącą wartość pensji pracownika, aktualną od czasu ostatniej podwyżki. Takie rozwiązanie nie jest wystarczające, jeśli w bazie miałaby być informacja o wszystkich wcześniejszych wynagrodzeniach pracownika. W celu zapamiętania historii zmian jakiegoś atrybutu wprowadzamy nową encję zależną, np. Płace, a w niej nowy atrybut Od_kiedy, reprezentujący czas. Atrybut ten włączamy do złożonego klucza

głównego.

Podobnie wygląda modelowanie związków, które zmieniają się w czasie. Jeśli w bazie np. chcemy zapisać informację o małżeństwach jako związkach binarnych pomiędzy encjami MĘŻCZYZNA i KOBIETA, to uwzględnimy związek typu 1:1 obustronnie opcjonalny. Przy takiej organizacji danych będzie możliwe zapamiętanie tylko informacji o aktualnie trwających związkach. Jeśli mielibyśmy zapamiętać jednak wszystkie przeszłe i przyszłe zdarzenia dotyczące zawieranych małżeństw, to musielibyśmy uwzględnić związek N:M pomiędzy encjami MĘŻCZYZNA i KOBIETA, a w encji asocjacyjnej zależnej o nazwie MAŁŻEŃSTWO uwzględnić atrybuty związane z czasem. Dodanie własnego identyfikatora dla encji MAŁŻEŃSTWO pozwoli rejestrować również związki małżeńskie zawierane ponownie przez te same osoby. Przedstawia to rysunek 1.11.



Rysunek 1.11. *Modelowanie czasu w związkach*

1.5.4. Elementy obiektowości w bazach relacyjnych — hierarchia encji

W bazach relacyjnych nie ma dziedziczenia atrybutów. Problem ten można obejść, definiując specjalny związek zwany hierarchią encji. Jeśli w modelowanej rzeczywistości mamy encje posiadające wspólne atrybuty, różniące się tylko częścią swoich cech, można w takim wypadku zdefiniować jedną encję, zwaną nadencją

(encją generalizacji) i zawierającą wspólne atrybuty, oraz pozostałe encje z własnymi atrybutami, zwane podencjami (encjami specjalizacji).

Jako przykład hierarchii encji rozważmy model dotyczący studentów. Na uczelni studenci uczęszczają na zajęcia na studiach dziennych i zaocznych. Wszyscy studenci posiadają wspólny zbiór atrybutów, np. Nr_albumu, PESEL, Nazwisko, Imię, Adres itd. Studenci studiów dziennych mają specyficzne atrybuty, np. stypendium, natomiast studenci studiów zaocznych: wysokość_czesnego.

W tym modelu mielibyśmy encję generalizacji (nadencję) STUDENT i dwie encje specjalizacji (podencje) DZIENNY i ZAOCZNY.

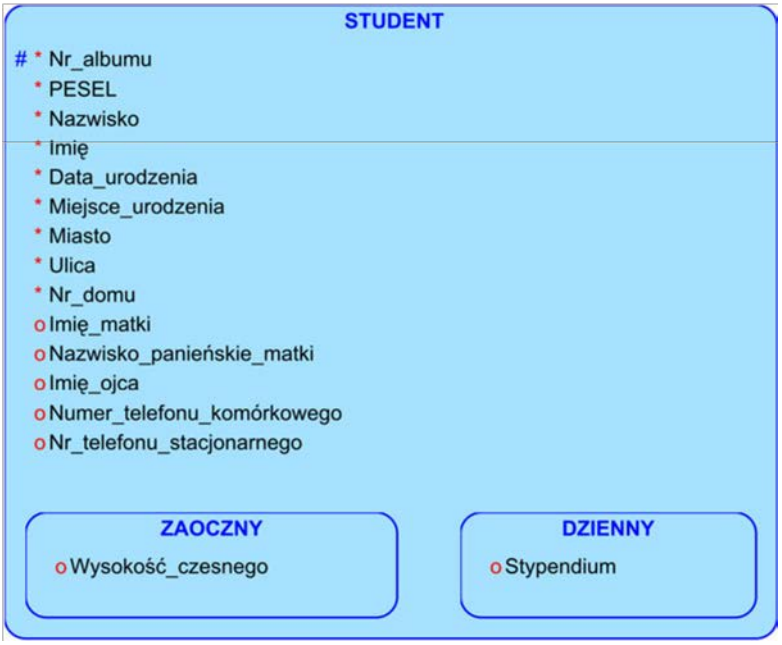
Podencje nie posiadają swoich identyfikatorów. Każde wystąpienie nadencji jest zawsze wystąpieniem jednej podencji (student jest albo studentem dziennym albo zaocznym) oraz każde wystąpienie podencji jest wystąpieniem nadencji (student dzienny jest studentem, student zaoczny jest studentem). Przykład hierarchii encji przedstawia rysunek 1.12.

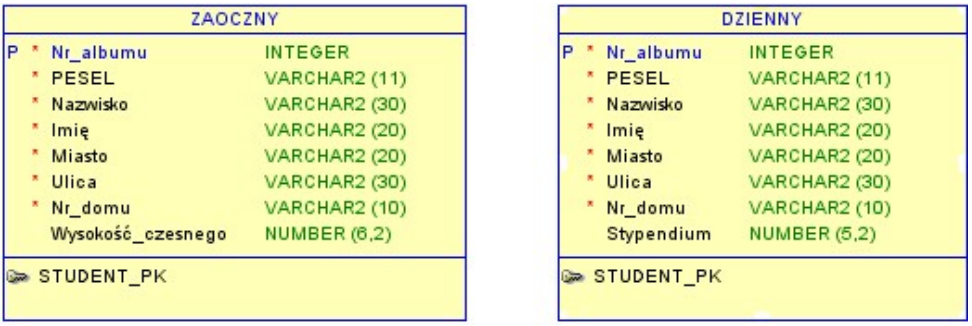
Hierarchia encji będzie implementowana w różny sposób:

- może być utworzona tablica dla każdej podencji, wówczas podencje odziedziczą wszystkie atrybuty swojej nadencji,
- może być utworzona tablica tylko dla nadencji, zawierająca wszystkie atrybuty własne i podencji. Jeśli atrybuty specyficzne dla danej podencji przyjmą określone wartości, to atrybuty pozostałych podencji będą miały wartości puste (NULL).

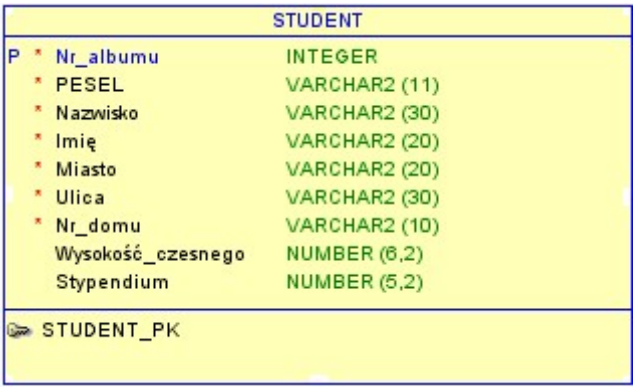
Przedstawiają to rysunki 1.13 i 1.14.

Rysunek 1.12. *Hierarchia encji, projekt logiczny*





Rysunek 1.13. *Hierarchia encji — projekt fizyczny. Tablice dla każdej podencji*



Rysunek 1.14. *Hierarchia encji — projekt fizyczny. Jedna tablica dla nadencji*

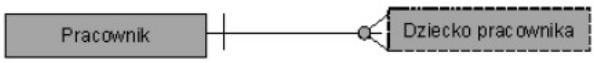
1.5.5. Alternatywne notacje stosowane w modelowaniu danych

Diagram związków encji można przedstawić w wielu różnych notacjach graficznych, różniących się zestawem znaków i symboli graficznych. Najpopularniejszymi są: notacja Chena i notacja Barkera (stosowana w produktach Oracle).

W zamieszczonych powyżej przykładach jest stosowana notacja Barkera. Został również podany dokładny opis ułatwiający interpretację wszystkich oznaczeń. Znajomość tej notacji ułatwi przeanalizowanie rysunku 1.15, przedstawiającego jeden związek zapisany w trzech różnych notacjach. Jest to związek typu 1:N pomiędzy encjami PRACOWNIK i DZIECKO_PRACOWNIKA. Wśród pracowników mogą być osoby bezdzietne, więc związek będzie opcjonalny od strony pracownika, zaś od strony dziecka będzie to związek obligatoryjny, bo informacje o dzieciach pracowników mogą pozostać w bazie tylko dopóty, dopóki rodzic jest zatrudniony.

- **Notacja CASE*Method (Oracle)**

- **Notacja IEM (Information Engineering Method) (Sybase)**



- **Standard International DEFinition (IDEF1X)**

- **Standard ER (*od ang.Entity-Relationship*) notacja Chena**

Rysunek 1.15. *Przykłady różnych notacji w modelowaniu*

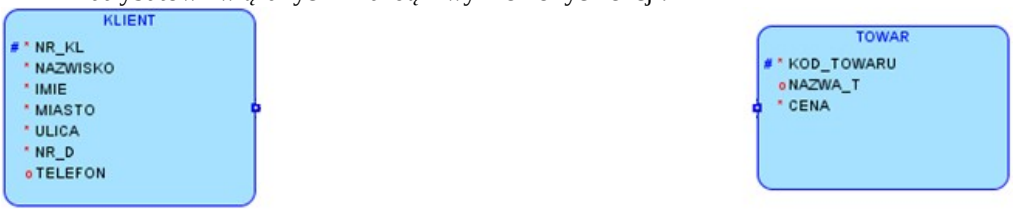
1.6. Przykład projektowania bazy dla sklepu metodą modelowania danych

W celu przybliżenia metody modelowania danych zostanie przedstawiony prosty przykład projektowania bazy dla sklepu. Niestandardowo zostanie poprowadzony pierwszy etap, który zobowiązuje projektującego do szerokiej analizy problemu, wywiadu z przyszłym użytkownikiem projektowanego systemu oraz pogłębienia wiedzy z danego zakresu. Ponieważ wskazany wycinek rzeczywistości, dla którego będziemy modelować bazę, wydaje się powszechnie znany i właściwy dobór encji zdaje się być problemem banalnym, nacisk zostanie położony na etap dopracowania diagramu związków encji — mianowicie na konwersję związków wieloznacznych do postaci związków prostych.

Definiując projekt logiczny, należy kolejno wykonać opisane niżej zadania.

1.6.1. Wybór potrzebnych encji i ich opis

Ponieważ w projektowanym systemie mamy rejestrować sprzedaż w sklepie, będziemy przechowywać informację o klientach i towarach, które są w ofercie. Stosując uogólnione i uproszczone podejście, uwzględnimy dwie encje KLIENT i TOWAR (rysunek 1.16). Uwzględniając informacje, jakie mamy rejestrować w systemie, ustalamy listę atrybutów związanych z każdą z wymienionych encji.



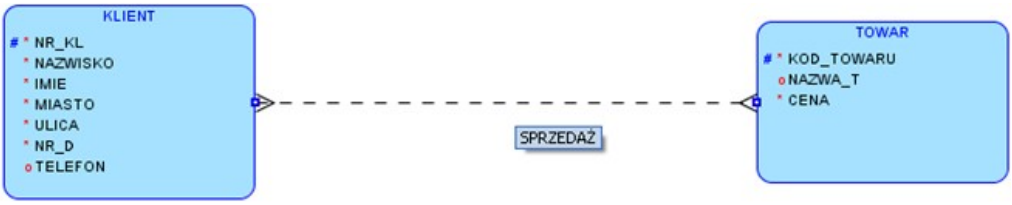
Rysunek 1.16. *Wybór encji i definiowanie opisu encji KLIENT i TOWAR*

1.6.2. Identyfikowanie związków między encjami

Informacja o dokonywanych zakupach jest związana ze związkiem typu N:M pomiędzy encjami KLIENT i TOWAR (rysunek 1.17). Klient może kupować wiele towarów, a towar może być kupowany przez wielu klientów. Przyjmijmy uczestnictwo opcjonalne po obu stronach tego związku, tzn. mogą być w bazie towary, które nie były kupowane (niekoniecznie towary niechodliwe), oraz mogą być klienci, którzy po zarejestrowaniu w bazie nie dokonywali zakupów.

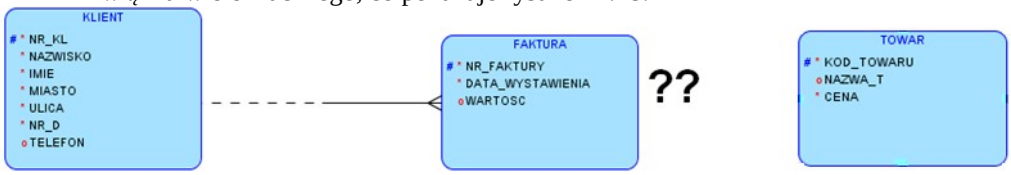
1.6.3. Konwersja związków wieloznacznych do postaci związków prostych

Usunięcie związku wieloznacznego pomiędzy encjami KLIENT-TOWAR w sposób uproszczony — poprzez wprowadzenie encji asocjacyjnej słabej — jest nie do zaakceptowania, ponieważ wprowadziłoby to niedopuszczalne ograniczenia. W tym związku nie



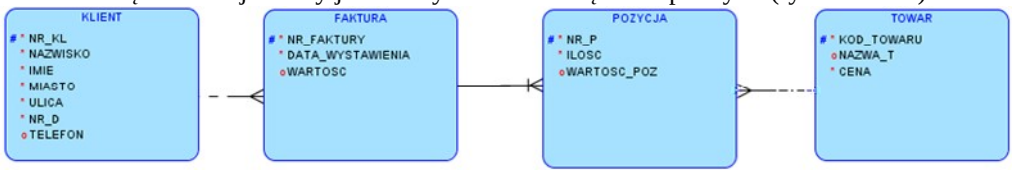
Rysunek 1.17. *Definiowanie związków. Związek Sprzedaż typu N:M pomiędzy encjami Klient i Towar*

chodzi przecież tylko o informację, czy klient kupował określony towar, czy nie, ale o wiele szczegółów transakcji dotyczących zakupów, które muszą być zapisywane w bazie. Ten związek musi być uproszczony poprzez dodatkowe encje. Wprowadzamy więc encję FAKTURA (lub RACHUNEK, ZAMÓWIENIE czy PARAGON), czyli encję, która będzie dotyczyła jednych określonych zakupów. Uwzględnienie tej encji nie usunie związku wieloznacznego, co pokazuje rysunek 1.18.



Rysunek 1.18. *Dopracowywanie związków typu N:M poprzez dodatkową encję — FAKTURA*

Jest tu konieczne dalsze uszczegółowienie związku i wprowadzenie encji POZYCJA_FAKTURY, zawierającej szczegóły pojedynczych zakupów. W diagramie związków encji mamy już do czynienia ze związkami prostymi (rysunek 1.19).

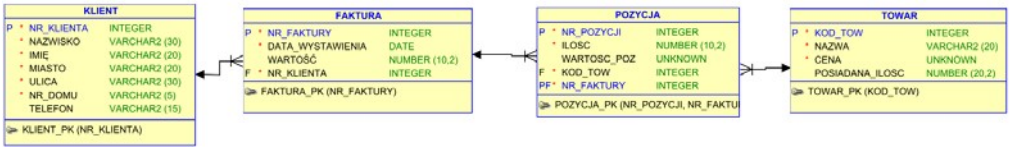


Rysunek 1.19. *Diagram związków encji w definiowanym projekcie*

1.6.4. Transformacja diagramu związków encji do modelu relacyjnego

Transformację przeprowadzamy zgodnie z omówionymi powyżej zasadami.

Projekt fizyczny wygenerowany za pomocą Oracle DataModeler przedstawia rysunek 1.20.



Rysunek 1.20. *Projekt fizyczny bazy dla sklepu*

1.6.5. Implementacja bazy danych

Następnym etapem jest implementacja bazy danych w wybranym Systemie Zarządzania Bazą Danych. Przykładowy skrypt SQL z poleceniami tworzącymi tabele przez narzędzie Oracle DataModeler jest zamieszczony poniżej:

```
-- Generated by Oracle SQL Developer Data Modeler Version: 2.0.0 Build: 584
-- type:      Oracle Database 11g CREATE TABLE FAKTURA
(NR_FAKTURY INTEGER  NOT NULL ,
DATA_WYSTAWIENIA DATE ,
WARTOSC NUMBER (10,2) ,
NR_KL INTEGER  NOT NULL) ;

ALTER TABLE FAKTURA
ADD CONSTRAINT FAKTURA_PK PRIMARY KEY (NR_FAKTURY);

CREATE TABLE KLIENT
(NR_KL INTEGER  NOT NULL ,
NAZWISKO VARCHAR2 (25) ,
IMIĘ VARCHAR2 (20)  NOT NULL ,
MIASTO VARCHAR2 (20)  NOT NULL ,
ULICA VARCHAR2 (30)  NOT NULL ,
NR_DOMU VARCHAR2 (10) ,
TELEFON VARCHAR2 (15) ) ;

ALTER TABLE KLIENT
ADD CONSTRAINT KLIENT_PK PRIMARY KEY (NR_KL) ;

CREATE TABLE POZYCJA
(NR_POZYCJI INTEGER  NOT NULL ,
ILOSC NUMBER (10,2) ,
WARTOSC_POZ NUMBER (10,2) ,
KOD_TOW INTEGER  NOT NULL ,
NR_FAKTURY INTEGER  NOT NULL , NR_KLIENTA
INTEGER  NOT NULL) ;

ALTER TABLE POZYCJA
ADD CONSTRAINT POZYCJA_PK PRIMARY KEY (NR_POZYCJI, NR_FAKTURY);

CREATE TABLE TOWAR
(KOD_TOWARU INTEGER  NOT NULL ,
NAZWA_T VARCHAR2 (20) ,
CENA NUMBER (10,2) ,
POSIADANA_ILOSC NUMBER (10,2));

ALTER TABLE TOWAR
ADD CONSTRAINT TOWAR_PK PRIMARY KEY ( KOD_TOWARU ) ;
```

```
ALTER TABLE POZYCJA
ADD CONSTRAINT FK_A FOREIGN KEY (NR_FAKTURY)
REFERENCES FAKTURA (NR_FAKTURY);

ALTER TABLE FAKTURA
ADD CONSTRAINT Relation_5 FOREIGN KEY (NR_KLIENTA)
REFERENCES KLIENT (NR_KLIENTA);

ALTER TABLE POZYCJA
ADD CONSTRAINT Relation_6 FOREIGN KEY (KOD_TOWARU)
REFERENCES TOWAR(KOD_TOWARU);
```

1.7. Zadania z modelowania logicznego

1. Określ, jakie własności musi posiadać obiekt, aby mógł być uwzględniony jako typ encji w projektowanym systemie.
2. Omów wady i zalety stosowania identyfikatorów naturalnych.
3. Jaka jest różnica w przedstawionych relacjach? Skomentuj i narysuj model pojęciowy dla obu wypadków (projekt logiczny). Dopisz pola, które mogą wystąpić w tabelach:
   R1(Id pokoju<PK,FK>, Id_osoby<PK,FK>) R2(Id pokoju<PK>, Id_osoby<FK>).
4. Podaj przykład diagramu związków encji, dla którego powstała tablica Osoba: OSOBA(Nr_osoby<PK>, Nazwisko, Imię, Nr_osoby_1<FK>, Nr_osoby_2<FK>, Nr_osoby_3<FK>). Podaj, jakie informacje będą rejestrowane w tej tablicy? Jakie więzy integralnościowe są przypisane niejawnie polom kluczowym?

   Zdefiniuj projekt logiczny i fizyczny bazy oraz napisz skrypt tworzący tablice w bazie, uwzględniając poniższe informacje:
5. Firma ma kilka oddziałów, w każdym z nich jest zatrudniony szereg pracowników. Pracownicy łączeni są w kilkuosobowe zespoły. Pracownik w danej chwili może być przydzielony tylko do jednego zespołu. Zespół może mieć przydzielonych kilka zadań. Jedno zadanie realizuje jeden zespół. Zadanie jest zatwierdzane przez pracownika będącego kierownikiem.
6. W domu akademickim student może być zakwaterowany po wcześniejszym przyznaniu mu miejsca. Prawo do zakwaterowania jest przydzielane na określony rok akademicki i określony czas (np. z wyłączeniem wakacji). Pokoje w akademikach są wieloosobowe, i są w nich meldowani studenci z różnych wydziałów. Należność za użytkowanie miejsca noclegowego jest wpłacana na bieżąco w kolejnych miesiącach i rejestrowana w systemie. Dodatkowo są naliczane ewentualne zaległości w opłatach każdego studenta, które mogą skutkować utratą prawa zamieszkiwania w akademiku.
7. Zaprojektuj bazę danych z jednym zbiorem encji Osoba, w której będą zapisywane informacje na temat członków rodziny z uwzględnieniem związków pokrewieństwa: macierzyństwa i ojcostwa. Wśród danych osobowych powinny znaleźć się m.in. imię, nazwisko, data_urodzenia, data_śmierci.

8. Firma organizuje szkolenia, które mogą kończyć się uzyskaniem certyfikatu. Każde szkolenie jest prowadzone przez jednego wykładowcę wybieranego spośród potencjalnych współpracowników. Wykładowca może prowadzić wiele szkoleń. Słuchaczami są studenci. Student może uczestniczyć w wielu szkoleniach, musi też zadeklarować swój udział w szkoleniu, aby mógł być zarejestrowany w bazie. Niektóre szkolenia są zhierarchizowane (np. aby rozpocząć kurs angielskiego drugiego stopnia, należy ukończyć wcześniej kurs angielskiego pierwszego stopnia).
9. W wypożyczalni filmów każdy nośnik ma indywidualnie nadany numer seryjny. Osoba musi być zarejestrowana jako klient wypożyczalni i posiadać unikalny numer. Są dwa typy członkostwa: członkostwo zwykłe (personalne) i klubowe.

   Członkostwo klubowe uzyskuje się przy wypożyczeniu trzydziestego filmu lub przekroczeniu kwoty 200 zł wynikającej z kosztów wypożyczeń. Członkostwo takie uprawnia do 15% zniżki. Cena wypożyczenia wynosi 5 zł za DVD oraz 10 zł za Blu-ray.
10. W bazie o nazwie Książka kucharska mają znaleźć się dokładne opisy potraw i składników oraz przepisy, w jaki sposób przygotować potrawę. Należy uwzględnić kategorie potraw oraz kaloryczność dań, która powinna być wyliczana na podstawie kalorii przypisanych jednostce wagi każdego składnika.
11. Zespół muzyczny wydaje wiele płyt. Płyta zawiera wiele utworów, a jeden utwór może pojawić się na wielu płytach. Każdy utwór może mieć wielu autorów słów oraz wielu autorów muzyki, ale zawsze co najmniej jednego.
12. Baza danych dla strony WWW, na której znajduje się galeria zdjęć. Użytkownicy, zakładając konto, podają login, hasło i e-mail. Użytkownik może dodawać do galerii swoje zdjęcia oraz wystawiać komentarze dla zdjęć umieszczonych na stronie. Zdjęcia są podzielone na kategorie.
13. Zaprojektuj bazę dla zmiennych w czasie danych dotyczących właścicieli nieruchomości. Uwzględnij, że związek własności może przestać istnieć również z powodu śmierci właściciela lub zniszczenia nieruchomości.
14. Zaprojektuj bazę, przechowującą dane pracowników (dane osobowe) wraz ze stanowiskami, jakie zajmowali i zajmują obecnie w firmie (z uwzględnieniem parametru: od kiedy). Ponadto pracodawca ma mieć wgląd w historię wszystkich wypłat pracowników uwzględniającą liczbę dni nieobecności, liczbę nadgodzin, premie i potrącenia w każdym miesiącu.

Rozdział 2. Normalizacja danych

Normalizacja jest tradycyjną metodą tworzenia projektu logicznego bazy. Jest metodą projektowania wstępującego (ang. *bottom-up design*). Jest realizowana jako proces weryfikowania poprawności projektowanych struktur tablic i przeprowadzania ewentualnej dekompozycji na mniejsze schematy relacji o pożądanych cechach. Ma za zadanie usunięcie nadmiarowości (powtarzania danych) i różnego rodzaju anomalii (anomalie wprowadzania, usuwania i modyfikacji danych).

Przykładem niestarannie zaprojektowanej tablicy, obciążonej wieloma wadami, może być relacja FILMOTEKA zawierająca informacje o filmach polskich reżyserów, przedstawiona na rysunku 2.1.

Rysunek 2.1. *Błędnie zaprojektowana tablica*

NR_REŻ	NAZWISKO_REZ	IMIE_REZ	TYTUŁ	ROK
10	WAJDA	ANDRZEJ	TATARAK	2009
10	WAJDA	ANDRZEJ	KATYŃ	2007
10	WAJDA	ANDRZEJ ...	PAN TADEUSZ	1999
15	ZANUSSI	KRZYSZTOF	SOLIDARNOSC...	2005
15	ZANUSSI	KRZYSZTOF	SUPLEMENT	2002
15	ZANUSSI	KRZYSZTOF	ILUMINACJA	1972

Ta relacja charakteryzuje się następującymi cechami:

1. Występuje w niej *nadmiarowość* — informacje są powielane w kolejnych krotkach. Opis reżysera jest powtarzany tyle razy, ile filmów danego reżysera jest wpisanych do bazy. Redundancja jest niekorzystna nie tylko ze względu na zwiększoną zajętość pamięci, ale przede wszystkim ze względu na problem niespójności, który może wystąpić np. podczas zmiany danych, jeśli modyfikacja nie byłaby powtórzona dokładnie w każdej krotce, która zawiera modyfikowaną wartość. Mielibyśmy wówczas do czynienia z *anomalią modyfikacji danych.*

2. Występuje *anomalia wprowadzania danych* polegająca na tym, że wpisanie do tablicy nowej informacji wymusza zapisanie również innych informacji powiązanych. W tej tabeli jest możliwa rejestracja nowego reżysera dopiero wtedy, gdy wprowadzamy również opis jego filmu.
3. Występuje *anomalia usuwania danych* polegająca na tym, że jeśli w trakcie usuwania zbiór określonych wartości stanie się pusty, wówczas możemy utracić inne informacje z bazy. W powyższej tabeli usuwając informacje o danym reżyserze, tracimy również informacje o wszystkich jego filmach.

Łatwo zauważyć, że przedstawiona tablica FILMOTEKA posiada niepożądane właściwości, ponieważ zawiera grupy różnych powiązanych ze sobą informacji, dotyczących filmów i reżyserów. Rozdzielenie tych grup atrybutów do oddzielnych struktur pozwoli uporać się z omówionymi problemami.

Eliminujemy anomalie występujące w tablicy poprzez podział jej na dwie lub więcej nowych relacji. Operacja ta, nazywana dekompozycją struktury tablicy, musi być przeprowadzona z zachowaniem następujących zasad:

- *Zasada zachowania atrybutów* — w procesie podziału muszą być uwzględnione wszystkie atrybuty, żaden z nich nie może być pominięty podczas tworzenia nowych tablic.
- *Zasada zachowania informacji* — krotki w nowych relacjach, otrzymywane poprzez operację rzutowania analizowanej relacji na wybrany zbiór atrybutów, muszą zachować wszystkie informacje przechowywane w kolumnach tablicy dekomponowanej.
- *Zasada zachowania zależności funkcyjnych* — w strukturach nowych tablic muszą być zachowane wszystkie związki między danymi, określone w tablicy dekomponowanej.

Rzutowanie na zestaw atrybutów polega na zawężeniu zbioru wartości krotki do tych, które odpowiadają atrybutom, na jakie krotka jest rzutowana. W wyniku tej operacji mogą powstać krotki, które nie różnią się wartością żadnego atrybutu, ale krotki takie traktowane są jako tożsame. W tablicach zdekomponowanych grupy wierszy zawierające te same wartości są reprezentowane przez pojedynczy wiersz. Żadna tablica w bazie nie może mieć powtarzających się całych wierszy — dla zapewnienia unikalności wierszy jest definiowany klucz główny. W przedstawionej powyżej tablicy Filmoteka w wyniku rzutowania na atrybut np. Nazwisko_rez otrzymujemy dwie wartości — Wajda i Zanussi.

W wyniku dekompozycji tablicy FILMOTEKA otrzymamy przedstawione poniżej dwie tablice REŻYSERZY i FILMY. W nowych relacjach są uwzględnione wszystkie atrybuty z tablicy Filmoteka (zgodnie z pierwszą zasadą — zachowania atrybutów). W tablicy Reżyserzy, otrzymanej w wyniku rzutowania tablicy Filmoteka na listę atrybutów: Nr_rez, Nazwisko_rez, Imię_rez, występują niepowtarzające się dane o wszystkich twórcach filmowych (zgodnie z drugą zasadą — zachowania informacji). Z analizy danych przechowywanych w tablicy Filmoteka wynika, że każdy film jest jednoznacznie przypisany konkretnemu reżyserowi. Te związki są uwzględnione w nowych tablicach (zgodnie z obowiązującą zasadą zachowania zależności funkcyjnych) poprzez dołączenie atrybutu Nr_rez do opisu filmów w strukturze tablicy Filmy. Atrybut Nr_rez w tablicy Filmy pełni rolę klucza obcego, czerpie wartości z kolumny klucza głównego (Nr_rez) w tablicy Reżyserzy i pozwala na złączenie wierszy zdekomponowanych tablic. Poprawnie przeprowadzona dekompozycja pozwala na dokładne odtworzenie dzielonej tablicy razem z jej zawartością poprzez złączenie tablic, otrzymanych w wyniku dekompozycji. Mówimy wówczas, że proces dekompozycji jest odwracalny. Zagadnienie weryfikowania odwracalności dekompozycji jest omówione w dalszej części opracowania.

Zastąpienie tablicy Filmoteka nowymi tablicami Reżyserzy i Filmy (rysunki 2.2 i 2.3) pozwala wyeliminować redundancję — informacje o konkretnym reżyserze są pamiętane tylko w jednym wierszu tabeli, niezależnie od liczby wyprodukowanych filmów.

Rysunek 2.2.
Zdekomponowana tablica Reżyserzy

NR_REŻ	NAZWISKO_REZ	IMIE_REZ
10	WAJDA	ANDRZEJ
15	ZANUSSI	KRZYSZTOF

Rysunek 2.3.
Zdekomponowana tablica Filmy

TYTUŁ	ROK	NR_REZ
TATARAK	2009	10
KATYŃ	2007	10
PAN TADEUSZ	1999	10
SOLIDARNOSC...	2005	15
SUPLEMENT	2002	15
ILUMINACJA	1972	15

W związku z tym nie ma problemu z modyfikacją danych o twórcach. Nie występują również anomalie wprowadzania i usuwania danych — można zarejestrować reżysera, którego filmów jeszcze nie posiadamy, jak również usunąć wszystkie filmy konkretnego reżysera, nie powodując utraty danych o twórcy.

Przedstawiony powyżej przykład dekompozycji tablicy Filmoteka na dwie tablice Filmy i Reżyserzy pozwala zrozumieć projektowanie poprawnych tablic w bazie metodą normalizacji danych. Normalizacja bowiem jest procesem wielokrotnej dekompozycji projektowanych tablic, realizowanym w etapach związanych ze ściśle określonymi wymogami, zdefiniowanymi jako postacie normalne. Choć doświadczenie i intuicja są niezwykle przydatne w tej dziedzinie, należy dokładnie poznać zasady stosowane podczas normalizacji tablic, aby uniknąć błędów związanych z nieodwracalnością przeprowadzanych dekompozycji. Ocena poprawności dekompozycji może polegać na przeprowadzeniu testu zwanego w literaturze testem na złączenie bezstratne. Jest on oparty na algorytmie chase opisanym w literaturze przedmiotu np. [Garcia-Molina, Ullman, 2011, str. 111]. Test ten jest omówiony w dalszej części opracowania.

Gdy przystępujemy do projektowania tablic metodą normalizacji danych, na początku wszystkie atrybuty, które mają znaleźć się w bazie, traktujemy jako pola jednej tablicy (w literaturze zwanej relacją uniwersalną [14]). Następnie, oceniamy, czy w tej tablicy są spełnione określone własności, związane z konkretnym etapem projektowania. Jeśli analizowana tablica nie spełnia określonych wymagań, poddajemy ją dekompozycji w taki sposób, aby w otrzymanych tablicach wynikowych występowały oczekiwane własności. Każdy etap analizy własności tablicy i ewentualnego podziału jej struktury na mniejsze jest związany z określoną postacią normalną. Proces normalizacji danych wymusza na projektującym konsekwentne przeanalizowanie własności wszystkich postaci normalnych — nawet wówczas, gdy na pewnym etapie projektu przejście do wyższej postaci normalnej nie wymaga wprowadzenia zmian w strukturach projektowanych tablic.

Początkowo zdefiniowano trzy postacie normalne — publikacji ich dokonał w 1972 r. E.F. Codd (nazywany ojcem relacyjnych baz danych). Następnie w roku 1974

wprowadzono definicję silniejszej, trzeciej postaci normalnej, zwanej postacią Boyce'a-Codda. Kilka lat później została zdefiniowana przez R. Fagina czwarta i wkrótce piąta postać normalna. Obecnie funkcjonuje już szósta postać normalna, która jest określana dla tych baz, w których dane są mocno uwarunkowane w czasie, czyli tzw. „temporalnych baz danych".

Wszystkie wymienione postacie normalne są związane z kluczami głównymi relacji i określonymi zależnościami funkcyjnymi pomiędzy atrybutami. Pamiętajmy, każdy atrybut w tablicy ma mieć prosty typ. Przeanalizujmy własności, które są oceniane w kolejnych etapach projektu.

2.1. Zależność funkcyjna i pierwsza postać normalna

Zależność funkcyjna (ang. *functional dependence*), oznaczona jako: A→B, jednoznacznie przyporządkowuje do każdej wartości atrybutu A jedną określoną wartość atrybutu B.

Uwzględniając tablicę z danymi osobowymi, można wskazać zależność funkcyjną pomiędzy dwoma atrybutami, np. PESEL→Nazwisko. Oznacza to, że każda wartość numeru PESEL determinuje określoną wartość atrybutu Nazwisko. Ta zależność jest jednokierunkowa: jeśli A→B, to znaczy, że nie występuje zależność B→A, nawet jeśli na podstawie zbioru wartości w tablicy można taką zależność wyróżnić — np. jeśli atrybut Nazwisko ma unikalne wartości we wszystkich wierszach tablicy i jest jednoznacznie przyporządkowany do wartości atrybutu PESEL. Pamiętajmy, że wszystkie zależności funkcyjne są określone dla schematu relacji, a nie określonych instancji, nie mogą więc być rozpatrywane na podstawie wartości w wierszach.

Wprowadźmy jeszcze kilka pojęć, które umożliwią sprecyzowanie cech zależności funkcyjnej i będą konieczne w dalszym procesie normalizacji.

Zależność funkcyjna może być określona dla zbiorów atrybutów A i B, jest wówczas opisana następującym zdaniem: „Jeśli dwie krotki relacji R są zgodne dla atrybutów $A_1A_2...A_n$ (tzn. obie krotki mają takie same wartości składowych dla wymienionych atrybutów), to muszą być również zgodne w pewnym innym atrybucie B" [9].

Jeśli zbiór atrybutów $A_1,A_2,...A_n$ określa więcej niż jeden atrybut, np. $A_1,A_2,...A_n \rightarrow B_1$ i $A_1,A_2,...A_n \rightarrow B_2$ oraz $A_1,A_2,...A_n \rightarrow B_3,...A_1A_2...A_n \rightarrow B_m$, to zapisujemy to jako zależność $A_1,A_2,...A_n \rightarrow B_1,B_2,...B_m$.

Uwzględnijmy uczelnianą bazę danych, w której m.in. funkcjonuje tablica Starostowie o podanej strukturze (Wydział, Typ_studiów, Rok_studiów, Kierunek, Numer_starosty_roku). W tablicy jest określona zależność funkcyjna: Wydział, Typ_studiów, Rok_studiów, Kierunek → Numer_starosty_roku. Jeśli w kilku wierszach wystąpiłyby te same wartości atrybutów Wydział, Typ_studiów, Rok_studiów, Kierunek, to w każdym z tych wierszy musi wystąpić ten sam numer starosty roku.

Występują trzy *typy zależności funkcyjnych* postaci $A_1,A_2,...A_n \rightarrow B_1,B_2,...B_m$:

- *trywialne* — jeśli zbiór atrybutów $B_1,B_2,...B_m$ jest podzbiorem zbioru atrybutów $A_1,A_2,...A_n$;
- *nietrywialne* — jeśli co najmniej jeden atrybut B_i nie należy do zbioru atrybutów $A_1, A_2,...A_n$;
- *całkowicie nietrywialne* — jeśli żaden z atrybutów B_i nie należy do zbioru atrybutów $A_1, A_2,...A_n$.

Zbiór $A_1,A_2,...A_n$ (po lewej stronie zależności) nazywa się *zbiorem determinującym* lub *wyznacznikiem*, natomiast zbiór $B_1,B_2,...B_m$ jest *zbiorem zależnym*.

Znając wartość atrybutu lub zbioru atrybutów stanowiących wyznacznik zależności, ustalamy wartość atrybutów zależnych. Na przykład, na podstawie zależności funkcyjnej PESEL→Nazwisko, znając PESEL danej osoby, określimy Nazwisko, jakie nosi ta osoba. Taka zależność jest całkowicie nietrywialna, bo zbiory wyznacznika i atrybutów zależnych są rozłączne. W przedstawionej powyżej tablicy Starostowie na podstawie zależności trywialnej wyznaczamy wartość atrybutu Wydział lub Kierunek, natomiast na podstawie zależności całkowicie nietrywialnej ustalamy Numer_starosty_roku.

Jeden lub więcej atrybutów należących do schematu relacji jest *kluczem głównym* (ang. *primary key*) relacji R, jeśli wszystkie pozostałe atrybuty relacji są funkcyjnie zależne od tych atrybutów.

Klucz główny musi mieć unikalne wartości w tablicy. Klucz musi również spełniać wymóg minimalnej długości, nie może więc istnieć podzbiór właściwy zbioru klucza, od którego wszystkie pozostałe atrybuty relacji byłyby funkcjonalnie zależne. Zbiór atrybutów, który zawiera klucz, nazywa się *nadkluczem*. Atrybuty niewchodzące w skład klucza nazywają się *niekluczowymi*.

W tablicy Starostowie kluczem będzie zbiór atrybutów Wydział, Typ_studiów, Rok_studiów, Kierunek. Ponieważ pozostały w tablicy atrybut niekluczowy Numer_starosty_roku zależy funkcjonalnie od klucza, jest to zależność nietrywialna. Zbiór klucza ma minimalną długość, próba usunięcia jakiegokolwiek atrybutu z tego zbioru powoduje utratę jedynej zależności funkcyjnej spełnionej w tej tablicy — np. na podstawie atrybutu Wydział nie ustalimy Numeru_starosty_roku.

Wróćmy do wymagań, które musi spełniać tablica na pierwszym etapie normalizacji.

Pierwsza postać normalna (1NF — ang. first normal form)

Relacja jest w *pierwszej postaci normalnej* wtedy i tylko wtedy, gdy każdy atrybut niekluczowy jest funkcjonalnie zależny od klucza głównego.

W takiej relacji w każdej komórce (na przecięciu kolumny i wiersza) występuje tylko jedna wartość. Zależność funkcyjna wymusza jednoznaczne przyporządkowanie

wartości klucza (który może być określony na zbiorze atrybutów) do wartości każdego atrybutu niekluczowego.

W tablicy nieznormalizowanej mogą wystąpić atrybuty, które dla jednej określonej wartości klucza przyjmują wiele wartości. Przykład tablicy nieznormalizowanej z rzeczywistymi danymi zamieszczono w tabeli 2.1.

Tabela zawiera informacje o fakturach, klientach, towarach i ich sprzedaży. Wybierając atrybut Nr_faktury do roli klucza głównego tej tabeli, widzimy, że dla konkretnej wartości klucza część atrybutów (Data_wystawienia, Nr_klienta, Nazwa_klienta, Adres_kl, Wartość_całej_faktury) przyjmuje pojedyncze wartości, natomiast pozostała grupa atrybutów (Nr_pozycji, Kod_towaru, Nazwa_towaru, Cena_towaru, Ilość_sprzedana, Wartość_sprzedanego_tow) ma wiele wartości.

Tabela 2.1. *Nieznormalizowana tablica z danymi*

Nrfaktury	Datawystawienia	Nrklienta	Nazwaklienta	Adresklienta	Nrpozycji	Kodtowaru	Nazwatowaru	Cenatowaru	Ilość sprzedana	Wartość sprzedanegotowaru	Wartość całejfaktury
					1	ZB16	Zeszyt	1,25	2	2,50	
75			Szkoła	Będzin,	2	ZB96	Zeszyt	3,50	1	3,50	21,00
	14/01/05	1527	Podst.	ul. Woli		ODK1	Ołówek	0,80	5	4,00	
			Nr 10	12/14	3	DA13	Długopis	5,50	2	11,00	
					4						

soby usu

Normalizacja tablicy do pierwszej postaci normalnej. Istnieją
powtarzających się grup atrybutów w celu znormalizowania ta
postaci normalnej [14] — oba zostaną zaprezentowane dla tego pr

Pierwszy sposób (spłaszczenie tablicy) polega na zachowaniu niez.

tablicy, powieleniu informacji w komórkach, w których ich brakuje, i wyborze właściwego klucza głównego. Przedstawia to zamieszczona poniżej uzupełniona tabela 2.2.

Tabela 2.2. *Tabela z uzupełnionymi danymi*

Nrfaktury	Datawystawienia	Nrklienta	Nazwaklienta	Adresklienta	Nrpozycji	Kodtowaru	Nazwatowaru	Cenatowaru	Ilość sprzedana	Wartość sprzedanegotowaru	Wartość ca łejfaktury
75	14/01/05	1527	Szkoła Podst. Nr 10	Będzin, ul. Woli 12/14	1	ZB16	Zeszyt	1,25	2	2,50	21,00
75	14/01/05	1527	Szkoła Podst. Nr 10	Będzin, ul. Woli 12/14	2	ZB96	Zeszyt	3,50	1	3,50	21,00
75	14/01/05	1527	Szkoła Podst. Nr 10	Będzin, ul. Woli 12/14	3	ODK1	Ołówek	0,80	5	4,00	21,00
75	14/01/05	1527	Szkoła Podst. Nr 10	Będzin, ul. Woli 12/14	4	DA13	Długopis	5,50	2	11,00	21,00

Można zaproponować dla tej tabeli złożony klucz główny taki, aby spełniała ona wymagania pierwszej postaci normalnej, np. klucz Nr_faktury, Nr_pozycji.

Struktura tej tablicy jest następująca (atrybuty klucza są podkreślone):

- **Nr_Faktury**,
- **Nr_pozycji**,
- Data_wystawienia,
- Nr_klienta,
- Nazwa_klienta,
- Adres_kl,
- Kod_towaru,
- Nazwa_towaru,
- Cena_towaru,
- Ilość_sprzedana,
- Wartość_sprzedanego_tow,
- Wartość_całej_faktury

Drugi sposób (dekompozycja): polega na usunięciu do oddzielnej relacji tych atrybutów, z powodu których tablica nie spełniała wymagań pierwszej postaci normalnej. Wyłączona grupa atrybutów w oddzielnej relacji będzie miała złożony klucz główny (własny klucz grupy wraz z kluczem relacji macierzystej). W wyniku tej operacji otrzymujemy dwie tablice o podanej strukturze:

- **Nr_Faktury**,
- Data_wystawienia,
- Nr_klienta,
- Nazwa_klienta,
- Adres_kl,
- Wartość_całej_faktury

- **Nr_Faktury**,
- Nr_pozycji,
- Kod_towaru,
- Nazwa_towaru,
- Cena_towaru,
- Ilość_sprzedana,
- Wartość_sprzedanego_tow

Tablice uzyskane zarówno pierwszym, jak i drugim sposobem spełniają wymagania pierwszej postaci normalnej. Proszę zwrócić uwagę, że na tym poziomie normalizacji, zależnie od wybranej metody, mamy różne tablice. Nie ma to znaczenia dla ostatecznego projektu bazy, ponieważ na dalszym etapie normalizacji otrzymamy jednolity projekt. Ta sytuacja uświadamia nam jednak, że normalizacja musi być przeprowadzona konsekwentnie do końca.

2.2. Pełna zależność funkcyjna i druga postać normalna

Druga postać normalna (2NF — ang. second normal form)

Relacja jest w drugiej postaci normalnej, jeśli jest już w pierwszej postaci normalnej i każdy atrybut niekluczowy jest w pełni funkcjonalnie zależny od klucza głównego.

Pełna zależność funkcyjna A→B (ang. *full functional dependence*), jest oceniana w wypadku, gdy klucz A jest określony na zbiorze atrybutów $A_1A_2...A_n$, i polega ona na tym, że atrybut B jest funkcjonalnie zależny od całego zbioru atrybutów A, lecz nie jest on funkcjonalnie zależny od żadnego właściwego podzbioru A.

Przypomnijmy tablicę Starostowie o strukturze (Wydział, Typ_studiów, Rok_studiów, Kierunek, Numer_starosty_roku) — występuje w niej pełna zależność funkcyjna pomiędzy złożonym kluczem głównym, określonym na zbiorze atrybutów Wydział, Typ_studiów, Rok_studiów, Kierunek, a atrybutem niekluczowym Numer_starosty_roku, ponieważ atrybut niekluczowy nie zależy funkcjonalnie od żadnego podzbioru klucza.

Relacja, która jest w pierwszej postaci normalnej i posiada klucz prosty (określony na jednym atrybucie), automatycznie spełnia wymagania drugiej postaci normalnej.

Normalizacja tablicy z 1NF do 2NF polega na wykryciu i usunięciu częściowych zależności funkcyjnych. W wypadku wykrycia niepełnej zależności funkcjonalnej jest wymagane utworzenie dodatkowych relacji, do których zostaną przeniesione atrybuty niekluczowe wraz z tą częścią klucza głównego, od której w pełni funkcjonalnie zależą.

W omawianym powyżej przykładzie dotyczącym faktury obie tablice w 1NF uzyskane drogą dekompozycji spełniają wymagania 2NF.

Tablica uzyskana metodą spłaszczania posiada klucz złożony, występuje w niej także niepełna zależność funkcyjna, musi więc na tym etapie normalizacji podlegać dekompozycji. Część atrybutów (Data_wystawienia, Nr_klienta, Nazwa_klienta, Adres_kl, Wartość_całej_faktury) zależy funkcjonalnie od podzbioru klucza — od pojedynczego atrybutu Nr_faktury — i powinna być przeniesiona do oddzielnej tablicy z takim właśnie kluczem. Kolejne atrybuty (Kod_towaru, Nazwa_towaru, Cena_towaru, Ilość_sprzedana, Wartość_sprzedanego_tow) zależą od całego złożonego klucza i pozostają w tablicy z tym kluczem. Klucz w tym wypadku jest najkrótszy, bo próba ograniczenia zbioru klucza powoduje utratę zależności funkcyjnych.

W celu zdefiniowania drugiej postaci normalnej musimy zdekomponować tablicę, tworząc dwie relacje, w których będzie występowała pełna zależność funkcyjna.

Otrzymujemy takie same tablice jak w pierwszej postaci normalnej zdefiniowanej metodą dekompozycji. Na tym etapie normalizacji, niezależnie od wcześniejszych decyzji, uzyskujemy już jednolity projekt tablic.

- **Nr_Faktury**,
- Data_wystawienia,
- Nr_klienta,
- Nazwa_klienta,
- Adres_kl,
- Wartość_całej_faktury

- **Nr_Faktury**,
- **Nr_pozycji**,
- Kod_towaru,
- Nazwa_towaru,
- Cena_towaru,
- Ilość_sprzedana,
- Wartość_sprzedanego_tow

2.3. Zależność tranzytywna i trzecia postać normalna

Trzecia postać normalna (3NF — ang. third normal form)

Relacja jest w trzeciej postaci normalnej wtedy i tylko wtedy, gdy jest w drugiej postaci normalnej i każdy atrybut niekluczowy jest zależny bezpośrednio (a nie tranzytywnie) od klucza głównego.

Zależność tranzytywna (ang. *transitive functional dependence*) występuje wówczas, gdy pomiędzy atrybutami A,B,C w relacji R występują następujące zależności funkcyjne: A→B i B→C. Mówimy wtedy, że atrybut C jest tranzytywnie (przechodnio) zależny od atrybutu A.

Normalizacja z 2NF do 3NF wymaga usunięcia zależności przechodnich występujących w danej relacji.

Oceniając pod tym względem podane wyżej dwie tablice w drugiej postaci normalnej, można wyróżnić w nich zależności tranzytywne.

W pierwszej tablicy występujące zależności funkcyjne pomiędzy atrybutami niekluczowymi: Nr_klienta→Nazwa_klienta, Adres_kl wskazują, że atrybuty Nazwa_klienta, Adres_kl zależą tranzytywnie od klucza Nr_faktury (poprzez Nr_klienta).

Podobnie w drugiej tablicy — atrybuty Nazwa_towaru, Cena_towaru zależą tranzytywnie od klucza (poprzez atrybut Kod_towaru).

Atrybuty zależne tranzytywnie od klucza muszą być przeniesione do oddzielnych tablic wraz z kopią ich wyznacznika. Pamiętajmy, że atrybut, poprzez który zależność tranzytywna jest wyróżniona, pozostawiony w tablicy macierzystej pełni rolę klucza obcego.

W trzeciej postaci normalnej uzyskujemy więc następujące struktury:

Do trzeciej postaci normalnej są również stosowane dodatkowe wymagania związane z zależnościami funkcyjnymi spełnionymi w relacji, określające bardziej restrykcyjną postać zwaną postacią Boyce'a-Codda.

2.4. Postać normalna Boyce'a-Codda (BCNF)

Relacja R jest w postaci Boyce'a-Codda (BCNF) wtedy i tylko wtedy, gdy dla każdej nietrywialnej zależności funkcyjnej $A_1,A_2,...A_n \rightarrow B_1,B_2,...B_m$ spełnionej w R zbiór $\{A_1,A_2,...A_n\}$ jest nadkluczem R.

Aby stwierdzić, czy dana relacja jest w postaci BCNF, należy określić wszystkie zależności funkcyjne spełnione w tej relacji i sprawdzić, czy wyznaczniki są kluczami lub zawierają klucz.

W tablicach z analizowanego przykładu we wszystkich spełnionych zależnościach wyznacznikami są klucze relacji, dlatego każda z nich spełnia warunki postaci Boyce'a-Codda.

Przykładem tablicy, która jest w trzeciej postaci normalnej, ale nie spełnia wymagań postaci Boyce'a-Codda, jest tablica z danymi adresowymi o strukturze: Miasto, Ulica, Nr_domu, Kod_pocztowy. Kluczem głównym takiej tablicy jest zbiór atrybutów: Miasto, Ulica, Nr_domu. Są w niej spełnione dwie zależności funkcyjne: Miasto, Ulica, Nr_domu→ Kod_pocztowy oraz Kod_pocztowy→Miasto. Ze względu na drugą zależność funkcyjną, która nie wynika z klucza głównego, tabela ta nie spełnia wymagań postaci Boyce'a-Codda.

Omawiana trzecia postać normalna jest najczęściej postacią wynikową dla procesu projektowania, chyba że w relacji występuje nadmiarowość spowodowana zależnością wielowartościową. Tablice definiowane w czwartej i piątej postaci normalnej są omawiane w pierwszej części tego opracowania dotyczącej modelowania związków wieloznacznych i N-arnych.

2.5. Zależność wielowartościowa i czwarta postać normalna

Zależność wielowartościowa (ang. *multivalued dependency*) zwana jest niefunkcyjną, bo do każdej wartości wyznacznika przyporządkowuje zbiór wartości atrybutu zależnego. Taką zależność oznaczamy graficznie: A→→B.

Załóżmy, że w tabeli z informacjami o pracownikach mamy m.in. atrybuty: Znajomość_języków_obcych oraz Znajomość_języków_programowania. Jeśli osoba o numerze 100 zna dwa języki obce — angielski i francuski — oraz dwa języki programowania — C# i Java, to języki obce i języki programowania powtarzają się we wszystkich możliwych kombinacjach, wprowadzając redundancję.

Przykładowe krotki w takiej tablicy przedstawiono w tabeli 2.3.

Tabela 2.3. *Występowanie zależności wielowartościowych*

	Nr_osoby	Znajomość_języków_obcych	Znajomość_j_programowania
v	100	Angielski	C#
t	100	Angielski	Java
u	100	Francuski	C#
	100	Francuski	Java

W takiej tablicy żaden z atrybutów nie jest funkcyjnie zależny od pozostałych, nie występuje też żadna nietrywialna zależność funkcyjna, więc wszystkie atrybuty muszą wchodzić w skład klucza. Na przykładzie tej tablicy przeanalizujmy definicję omawianej

zależności.

Zależność wielowartościowa [9] jest spełniona w relacji R, jeśli dla każdej pary krotek *t* i *u* mających zgodne wartości atrybutu A, można znaleźć krotkę *v*, która ma zgodne:
1) A z krotkami *t* i *u*,

2) B z krotką *t*,

3) wszystkie atrybuty spoza A i B z krotką *u*.

Oznaczając w przykładowej tablicy kolejne atrybuty jako A-(Nr_osoby), B-(Znajomość_języków_obcych), C-(Znajomość_języków_programowania), krotkę drugą jako *t*, trzecią jako *u*, możemy wskazać pierwszą krotkę jako *v*, która będzie miała tą samą wartość atrybutu A (Nr_osoby = 100), wartość atrybutu B taką jak w krotce *t* (język angielski) oraz w atrybucie C wartości zgodne z krotką *u* (język programowania = C#). Dla ułatwienia, w pierwszej kolumnie tablicy z przykładowymi danymi zamieszczonej powyżej, wprowadzone zostały oznaczenia krotek *v*, *t* i *u*.

Czwarta postać normalna (4NF — ang. fourth normal form)

Relacja R jest w czwartej postaci normalnej wtedy i tylko wtedy, gdy jest w postaci BCNF i nie zawiera żadnych nietrywialnych zależności wielowartościowych, które to zależności muszą być wyłączone do oddzielnych schematów. Innymi słowy, czwarta postać stanowi uogólnienie warunku BCNF.

Dekompozycja do czwartej postaci normalnej polega na znalezieniu zależności wielowartościowej, która nie spełnia wymagania 4NF, np. $A_1,A_2,...A_n \rightarrow\rightarrow B_1,B_2,...B_m$, gdzie zbiór $A_1,A_2,...A_n$ nie jest nadkluczem, wówczas schemat relacji R dzielimy na dwa schematy:

- pierwszy schemat zawiera wszystkie atrybuty typu A i typu B,
- drugi schemat zawiera wszystkie atrybuty typu A oraz te wszystkie atrybuty z R, które nie są atrybutami ani typu A, ani typu B.

Tablica z przykładu zamieszczonego powyżej musi być zdekomponowana tak, aby w nowych schematach występowały co najwyżej pojedyncze zależności wielowartościowe:

Nr_osoby →→ Znajomość_języków_obcych,

Nr_osoby →→ Znajomość_języków_programowania.

W obu tablicach zależności wielowartościowe są trywialne — jedynymi kluczami są wszystkie atrybuty relacji.

W żadnym z tych schematów nie ma nietrywialnych zależności wielowartościowych (ani funkcyjnych), zatem spełniają one warunki 4NF.

2.6. Zależność połączeniowa i piąta postać normalna

Piąta postać normalna (5NF — ang. fifth normal form)

Relacja jest w piątej postaci normalnej, jeśli jest w czwartej postaci normalnej i nie istnieje jej rozkład odwracalny na zbiór mniejszych tabel, czyli nie zawiera zależności połączeniowych.

Rozkład odwracalny, inaczej określany złączeniem bezstratnym, jest omówiony w dalszej części opracowania. Oznacza taką dekompozycję tablicy na mniejsze, że przy złączeniu naturalnym otrzymanych tablic uzyskamy niezmienioną relację pierwotną. W procesie łączenia nie pojawią się żadne nieautentyczne wiersze dekomponowanej tablicy ani też żadna krotka nie zostanie utracona.

Zależność połączeniowa (ang. *join dependency*) jest spełniona w relacji R, zawierającej wielokrotne zależności wielowartościowe, jeśli prawidłowy stan relacji R (zawartość)

jest równy złączeniu relacji R_1, R_2, R_N uzyskanych poprzez rzutowanie na wybrane podzbiory atrybutów ze schematu relacji R.

Dekompozycja w 5NF dotyczy tablic (np. o strukturze A,B,C) zawierających wielokrotne, niepodzielne dotąd zależności wielowartościowe (np. A→→B, A→→C, C→→B). Takich tablic nie można bezstratnie podzielić na dwie relacje, ale można je rozłożyć na trzy (dla struktury A,B,C) lub więcej relacji.

Przykład 2.1.

Załóżmy, że mamy rozliczać koszty poszczególnych inwestycji, uwzględniając zużywane materiały pochodzące od różnych producentów. W bazie musi być tablica o schemacie Dostawca_Część_Inwestycja przechowująca zbiorcze informacje. Taka tabela jest znormalizowana (tabela 2.4), jest to jednak dość szczególny przypadek, gdyż występują w niej tylko zależności trywialne, bowiem wszystkie kolumny wchodzą w skład klucza głównego.

Taka tablica może być na poziomie piątej postaci normalnej zdekomponowana na podstawie zależności połączeniowej. Ta dekompozycja musi być odwracalna, tzn. podczas wykonywania złączenia naturalnego zdekomponowanych tablic musi być możliwe wierne odtworzenie podzielonej tablicy wraz z jej wszystkimi danymi.

Uwzględniając zależności wielowartościowe występujące w tablicy: D→→CZ, D→→I, I→→CZ, wyłączamy każdą z nich do oddzielnej tablicy (tabela 2.5).

Tabela 2.4. *Tabela ze złożonym kluczem głównym*

DOSTAWCA	CZĘŚĆ	INWESTYCJA
D1	CZ1	I1
D1	CZ2	I1
D1	CZ1	I2
D2	CZ2	I1
D3	CZ1	I1
D3	CZ2	I1

Tabela 2.5. *Trzy tablice binarne w 5PN*

DOSTAWCA	CZĘŚĆ
D1	CZ1
D1	CZ2
D2	CZ2
D3	CZ1
D3	CZ2

DOSTAWCA	INWESTYCJA
D1	I1
D1	I2
D2	I1
D3	I1

INWESTYCJA	CZĘŚĆ
I1	CZ1
I1	CZ2
I2	CZ1

Łączenie naturalne trzech powyżej zamieszczonych tablic daje możliwość odtworzenia zawartości tablicy sprzed podziału. Żadna krotka nie zostanie utracona ani nie pojawią się nieautentyczne dane.

2.7. Reguły dotyczące zależności funkcyjnych

Warto poznać kilka użytecznych zasad dotyczących zależności funkcyjnych, które są niezbędne przy projektowaniu dobrego schematu relacji.

Wyznaczenie zależności funkcyjnych spełnionych w relacji R zaczyna się od określenia tych zależności, które wynikają z semantyki atrybutów. Następnie, stosując określone reguły, zwane aksjomatami Armstronga, czy wykorzystując domknięcia zbioru atrybutów, można wywnioskować nowe zależności, które wynikają z istniejących.

Precyzyjne określenie zbioru zależności funkcyjnych spełnionych w tablicy pozwala na ocenę poprawności projektu tablicy, przeprowadzenie odpowiedniej dekompozycji czy wskazanie właściwego klucza tablicy.

2.7.1. Aksjomaty Armstronga

Przyjmijmy, że A,B,C są podzbiorami atrybutów relacji R.

Oto aksjomaty Armstronga [9]:

1. *Zwrotność* (ang. *Reflexivity*). Jeśli B jest podzbiorem A, to A→B. Nazywaliśmy to zależnością trywialną.
2. *Rozszerzenie* (ang. *Augmentation*). Jeśli A→B, to A,C→B,C.
3. *Przechodniość* (ang. *Transitivity*). Jeśli A→B i B→C, to A→C.

Z powyższych aksjomatów zostały wyprowadzone dalsze reguły:

- *Samookreślenie*. A→A.
- *Rozkład*. Jeśli A→B,C, to A→B i A→C.
- *Suma*. Jeśli A→B i A→C, to A→B,C.
- *Złożenie*. Jeśli A→B i C→D, to A,C→B,D.

2.7.2. Domknięcie zbioru atrybutów

Podstawowym sposobem ustalania zbioru zależności funkcyjnych spełnionych w relacji R jest obliczanie domknięcia zbioru atrybutów. [18]

Domknięciem zbioru wybranych atrybutów z tablicy R jest zbiór takich atrybutów, które jesteśmy w stanie wyznaczyć na podstawie wszystkich zależności funkcyjnych spełnionych w tej relacji. Przeanalizujmy definicję tego pojęcia.

Przyjmijmy, że w relacji R mamy podany zbiór S zawierający zależności funkcyjne. Domknięciem zbioru atrybutów $\{A_1,A_2,...A_n\}$ należących do relacji R nad zbiorem zależności S nazywamy taki zbiór atrybutów B, że jeśli w relacji R są spełnione wszystkie zależności ze zbioru S, to także zależność $A_1,A_2,...A_n \rightarrow B$ wynika z S. Domknięcie zbioru atrybutów $A_1,A_2,...A_n$ oznaczamy przez $\{A_1,A_2,...A_n\}^+$.

Na początku uwzględniamy zależności trywialne i przyjmujemy, że badany zbiór $A_1,A_2,...A_n$ jest zawarty w domknięciu $\{A_1,A_2,...A_n\}^+$. Następnie uzupełniamy początkowy zbiór atrybutów $A_1,A_2,...A_n$ tymi atrybutami, które występują z prawych stron tych zależności funkcyjnych ze zbioru S, jakie wynikają z $A_1,A_2,...A_n$. Postępujemy tak długo, jak to jest możliwe. Jeśli nie można już rozszerzyć otrzymanego zbioru atrybutów, to znaczy, że stanowi on domknięcie. Innymi słowy — domknięcie zawiera wszystkie atrybuty relacji funkcjonalnie zależne od badanego zbioru atrybutów A.

Przykład 2.2.

Dana jest relacja R o schemacie (A,B,C,D) oraz zbiór zależności funkcyjnych spełnionych w niej: B→C i AB→D. Ustalmy domknięcie dla atrybutu B.

Na początku na podstawie zależności trywialnych:

$\{B\}^+ = \{B\}$.

Na podstawie zależności B→C dołączamy do zbioru domknięcia atrybut C:

$\{B\}^+ = \{B,C\}$.

W podanym zbiorze zależności funkcyjnych nie ma żadnej innej zależności, której wyznacznik występowałby w zbiorze domknięcia, a więc nie ma sposobu rozszerzenia tego zbioru atrybutów. Ostatecznie, otrzymujemy domknięcie dla atrybutu B:

$\{B\}^+ = \{B,C\}$.

Przykład 2.3.

Uwzględnijmy relację o schemacie R(A,B,C,D) oraz zbiór S zawierający spełnione w tej relacji zależności funkcyjne: A,B→C, C→D, D→A.

Analizując domknięcia podzbiorów schematu tej relacji, możemy określić dodatkowe zależności funkcyjne spełnione w tej relacji i wynikające z zależności podanych w S oraz możemy ustalić klucze tej relacji.

Zacznijmy od ustalenia domknięć dla pojedynczych atrybutów:

$\{A\}^+ = \{A\}$;

$\{B\}^+ = \{B\}$; $\{C\}^+$

$= \{C,D,A\}$; $\{D\}^+$

$= \{D,A\}$.

Wnioski:

Po pierwszym etapie analizy można zauważyć, że jeśli jakiś atrybut (w naszym przykładzie są to A oraz B) nie występuje po lewej stronie w żadnej obowiązującej zależności, to jego domknięcie — na podstawie reguły samookreślenia — zawiera tylko ten atrybut i jego obecność w schemacie relacji nie ma wpływu na zbiór zależności tam spełnionych. Analizę domknięć można więc ograniczyć do badania zbiorów atrybutów zawierających wyznaczniki podanych zależności.

Z domknięcia atrybutu C wynikają zależności:

C→D (zależność jest już określona w zbiorze S);

C→A (to nowa zależność, którą można postrzegać jako zależność tranzytywną pomiędzy atrybutami C,D,A).

Domknięcia dla par atrybutów i wynikające z nich nowe zależności funkcyjne:

$\{A,B\}^+ = \{A,B,C,D\}$, nowa zależność A,B→D;

$\{A,C\}^+ = \{A,C,D\}$, nowa zależność A,C→D;

$\{A,D\}^+ = \{A,D\}$;

$\{B,C\}^+ = \{A,B,C,D\}$, nowe zależności B,C → A oraz B,C→D;

$\{B,D\}^+ = \{A,B,C,D\}$, nowe zależności B,D→A oraz B,D→C; $\{C,D\}^+$

$= \{A,C,D\}$, nowa zależność C,D→A.

Wnioski:

Jeśli dla badanego zbioru A_1, A_2,...A_n otrzymujemy domknięcie $\{A_1, A_2,...A_n\}^+$ zawierające pełny schemat relacji, wtedy badany zbiór jest nadkluczem relacji.

Jeśli spełnia on wymóg minimalnej długości, wówczas jest kluczem relacji. Należy to sprawdzić, obliczając domknięcia dla podzbiorów uzyskanych poprzez usunięcie choćby jednego atrybutu spośród A_1, A_2,...A_n.

Widać, że w kilku wypadkach dla podzbiorów {A,B}, {B,C} i {B,D} domknięciem jest pełny schemat relacji R. Ponieważ zaczęliśmy analizę od najkrótszych podzbiorów

relacji R, otrzymaliśmy więc trzy klucze kandydujące relacji: {A,B}, {B,C}, {B,D}, bez konieczności upewniania się, czy ustalone klucze spełniają wymóg minimalnej długości zbioru. Jeden z tych kluczy może być wybrany do roli klucza głównego.

Analiza domknięć dla podzbiorów trójelementowych pozwoli na zidentyfikowanie kolejnych zależności funkcyjnych spełnionych w relacji R i ustalenie nadkluczy.

$\{A,B,C\}^+ = \{A,B,C,D\}$, nowa zależność A,B,C→D;

$\{A,B,D\}^+ = \{A,B,C,D\}$, nowa zależność A,B,D→C;

$\{A,C,D\}^+ = \{A,C,D\}$;

$\{B,C,D\}^+ = \{A,B,C,D\}$, nowa zależność B,C,D→A.

Wnioski:

Podzbiory: {A,B,C}, {A,B,D} oraz {B,C,D} są nadkluczami relacji R, każdy z nich stanowi rozszerzenie zbioru klucza o jeden dodatkowy atrybut.

Domknięcie dla całego schematu relacji nie daje nowych zależności:

$\{A,B,C,D\}^+ = \{A,B,C,D\}$.

W wyniku przeprowadzonych obliczeń uzyskujemy pełny zbiór zależności funkcyjnych spełnionych w relacji R: A,B→C, C→D, D→A, C→A, A,B→D, A,C→D, B,C→A, B,C→D, B,D→A, B,D→C, C,D→A, A,B,C→D, A,B,D→C, B,C,D→A.

Wnioski:

Analizując kompletny zbiór zależności funkcyjnych spełnionych w relacji R, należy zwrócić uwagę na kolejną zasadę dotyczącą zależności. Mianowicie, poszerzenie zbioru wyznacznika również spełnia daną zależność, co można zobrazować następującymi przykładami: C→D, A,C→D, A,B,C→D. Niektóre zależności są więc zbędne w tym zbiorze, ale optymalizację tego zbioru zależności omówimy w dalszej części opracowania.

Analizę domknięć przeprowadza się również w celu zidentyfikowania wszystkich zależności funkcyjnych i ustaleniu kluczy w relacjach uzyskanych w drodze dekompozycji. Kolejny przykład będzie dotyczył tego problemu.

Przykład 2.4.

W wyniku dekompozycji relacji R(A,B,C,D,E) otrzymano między innymi tablicę R1(A,B,C). W relacji R były spełnione zależności: A→D, B→E, D,E→C.

Aby określić, jakie własności ma tablica R1, jakie zależności funkcyjne są w niej spełnione oraz co jest kluczem, należy obliczyć domknięcia wszystkich podzbiorów atrybutów należących do schematu tej tablicy.

$\{A\}^+ = \{A,D\}$ — ponieważ atrybut D nie występuje w relacji R1, zależność A→D nie jest spełniona w relacji R1.

Analogicznie, domknięcia dla atrybutów B i C nie wprowadzają żadnych zależności funkcyjnych spełnionych w R1. $\{B\}^+ = \{B,E\}$, $\{C\}^+ = \{C\}$.

Kolejno, należy przeanalizować pary atrybutów:

$\{A,B\}^+ = \{A,B,C,D\}$.

Wnioski:

Spośród dwóch zależności A,B→C i A,B→D tylko pierwsza jest spełniona w relacji R1.

Domknięcia dla kolejnych dwóch par atrybutów nie wprowadzą żadnych nowych zależności, które byłyby spełnione w R1: $\{A,C\}^+ = \{A,C,D\}$, $\{B,C\}^+ = \{B,C,E\}$.

Wnioski:

Domknięcie dla wszystkich trzech atrybutów ($\{A,B,C\}^+ = \{A,B,C,D,E\}$) również nie dostarczy żadnej nowej zależności funkcyjnej dla R1, jedynie pozwala określić zależność trywialną A,B,C→A,B,C.

Zatem jedyną zależnością, która zostaje przeniesiona do relacji R1 po dekompozycji relacji R, jest zależność A,B→C i z niej wynika klucz {A,B} tej relacji.

2.7.3. Równoważność oraz minimalne pokrycie zbioru zależności

Do przeanalizowania algorytmu przeprowadzania dekompozycji dowolnej relacji R do 3NF jest potrzebne przybliżenie kilku pojęć takich jak: równoważność zbiorów zależności funkcyjnych oraz minimalne pokrycie zbioru zależności. [14]

Z równoważnością zbiorów zależności funkcyjnych E i F mamy do czynienia wówczas, jeśli E^+ i F^+, tzn. zbiory zależności funkcyjnych ustalonych poprzez obliczanie domknięć, są takie same.

Zbiory zależności funkcyjnych E i F pokrywają się, jeśli każda zależność funkcyjna ze zbioru E występuje w domknięciu F^+, to znaczy, że każdą zależność funkcyjną ze zbioru E można wywnioskować na podstawie zbioru F.

Minimalne pokrycie zbioru E to minimalny zbiór zależności funkcyjnych F, równoważny zbiorowi E.

Minimalne pokrycie zbioru [9] określane w literaturze przedmiotu jako baza minimalna relacji (w skrócie *baza B*) musi spełniać trzy warunki:

- wszystkie zależności funkcyjne w tym zbiorze B mają jednoelementowe prawe strony,
- po usunięciu dowolnej zależności funkcyjnej z B wynikowy zbiór nie będzie bazą, czyli zbiór zależności funkcjonalnych w B musi być minimalny,
- jeśli z dowolnej zależności funkcjonalnej z B usuniemy jeden lub kilka atrybutów z lewej strony zależności, wynik nie będzie bazą — to znaczy, że w każdej zależności wyznacznik musi być najkrótszy.

2.8. Projektowanie schematów relacyjnych baz danych

Obliczanie domknięć dla poszczególnych zbiorów atrybutów pozwala ustalić zbiór wszystkich możliwych zależności funkcjonalnych spełnionych w danej relacji.

Poprawnie zidentyfikowane zależności są podstawą zdefiniowania takich struktur tablic w bazie, które spełniają wymagania trzeciej postaci normalnej. Obecnie normalizację przeprowadza się z pominięciem dwóch pierwszych postaci normalnych, a baza minimalna relacji jest podstawą definiowania poprawnych tablic.

Cały proces projektowania jest realizowany w dwóch etapach. Pierwszy to ustalenie minimalnego pokrycia dla zbioru zależności, drugi — definiowanie gotowych struktur. Opis algorytmów wykonujących te zadania oraz przykład ich praktycznego zastosowania jest omówiony poniżej.

2.8.1. Algorytm znajdowania pokrycia minimalnego dla zbioru zależności

Przeanalizujmy algorytm znajdowania pokrycia minimalnego F dla zbioru zależności funkcjonalnych E. [14]

1. W pierwszym kroku przyjmujemy, że w zbiorze pokrycia minimalnego F znajdą się wszystkie zależności funkcjonalne E.
2. Ponieważ wszystkie zależności funkcjonalne w tym zbiorze muszą mieć jednoelementowe prawe strony, zastępujemy zależności typu $X \rightarrow A_1A_2...A_n$ zależnościami prostymi: $X \rightarrow A_1$, $X \rightarrow A_2$ ... $X \rightarrow A_n$.
3. Ponieważ wszystkie wyznaczniki muszą być najkrótszymi zbiorami, badamy, czy po usunięciu jakiegokolwiek atrybutu ze zbioru wyznacznika jest zachowana dana zależność. Jeśli tak, wówczas eliminujemy nadmiarowy atrybut ze zbioru wyznacznika.
4. Usuwamy dublujące się zależności w zbiorze F.

Przeanalizujmy praktyczne zastosowanie tego algorytmu.

Przykład 2.5. _____

Przyjmijmy, że mamy daną relację R(A,B,C,D,E,F,G,H,I) oraz zbiór E zależności spełnionych w tej relacji:

$A,B \rightarrow C,D,E,F,G,H;$

$A,C \rightarrow D,E;$ $A \rightarrow C,D,E;$

$B \rightarrow F,G;$

$D \rightarrow I;$

$B,F \rightarrow G.$

Zgodnie z pierwszym krokiem algorytmu przyjmujemy, że w zbiorze minimalnego pokrycia znajdą się wszystkie zależności ze zbioru E. Następnie zastępujemy zależności złożone zależnościami prostymi.

A,B→C;

A,B→D;

A,B→E;

A,B→F;

A,B→G;

A,B→H;

A,C→D;

A,C→E;

A→C;

A→D;

A→E;

B→F;

B→G;

D→I;

B,F→G.

Następnie usuwamy zbędne zależności przedstawione poniżej.

Jeśli mamy zależności:

A→C;

A→D;

A→E.

Wówczas zbędne będą następujące:

A,B→C;

A,B→D; A,B→E.

Podobnie, jeśli mamy:

B→G.

Wówczas do usunięcia pozostaną:

B,F→G.

Usuwamy również zależność oraz A,B→F, bo w zbiorze F mamy już zależność B→F. Usuwamy też A,B→G, ponieważ w zbiorze F mamy już zależność B→G.

Ostatecznie mamy następujący zbiór zależności:

A,B→H;

A→C;

A→D;

A→E;

B→F;

B→G; D→I.

Otrzymany zbiór zależności, zwany *bazą minimalną*, jest podstawą definiowania tablic znormalizowanych już do trzeciej postaci normalnej.

2.8.2. Algorytm tworzenia dekompozycji relacji R do 3NF

Ten algorytm pozwala wykonać dekompozycję relacji R na zbiór relacji o następujących cechach:

- Wszystkie relacje uzyskane w wyniku dekompozycji są już w 3NF,
- Dekompozycja jest procesem odwracalnym, tzn. istnieje dla niej złączenie bezstratne.
- Dekompozycja pozwala zachować zależności.

Dane wejściowe to relacja uniwersalna R i określony zbiór zależności funkcyjnych spełnionych w tej relacji.

Algorytm jest wykonywany w następujących krokach:

1. Uwzględniamy *bazę minimalną* otrzymaną w wyniku przeprowadzenia poprzedniego algorytmu.
2. Dla każdego zbioru X występującego na lewej stronie zależności funkcjonalnych w pokryciu F tworzymy oddzielny schemat relacji. Zbiór atrybutów X jest kluczem tej relacji, a pozostałe atrybuty to te, dla których X był wyznacznikiem.
3. Pozostałe atrybuty, które nie znalazły się w żadnej relacji tworzonej w drugim kroku tego algorytmu, są umieszczane w nowym schemacie.

Przykład 2.6.

Kontynuujmy poprzedni przykład — dla relacji R(A,B,C,D,E,F,G,H,I) ustaliliśmy pokrycie minimalne zbioru zależności (w przykładzie 2.1). Zgodnie z drugim algorytmem zdekomponujmy relację R na zbiór relacji w trzeciej postaci normalnej. Ustalona baza minimalna zawiera:

A,B→H;

A→C;

A→D;

A→E;

B→F;

B→G; D→I.

Dla każdego wyznacznika zależności występującej w pokryciu minimalnym tworzymy oddzielny schemat relacji, w której wyznacznik jest kluczem głównym. Otrzymamy więc następujące tablice (klucze wyróżnione podkreśleniem):

R1 = (**A**,**B**,H); R2
= (**A**,C,D,E);
R3 = (**B**,F,G); R4
= (**D**,I).

Ten algorytm ma szczególne zastosowanie, gdy analiza semantyczna atrybutów pozwala pogrupować je, umożliwiając wyróżnienie opisu poszczególnych typów obiektów występujących w bazie.

Przeanalizujmy, jakie rezultaty przyniesie zastosowanie tego algorytmu do normalizacji danych pochodzących z faktury.

Przykład 2.7.

Mamy podaną do zdekomponowania tablicę R (Nr_Faktury, Data_wystawienia, Nr_klienta, Nazwa_klienta, Adres_kl, Nr_pozycji, Kod_towaru, Nazwa_towaru, Cena_towaru, Ilość_sprzedana, Wartość_sprzedanego_tow, Wartość_całej_faktury)

1. Analizując semantykę podanych atrybutów, określamy występujące zależności funkcyjne:

   Z1: Nr_klienta→Nazwa_klienta, Adres_kl Z2:
   Kod_towaru→Nazwa_towaru, Cena_towaru

   Z3: Nr_Faktury→Data_wystawienia, Nr_klienta, Nazwa_klienta, Adres_kl, Wartość_całej_faktury

   Z4: Nr_Faktury, Nr_pozycji→Kod_towaru, Nazwa_towaru, Cena_towaru, Ilość_sprzedana, Wartość_sprzedanego_tow

2. Tworzymy pokrycie minimalne, odrzucając zależności tranzytywne:

   a) Nr_klienta→Nazwa_klienta;

   b) Nr_klienta→Adres_kl;

   c) Kod_towaru→Nazwa_towaru;

   d) Kod_towaru→Cena_towaru;

   e) Nr_Faktury→Data_wystawienia;

   f) Nr_Faktury→Nr_klienta;

   g) Nr_Faktury→Wartość_całej_faktury;

   h) Nr_Faktury, Nr_pozycji→Kod_towaru;

ö

i) Nr_Faktury, Nr_pozycji Ílloħè_sprzedana;

j) Nr_Faktury, Nr_pozycji ÍWartoħè_sprzedanego_tow.

3. Tworzymy oddzielne schematy relacji dla kaĪdego wyznacznika zaleĪnoĞci wystĊpującej w pokryciu minimalnym: Klient(Nr_klienta, Nazwa_klienta, Adres_kl);

Towar(Kod_towaru, Nazwa_towaru, Cena_towaru);

Faktura(Nr_Faktury, Data_wystawienia, Nr_klienta, Wartoħè_cađej_faktury);

Pozycje(Nr_Faktury, Nr_pozycji, Kod_towaru, Iloħè_sprzedana, Wartoħè_sprzedanego_tow).

Uzyskane relacje są w trzeciej postaci normalnej i porównanie ich z tablicami otrzymanymi inną metodą projektowania nasuwa wniosek, Īe zarówno normalizacja klasyczna, jak i zastosowanie algorytmu daje podobne rezultaty.

Omówiony algorytm tworzenia dekompozycji relacji do 3NF w literaturze przedmiotu jest nazywany algorytmem syntezy do schematów o trzeciej postaci normalnej. [zob. Garcia-Molina, Ullman, 2011, s. 117]. Taka dekompozycja zachowuje zaleĪnoĞci funkcjonalne i umoĪliwia záączenie bezstratne.

Przyszáa pora na przybliĪenie terminu záączenia bezstratnego.

2.8.3. Zäñczenie bezstratne

Przeprowadzona dekompozycja jest procesem odwracalnym, tzn. za pomocą záączenia naturalnego relacji uzyskanych w wyniku podziaáu jest moĪliwe odtworzenie dokáadnej postaci relacji pierwotnej. Nie kaĪdy podziaá tablicy przynosi takie efekty — porównajmy dwie róĪne dekompozycje tej samej tablicy obrazujące omawiany problem.

Przykäad 2.8.

Dekompozycja nieodwracalna. Relacja R o schemacie (A,B,C) zostaáa zdekomponowana do dwóch schematów: R1(A,B) oraz R2(B,C). Wraz z przykáadowymi wartoĞciami jest zamieszczona na rysunku 2.4.

R:

A	B	C
1	6	3
4	6	5

R1:

A	B
1	6
4	6

R2:

B	C
6	3
6	5

Rysunek 2.4. *Przykáad relacji R (przed dekompozycją) oraz R1 i R2 uzyskanych po dekompozycji relacji R z wartoĞciami*

Aby odzyskaü instancjĊ relacji, która zostaáa zdekomponowana, naleĪy kaĪdą krotkĊ wystĊpującą w jednej relacji powstaáej po dekompozycji poáączyü z tymi wszystkimi krotkami wystĊpującymi w drugiej relacji, w których atrybuty wspólne z daną krotką są identyczne.

Próbując odtworzyü relacjĊ R, naleĪy dokonaü záączenia naturalnego relacji R1 i R2. W wyniku tej operacji zostaje uzyskana relacja o schemacie (A,B,C) z wartoĞciami przedstawionymi w tabeli 2.6.

Tabela 2.6. *Wynik záączenia naturalnego relacji R1 i R2*

A	B	C
1	6	3
1	6	5
4	6	3
4	6	5

Otrzymana tablica róĪni siĊ od pierwotnej tablicy R — zawiera dwie krotki: (1,2,5) oraz (4,2,3), które nie wystĊpowaáy w relacji R przed dekompozycją. Taki podziaá tablicy R jest wiĊc przykáadem dekompozycji nieodwracalnej.

Przykäad 2.9. _____

Dekompozycja odwracalna. Przeanalizujmy inny podziaá tablicy R (rysunek 2.5) na dwie relacje o schematach R3(A,B) i R4(A,C).

R:

A	B	C
1	6	3
4	6	5

R3:

A	B
1	6
4	6

R4:

A	C
1	3
4	5

Rysunek 2.5. *Przykáad relacji R (przed dekompozycją) oraz R3 i R4 uzyskanych po dekompozycji relacji R z wartoĞciami*

W wyniku záączenia naturalnego relacji R3 i R4 zostaje otrzymana tablica toĪsama ze zdekomponowaną relacją R (tabela 2.7). W tym przykáadzie mamy do czynienia z dekompozycją odwracalną.

Tabela 2.7. *Wynik záączenia naturalnego relacji R3 i R4*

A	B	C
1	6	3
4	6	5

Trzeba pamiętać, że złączenie naturalne jest łączne i przemienne. Nie ma znaczenia kolejność złączania otrzymanych po dekompozycji relacji.

Z przytoczonych przykładów wynika, że sposób przeprowadzenia dekompozycji tablicy ma niebagatelne znaczenie. Aby uniknąć powstawania wielu błędów w bazie, dekompozycja musi być procesem odwracalnym.

2.8.4. Test na złączenie bezstratne oparty na algorytmie chase

Istnieje test na złączenie bezstratne oparty na algorytmie chase [Garcia-Molina, Ullman, 2011, str. 111]. Test ten pozwala w uporządkowany sposób udowodnić, że przy spełnionych określonych zależnościach funkcjonalnych w tablicy R każda krotka otrzymana w wyniku złączenia naturalnego relacji po dekompozycji znajduje się także w tablicy R, co świadczy o odwracalności dekompozycji.

W teście należy zapisać dane w postaci tzw. obrazu (ang. *tableau*). Obraz przedstawia tablicę o tylu kolumnach, ile atrybutów ma schemat relacji R przed dekompozycją (kolumna czerpie nazwę z atrybutu), i tylu wierszach, ile relacji uzyskano w wyniku dekompozycji. W każdym wierszu oddzielnie są zapisywane schematy relacji po dekompozycji. Jeśli dana relacja ma schemat {A,B}, bo została utworzona przez projekcję na atrybuty A i B relacji R, to w odpowiednim wierszu obrazu będą zapisane wartości *a*, *b* jako komponenty w kolumnach A i B, natomiast w pozostałych komórkach będą wpisane wartości wskazujące na daną kolumnę, ale z indeksem dolnym wskazującym numer wiersza obrazu. Elementy *a* i *b* bez indeksów dolnych oznaczają, że kolumny A i B w tablicy po dekompozycji zawierają takie same dane jak w tablicy przed dekompozycją. Natomiast kolejne komponenty z indeksami oznaczają nieznaną wartość, którą możemy odtworzyć na podstawie zależności funkcyjnych, łącząc z powrotem zdekomponowane tablice.

Uwzględniamy kolejne zależności funkcyjne spełnione w relacji R (kolejność jest dowolna). Dla wybranej zależności np. A→B sprawdzamy, czy w obrazie występują dwa wiersze (lub więcej), które mają zgodne komponenty A — wówczas z zależności funkcyjnej wynika, że muszą mieć takie same komponenty B. Zrównujemy więc komponenty B w analizowanych wierszach, zgodnie z następującą regułą: jeśli któryś z komponentów nie ma indeksu dolnego, zastępujemy nim drugi. Jeśli oba symbole mają własny indeks dolny, można je zastąpić dowolnie jeden drugim. Test przeprowadzamy tak długo, dopóki wprowadzanie zmian w obrazie będzie możliwe, lub do czasu, aż uzyskamy jeden wiersz w obrazie zawierający same symbole bez indeksów dolnych. Taki wynik testu udowodni, że uzyskana krotka w złączeniu tablic po dekompozycji jest krotką z relacji R, czyli podział tablicy R jest odwracalny.

Zastosujmy omówiony test do tablicy z przykładowymi danymi, którą próbowaliśmy dekomponować powyżej.

Przykład 2.10.

Uwzględnijmy w przykładzie tablicę R(A,B,C), która została podzielona na dwie relacje R3(A,B) i R4(A,C). Taka dekompozycja jest przedstawiona w przykładzie poprzedzającym test na złączenie bezstratne. W tablicy R są spełnione zależności funkcyjne A→B i A→C. Obraz tej dekompozycji byłby taki jak w tabeli 2.8.

Tabela 2.8. *Obraz dekompozycji R(A,B,C) na R3(A,B) i R4(A,C)*

A	B	C
a	b	c_1
a	b_2	c

Pierwszy wiersz obrazu odpowiada tabeli R3 — komponenty dla atrybutów A i B są oznaczone jako *a* i *b* bez indeksów dolnych. Dla atrybutu C komponent *c* ma dodany indeks dolny 1, aby okreĞliü, Īe atrybut C jest poza schematem tabeli R3 i ma dowolną wartoĞü. Analogicznie jest zbudowany drugi wiersz obrazu, odpowiadający tabeli R4.

UwzglĊdniając pierwszą zaleĪnoĞü funkcjonalną AÍB, zauwaĪamy, Īe jeĞli oba wiersze obrazu mają zgodne komponenty A, to muszą mieü takie same komponenty B, zastĊpujemy wiĊc b_2 przez *b* (tabela 2.9).

Tabela 2.9. *Obraz dekompozycji R(A,B,C) na R3(A,B) i R4(A,C) po przeprowadzeniu testu*

A	B	C
a	b	c_1
a	b	c

Na tym etapie juĪ widaü, Īe drugi wiersz obrazu jest identyczny z krotką relacji R. W ten sposób udowodniliĞmy, Īe jeĞli w relacji R(A,B,C) są speánione zaleĪnoĞci funkcyjne AÍB i AÍC, to po przeprowadzeniu projekcji na (A,B) i (A,C) oraz záączeniu zdekomponowanych tablic moĪemy w sposób bezstratny odtworzyü relacjĊ R.

Przykäad 2.11.

PrzeprowadĪmy teraz test chase dla drugiego przypadku: podziaáu tablicy R na R1(A,B) oraz R2(B,C). Obraz tej dekompozycji jest zaprezentowany w tabeli 2.10.

Tabela 2.10. *Obraz dekompozycji R(A,B,C) na R3(A,B) i R4(B,C)*

A	B	C
a	b	c_1
a_2	b	c

Analizując kolejno zaleĪnoĞci funkcyjne AÍB i AÍC speánione w R, nie moĪemy wprowadziü do obrazu Īadnych zmian, gdyĪ nie znajdujemy dwóch wierszy, które

miaáyby zgodne komponenty A. KoĔczymy test, pozostawiając w kaĪdym wierszu obrazu zarówno komponenty z indeksami dolnymi, jak i te bez indeksów, co dowodzi, Īe dekompozycja R(A,B,C) na R1(A,B) oraz R2(B,C) jest nieodwracalna.

W obu wypadkach przeprowadzony test potwierdziá wcześĜniejszą ocenĆ dokonanej dekompozycji, zobrazowanej rzeczywistymi danymi w przykáadzie powyĪej.

2.9. Zadania

1. Na podstawie zależności funkcyjnych występujących w relacji R(A,B,C,D) określ, w której postaci normalnej jest ta tablica. Odpowiedź krótko uzasadnij.
   a) A→B, B→C, C→D;
   b) AB→C, D→C;
   c) CD→B, C→A;
   d) A→C, D→B;
   e) D→C, D→A, A→B;
   f) A→→B, AB→C AB→D.
2. Określ, jaka zależność funkcjonalna musi być usunięta z podanej tablicy i w której postaci normalnej zostanie wyeliminowana:
   a) Wypożyczenie(id_wypożyczenia <pk>, id_książki, id_czytelnika, imię_czytelnika, nazwisko_czytelnika);
   b) PozycjaZamówienia(id_towaru <pk>, id_zamówienia<pk>, nazwa_towaru, cena_towaru, ilość).
3. Odpowiedz, w której postaci normalnej występuje zaproponowany poniżej schemat — jeśli to możliwe, przekształć go do 3NF.

   Nr_zamówienia <PK>
   Kod_towaru <PK>
   Nazwa_towaru
   Cena_towaru
   Ilość_ zamówiona
4. Oceń postać normalną i następnie znormalizuj poniższą tablicę:

   Nr_Faktury <PK>
   Nr_Klienta
   Wartość_faktury
   Data_wystawienia_faktury
   Imię_klienta
   Nazwisko_klienta
5. Odpowiedz, w której postaci normalnej występuje zaproponowany schemat — jeśli to możliwe, przekształć go do 3NF, wiedząc, że operacja wykonywana przez

wielu lekarzy dotyczy jednego pacjenta. Kod_operacji <PK> data_operacji godz_operacji Nr_pacjenta

Nazwisko_pacjenta

Data_przyjęcia_do_szpitala

Nr_lekarza

Nazwisko_lekarza

6. Określ zależności funkcyjne oraz klucz w relacji R1(A,B,C) otrzymanej po dekompozycji relacji R(A,B,C,D), jeśli w R występowały następujące zależności funkcyjne:

a) AB→D, AC→E, BC→D, D→A, E→B;

b) A→B, B→C, C→D, D→E, E→A;

c) A→D, BD→E, AC→E, DE→B.

7. Sprawdź, czy rozkład relacji R(A,B,C,D,E) na relacje: R1(A,B), R2(C,E), R3(A,D), R4(C,D,E) przy spełnionych zależnościach A→C, C→DE, D→A, AC→B jest odwracalny.

8. Sprawdź, czy rozkład relacji R(A,B,C,D,E) na relacje: R1(A,D), R2(B,C), R3(A,E), R4(C,D,E) przy spełnionych zależnościach A→D, B→C, D→E, DE→C jest odwracalny.

9. Znormalizuj tablicę reprezentującą arkusz zaliczeń przedmiotów, określając najpierw zależności występujące miedzy danymi:

Nazwa_przedmiotu

Nr_prowadzącego_przedmiot

Nazwisko_pracownika

Imię_pracownika

Nr_studenta

Nazwisko_studenta

Imię_studenta

typ_oceny(zal/egz) Ocena

10. W relacji R(I,N,A,K,P,O,T) poszczególne dane to:

I — numer indeksu studenta,

N — nazwisko,

A — adres studenta,

K — kierunek studiów,

P — przedmiot,

O — ocena końcowa,

T — tematy realizowane w ramach przedmiotu.

W relacji R spełnione są zależności: I→NAK, IP→O, P→→T.

Znormalizuj relację R.

11. Znormalizuj tablicę reprezentującą arkusz zaliczeń przedmiotów, określając najpierw zależności występujące miedzy danymi:

Nazwa_przedmiotu

Nr_prowadzącego_przedmiot

Nazwisko_pracownika

Imię_pracownika
Nr_studenta
Nazwisko_studenta Imię_studenta
typ_oceny(zal/egz)
Ocena

Rozdział 3. Język baz danych SQL — podstawy

Język SQL (skrót od ang. *Structured Query Language*), strukturalny język zapytań, służy głównie do komunikowania się z bazą danych. Został opracowany w laboratorium IBM w Kalifornii w latach siedemdziesiątych. W roku 1986 SQL został uznany przez Międzynarodową Organizację Normalizacyjną (ISO) i Amerykański Narodowy Instytut Normalizacji (ANSI) jako standardowy język dla systemów zarządzania relacyjnymi bazami danych i nazwany SQL-86 (SQL-1). Pierwszą firmą, która zaimplementowała ten język do komercyjnej bazy, był Oracle. Standard szybko doczekał się kolejnej rozszerzonej wersji, wprowadzonej na rynek w roku 1992 pod nazwą SQL-92 (SQL-2). Choć druga wersja standardu była pięciokrotnie obszerniejsza od pierwszej, nadal język SQL pozostał językiem nieproceduralnym — niepozwalającym na tworzenie kompletnych programów. Do standardu wprowadzono więc elementy modelu obiektowego i opublikowano jego nową wersję w roku 1999 pod nazwą SQL-99 (SQL-3) — dopiero wówczas SQL stał się w pełni językiem, w którym można programować. Ta wersja została powszechnie zaimplementowana w większości systemów zarządzania bazami danych, jednak z pewnymi modyfikacjami i rozszerzeniami w poszczególnych implementacjach. Nowe funkcje i obsługę danych zapisanych w formacie XML włączono w kolejnej wersji standardu opublikowanego w roku 2003 pod nazwą standard SQL4. Trzy lata później, w 2006 roku, ukazał się standard SQL5 całkowicie poświęcony danym w formacie XML i ich obsłudze. Cały czas, do chwili obecnej, trwają prace nad kolejnymi wersjami standardu.

Polecenia SQL ze względu na realizowane zadania są podzielone na grupy (zwane podjęzykami):

- DDL — ang. *Data Definition Language* — język definiowania danych umożliwiający tworzenie, modyfikację i usuwanie struktur takich jak tabele, indeksy, widoki, bazy. Obejmuje polecenia: CREATE, ALTER, DROP.
- DML — ang. *Data Manipulation Language* — język manipulowania danymi, umożliwia wprowadzanie, usuwanie i modyfikację danych w bazie. Obejmuje polecenia: INSERT, UPDATE, DELETE.

- DQL — ang. *Data Query Language* — język formułowania zapytań do bazy danych (wyszukiwania danych), często zaliczany do grupy DML. Obejmuje polecenie SELECT.
- DCL — ang. *Data Control Language* — język kontroli danych, sterowania danymi, umożliwia nadawanie użytkownikom uprawnień do obiektów w bazie. Najważniejsze polecenia to GRANT i REVOKE.

Inne grupy poleceń wprowadzone w standardzie SQL-3 to:

- *Connection Statement* — polecenia odpowiedzialne za połączenia z serwerem, np. CONNECT i DISCONNECT.
- *Session Statement* — polecenia umożliwiające kontrolę sesji użytkownika.
- *Diagnostic Statement* — narzędzia diagnostyczne i służące do obsługi błędów.
- *Transaction Statement* — obejmuje instrukcje związane z transakcjami: START, COMMIT, ROLLBACK.

SQL jest językiem deklaratywnym, czyli umożliwia zdefiniowanie żądania dotyczącego określonych danych w bazie, bez precyzowania sposobu przechowywania i udostępniania tych danych, bo o tym decyduje System Zarządzania Bazą Danych. Ze względu na sposób wykorzystania z tego języka wyróżnia się trzy formy SQL — interakcyjny, statyczny i dynamiczny.

SQL interakcyjny (samodzielny) — uruchamiany przez użytkownika w celu wyszukania danych w bazie lub wstawienia danych do bazy.

SQL statyczny — nie ulega zmianom, jest uruchamiany z aplikacją pisaną w innym języku. Może wystąpić w dwóch odmianach: jako SQL osadzony, kiedy instrukcje realizujące odwołania do bazy są umieszczane w kodzie aplikacji, albo jako język modułów, kiedy polecenia SQL stanowią całe moduły i są łączone z modułami pisanymi w innym języku.

SQL dynamiczny — używany, kiedy w trakcie pracy aplikacji użytkownik definiuje żądanie do bazy, wykorzystując takie narzędzia, jak graficzne języki zapytań.

3.1. Typy danych i literały w bazie ORACLE

Typ danych to podstawowa własność kolumny w tablicy lub zmiennej w procedurze — determinuje on rodzaj informacji przechowywanej w kolumnie oraz ogranicza zakres wykonywanych operacji.

W Oracle występują:

- Typy wbudowane, dostarczane z oprogramowaniem,
- Typy danych zdefiniowane przez użytkownika, z wykorzystaniem typów wbudowanych i innych zdefiniowanych przez użytkownika typów danych.

Dostępne w Oracle typy danych można podzielić na: typy znakowe, liczbowe, daty i pozostałe.

3.1.1. Znakowe typy danych

Znakowe typy danych mogą przechowywać dowolną wartość łańcuchową, w tym wartości liczbowe. Do znakowych typów wbudowanych w Oracle należą: VARCHAR2(*rozmiar*) i CHAR[(*rozmiar*)].

Typ VARCHAR2(*rozmiar*) pozwala na przechowywanie ciągów znaków zmiennej długości. Rozmiar wskazuje nieprzekraczalną długość łańcucha znaków. Maksymalnie może to być 4000 znaków, minimalnie 1 znak.

Typ CHAR[(*rozmiar*)] przechowuje ciąg znaków o stałej długości określonej przez rozmiar. Domyślnie rozmiar wynosi 1 znak, maksymalnie 2000 znaków. Jeśli przypiszemy zmiennej wartość krótszą niż zdefiniowany rozmiar, to pozostałe znaki zostaną uzupełnione spacjami.

Typy NCHAR i NVARCHAR2 przechowują łańcuchy znaków odpowiednio o stałej i zmiennej długości, przy wykorzystaniu zestawu znaków języka narodowego, alternatywnego dla reszty bazy. Dane zapisane w Unicode: UTF8 i AL16UTF16.

Typ LONG pozwala przechowywać do 2GB danych znakowych, wiąże się jednak z wieloma ograniczeniami — zmiennych takich typów nie można stosować w klauzulach WHERE, GROUP BY, ORDER BY i z klauzulą DISTINCT. Nie można definiować dla nich indeksów i ograniczeń integralnościowych. Tabela może zawierać tylko jedną kolumnę typu LONG. W nowych wersjach Oracle dla dużych łańcuchów znakowych są rekomendowane typy CLOB i NCLOB.

3.1.2. Liczbowe typy danych

Wbudowanym typem liczbowym, jedynym w bazie Oracle, jest NUMBER. Są w nim przechowywane następujące typy: DECIMAL, NUMBER, INTEGER, FLOAT, DOUBLE PRECISION i REAL. Ten typ możemy również zadeklarować z użyciem dwóch kwalifikatorów: precyzji i skali.

Typ NUMBER[(*p,s*)] przechowuje wartości numeryczne o ściśle określonej długości. Precyzja (*p*) oznacza łączną dopuszczalną liczbę znaczących cyfr, a jej wartość należy do przedziału od 1 do 38. Skala (*s*) to maksymalna liczba cyfr po prawej stronie przecinka. Jeśli, deklarując typ NUMBER, nie podajemy precyzji ani skali, to domyślnie precyzja ma wartość 38, skala zaś automatycznie przyjmie wartość 0. Skala musi należeć do przedziału od –84 do 127. Jeśli skala będzie miała wartość ujemną, to Oracle będzie zaokrąglać wartości o odpowiednią liczbę miejsc w lewo.

3.1.3. Typ daty

Oracle korzysta z wbudowanego formatu DATE — jego składnia to DD-MIE-RR GG:MI:SS, gdzie DD to dzień, MIE to trzyliterowy skrót nazwy miesiąca, RR to rok

zapisany za pomocą dwóch cyfr, natomiast GG,MI,SS to odpowiednio godziny, minuty, sekundy zapisane z użyciem dwóch cyfr. Jeśli wraz z datą nie podamy dokładnego czasu, to wszystkie wartości czasu będą miały domyślną wartość zero. Domyślny format daty jest określony przez parametr NLS_DATE_FORMAT, a modyfikując sesję, można zmienić format daty.

Typ TIMESTAMP (*rozmiar*) przechowuje czas z dokładnością do ułamków sekundy, *rozmiar* określa liczbę znaków przeznaczoną na znacznik czasu — domyślnie jest to 6 znaków, maksymalnie 9 znaków.

Od wersji Oracle9i są również używane dwa interwałowe typy danych: INTERVAL YEAR TO MONTH oraz INTERVAL DAY TO SECOND przeznaczone do zapamiętania przedziału czasu wyrażonego liczbą lat i miesięcy oraz — w drugim wypadku — wyrażonego liczbą dni, minut i sekund. Te typy zostały zdefiniowane w rozszerzonej wersji standardu SQL99.

3.1.4. Pozostałe typy danych

Oracle posiada również specjalne typy danych:

- ROWID (od ang. *row ID*) — identyfikator wiersza, zawiera adres pozwalający na fizyczną lokalizację rekordu. W bazie Oracle (od wersji 7) ROWID jest typu znakowego, przechowuje 18-znakowy łańcuch ustalany przez system. Dostęp do określonej wartości w tabeli jest realizowany albo poprzez wykonanie pełnego przeglądu tabeli albo poprzez skorzystanie z identyfikatorów ROWID, jeśli na kolumnie, która występuje w warunku wyszukiwania, jest zdefiniowany indeks. Wartość ROWID nie zmienia się przez cały czas istnienia wiersza, jednak w czasie wykonywania eksportu i importu danych z tablic wartości ROWID mogą się zmieniać. Po usunięciu wiersza z tablicy ta sama wartość ROWID może być wykorzystana dla wiersza wprowadzonego później.
- RAW(*size*) — typ służący do przechowywania danych binarnych, ma rozmiar od 1 do 2000 bajtów.
- LOB (ang. *large object*) — typ dużych obiektów, przechowuje do 4 GB informacji (w wersjach wcześniejszych niż Oracle 10 g). Aktualnie górna granica rozmiaru to nawet 128 TB, w zależności od rozmiaru bloku bazy. Mogą to być dane różnych typów, stąd podział na kategorie:
- CLOB, przechowujący dane tekstowe kodowane zestawem znaków domyślnym dla bazy danych, podobnie jak NCLOB jest odpowiednikiem typu TEXT,
- NCLOB przechowujący dane tekstowe, może korzystać z zestawu znaków NLS (ang. *National Language Support* — wsparcie języków narodowych),
- BLOB przechowujący dane w postaci binarnej, np.pliki wideo czy audio. Zastąpił dawniej stosowany typ LONG RAW.

  Dane obiektów LOB mogą być składowane wewnątrz bazy Oracle lub w zewnętrznym pliku danych.

- BFILE — dane tego typu działają jak wskaźniki do plików składowanych poza bazą Oracle, można je jedynie odczytywać, nie biorą udziału w transakcjach.
- XMLType — w kolumnach tego typu są przechowywane dokumenty XML. Typ pojawił się od wersji Oracle9i.

Do przechowywania i zarządzania danymi audio w Oracle służą specjalne typy obiektowe ORD np. ORDAudio, ORDImage, ORDVideo i in.

W Oracle występują też specjalne typy do przechowywania informacji geogeficznych dla systemów GIS np. SDO_GEOMETRY, SDO_GEORASTER i in.

3.1.5. Literały

Literały to konkretne wartości, które mogą być uwzględnione w poleceniu SQL. W Oracle są następujące rodzaje literałów:

- Boolowskie, np. FALSE.
- Tekstowe to ciągi znaków ujęte w apostrofy, np. 'Katowice'.
- Liczbowe, mogą to być liczby całkowite oraz rzeczywiste, które mogą być poprzedzone znakiem + lub – np. 15, +3.14, 0.5, 5e-03.
- Literały czasowe to mogą być wartości typu daty ze znacznikiem czasowym lub bez, wartości wskazujące tylko czas lub przedział czasowy.
- Słowo kluczowe DATE ustala domyślny format daty *rrrr-mm-dd* dla literału czasowego, podanego jako łańcuch znaków np.:

  DATE '2014-01-01'

- Słowo kluczowe TIME ustala format czasu *gg:mm:ss:[nn]* z dokładnością (opcjonalnie) do ułamkowych części sekundy dla wartości podanych jako ciąg znaków, np.: TIME '12:15:10:20'
- Słowo kluczowe INTERVAL ustala interpretację wartości podanych po słowie kluczowym jako przedział czasowy, np.:

  INTERVAL '1' YEAR, INTERVAL '1' MONTH, INTERVAL '1'DAY,
  oznaczają kolejno: 1 rok, 1 miesiąc oraz 1 dzień,
  INTERVAL '1-6' YEAR TO MONTH oznacza półtora roku.

3.2. Wartość NULL

NULL nie reprezentuje żadnej wartości, oznacza jej brak. NULL może wystąpić w kolumnie dowolnego typu. Gdy w kolumnie tabeli obligatoryjnie muszą znajdować się wartości, wówczas musi zostać zdefiniowane ograniczenie NOT NULL w tej kolumnie. W takim wypadku pominięcie tej kolumny przy próbie wstawienia wiersza do tabeli nie powiedzie się, lecz zostanie zgłoszony błąd.

Ze względu na występowanie wartości NULL SQL stosuje trójwartościową logikę. Tradycyjne porównania oraz warunki definiowane w poleceniach SQL mogą mieć

wartość prawdy (TRUE), fałszu (FALSE) lub mogą być nieokreślone, jeśli występuje w nich kolumna zawierająca wartość NULL.

3.3. Operatory

Wyróżniamy następujące operatory:

- operatory arytmetyczne: +, –, *, /,
- operatory porównań: <, <=, >, >=, =, <>, !=,
- operatory logiczne: NOT, AND, OR,
- operator konkatenacji: ||,
- operator zakresu: Z [NOT] BETWEEN x AND y, sprawdza, czy wartość Z mieści się (lub nie mieści się) w przedziale domkniętym <x,y>,
- operator wzorca: x [NOT] LIKE y, sprawdza, czy łańcuch znaków x odpowiada (lub nie odpowiada) wzorcowi y,
- operator testowania wartości NULL: x IS [NOT] NULL, sprawdza, czy x zawiera (lub nie zawiera) NULL,
- operator przynależności do listy: x [NOT] IN (x1, x2, x3) , sprawdza, czy wartość x występuje w liście wartości.

3.4. Wstawianie komentarzy do instrukcji SQL

Do poleceń SQL można dodawać komentarze, które są pomijane podczas ich wykonywania. Możemy to zrobić na dwa poniższe sposoby:

- Znakami – – (podwójny myślnik) poprzedza się komentarz umieszczony w jednej linii,
- Pomiędzy znaki */* *komentarz* */* (prawy ukośnik (ang. *slash*) oraz gwiazdka) wstawia się komentarz dłuższy, wpisany w więcej niż jednej linii.

Np.:

```
-- ta instrukcja wybiera dane osób bez telefonu – komentarz jednowierszowy,
SELECT * FROM PRACOWNICY
WHERE TELEFON IS NULL;
/* komentarz wielowierszowy: w poleceniu warunek zapisany jest słownie, z operatorem IS NULL, by
wybrać wiersze, w których nie podano numeru telefonu */
```

3.5. Operacje algebry relacji

Algebra relacyjna jest zbiorem 8 operacji, w których argumentem jest jedna lub więcej relacji, natomiast w wyniku jest uzyskiwana jedna relacja.

Operacje te można przyporządkować do dwóch grup:

- działania na zbiorach: suma, przecięcie, różnica,
- specjalne operacje relacyjne: selekcja (wybór), projekcja (rzut), iloczyn kartezjański, iloraz, złączenie.

Każda relacja (tablica) musi mieć swoją nazwę i spełniać założenia Codda.

W bazie szkoleniowej są stosowane m.in. następujące tablice:

```
PRACOWNICY
(IDENTYFIKATOR      CHAR(5),
NAZWISKO                CHAR(20),  IMIĘ
CHAR(20),
KOD DZIAŁU          CHAR(2),
DATA URODZENIA      DATE,
 CZAS PRACY             NUMBER(38),
STAWKA                  NUMBER(38),
 UBEZPIECZENIE     CHAR(3),
KOMENTARZ               VARCHAR2(70));

ADRESY
(IDENTYFIKATOR    CHAR(5),
ULICA                   VARCHAR2(20),
MIASTO                  VARCHAR2(20),
WOJEWODZTWO             VARCHAR2(20),
KOD                     CHAR(6),
TELEFON                 CHAR(11));

DZIAŁY
(KOD DZIAŁU             CHAR(2),
NAZWA DZIAŁU            CHAR(20));

KIEROWNICY
(KOD DZIAŁU             CHAR(2),
IDENTYFIKATOR     CHAR(5));
```

Rozdział 4. Język zapytań

DQL — polecenie SELECT

Instrukcja SELECT stanowi oddzielny język zapytań do bazy DQL (ang. *Data Query Language*), choć najczęściej jest zaliczana do języka manipulowania danymi DML (ang. *Data Manipulation Language*). Służy do wybierania informacji z bazy i umożliwia wykonywanie wymienionych wyżej operacji relacyjnych. Może być stosowana jako samodzielna instrukcja lub włączona do innych poleceń, takich jak np. INSERT, UPDATE lub DELETE.

Składnia polecenia SELECT może zawierać następujące klauzule:

```
SELECT * | {[DISTINCT] nazwa_kolumny [, nazwa_kolumny,...] |
                    [nazwa_tablicy.] nazwa_kolumny... [, wyrażenie [alias]]}
FROM tabela | lista tabel
[WHERE warunek_selekcji_wierszy]
[GROUP BY  {[nazwa_tablicy.] nazwa_kolumny, ...}
[HAVING  warunek_selekcji_grup ]]
[{UNION [ALL]  |  INTERSECT  |  MINUS }]
[ ORDER BY {nazwa_kolumny [ASC | DESC ] [,nazwa_kolumny...]};
```

Zapytania są najczęstszymi poleceniami uruchamianymi przez użytkownika w trakcie eksploatacji bazy. Zawartość instrukcji SELECT różni się w zależności od realizowanego zadania. W dalszej części opracowania zostanie omówiona składnia tego polecenia w kontekście wykonywanych operacji relacyjnych.

4.1. Projekcja

Najkrótsze samodzielne zdanie SQL zawiera dwie klauzule — SELECT oraz FROM — i kończy się średnikiem.

- W klauzuli **SELECT** określa się, które dane mają być uwzględnione w wyniku. W tym celu można użyć gwiazdki * lub wymienić listę oczekiwanych atrybutów, rozdzielonych przecinkami. Gwiazdka zastępuje nazwy wszystkich kolumn, a wyświetlają się one zgodnie z kolejnością ustaloną podczas definiowania

tablicy.

- Podając w poleceniu listę kolumn, ustalamy porządek ich wyświetlania. Takie polecenie realizuje **operację projekcji** (**rzutowania**). Poprzez wybranie wartości wskazanych kolumn jest tworzony pionowy podzbiór tabeli. Ta sama kolumna może być wielokrotnie użyta w poleceniu.
- W klauzuli **FROM** wskazujemy tablicę lub kilka tablic z danymi źródłowymi, których dotyczy zapytanie.

Aby zwiększyć czytelność zapisu polecenia, można używać spacji, tabulatorów, przejścia do nowej linii. Zazwyczaj słowa kluczowe języka SQL zapisuje się dużymi literami, a każda klauzula zaczyna się od nowego wiersza.

Przybliżą to przykłady zamieszczone w dalszej części opracowania.

Przykład 4.1.

Przegląd zawartości całej tabeli.

```
SELECT * FROM PRACOWNICY;
```

Przykład 4.2.

Sporządź listę osób zarejestrowanych w tablicy Pracownicy — projekcja, wybieranie wartości określonych atrybutów z tabeli.

```
SELECT NAZWISKO, IMIĘ
FROM PRACOWNICY;
```

Przykład 4.3.

Wymień działy, w których pracownicy są zatrudnieni— projekcja oraz eliminowanie powtarzających się wartości.

```
SELECT DISTINCT "KOD DZIAŁU" FROM
PRACOWNICY;
```

W wyniku każdy dział będzie uwzględniony jednokrotnie.

- Klauzula **DISTINCT** występuje w poleceniu zaraz po klauzuli SELECT i może być użyta tylko jeden raz. Spowoduje wybieranie z tabeli tych wierszy, które zawierają unikalne wartości w wyświetlanych kolumnach.

Przykład 4.4.

Sporządź alfabetycznie uporządkowaną listę osób — projekcja i sortowanie zbioru wynikowego.

```
SELECT NAZWISKO, IMIĘ FROM
PRACOWNICY
ORDER BY NAZWISKO;
```

Polecenie powyższe utworzy alfabetyczną listę osób (uporządkowanie od A do Z).

♦

Ostatnia w poleceniu klauzula **ORDER BY** odpowiada za uporządkowanie wierszy w zbiorze wynikowym. Pominięcie tej klauzuli powoduje wyświetlanie danych wynikowych zgodnie z kolejnością ich wprowadzania do bazy — wówczas obowiązuje uporządkowanie według wartości klucza głównego tablicy. W klauzuli ORDER BY można wskazać dowolny atrybut lub atrybuty typu liczbowego, znakowego, daty lub logiczne, według wartości których zbiór wynikowy będzie uporządkowany. Są dwa możliwe kierunki porządkowania określone słowami kluczowymi uzupełniającymi definicję sortowania — ASC (od ang. *ascend*) oznacza kierunek rosnący oraz DESC (od ang. *descend*) oznacza kierunek malejący. Domyślnie porządek sortowania jest rosnący (ASC nie trzeba deklarować). Wróćmy do przykładu — jeśli nazwiska mają być posortowane malejąco (od Z do A), należy zadeklarować to jawnie: ORDER BY NAZWISKO DESC. Jeśli sortowanie odbywa się według atrybutu, który w niektórych wierszach zawiera wartości NULL, to przy rosnącym porządku sortowania takie wiersze będą uwzględnione na początku zbioru wynikowego. Klauzula ORDER BY występuje na końcu polecenia. Sortowanie znacznie obciąża serwer bazy, dlatego nie należy nadużywać tej operacji w nieuzasadnionych wypadkach.

Przykład 4.5.

Przygotuj listę osób i uporządkuj wynik według danych osobowych i stawki — projekcja i sortowanie danych według wartości kilku atrybutów.

```
SELECT NAZWISKO, IMIĘ
FROM PRACOWNICY
ORDER BY NAZWISKO, IMIĘ, STAWKA;
```

W wypadku sortowania według kilku atrybutów porządkowanie zaczyna się od pierwszej kolumny wymienionej w klauzuli ORDER BY. Jeśli w kilku wierszach wynikowych wartości tej kolumny się powtarzają, to te wiersze podlegają uporządkowaniu według wartości kolejnego atrybutu. W przykładzie, jeśli w tablicy Pracownicy będą osoby o powtarzających się nazwiskach i imionach, to rekordy zawierające takie dane zostaną dodatkowo uporządkowane według stawki.

Kolejne przykłady prezentują, jak można zmienić sposób wyświetlania wyniku zapytania. W bardziej rozbudowanej instrukcji SELECT, poza atrybutami, można uwzględniać literały i wyrażenia. Literały to stałe wartości typu znakowego lub daty zapisane w apostrofach oraz wartości numeryczne. Definiowane wyrażenia arytmetyczne mogą mieć dowolnie złożoną postać — ich wynik będzie wyświetlany w dodatkowej kolumnie obliczeniowej. Można również zmienić sposób wyświetlania danych pobranych z tabeli, używając do tego odpowiednich funkcji.

♦

Przykład 4.6.

Dla każdego pracownika wylicz jego premię, równą iloczynowi stawki godzinowej i czasu pracy — definiowanie wyrażenia arytmetycznego w instrukcji SELECT oraz deklarowanie nowej nazwy (aliasu) dla kolumny obliczeniowej.

```
SELECT NAZWISKO, IMIĘ, STAWKA * "CZAS PRACY"  PREMIA FROM
PRACOWNICY;
```

W wyniku, w trzeciej kolumnie PREMIA, będzie wyświetlana wartość wyrażenia.

Alias dla kolumny — to nowa nazwa, która pojawi się w nagłówku kolumny wyświetlanej w wyniku zapytania. Nową nazwę podajemy po definicji kolumny, oddzielając ją spacją lub dodatkowo po słowie kluczowym AS. Aliasy poprawiają czytelność danych zwracanych przez pytanie. Są szczególnie przydatne, gdy w poleceniu są definiowane złożone wyrażenia albo nazwy kolumn w projekcie tablicy są niezrozumiałe dla użytkownika. Poprzez alias można odwołać się do danej kolumny w innych klauzulach. W przykładzie powyżej można zdefiniować sortowanie wyniku według kolumny Premia przy użyciu polecenia ORDER BY PREMIA.

♦ **Nazwy obiektów** w bazie (np. nazwy tablic, kolumn i in.) możemy pisać w cudzysłowie — wówczas należy przestrzegać wielkości liter. Jeśli w nazwie występuje spacja, cudzysłów jest konieczny. Nazwy podane bez cudzysłowu będą zawsze wyświetlane dużymi literami, natomiast w nazwach podanych w cudzysłowie zachowana zostanie wielkość znaków. Maksymalna długość nazwy to 30 znaków.

Przykład 4.7.

Sporządź listę osób, wyświetlając imię i nazwisko w jednej kolumnie — zastosowanie operatora konkatenacji w instrukcji SELECT w celu wyświetlania w jednej kolumnie złączonych wartości atrybutów znakowych.

```
SELECT NAZWISKO ||'    '|| IMIĘ  OSOBY FROM
PRACOWNICY;
```

Powyższe polecenie zwróci nazwiska i imiona osób (oddzielone trzema spacjami) jako jeden ciąg znaków — jako wartości kolumny znakowej o nazwie OSOBY. W poleceniu użyto funkcji konkatenacji (CONCAT) oznaczonej skrótem ||. Konkatenacja jest omówiona poniżej, w rozdziale dotyczącym funkcji znakowych.

Przykład 4.8.

Wykonaj zestawienie zawierające dane osobowe pracowników i informacje o miejscu pracy, uzupełnione dodatkowym komentarzem, poprawiającym czytelność wyniku — uwzględnianie literałów znakowych uzupełniających wyświetlany wynik.

```
SELECT NAZWISKO ||' '|| IMIĘ OSOBA,
'pracuje w ' || "KOD DZIAŁU" DZIAŁ FROM
PRACOWNICY;
```

♦

W kolumnie wynikowej OSOBA będą uwzględnione dane osobowe — NAZWISKO i IMIĘ pracownika, zaś wprowadzona wartość: pracuje w pojawi się jako uzupełniający komentarz, razem z wartością atrybutu "KOD DZIAŁU", w kolumnie o nazwie DZIAŁ.

4.2. Selekcja

- **Selekcja** to operacja tworzenia poziomego podzbioru tablicy poprzez wybór rekordów, w których są spełnione określone warunki. Ta operacja nie ogranicza struktury zbioru wynikowego, zaś klauzula ma postać: SELECT *.

  W warunku selekcji zdefiniowanym w klauzuli **WHERE** mogą występować atrybuty, operatory, stałe, a nawet dowolnie złożone wyrażenia arytmetyczne.

- Pojedynczy warunek selekcji ma postać: *atrybut-operator-wartość*.
  Dodatkowo, każdy warunek może być zanegowany operatorem NOT. Bardziej złożone predykaty można definiować, używając spójników logicznych AND lub OR oraz NAWIASÓW. Operatory mają różne priorytety. Najwyższy priorytet mają operatory arytmetyczne (z wagami takimi jak w działaniach matematycznych) i negator (NOT), następnie operatory porównania, dalej operatory logiczne w kolejności: AND, OR. Warunki definiowane w klauzuli WHERE podlegają zasadom logiki boolowskiej — wynik zawsze ma wartość logiczną TRUE lub FALSE. Gdy wynik wyrażenia w danym wierszu przyjmuje wartość prawdy, jest on włączony do zbioru wynikowego, zaś w przeciwnym wypadku jest pomijany.

Należy pamiętać, że NULL wskazuje na brak konkretnej wartości atrybutu i nie podlega operacjom porównania. Do przetestowania wierszy tablicy pod kątem braku jakiejkolwiek wartości wybranego atrybutu służy warunek, w którym używa się wyrażenia IS NULL albo IS NOT NULL, aby wybrać wiersze, w których atrybut ma przypisaną wartość.

Przykład 4.9.

Wybierz pracowników z działu administracji o podanym kodzie 'AD'.

```
SELECT * FROM PRACOWNICY
WHERE "KOD DZIAŁU" = 'AD';
```

Zwróćmy uwagę, że operacja selekcji nie ogranicza struktury zbioru wynikowego, wyświetlając wszystkie kolumny tablicy z danymi źródłowymi. Najczęściej jednak w poleceniu SELECT wykonuje się jednocześnie operacje projekcji i selekcji. Poniższe przykłady prezentują wykonanie obu operacji w poleceniu oraz użycie operatorów IN i BETWEEN, umożliwiających zdefiniowanie złożonych warunków selekcji w skrócony

sposób.

♦

Przykład 4.10.

Wypisz pracowników z trzech wskazanych działów — sprawdzenie wielokrotnego, alternatywnego warunku selekcji, dodatkowo jest wykonywane rzutowanie na kolumny NAZWISKO i IMIĘ.

```
SELECT NAZWISKO, IMIĘ
FROM PRACOWNICY
WHERE "KOD DZIAŁU" IN ( 'AD', 'CH', 'TR' );
```

W wyniku są wyświetlone wiersze, w których jest spełniony warunek selekcji. Warunek jest prawdziwy, jeśli atrybut "KOD DZIAŁU" jest równy którejkolwiek wartości z listy. W taki sposób można zapisać złożony, alternatywny warunek dotyczący atrybutu typu znakowego, numerycznego lub daty.

Przykład 4.11.

Wypisz pracowników, mających stawkę godzinową pomiędzy 10 a 20 złotych — selekcja poprzez określanie zakresu wartości atrybutu.

```
SELECT NAZWISKO, IMIĘ
FROM PRACOWNICY
WHERE STAWKA BETWEEN 10 AND 20;
```

Zawarta w przykładzie instrukcja wybiera dane o osobach, których stawka godzinowa należy do przedziału domkniętego pomiędzy 10 a 20 złotych. Z operatorem BETWEEN możemy zapisać warunek dla atrybutu znakowego, numerycznego lub daty.

Przykład 4.12.

Wyszukaj informacje o osobach nie posiadających telefonu — testowanie wartości pustych.

```
SELECT NAZWISKO, IMIĘ
FROM PRACOWNICY
WHERE TELEFON IS NULL;
```

W warunku selekcji użyty operator IS do testowania wartości NULL może dotyczyć atrybutu dowolnego typu.

Przykład 4.13.

Wybierz osoby o imionach kończących się literą 'a' — testowanie, czy atrybut znakowy odpowiada zdefiniowanemu wzorcowi; warunek zapisany jest z operatorem LIKE.

```
SELECT NAZWISKO, IMIĘ FROM
PRACOWNICY
WHERE IMIĘ LIKE '%a';
```

Znaki % i _ (podkreślnik) to znaki specjalne. We wzorcu symbol % zastępuje dowolny ciąg znaków, znak _ zastępuje jeden konkretny znak w łańcuchu. Polecenie z przykładu pozwala wybrać kobiety (warunek sprawdza się tylko w wypadku imion polskich). Jest to bardzo uproszczony warunek, posiada bowiem kilka wyjątków.

Przykład 4.14.

Wyszukaj osoby o 6-znakowych nazwiskach kończących się na litery 'ski'.

```
SELECT NAZWISKO, IMIĘ
FROM PRACOWNICY
WHERE NAZWISKO LIKE '___ski';
```

We wzorcu są uwzględnione trzy podkreślniki i znaki ski.

A co, jeśli musimy znaleźć łańcuch znaków, w którym występuje znak specjalny? Rozwiązaniem jest tzw. znak ucieczki, definiowany za pomocą słowa kluczowego ESCAPE. Jest to znak, który — użyty we wzorcu — powoduje, że wymieniony po nim znak specjalny jest traktowany jak każdy inny zwykły znak.

Przykład 4.15.

Wyświetl te wiersze z tablicy Pracownicy, w których w kolumnie Komentarz występuje znak '%' — zastosowanie słowa ESCAPE do zdefiniowania znaku ucieczki i wyszukania w łańcuchu znaku specjalnego %.

```
SELECT NAZWISKO, IMIĘ
FROM PRACOWNICY
WHERE KOMENTARZ LIKE '%!%%'  ESCAPE '!';
```

Zostaną wyświetlone dane dotyczące tych osób, które w komentarzu będą miały znak %.

Sytuacja trochę się komplikuje, jeśli zamierzamy wyszukać łańcuch znaków zawierający znak specjalny (%) i znak ucieczki (!), np. znaki abc%! w tekście komentarza. Rozwiązanie przedstawia kolejny przykład.

Przykład 4.16.

Wyświetl te wiersze, które w komentarzu mają ciąg znaków abc%! — zastosowanie znaku ucieczki we wzorcu.

```
SELECT NAZWISKO, IMIĘ
FROM PRACOWNICY
WHERE KOMENTARZ LIKE '%abc!%!!%'  ESCAPE '!';
```

4.3. Stosowanie wyrażeń, operatorów i funkcji w instrukcji SELECT

W poleceniu SELECT jest często wykonywane wyliczenie wartości, których nie przechowuje się w bazie. Przykładami takich operacji może być np. zaokrąglenie obliczonej pensji z ustaloną dokładnością, odczytanie roku z daty urodzenia czy zastąpienie wszystkich znaków w nazwiskach dużymi literami itp. Funkcje realizujące takie zadania operują na wartościach atrybutów znajdujących się w tym samym rekordzie, dla którego ustalają nowe wartości.

Ze względu na ten zakres działania nazywają się funkcjami wierszowymi. Wyróżnia się *funkcje wierszowe* znakowe, numeryczne, operujące na datach, na interwałach czasowych, funkcje konwersji i inne, ich nazwy są związane z typami danych, na jakich

działają.

Funkcje znakowe pobierają jako parametr ciąg lub ciągi znaków i zwracają inny ciąg znaków (np. zastępując wielkie litery w łańcuchu małymi) lub wartości liczbowe (np. obliczając długość łańcucha).

Funkcje znakowe zwracające wynik tekstowy to m.in.:

- CONCAT(*tekst1,tekst2*) — łączy łańcuchy znaków podane jako argumenty *tekst1* i *tekst2* w jeden ciąg.
- INITCAP(*tekst*) — zamienia pierwszą literę w tekście na wielką, pozostałe na małe.
- LOWER(*tekst*) — zamienia w tekście wszystkie litery na małe.
- LPAD(*tekst1,n[tekst2]*) — uzupełnia od lewej strony *tekst1* do długości łańcucha podanej w parametrze *n*, używając do tego celu znaków zawartych w *tekście2*. Jeśli *n* jest mniejsze od długości *tekstu1*, następuje obcięcie łańcucha *tekst1*. Analogicznie, RPAD uzupełnia ciąg *tekst1* od prawej strony.
  Łańcuch wynikowy w obu wypadkach ma długość *n*-znaków.
- LTRIM(*tekst1[,tekst2]*) — usuwa znaki w *tekst1* od lewej strony do pierwszego znaku nie należącego do *tekst2*. RTRIM natomiast wykonuje to przetwarzanie od prawej strony. Jeśli drugi argument zostanie pomięty, *tekst2* domyślnie oznacza spację. Funkcja TRIM() łączy działanie obu wymienionych funkcji. Funkcja TRIM(*atrybut*) jest stosowana w celu wyeliminowania nieznaczących spacji w kolumnie znakowej, wymienionej jako argument, eliminowanie spacji następuje z początku i z końca łańcucha znaków.
- REPLACE(*tekst, fragment1[,fragment2]*) — zamienia w *tekście* każdy *fragment1* na *fragment2*. Jeśli *fragment2* zostanie opuszczony, usunięty będzie *fragment1*.
- SUBSTR(*tekst,m[,n]*) — z łańcucha *tekst* wycina *n* kolejnych znaków, zaczynając od *m*-tej pozycji. Jeśli *n* jest pominięte, funkcja zwraca kolejne znaki od m-tej pozycji do końca łańcucha *tekst*.
- TRANSLATE(*tekst,zbiór1,zbiór2*) — zamienia znaki w *tekście* w oparciu o zależności między zbiorami znaków. Każde wystąpienie w *tekście* znaku ze *zbioru1* będzie zastąpione odpowiadającym mu znakiem ze *zbioru2*. Zazwyczaj zbiory 1 i 2 są równoliczne.
- UPPER(*tekst*) — zamienia w *tekście* wszystkie znaki na duże.

Funkcje znakowe zwracające wynik numeryczny to:

- ASCII(*znak*) — zwraca kod ASCII podanego znaku.
- INSTR(*tekst, fragment[,n[,m]]*) — poszukuje *m*-tego wystąpienia fragmentu w tekście, zaczynając od *n*-tego znaku. Zwraca pozycję pierwszego znaku znalezionego fragmentu w *tekście*.
- LENGTH(*tekst*) — zwraca długość *tekstu*.

Zastosowanie omówionych funkcji prezentują przykłady poniżej.

Przykład 4.17.

Zdefiniuj prosty warunek, który pozwoli wybrać osoby z działu administracji, niezależnie od wielkości znaków występujących w tej kolumnie — zastosowanie funkcji znakowej UPPER w warunku selekcji.

```
SELECT NAZWISKO, IMIĘ
FROM PRACOWNICY
WHERE UPPER("KOD DZIAŁU") ='AD';
```

Należy pamiętać, że wielkość liter w ciągach znaków wprowadzanych do bazy ma znaczenie, gdyż jest rozróżniana przez SZBD, dlatego w definiowaniu warunków dla pól znakowych należy używać np. funkcji UPPER, LOWER, INITCAP.

Jeśli w tablicy Pracownicy atrybut "KOD DZIAŁU" może przyjmować wartości np.: 'AD', 'Ad', 'ad', 'aD', to — chcąc wybrać pracowników tego właśnie działu administracji — należy w warunku podać listę wszystkich możliwych wartości (z operatorem IN) lub posłużyć się wybraną funkcją znakową, co przedstawia omawiany przykład.

Przykład 4.18. _____

Przygotuj zestawienie pracowników zawierające imię i nazwisko osoby, inicjały oraz obliczoną długość nazwiska — zastosowanie funkcji znakowych INITCAP(), SUBSTR(), CONCAT(), TRIM() i LENGTH() w klauzuli SELECT.

```
SELECT INITCAP(NAZWISKO) NAZWISKO,
INITCAP(IMIĘ) IMIĘ,
CONCAT(SUBSTR(IMIĘ,1,1),SUBSTR(NAZWISKO,1,1)) INICJAŁ,
LENGTH(TRIM(NAZWISKO)) DŁUGOŚĆ
FROM PRACOWNICY;
```

Pytanie wyświetli cztery kolumny — w dwóch pierwszych (z zadeklarowanymi nazwami NAZWISKO i IMIĘ) jest użyta funkcja INITCAP, która spowoduje, że wszystkie pierwsze znaki nazwiska i imienia będą wyświetlane jako duże, pozostałe zaś litery jako małe. W trzeciej kolumnie (nazwanej INICJAŁ) są wyświetlane inicjały każdego pracownika.

Zastosowanie aliasu do kolumn poprawia czytelność wyniku. Na przykład, jeśli w trzeciej kolumnie nie podamy aliasu, pojawi się w jej nagłówku pełna definicja kolumny, tzn. CONCAT(SUBSTR(IMIĘ,1,1),SUBSTR(NAZWISKO,1,1)).

Funkcja SUBSTR(IMIĘ,1,1) wybiera z łańcucha IMIĘ jeden znak, zaczynając od pierwszego znaku. Analogicznie, drugie użycie tej funkcji spowoduje wybranie pierwszego znaku z NAZWISKA. Dodatkowo jest w tej kolumnie zdefiniowana funkcja CONCAT, która łączy dwa łańcuchy znaków podane jako argumenty. W czwartej kolumnie (o nazwie DŁUGOŚĆ) jest wyświetlana liczba znaków w nazwisku pracownika — wartość ta jest wynikiem funkcji LENGTH(*argument*). Kolumna NAZWISKO ma typ CHAR(20), więc wprowadzane wartości są uzupełniane spacjami do stałej długości łańcucha. Do wyliczenia rzeczywistej długości nazwisk musi zostać dodatkowo użyta funkcja TRIM, która wyeliminuje spacje z lewej i prawej strony w łańcuchu. Funkcja TRIM(*tekst1[,tekst2]*) jest dwuargumentowa, *tekst1* — to tekst podlegający przetwarzaniu (w naszym wypadku to *NAZWISKO*), drugi argument *tekst2* — to znaki, które mają być wyeliminowane na początku i na końcu łańcucha wejściowego. Jeśli drugi argument jest pominięty, oznacza on domyślnie spację.

Otrzymany wynik jest prezentowany na rysunku 4.1.

Rysunek 4.1. *Wynik zastosowania funkcji znakowych INITCAP, SUBSTR, TRIM, CONCAT i LENGTH w zapytaniu*

	NAZWISKO	IMIE	INICJAL	DLUGOSC
1	Jacki	Tomasz	TJ	5
2	Górska	Hanna	HG	6
3	Padek	Paulina	PP	5
4	Binder	Julia	JB	6
5	Sanderska	Maria	MS	9
6	Elbaj	Klaudia	KE	5
7	Martecka	Joanna	JM	8
8	Binga	Alicja	AB	5
9	Czapski	Bogdan	BC	7
10	Obarski	Dominik	DO	7

Przykład 4.19.

Przygotuj zestawienie zawierające adresy osób — zastąp znaki / na znaki m. wyrażające skrót od „numer mieszkania” — zastosowanie funkcji znakowych REPLACE, INSTR, TRANSLATE w poleceniu.

```
SELECT ULICA,
REPLACE(ULICA,'/','m.'),    -- zamienia znak / na skrót m.
INSTR(ULICA,'Al.'),         -- zwraca miejsce wystąpienia Al. w ciągu
TRANSLATE(ULICA,'ar','XY'), -- zastępuje znak a znakiem X i znak r znakiem Y
TRANSLATE(ULICA,'arBk','XY') -- zastępuje znaki a i r (jak wyżej), znaki nie mające odpowiednika są
                                usuwane (B i k)
FROM ADRESY
ORDER BY ULICA;
```

Otrzymujemy wynik jak na rysunku 4.2.

ULICA	REPLACE(ULICA,'/','M.')		TRANSLATE(ULICA,'AR','XY')	TRANSLATE(ULICA,'ARBK'
Al. Ujazdowskie 4/6	Al. Ujazdowskie 4m.6	1	Al. UjXzdowskie 4/6	Al. UjXzdowsie 4
Batorego 5/90	Batorego 5m.90	0	BXtoYego 5/90	XtoYego 5/90
Bednarska Al. 2/78	Bednarska Al. 2m.78	11	BednXYskX Al. 2/78	ednXYsX Al. 2/78
Berlinga 2/6	Berlinga 2m.6	0	BeYlingX 2/6	eYlingX 2/6
Bracka 1/9	Bracka 1m.9	0	BYXckX 1/9	YXcX 1/9
Bracka 4/8	Bracka 4m.8	0	BYXckX 4/8	YXcX 4/8

Rysunek 4.2. *Wynik działania funkcji znakowych REPLACE, INSTR, TRANSLATE w poleceniu*

Przykład 4.20.

W liście nazwisk pracowników dokonaj zamiany pierwszych znaków w następujący sposób: znaki K zamień na L, A na B, natomiast znak S na W.

```
SELECT NAZWISKO,
TRANSLATE    ((INITCAP(NAZWISKO)),'KAS','LBW')    FROM
PRACOWNICY;
```

Funkcja INITCAP zapewni, że tylko pierwsze litery w nazwisku będą duże, wszystkie pozostałe zaś będą małe. Użycie tej funkcji zagwarantuje, że przeprowadzona zamiana będzie dotyczyć tylko pierwszych znaków łańcucha NAZWISKO. Następnie funkcja TRANSLATE zastąpi ustalone znaki innymi, zgodnie z ich wzajemnym przyporządkowaniem w zbiorach znaków podanych jako argumenty tej funkcji.

Przykład 4.21.

Wyświetl uporządkowane rosnąco według długości nazwiska pracowników, wyrównując je do prawej krawędzi kolumny NAZWISKO. Zastosowana kolumna ma mieć szerokość 30 znaków.

```
SELECT LPAD(TRIM(NAZWISKO),30,' ') AS NAZWISKO,
LENGTH(TRIM(NAZWISKO)) DLUGOSC
FROM PRACOWNICY
ORDER BY DLUGOSC;
```

Funkcja TRIM wyeliminuje obustronnie nieznaczące spacje w kolumnie NAZWISKO. Następnie funkcja LPAD uzupełni spacjami od lewej strony łańcuch NAZWISKO do długości 30 znaków. W drugiej kolumnie o nazwie DLUGOSC funkcja LENGTH zwróci rzeczywistą długość każdego nazwiska, działa bowiem na wyniku funkcji TRIM. Wynik jest sortowany rosnąco według długości nazwiska.

Przykład 4.22.

Wyświetl nazwy działów z tablicy DZIAŁY, wyśrodkowane w kolumnie o szerokości 20 znaków.

```
SELECT LPAD(TRIM("NAZWA DZIAŁU"),
(LENGTH(TRIM("NAZWA DZIAŁU")) +
(20 - LENGTH(TRIM("NAZWA DZIAŁU"))/2))) AS NAZWA_DZIAŁU
FROM DZIAŁY
ORDER BY LENGTH(TRIM("NAZWA DZIALU"));
```

Zawsze kiedy odwołujemy się do atrybutu, który ma typ znakowy stałej długości (w bazie ćwiczeniowej to wszystkie atrybuty znakowe: NAZWISKO, IMIĘ, "NAZWA DZIAŁU" itd.), stosujemy funkcję TRIM w celu wyeliminowania nieznaczących spacji. Funkcja LPAD uzupełnia od lewej strony spacjami otrzymaną nazwę do długości ustalonej wyrażeniem w nawiasie. Spacje mają wypełnić różnice pomiędzy szerokością kolumny a rzeczywistą długością łańcucha nazwy, symetrycznie dodając tę wyliczoną różnicę po połowie z każdej strony. Łańcuch spacji o wyliczonej długości jest dodawany z lewej strony. Z prawej strony będzie uzupełniony automatycznie przez system.

Funkcje numeryczne pobierają liczby jako parametr wejściowy i po przekształceniu zwracają liczbowy wynik (np. obliczając pierwiastek kwadratowy danej liczby). Przykładami takich funkcji są m.in.:

- ABS(*x*) — zwraca wartość bezwzględną argumentu.
- CEIL(*x*) — zwraca najmniejszą liczbę całkowitą większą lub równą podanej wartości, odpowiednio FLOOR(*x*) — zwraca największą liczbę całkowitą mniejszą lub równą podanej wartości.
- MOD(*x,y*) — zwraca resztę z dzielenia liczby *x* przez *y*.
- POWER(*x,y*) — zwraca liczbę *x* podniesioną do potęgi *y*.

- ROUND(*x[,n]*) — zaokrągla *x* z określoną dokładnością. Jeśli *n* jest pominięte, to zaokrągla do liczby całkowitej. Jeśli *n* jest liczbą dodatnią, to wartość jest zaokrąglona do *n*-miejsc dziesiętnych po przecinku, jeśli *n* jest liczbą ujemną, to zaokrąglenie jest do wskazanego rzędu wielkości, na lewo od przecinka.
- SIGN(*x*) — zwraca –1 dla $x < 0$, 0 dla $x = 0$ i 1 dla $x > 0$.
- SQRT(*x*) — oblicza pierwiastek kwadratowy liczby *x*.
- TRUNC(*x[,n]*) — obcina *x* z dokładnością do *n*-tego dziesiętnego miejsca po przecinku. Jeżeli *n* jest liczbą ujemną, funkcja zastępuje zerami *n* – 1 cyfr na lewo od przecinka.

Przykład 4.23.

Wylicz średnią stawkę pracowników (funkcja AVG(stawka) omówiona w rozdziale „Operacje grupowania") i porównaj zaokrąglenie i obcięcie wyniku do dwóch miejsc dziesiętnych — zastosowanie funkcji numerycznych TRUNC i ROUND w poleceniu.

```
SELECT AVG(STAWKA),      --wyliczona średnia
TRUNC(AVG(STAWKA),2),    --obcięta do 2 miejsc dziesiętnych
ROUND(AVG(STAWKA),2)     --zaokrąglona do 2 miejsc dziesiętnych
FROM PRACOWNICY;
```

Otrzymamy wynik, który został zaprezentowany na rysunku 4.3.

	AVG(STAWKA)	TRUNC(AVG(STAWKA),2)	ROUND(AVG(STAWKA),2)
1	16,819672131147540983606557377049180327 87	16,81	16,82

Rysunek 4.3. *Wynik zastosowania funkcji numerycznych TRUNC I ROUND w poleceniu*

Przykład 4.24.

Porównaj wynik zastosowania funkcji ROUND i TRUNC dla tego samego argumentu — liczby 12,56.

```
SELECT ROUND(12.56,1), ROUND(12.56), ROUND(12.56,-1),
TRUNC(12.56,1),  TRUNC(12.56),  TRUNC(12.56,-1)  FROM
DUAL;
```

Zależnie od wartości drugiego argumentu, który przy pierwszym użyciu każdej z tych funkcji ma wartość 1, przy drugim użyciu został pominięty, przy trzecim ma wartość –1, otrzymamy wartości w kolejnych kolumnach: 12,6, 13, 10, 12,5, 12, 10. Warto przeanalizować składnię tego polecenia.

Przykład 4.25.

Porównaj zastosowanie funkcji numerycznych FLOOR i CEIL dla tego samego argumentu — liczby 2.5.

```
SELECT FLOOR(2.5), CEIL(2.5)
FROM DUAL;
```

Funkcja FLOOR zwróci wartość 2 jako największą liczbę całkowitą mniejszą od podanej, natomiast funkcja CEIL zwróci wartość 3, jako najmniejszą liczbę całkowitą większą od podanej.

Przykład 4.26.

Porównaj zastosowanie funkcji numerycznych MOD(), POWER() i SQRT().

```
SELECT MOD(5,2), POWER(5,2), SQRT(5) FROM
DUAL;
```

W pierwszej kolumnie otrzymamy wynik funkcji MOD zwracającej resztę z dzielenia liczby 5 przez 2 — będzie to 1. W drugiej kolumnie funkcja POWER zwróci wynik podniesienia liczby 5 do potęgi 2, co jest równoważne działaniu 5 * 5, dając wynik 25. W trzeciej kolumnie funkcja SQRT zwróci pierwiastek kwadratowy liczby 5, czyli 2,236.

W trzech przykładach powyżej w klauzuli FROM występuje **tablica DUAL**. Jest to tablica systemowa, która nie przechowuje danych. Mają do niej dostęp wszyscy użytkownicy, a posiada ona tylko jedną kolumnę o nazwie DUMMY (typu VARCHAR2(1)) i jeden wiersz. Nie przechowuje na stałe żadnych wartości. Nie można wykonać na niej operacji DDL, DML.
Ta tabela jest konieczna ze względu na składnię poleceń Oracle.

W dialekcie SQL implementowanym w grupie ANSI, do której należy Oracle, jest wymagana klauzula FROM w każdym pytaniu — również wtedy, gdy nie korzysta się z żadnej tabeli przechowującej dane w bazie. Aby ustalić np. bieżącą datę, użytkownika sesji lub wynik innej funkcji systemowej czy następną wartość sekwencji, należy w poleceniu użyć pełnej składni, podając dwie klauzule SELECT i FROM. W tych wypadkach jako źródłowa będzie podana tablica DUAL. Zapytamy więc o datę bieżącą, używając polecenia: SELECT SYSDATE FROM DUAL;.

W innym dialekcie SQL, w grupie SYBASE, choć jest tam zaimplementowana ta sama wersja standardu SQL, klauzulę FROM stosujemy tylko w pytaniach pobierających dane z tablic z bazy. Datę bieżącą ustalamy przy użyciu polecenia SELECT NOW();.

Przechodząc do kolejnych funkcji związanych z datą i czasem, zwróćmy uwagę, że w rozmaitych Systemach Zarządzania Bazą Danych elementy czasowe są w różny sposób przechowywane. Podstawowy typ — to typ DATE.

W Oracle typ DATE przechowuje zarówno datę, jak i czas, z uwzględnieniem minut i sekund. Zakres dat to 1 stycznia 4712 r. p.n.e. do 31 grudnia 9999 r. n.e. Dodatkowy typ TIMESTAMP służy do przechowywania znaczników czasowych — daty i czasu. Nie ma zaimplementowanego typu TIME.

W standardzie SQL-99 typ DATE przechowuje tylko daty z dokładnością do dni. Inny jest również zakres przechowywanych dat — od 1 stycznia 1 r. n.e. do 31 grudnia 9999 r. n.e.. Oddzielnie jest zaimplementowany typ TIME, przechowujący czas z uwzględnieniem godzin, minut i sekund, z dokładnością do części ułamkowych

sekundy. Typ TIMESTAMP, podobnie jak w Oracle, jest przeznaczony do przechowywania znaczników czasowych daty i czasu.

Do zdefiniowania pytania uwzględniającego atrybuty typu data są używane również literały czasowe. Słowo kluczowe DATE służy do reprezentacji literałów typu DATE z domyślnym formatem rrrr-mm-dd. Słowo kluczowe TIME reprezentuje literał związany z formatem czasu gg:mi:ss, opcjonalnie mogą być też uwzględnione części dziesiętne sekundy. Słowo kluczowe INTERVAL służy do reprezentacji literałów typu interwał czasowy. Ustala, jaki przedział czasowy określa wartość podana po słowie kluczowym, np. INTERVAL '1' YEAR oznacza 1 rok, INTERVAL '1 12:30:10' DAY TO SECOND oznacza 1 dzień, 12 godzin, 30 minut i 10 sekund.

Daty są przechowywane w systemie jako liczby. Jest możliwe porównywanie dat oraz wykonywanie operacji arytmetycznych na datach. Te operacje prezentuje tabela 4.1, a dalej jest zamieszczony zestaw przykładów poleceń wykonujących działania na datach.

Przykład 4.27. _____

Wybierz osoby urodzone po 1 stycznia 1990 roku — wykonywanie porównania dat.

```
SELECT *
FROM PRACOWNICY
WHERE "DATA URODZENIA" > DATE '1990-01-01';
```

Przykład 4.28. _____

Wskaż datę przesuniętą o 1 rok względem podanej daty — dodawanie przedziałów czasowych do daty.

Tabela 4.1. *Operacje arytmetyczne na datach*

Działanie	Komentarz dot. wyniku
data + data	DZIAŁANIE NIEMOŻLIWE
data – data	różnica dat wyrażona w dniach
data + liczba	data przesunięta o liczbę dni w przyszłość
data – liczba	data przesunięta o liczbę dni w przeszłość
data + liczba/24	zwiększenie daty o liczbę godzin
*data + liczba/(24*60)*	zwiększenie daty o liczbę minut
*data + liczba/(24*60*60)*	zwiększenie daty o liczbę sekund
data+|- interwał czasowy	przesunięcie daty zależnie od drugiego składnika

```
SELECT DATE '2014-01-01' + INTERVAL '1' YEAR FROM DUAL;
```

Polecenie zwróci datę 1 stycznia 2015 r.

Przykład 4.29.

Wskaż datę przesuniętą o 1 miesiąc względem podanej daty — dodawanie przedziałów czasowych do daty.

```
SELECT DATE '2014-01-01' + INTERVAL '1' MONTH FROM
DUAL;
```

Otrzymamy wynik wskazujący na luty 2014 r.

Przykład 4.30.

Ustal łączny przedział czasowy dla lat i miesięcy — dodawanie przedziałów czasowych.

```
SELECT INTERVAL '10' YEAR + INTERVAL '20' MONTH FROM
DUAL;
```

Pytanie zwróci wynik — 11 lat i 8 miesięcy.

Przykład 4.31.

Ustal łączny przedział czasowy dla dni i godzin — dodawanie przedziałów czasowych.

```
SELECT INTERVAL '5' DAY + INTERVAL '40' HOUR FROM DUAL;
```

W wyniku otrzymamy przedział czasowy wynoszący 6 dni i 16 godzin.

Wykorzystując interwały czasowe, można wykonywać operacje na datach, dodając lub odejmując przedziały czasowe i daty — w wyniku uzyskujemy daty. Możemy również dodawać do siebie lub odejmować przedziały czasowe: miesiące do roku albo czas (godziny, minuty, sekundy) do dni, ale nie można np. dodawać godzin do miesięcy czy lat. Interwał może wskazywać wartość ujemną oznaczającą dany przedział czasowy wcześniej, np. INTERVAL'-5' oznacza wcześniejsze 5 dni. Sprawdźmy to, używając konkretnych poleceń.

Przykład 4.32.

Ustal łączny przedział czasowy dla „minionych" dni i następnych godzin — dodawanie przedziałów czasowych.

```
SELECT INTERVAL '-2' DAY + INTERVAL '50' HOUR FROM DUAL;
```

Wynik zwrócony przez pytanie to 2 godziny.

Przykład 4.33.

Ustal przedział czasowy dla dni i czasu — dodawanie przedziałów czasowych.

```
SELECT INTERVAL '1 12:30:10' DAY TO SECOND
+ INTERVAL '10' SECOND FROM
DUAL;
```

Aby możliwe było wykonanie tego działania ciąg znaków, musi zostać dokładnie zinterpretowany z wyróżnieniem dnia i znacznika czasowego, jako 1 dzień 12 godzin 30 minut i 10 sekund. Interwał musi być zdefiniowany tak jak w poleceniu, tzn. INTERVAL '1 12:30:10' DAY TO SECOND. Wówczas po dodaniu 10 sekund pytanie zwróci wynik — 1 dzień 12 godzin 30 minut i 20 sekund.

Niżej wymienione funkcje rozszerzają możliwości wykonywania operacji na datach, uzupełniając lub zastępując w niektórych zastosowaniach wykorzystanie interwałów czasowych. Operacje na datach są bardzo często wykorzystywane w tych systemach informatycznych, w których przetwarzane dane mają mocne uwarunkowania czasowe, np. w obsłudze kredytów, gdzie należy wyznaczyć datę kolejnej raty, uwzględniać datę rzeczywistej wpłaty, oceniać spóźnienie wpłaty, liczyć odsetki za określony czas czy też zdefiniować w procedurach zabezpieczenia, aby planowana wpłata nie była przewidziana np. na sobotę itp.

Funkcje operujące na datach przekształcają datę podaną jako parametr wejściowy na inną datę (np. wskazują datę ostatniego dnia miesiąca odpowiadającego dacie podanej jako argument) lub zwracają wartości liczbowe (np. liczbę miesięcy różniącą dwie daty).

- CURRENT_DATE, CURRENT_TIMESTAMP, SYSDATE — funkcje zwracają datę systemową, druga z nich zwraca również czas. Funkcja CURRENT_TIME nie jest zaimplementowana w Oracle.
- ADD_MONTHS(*data,n*) — zwraca datę przesuniętą o *n* miesięcy kalendarzowych w przyszłość. Drugi argument funkcji ADD_MONTHS może być również liczbą ujemną. Pozwala to przesuwać daty nie tylko w przyszłość, ale i wstecz.
- LAST_DAY(*data*) — zwraca datę ostatniego dnia miesiąca, w którym zawiera się podana *data*.
- NEXT_DAY(*data, dzień_tygodnia*) — zwraca najbliższą datę, jaka wypadnie we wskazany dzień tygodnia.
- MONTHS_BETWEEN(*data1,data2*) — zwraca liczbę miesięcy różniących dwie daty.
- EXTRACT(*składnik FROM data*) — to funkcja pozwalająca na wydobycie z *daty* określonego składnika: roku, miesiąca, dnia, godziny, minuty lub sekundy. Argumentem jest ten właśnie składnik — odpowiednio: YEAR, MONTH, DAY, HOUR, MINUTE lub SECOND, a po słowie FROM data wejściowa.
- ROUND(*data*) — funkcja zaokrągla datę do północy, jeśli jest czas przed południem, lub do północy dnia następnego, jeśli jest po południu. Po podaniu dodatkowego parametru zaokrąglenie może być do pełnego miesiąca lub roku.

Przykład 4.34.

Ustalenie daty bieżącej w Oracle:

```
SELECT SYSDATE FROM DUAL; lub
SELECT CURRENT_DATE FROM DUAL;
```

Przykład 4.35.

Wykonaj zestawienie zawierające informacje o osobach (Nazwisko i imię) oraz wieku wyrażonym w pełnych latach — zastosowanie funkcji numerycznej ROUND oraz funkcji MONTHS_BETWEEN w instrukcji SELECT.

```
SELECT NAZWISKO, IMIĘ,
ROUND(MONTHS_BETWEEN(SYSDATE,
"DATA URODZENIA")/12) WIEK FROM
PRACOWNICY;
```

Precyzyjnie ustalamy wiek poprzez wyliczenie liczby miesięcy, które upłynęły od daty urodzenia do dnia dzisiejszego (do daty bieżącej), następnie dzielimy liczbę miesięcy przez 12. Funkcja ROUND zaokrągla wynik do wartości całkowitej, zwracając wiek wyrażony w pełnych latach.

Przykład 4.36.

Wylicz wiek każdego pracownika — inny sposób wyliczenia wieku osoby poprzez zastosowanie funkcji EXTRACT.

```
SELECT NAZWISKO,IMIĘ,
(EXTRACT(YEAR FROM SYSDATE) -
EXTRACT(YEAR FROM "DATA URODZENIA")) WIEK FROM
PRACOWNICY;
```

Zapytanie zwraca wiek pracownika obliczony jako różnicę roku bieżącego wydobytego funkcją EXTRACT z daty systemowej i roku wydobytego z daty urodzenia.

Przykład 4.37.

Wylicz dokładny wiek pracowników, ustalając liczbę lat i miesięcy — zastosowanie interwałów czasowych.

```
SELECT NAZWISKO,IMIĘ,
((SYSDATE - "DATA URODZENIA") YEAR TO MONTH) WIEK FROM
PRACOWNICY;
```

Podany wiek będzie wyrażony w pełnych latach i miesiącach, jakie upłynęły od daty urodzenia do daty bieżącej.

Przykład 4.38.

Wylicz wiek pracownika, podając tylko liczbę pełnych lat — zastosowanie interwałów czasowych i funkcji EXTRACT.

```
SELECT NAZWISKO,IMIE,
EXTRACT(YEAR FROM (SYSDATE-"DATA URODZENIA")
YEAR TO MONTH) WIEK
FROM PRACOWNICY;
```

Różnica daty systemowej i daty urodzenia, wyrażona jako przedział czasu w latach i miesiącach, będzie ograniczona tylko do pełnych lat poprzez funkcję EXTRACT.

Przykład 4.39.

Wylicz, ile miesięcy upłynęło od początku XXI wieku — zastosowanie funkcji ROUND i MONTHS_BETWEEN.

```
SELECT ROUND(MONTHS_BETWEEN(SYSDATE,'2000/01/01'),0) FROM DUAL;
```

Drugi argument funkcji numerycznej ROUND może być pominięty (zero jest wartością domyślną) — wówczas następuje zaokrąglenie do wartości całkowitej.

Przykład 4.40.

Wyświetl datę bieżącą, datę najbliższego dnia tygodnia wskazanego jako argument — datę wypadającą w najbliższą sobotę — oraz datę ostatniego dnia bieżącego miesiąca — zastosowanie funkcji NEXT_DAY i LAST_DAY w poleceniu.

```
SELECT SYSDATE,NEXT_DAY(SYSDATE, 'SOBOTA'), LAST_DAY(SYSDATE) FROM
DUAL;
```

Przykład 4.41.

Sprawdź, ile dni ma luty w roku 2020 — zastosowanie funkcji LAST_DAY i EXTRACT.

```
SELECT EXTRACT(DAY FROM (LAST_DAY('2020/02/01'))) FROM
DUAL;
```

Funkcja LAST_DAY zwróci datę ostatniego dnia miesiąca lutego 2020 roku, z tej daty funkcja EXTRACT wydobędzie tylko dzień.

Przykład 4.42.

Ustal, jaka data wypadnie za 15 miesięcy — zastosowanie funkcji ADD_MONTHS.

```
SELECT ADD_MONTHS(SYSDATE,15)
FROM DUAL;
```

W wyniku uzyskamy datę oddaloną o 15 miesięcy od dnia bieżącego.

Przykład 4.43.

Ustal datę, jaka wypadnie po 100 dniach od bieżącego dnia.

```
SELECT SYSDATE + 100 FROM DUAL;
```

Ponieważ wewnętrznie daty są przechowywane jako liczby (dni), można wykonać działania — dodawania i odejmowania — przesuwając datę wprzód i w tył. Dodawana liczba rzeczywista będzie automatycznie zaokrąglana do liczby całkowitej.

Przykład 4.44.

Porównaj trzy różne rodzaje zaokrągleń daty systemowej.

```
SELECT SYSDATE, ROUND(SYSDATE),
ROUND(SYSDATE,'YEAR'),  ROUND(SYSDATE,'MONTH')  FROM
DUAL;
```

Wynik polecenia jest zaprezentowany na rysunku 4.4. O jakiej porze dnia było uruchamiane polecenie z przykładu?

Rysunek 4.4. *Różne rodzaje zaokrągleń daty*

	SYSDATE	ROUND(SYSDATE)	ROUND(SYSDATE,'YEAR')	ROUND(SYSDATE,'MONTH')
1	14/10/11	14/10/12	15/01/01	14/10/01

Funkcje konwertujące dokonują konwersji pomiędzy wartościami różnych typów. Są to m.in.

- CAST(*wartość AS typ*) — funkcja zdefiniowana w standardzie SQL-99. Pozwala konwertować *wartość* do docelowego typu podanego jako drugi parametr po słowie kluczowym AS.
- TO_CHAR(*wyrażenie[,format]*) — pozwala na konwersję wyrażenia, które może być datą lub liczbą, na ciąg znaków. Struktura parametru *[,format]*) zależy od typu *wyrażenia*. Jeśli format został pominięty, to długość tekstu wynikowego jest równa ilości znaków potrzebnych do zapisania liczby.
- TO_NUMBER(*tekst[,format]*) — dokonuje konwersji wartości tekstowej na liczbową według formatu.
- TO_DATE(*tekst[,format]*) — dokonuje konwersji wartości tekstowej na typ daty według formatu.

Nowy format daty uwzględnia zarezerwowane sekwencje znaków ograniczonych apostrofami. Najczęściej używane obrazy formatu wykorzystują następujące znaki:

- DD — numer dnia tygodnia,
- MM — numer miesiąca,
- YY lub YYYY — dwie ostatnie lub cztery cyfry roku,
- DAY lub Day — nazwa dnia tygodnia pisana dużymi literami lub tylko pierwszy znak duży, pozostałe małe,
- MONTH lub Month — nazwa miesiąca napisana dużymi literami lub tylko pierwszy znak duży,
- HH24 — godziny w systemie 24-godzinnym, ♦ MI lub mi — minuty.

Parametr formatujący liczby jest zbudowany z zarezerwowanych sekwencji znaków, najczęstsze to:

- 9 — oznacza jedną cyfrę, np. format 999 — zwraca cyfry,
- 0 — zero poprzedzające liczbę,

- , — (przecinek) oddziela cyfry na określonej pozycji, np. tysiące,
- . — (kropka) oddziela części dziesiętne,
- D — znak domyślnego separatora dziesiętnego,
- $ — ruchomy znak dolara, ♦ L — znak waluty lokalnej.

Przykład 4.45.

Wyświetl nazwisko pracownika i wyliczoną wysokość jego pensji w jednej kolumnie znakowej, wynik uzupełnij stałym komentarzem — wykorzystanie funkcji CAST i literałów znakowych w poleceniu.

```
SELECT NAZWISKO, 'otrzymuje'||' '||
CAST(STAWKA * "CZAS PRACY" AS VARCHAR2(20))
||  '  tygodniowo'  PENSJA  FROM
PRACOWNICY;
```

Wstawione stałe wartości znakowe poprawiają czytelność wyniku. Zastosowana funkcja CONCAT (oznaczona skrótem ||) pozwala na łączenie wielu łańcuchów znaków w jeden. W tym wypadku jest konieczna konwersja typu numerycznego wyliczonej pensji na typ znakowy — zastosowano do tego celu funkcję CAST.

Przykład 4.46.

Przeanalizuj zastosowanie funkcji TO_CHAR do konwertowania liczby 123.45 na napisy — tabela 4.2 prezentuje wyniki polecenia z różnymi formatami:

```
SELECT TO_CHAR(123.45, format) FROM DUAL;
```

Tabela 4.2. *Wyniki zastosowania różnego formatu do konwertowania liczby na znaki*

Format	Wynik	Komentarz
'99'	###	Zwrócony błąd, gdyż liczba zawiera więcej cyfr niż limit dopuszczony przez format.
'999'	123	Tak zdefiniowany format dotyczy części całkowitej liczby.
'9999'	123	Uwzględniona tylko część całkowita liczby.
'99999'	123	jw.
'099'	123	Nie będzie uzupełniania zerem, bo są przewidziane tylko trzy pozycje dla części całkowitej liczby.
'0999'	0123	Zero poprzedza liczbę.
'9990'	123	Nie będzie uzupełniania zerem.
'999.9900'	123.4500	Uzupełnianie zerami części dziesiętnych.
'999.99'	123.45	Kropka jako separator dziesiętny.
'999,99'	1,23	Przecinek na trzeciej pozycji.

'9,999'	123	Przecinek na czwartej pozycji pominięty.
'9,999.99'	123.45	Kropka jako separator dziesiętny.
'9,99.99'	1,23.45	Przecinek na określonej pozycji.
'$999.99'	$123.45	Znak dolara poprzedzający liczbę.
'999.99L'	123.45zł	Symbol lokalnej waluty pochodzi z parametru NLS_CURRENCY bazy.
'999D99'	123,45	W polskich ustawieniach narodowych domyślnym separatorem jest przecinek. Parametr jest ustawiony w NLS_NUMERIC_CHARACTER bazy.

Funkcja TO_CHAR wykonuje konwersje daty na tekst według podanego formatu.

Jeśli nie podano formatu, wynik jest wyświetlany zgodnie z formatem obowiązującym w sesji.

Można zmieniać kolejność członów daty, można ustalać własne separatory.

Można wybierać wartość numeryczną lub nazwę dla poszczególnych członów daty.

Do formatowania daty można używać różnych znaków alfanumerycznych.

Ponieważ format daty przechowuje też znacznik czasowy, można wybierać w formacie również godziny, minuty, sekundy.

Przykład 4.47. _____

Przeanalizuj zastosowanie funkcji TO_CHAR do konwertowania daty na napisy — tabela 4.3 prezentuje wyniki polecenia z różnymi formatami daty:

```
SELECT TO_CHAR (SYSDATE  [, format])  FROM DUAL;
```

Tabela 4.3. *Wyniki zastosowania różnego formatu do konwertowania daty na znaki*

Format	Wynik
nie podano	14/11/10
'DD-MM-YYYY'	10-11-2014
'DD-MON-YY'	10-LIS-14
'year/YYYY'	twenty fourteen/2014
'YYYY-MONTH-DAY'	2014-LISTOPAD-PONIEDZIAŁEK
'DD-MM-YYYY, DAY,HH24:MI:SS'	10-11-2014, PONIEDZIAŁEK,15:10:25
'YYMMDD'	141110
'MM.yy.DD'	11.14.10
'(YY)(MM)(DD)'	(14)(11)(10)
'<YY><MM><DD>'	<14><11><10>
'yy.mm.dd :hh :mi :ss'	14.11.10 :15 :10 :25

'(DD)(MM)(YY) <HH><MM>'	(10)(11)(14) <15><10>

Funkcja TO_DATE (tekst[,format]) dokonuje konwersji tekstu na datę według podanego formatu. Zasada formatowania jest identyczna jak w wypadku funkcji TO_CHAR.

Przykład 4.48.

Znajdź wśród pracowników osoby urodzone po roku 1969 — wykorzystanie funkcji konwertującej TO_DATE.

```
SELECT NAZWISKO,IMIE FROM
PRACOWNICY
WHERE "data urodzenia" <
TO_DATE('01-01-1970','DD-MM-YYYY');
```

Inne funkcje to:

- NVL(*wyrażenie1,wyrażenie2*) — jeśli pierwszy argument nie ma określonej wartości (NULL), to jest zwracana wartość *wyrażenia2*; w przeciwnym wypadku zwrócona będzie wartość *wyrażenia1*.
- NVL2(*wyrażenie1,wyrażenie2,wyrażenie3*) — jeśli *wyrażenie1* ma wartość różną od NULL, wówczas będzie zwrócona wartość *wyrażenie2*; w przeciwnym wypadku — *wyrażenie3*.
- GREATEST(*w1,w2...*) — zwraca największą wartość z listy swoich argumentów, natomiast LEAST(*w1, w2...*) — zwraca najmniejszą wartość z listy.
- USER — funkcja zwraca nazwę użytkownika aktualnej sesji.
- UID — funkcja zwraca liczbę identyfikującą użytkownika aktualnej sesji.

Przykład 4.49.

Wybierz wszystkie wiersze z tablicy ADRESY. W kolumnie Nr_telefonu wyświetl numer telefonu lub informację słowną BRAK, jeśli nie będzie danych do wyświetlenia. W poleceniu zastosowana jest funkcja NVL, która zamienia puste wartości atrybutu na podaną wartość zastępczą. Typ wprowadzanej wartości musi być zgodny z typem atrybutu.

```
SELECT ADRESY.*,NVL(TELEFON,'BRAK') Nr_TELEFONU FROM
ADRESY;
```

Przykład 4.50.

Ustal użytkownika sesji i jego identyfikator systemowy.

```
SELECT USER, UID FROM DUAL;
```

4.4. Złączenia tabel

Złączenie tabel oznacza zapytanie łączące rekordy z jednej lub wielu tablic lub widoków. W klauzuli SELECT mogą pojawić się dowolne atrybuty tabel wymienionych w klauzuli FROM, natomiast w klauzuli WHERE są zawarte warunki złączenia.

W bazie danych Oracle występują następujące złączenia:

- *Równościowe* — warunek złączenia zawiera znak równości,
- *Nierównościowe* — w warunku nie występuje znak równości,
- *Naturalne* — kolumny złączenia mają takie same nazwy (NATURAL JOIN),
- *Wewnętrzne* — zwracające te dane z tablic, które mają swoje odpowiedniki w drugiej łączonej tablicy i spełniają zadeklarowany warunek złączenia (INNER JOIN),
- *Zewnętrzne* — zwracające — oprócz rekordów takich jak w wyniku złączenia wewnętrznego — również rekordy z jednej tablicy, nie mające swoich odpowiedników w drugiej tablicy (OUTER JOIN). Złączenie zewnętrzne występuje jako: lewostronne, prawostronne i pełne, czyli obustronne (LEFT JOIN, RIGHT JOIN i FULL JOIN, nie jest konieczne używanie pełnego operatora złączenia, odpowiednio — LEFT OUTER JOIN, RIGHT OUTER JOIN itd.),
- *Krzyżowe*, *krosowe* lub *kartezjańskie* — tworzące w wyniku iloczyn kartezjański tablic (CROSS JOIN),
- *Złączenia oparte na podzapytaniu*,
- *Samozłączenia* — łączenie tablic samych ze sobą.

W jednym zapytaniu można zdefiniować różne rodzaje złączeń.

Złączenia są wykonywane w zapytaniach udostępniających dane z więcej niż jednej tabeli. Dane z poszczególnych tabel są ze sobą łączone przez porównanie wartości wybranych kolumn występujących w warunku złączenia umieszczonym w klauzuli WHERE.

Przykład 4.51. _____

Wyświetl w zapytaniu dane osobowe pracownika i jego adres — zdefiniuj złączenie tabel PRACOWNICY i ADRESY.

Kolumnami łączącymi obie tablice będą IDENTYFIKATORy. W obu tablicach kolumny mają te same nazwy, ale nie jest to konieczne — można łączyć tablice, uwzględniając kolumny o różnych nazwach (jest tylko wymagana zgodność typów danych). Dla jednoznacznego wskazania kolumny uwzględnionej w poleceniu należy kwalifikować nazwy atrybutów, poprzedzając je nazwami tablic i rozdzielając nazwy kropką — (np. PRACOWNICY.IDENTYFIKATOR).

```
SELECT NAZWISKO, IMIĘ, ADRESY.*
FROM PRACOWNICY, ADRESY
WHERE PRACOWNICY.IDENTYFIKATOR =
ADRESY.IDENTYFIKATOR
ORDER BY NAZWISKO;
```

Przykład 4.52.

Wyświetl dane osobowe i adresowe pracownika, zastosuj w pytaniu aliasy do tablic.

W zapytaniu można zadeklarować **aliasy do tablic**, podając w klauzuli FROM po nazwie tablicy (po spacji) nową nazwę, która może być ograniczona do jednego znaku. Nowa nazwa tablicy obowiązuje w całym pytaniu. Obrazuje to poniższy przykład:

```
SELECT NAZWISKO, IMIĘ, A.* FROM
PRACOWNICY P, ADRESY A
WHERE P.IDENTYFIKATOR = A.IDENTYFIKATOR ORDER
BY NAZWISKO;
```

Pierwszym etapem wykonania tego złączenia jest utworzenie iloczynu kartezjańskiego, a następnie selekcja zgodnie z warunkiem umieszczonym w klauzuli WHERE. Powyższe pytanie zwróci uporządkowaną alfabetycznie listę pracowników z adresami. Udostępnimy w ten sposób nazwiska i imiona razem z wszystkimi danymi adresowymi pracownika, zestawiając dane z dwóch tablic razem, jeśli wartości pola Identyfikator przypisanego danym osobowym i adresowym są takie same. W wyniku tego zapytania będą udostępnione tylko spójne dane z obu tablic, tzn. będą to informacje o osobach zarejestrowanych w tablicy Pracownicy, które mają swoje dane w tablicy Adresy. Zostaną pominięte osoby bez adresu oraz te adresy, które nie są przyporządkowane do konkretnej osoby.

Iloczyn kartezjański tablic zawiera wszystkie możliwe kombinacje wierszy złączonych tabel. Możemy go utworzyć, niepoprawnie definiując złączenie, zapominając o podaniu warunku złączenia lub w sposób niepełny definiując ten warunek, co ma miejsce najczęściej podczas łączenia więcej niż dwóch tablic.

Przykład 4.53.

Wykonaj iloczyn kartezjański tablic Pracownicy i Adresy.

```
SELECT NAZWISKO, IMIĘ, MIASTO
FROM PRACOWNICY, ADRESY ;
```

W wyniku uzyskamy wszystkie możliwe kombinacje wartości — każda osoba z tablicy Pracownicy będzie połączona z każdym adresem.

Przykład 4.54.

Wykonaj krosowe złączenie tablic. Utworzenie iloczynu kartezjańskiego dwóch tablic można uzyskać, deklarując jawnie ten typ złączenia.

```
SELECT NAZWISKO, IMIĘ, MIASTO
FROM PRACOWNICY CROSS JOIN ADRESY ;
```

Przykład 4.55.

Wykonaj złączenie czterech tablic z ćwiczeniowej bazy — w wyniku mają być uwzględnione: Nazwisko i Imię z tablicy Pracownicy, Miasto z tablicy Adresy, identyfikator kierownika działu z tablicy Kierownicy (wyświetlony w kolumnie o zadeklarowanej nazwie Id_szefa) oraz "Nazwa działu" z tablicy Działy.

Z wszystkich tablic zostaną wybrane tylko spójne dane o pracowniku, miejscu zamieszkania, szefie i miejscu pracy.

```
SELECT NAZWISKO, IMIĘ,
MIASTO,
K.IDENTYFIKATOR  ID_SZEFA,  "NAZWA
DZIAŁU"
  FROM  PRACOWNICY P,  ADRESY
     A,
          KIEROWNICY K,
            DZIAŁY D
WHERE  P.IDENTYFIKATOR = A.IDENTYFIKATOR
  AND P. "KOD DZIAŁU" = K. "KOD DZIAŁU"
       AND P. "KOD DZIAŁU" = D. "KOD DZIAŁU" ORDER BY
NAZWISKO;
```

W wypadku niedostatecznie zdefiniowanego warunku złączenia w wyniku pojawi się iloczyn kartezjański wierszy. Podczas łączenia N tablic musi być N-1 poprawnie zdefiniowanych warunków złączenia.

Identyczne wyniki jak przy równozłączeniu uzyskujemy, wykonując **złączenie wewnętrzne tablic**.

Przykład 4.56.

Wykonaj zestawienie danych o pracownikach z ich adresami poprzez złączenie wewnętrzne dwóch tablic.

```
SELECT NAZWISKO, IMIĘ, A.*
FROM  PRACOWNICY P  INNER JOIN ADRESY A
ON P.IDENTYFIKATOR = A.IDENTYFIKATOR
ORDER BY NAZWISKO;
```

Przykład 4.57.

Wykonaj złączenie naturalne tablic Pracownicy i Adresy.

```
SELECT  NAZWISKO,  IMIĘ,  MIASTO,  ULICA
FROM  PRACOWNICY  NATURAL JOIN ADRESY
ORDER BY NAZWISKO;
```

Złączenie naturalne można zdefiniować, jeśli kolumna uwzględniona w warunku złączenia ma taką samą nazwę w obu tablicach. W składni polecenia można użyć również słowa USING specyfikującego nazwę kolumny złączenia. W zapytaniu operujemy prostymi nazwami atrybutów, bez kwalifikatorów.

Przykład 4.58.

Złączenie naturalne z użyciem frazy USING.

```
SELECT NAZWISKO, IMIE, MIASTO
FROM PRACOWNICY  JOIN ADRESY USING (IDENTYFIKATOR);
```

Przykładowe zapytania o numerach 4.51, 4.52 oraz 4.56 i 4.57 zwrócą te same wiersze. Warunek złączenia we wszystkich przykładach uwzględnia te same kolumny oraz znak równości. Wyświetlane wiersze wynikowe będą prezentować spójne dane z obu tablic.

Kolejne typy złączenia — **złączenia zewnętrzne** — pozwalają na wybranie z tablicy również niespójnych danych, czyli takich, które nie spełniają warunku złączenia. W złączeniu lewo- i prawostronnym jedna z tablic staje się nadrzędna i to z niej są wybierane wszystkie rekordy, natomiast z drugiej tablicy są dołączone rekordy spełniające warunek złączenia. W lewostronnym złączeniu zewnętrznym nadrzędną jest lewa tablica, czyli ta, która jako pierwsza jest zadeklarowana w klauzuli FROM. W prawostronnym złączeniu zewnętrznym są uwzględniane wszystkie wiersze z prawej tablicy (czyli drugiej w kolejności w zapytaniu) i uzupełniane wartości spełniające warunek złączenia z pierwszej

tablicy.

Wykonując ten typ złączenia, możemy sprawdzić, które dane są niespójne oraz które wiersze z danej tablicy nie spełniają warunku złączenia. W przykładowych tablicach możemy więc sprawdzić, którzy pracownicy nie mają swojego adresu, lub upewnić się, czy wszystkie adresy są przyporządkowane do konkretnych osób.

Przykład 4.59.

Wyszukaj w bazie osoby bez adresu — lewostronne złączenie zewnętrzne, sprawdzanie spójności danych w tablicach Pracownicy i Adresy. Obie tablice mają zadeklarowane aliasy.

```
SELECT PRACOWNICY.*
FROM PRACOWNICY P LEFT JOIN ADRESY A
ON  P.IDENTYFIKATOR =  A.IDENTYFIKATOR
WHERE A.IDENTYFIKATOR IS NULL;
```

Przykład 4.60.

Sprawdź, czy są w bazie adresy do usunięcia, niepowiązane z żadną osobą — prawostronne złączenie zewnętrzne.

```
SELECT ADRESY.*
FROM PRACOWNICY P RIGHT JOIN ADRESY A
ON  P.IDENTYFIKATOR = A.IDENTYFIKATOR
WHERE P.IDENTYFIKATOR IS NULL;
```

Złączenie prawo- i lewostronne można definiować, używając składni zawierającej znak (+). Jeśli przy atrybucie uwzględnionym w warunku złączenia pojawi się znak (+), to znaczy, że tabela, z której pochodzi ten atrybut, jest tabelą dołączaną, podrzędną.

Przykład 4.61.

Złączenie lewostronne ze znakiem (+) — w poniższym poleceniu otrzymujemy takie same wyniki jak przy lewostronnym złączeniu zewnętrznym.

```
SELECT NAZWISKO, IMIĘ, ADRESY.*
FROM PRACOWNICY, ADRESY
WHERE PRACOWNICY.IDENTYFIKATOR =ADRESY.IDENTYFIKATOR(+) ORDER
BY NAZWISKO;
```

Operatora (+) nie można użyć po obu stronach. Aby wybrać rekordy z obu łączonych tabel, niezależnie, czy spełniają warunek złączenia czy też nie, jest wykonywane pełne złączenie zewnętrzne przy użyciu operatora FULL JOIN.

Możemy uniknąć złączeń, stosując w zapytaniach uwzględniających dane z kilku tablic specjalne konstrukcje zdaniowe, definiujące **złączenia oparte na podzapytaniu**. Podzapytania są omówione w rozdziale 4.6.

Przykład poniżej przedstawia wyszukiwanie osób, które nie mają w tablicy Adresy swojego adresu. Choć do ustalenia wyniku musimy uwzględnić dane z dwóch tablic, nie łączymy ich w sensie dosłownym tak jak w poprzednich przykładach, a stosujemy dwa powiązane ze sobą pytania. Jako pierwsze zostanie wykonane podzapytanie (w nawiasie), które ustali wynik cząstkowy — zbiór identyfikatorów zarejestrowanych w tablicy adresy. Następnie pytanie główne, które generuje wynik końcowy, ustali zbiór osób, których identyfikatory nie występują w zbiorze wyników podzapytania.

Przykład 4.62.

Wyszukaj osoby bez adresu — wyszukiwanie niespójnych danych poprzez podzapytanie.

```
SELECT * FROM PRACOWNICY
WHERE IDENTYFIKATOR NOT IN
  (SELECT IDENTYFIKATOR
  FROM ADRESY);
```

W bazie danych Oracle można również definiować złączenia tabel oparte na warunku, który nie zawiera znaku równości. **Złączenia nierównościowe** są stosowane bardzo rzadko, najczęściej przy łączeniu tabeli z nią samą, wyniki takich złączeń zawierają mnóstwo powtarzanych danych. Jednym z nielicznych zastosowań tych złączeń jest analiza danych polegająca na wyszukaniu zależności między wierszami.

Samozłączenia są konieczne wówczas, gdy w jednym zapytaniu trzeba się kilkakrotnie odwołać do tej samej tabeli. Złączenia tabeli z nią samą są wykonywane w taki sposób, jak złączenia różnych tablic. Wszystkie operacje są wykonywane tak, jakby dotyczyły dwóch lub więcej identycznych tabel. Te tabele muszą być rozróżniane przez nazwę, dlatego koniecznie należy w pytaniu zadeklarować różne aliasy dla tablic i

kwalifikować nazwy wszystkich atrybutów. Tego typu złączenia są stosowane do rekurencyjnego odczytywania danych, np. o członkach rodziny, jeśli wszyscy członkowie rodziny zarejestrowani są w jednej tabeli. O rekursywnym powiązaniu wierszy świadczą klucze obce, które powstały przez powtórzenie klucza głównego tablicy w nowej roli. Klucz obcy wskazuje na rekord nadrzędny w tej samej tablicy. Przykładowe dane powiązane hierarchicznie w tablicy OSOBA prezentuje rysunek 4.5.

Przykład 4.63.

Wybierz dane o pracownikach i kierownikach działów, w których są zatrudnione poszczególne osoby — samozłączenie tablicy Pracownicy (należy dwukrotnie odwołać się do tej tabeli).

Tablica Pracownicy będzie źródłem danych o pracownikach (ma zadeklarowany alias P), użyta drugi raz (z aliasem Sz) udostępni informacje o kierownikach, którzy też przecież są pracownikami. Aby poprawnie przyporządkować dane „szefa" do „podwładnego", należy dodatkowo uwzględnić tablicę Kierownicy (z aliasem K) — tam mamy przypisany identyfikator kierownika konkretnemu działowi. Tablice są łączone według atrybutu "Kod działu".

```
SELECT P.NAZWISKO, P.IMIE,P."KOD DZIALU",
TRIM(Sz.NAZWISKO)||'  '||Sz.IMIE KIEROWNIK
FROM PRACOWNICY P,
PRACOWNICY Sz,
KIEROWNICY K
WHERE P."KOD DZIALU" = D."KOD DZIALU"
AND P. "KOD DZIALU" = K."KOD DZIALU";
```

Przykład 4.64.

Wybierz osoby z poszczególnych działów zarabiające więcej niż inni z danego działu — samozłączenie tablic Pracownicy oraz złączenie nierównościowe.

```
SELECT P.NAZWISKO, P."KOD DZIAŁU", P.STAWKA, D.NAZWISKO, D.STAWKA
FROM PRACOWNICY P, PRACOWNICY D
WHERE P."KOD DZIAŁU" = D."KOD DZIAŁU"
AND P.STAWKA * P."CZAS PRACY" >
D.STAWKA * D."CZAS PRACY";
```

Przykład 4.65.

Wyświetl informacje o każdej osobie (z tablicy OSOBY) i jej rodzicach — rekurencyjne odczytywanie danych z tablicy.

Do samozłączenia uwzględnijmy tabelę o podanej strukturze OSOBY(ID, NAZWISKO, IMIE, DATA_UR, ID_OJCA, ID_MATKI). W tym pytaniu musimy złączyć trzykrotnie tabelę OSOBY, raz — jako reprezentującą podstawowe informacje o osobie (tabela jest przemianowana na P), drugi raz — jako tabelę zawierającą informacje o ojcach — tabela O, oraz trzeci raz, tym razem tabela OSOBY przechowująca informacje o matkach,

jako tabela M. W wyniku wyświetlamy informacje — ID i Nazwisko, Imię i Datę urodzenia każdej osoby oraz Nazwiska, Imiona i Daty urodzenia matki i ojca.

```
SELECT P.ID,P.NAZWISKO P.IMIE,P.DATA_UR,
  M.NAZWISKO, M.IMIE,M.DATA_UR ,
  O.NAZWISKO, O.IMIE,O.DATA_UR
FROM OSOBY P, OSOBY M, OSOBY O
WHERE P.ID_OJCA=O.ID  AND  P.ID_MATKI=M.ID;
```

Zapytania hierarchiczne pozwalają na rekurencję w tablicach, gdzie występują hierarchiczne dane. Służą do wyświetlania informacji powiązanych, jak np. dane z drzewa genealogicznego o rodzicach i dzieciach czy te dotyczące podległości zawodowej pracowników itp.

Przykład 4.66.

Wyświetl informacje o dzieciach i wnukach wybranej osoby z bazy, np. pani Górskiej.

```
SELECT ID, ID_MATKI, NAZWISKO, LEVEL
FROM OSOBY
CONNECT BY PRIOR ID = ID_MATKI
START WITH NAZWISKO = 'Górska'
ORDER BY LEVEL;
```

Pseudokolumna LEVEL określa poziom rekurencji w drzewie hierarchii dla korzenia drzewa LEVEL=1.

Klauzula CONNECT BY określa sposób łączenia wierszy.

Operator PRIOR służy do odwoływania się do rodzica danego węzła.

Klauzula START WITH definiuje korzeń drzewa.

W poleceniu z przykładu 4.66 zaczynamy drzewo powiązań od osoby o nazwisku Górska (poziom 1). Na drugim poziomie będą informacje (m.in. ID, nazwisko) o dzieciach pani Górskiej, na trzecim poziomie dane o wnukach itd.

Zapytanie hierarchiczne w Oracle można również zapisać z rekurencyjną klauzulą WITH — otrzymamy wówczas identyczne wyniki, jak w poprzednim poleceniu. Klauzula WITH musi używać operacji UNION ALL i mieć listę aliasów kolumn.

Przykład 4.67.

Wyświetl informacje o potomkach wybranej osoby, np. P. Górskiej o identyfikatorze osoby EN03 — pytanie hierarchiczne z klauzulą WITH.

```
WITH
DZIECIEN03 (IDENTYFIKATOR, ID_M, NAZWISKO, POZIOM) AS
  (SELECT ID, ID_M, NAZWISKO, 1 POZIOM
  FROM OSOBA WHERE ID = 'EN03'
  UNION ALL
  SELECT O.ID ,O.ID_M, O.NAZWISKO, POZIOM+1
  FROM DZIECIEN03 S JOIN OSOBA O
```

```
    ON S.ID = O.ID_M)
    SELECT ID, ID_M, NAZWISKO, POZIOM    FROM
DZIECIEN03;
```

Przykładowo wybrane wiersze z tablicy OSOBA z atrybutami ID_O (identyfikator ojca) oraz ID_M (identyfikator matki) umożliwiajacymi określenie hierarchicznych powiązań między rekordami prezentuje rysunek 4.5.

Rysunek 4.5. *Wiersze w tablicy OSOBA zawierające informacje orodzicielstwie*

	IDENTYFIKATOR	NAZWISKO	IMIE	...	ID_O	ID_M
1	EN03	Górska	Hanna	...	(null)	(null)
2	EN13	Obarski	Dominik	...	EN12	EN03
3	EN19	Czekańska	Weronika	...	EN12	EN03
4	EN36	Celeborski	Adam	...	EN12	EN03
5	EN39	Elkora	Natalia	...	EN26	EN19
6	EN50	Cline	Dariusz	...	(null)	EN19

Wynik pytania hierarchicznego przedstawia tabela prezentowana na rysunku 4.6.

Rysunek 4.6. *Wynik pytania hierarchicznego*

	IDENTYFIKATOR	ID_M	NAZWISKO	POZIOM
1	EN03	(null)	Górska	1
2	EN13	EN03	Obarski	2
3	EN19	EN03	Czekańska	2
4	EN36	EN03	Celeborski	2
5	EN39	EN19	Elkora	3
6	EN50	EN19	Cline	3

Możemy również ustalić liczbę poziomów w hierarchii poprzez użycie funkcji COUNT(), korzystając z polecenia prezentowanego w przykładzie 4.68.

Przykład 4.68.

Wyświetl, ile pokoleń obejmuje drzewo genealogiczne wybranej osoby.

```
SELECT COUNT(DISTINCT LEVEL)
FROM OSOBA
START WITH identyfikator='EN03'
CONNECT BY PRIOR IDENTYFIKATOR = ID_M;
```

4.5. Operacje grupowania

Operacja grupowania jest wykonywana na wierszach spełniających warunek selekcji z klauzuli WHERE, ponieważ najpierw jest realizowana selekcja. W poleceniu występuje klauzula **GROUP BY**, która dokonuje partycjonowania zbioru wierszy na rozłączne podgrupy. Każdy wiersz w grupie ma te same wartości atrybutu będącego podstawą definicji grupy. W zapytaniu z grupowaniem są obliczane funkcje agregujące.

W tabeli wynikowej, zwanej tabelą zgrupowaną, każdy wiersz wyniku reprezentuje pojedynczą grupę, zaś każda wartość wyliczonej funkcji agregującej dotyczy danej grupy jako całości. Wartość NULL, jako wartość nieznana, jest ignorowana przez funkcje agregujące przy zliczaniu wartości kolumn.

Klauzula **HAVING** występuje w poleceniach po klauzuli GROUP BY. Pozwala ograniczyć zbiór wyników do tych grup, w których prawdziwy jest warunek zdefiniowany za pomocą dowolnej funkcji agregującej. Klauzula GROUP BY może wystąpić w poleceniu samodzielnie, bez HAVING.

Funkcje agregujące:

- COUNT(*) — zlicz,
- AVG(*expr*) — wartość średnia,
- MAX(*expr*) — wartość maksymalna,
- MIN(*expr*) — wartość minimalna,
- SUM(*expr*) — suma,
- MEDIAN(*expr*) — mediana,
- STDDEV(*expr*) — odchylenie standardowe, ♦ VARIANCE(*expr*) — wariancja.

Pierwsze pięć funkcji jest opisanych w standardzie ANSI, pozostałe, obecne we wszystkich implementacjach SQL, również są włączone do standardu, choć w poszczególnych implementacjach są używane pod różnymi nazwami.

Funkcja COUNT() zwraca liczbę rekordów spełniających warunek określony w klauzuli WHERE lub liczbę wierszy w grupie. Ta funkcja może przyjmować dwie postaci.

Jedna z postaci — COUNT(*) — zlicza wiersze zwracane przez zapytanie. W tym wypadku nie są sprawdzane wartości poszczególnych kolumn (czy zawierają wartości NULL, czy się powtarzają). Jeśli w tabeli nie ma danych lub pytanie nie zwraca żadnych wierszy, funkcja COUNT(*) przyjmuje wartość 0.

Druga postać tej funkcji — COUNT(*wyrażenie*) — zlicza każde wystąpienie, łącznie z powtórzeniami wartości atrybutu, wyrażenia wskazanego jako argument, pomijając wartości NULL. Domyślną opcją tej funkcji jest COUNT(ALL<*wyrażenie*>). Aby zliczyć unikatowe wartości zbioru <*wyrażenie*>, należy użyć następującej składni: COUNT(DISTINCT <*wyrażenie*>). Użycie słów kluczowych ALL|DISTINCT w funkcji COUNT jest identyczne jak w klauzuli SELECT.

Typem wartości funkcji COUNT(*) jest INTEGER, inne funkcje agregujące dziedziczą typ danych od wyrażeń, do których się odnoszą.

Funkcja AVG(*x*) oblicza średnią wartość atrybutu numerycznego, podanego jako argument tej funkcji. AVG(*x*) nie jest tym samym, co SUM(*x*)/COUNT(*), ponieważ funkcja SUM(*x*) odrzuca wartości NULL, a COUNT(*) tego nie robi.

Funkcje MAX(*x*) i MIN(*x*) działają ze wszystkimi typami danych, włącznie z napisami i datami. Zwracają odpowiednio najwyższą i najniższą wartość atrybutu numerycznego lub ciągi znaków o najwyższym i najniższym kodzie. W wypadku dat „maksymalną datą" jest najpóźniejsza data, „minimalną" — jest najwcześniejsza.

Funkcja SUM(*x*) zwraca w wyniku sumę wszystkich wartości atrybutu podanego jako argument.

Funkcja MEDIAN(*x*) zwraca medianę, czyli wartość środkową w zbiorze uporządkowanym, powyżej i poniżej której znajduje się tyle samo wartości.

Funkcja STDDEV(*x*) oblicza odchylenie standardowe x. Odchylenie standardowe informuje, jak „daleko" odbiegają wartości danej zmiennej od średniej wartości tej zmiennej. Funkcja ta zwraca wartość 0 dla zbioru składającego się z jednego elementu.

Funkcja VARIANCE(*x*) — to funkcja statystyczna, jest równa kwadratowi odchylenia standardowego. Wariancja zbioru liczb jest miarą zróżnicowania tego zbioru.

Użycie klauzuli DISTINCT we wszystkich powyższych funkcjach powoduje wyeliminowanie powtarzania wartości w agregowanych wyrażeniach.

Przykład 4.69.

Wylicz sumę pensji wszystkich pracowników — wyznaczenie funkcji agregującej (SUM) dla całej tabeli.

```
SELECT SUM(STAWKA * "CZAS PRACY" * 21/5) PENSJE_Razem FROM PRACOWNICY;
```

Przykład 4.70.

Wylicz sumę pensji w poszczególnych działach — wyznaczenie funkcji agregującej (SUM) dla grupy.

```
SELECT "KOD DZIAŁU",
SUM(STAWKA * "CZAS PRACY" * 21/5)
SUMA_PENSJI_W_DZIAŁACH
FROM PRACOWNICY
GROUP BY "KOD DZIAŁU";
```

Przykład 4.71.

Sprawdź, czy w trzech wybranych działach ('AD', 'CH' i 'EE') średnie stawki pracowników są większe niż 20 zł — wyznaczenie funkcji agregujących (COUNT, AVG, MAX) dla grupy łącznie z operacją selekcji.

```
SELECT "KOD DZIAŁU", COUNT(NAZWISKO),
  AVG(STAWKA) SREDNIA, MAX(STAWKA)
FROM PRACOWNICY
WHERE "KOD DZIAŁU" IN ('AD', 'CH', 'EE')
GROUP BY "KOD DZIAŁU"
HAVING AVG(STAWKA)>20
ORDER BY SREDNIA DESC;
```

W klauzuli HAVING użycie aliasów jest niedozwolone, natomiast w ORDER BY — aliasy są dozwolone (patrz przykład powyżej).

W powyższym przykładzie na początku wykonana jest selekcja z warunkiem zdefiniowanym w klauzuli WHERE. Zbiór wierszy podlegający dalszemu przetwarzaniu zostaje ograniczony do tych, w których warunek selekcji ma wartość prawdy. Wiersze dotyczące pracowników trzech wymienionych działów zostają przyporządkowane do grup utworzonych dla każdego działu oddzielnie. Następnie dla każdej grupy wyliczone są wartości funkcji COUNT(), AVG() i MAX(). Każda grupa jest reprezentowana przez jeden wiersz wynikowy. Na końcu wynik zostaje ograniczony do tych grup, w

których warunek zapisany w klauzuli HAVING jest prawdziwy. Wybrane są grupy (działy), w których średnia stawka jest większa od 20 zł.

Przykład 4.72.

Wykonaj zestawienie zawierające opis działu — KOD i NAZWĘ DZIAŁU oraz liczbę zatrudnionych osób w dziale — należy uwzględnić dwie złączone tablice PRACOWNICY i DZIAŁY (grupowanie na złączonych tablicach).

```
SELECT P."KOD DZIAŁU",
D."NAZWA DZIAŁU", COUNT(*) ILOSC_PRAC FROM
PRACOWNICY P, DZIAŁY D
WHERE P."KOD DZIAŁU" = D."KOD DZIAŁU"
GROUP BY P."KOD DZIAŁU",D."NAZWA DZIAŁU" HAVING COUNT(*)>5;
```

W zapytaniu z grupowaniem, poza funkcjami agregującymi, zbiór atrybutów wymienionych w SELECT może obejmować całą definicję grupy, czyli może być pełną listą atrybutów wymienionych w GROUP BY, może też być jej podzbiorem, ale nie odwrotnie. Nie można wyświetlać atrybutów spoza definicji grupy.

Rozszerzona specyfikacja grupowania

Frazy CUBE, ROLLUP, GROUPING SETS umożliwiają rozszerzoną specyfikację grupowania. Fraza ROLLUP powoduje wyliczenie wskazanych funkcji agregujących na różnych poziomach agregowania. CUBE rozszerza działanie frazy ROLLUP na wszystkie możliwe kombinacje poziomów grupowania. Fraza GROUPING SET pozwala na jawne definiowanie poziomów grupowania, eliminując inne zbędne poziomy i związany z tym nadmiar informacji.

Dodanie ROLLUP do GROUP BY spowoduje wyświetlenie podsumowania dla każdej grupy.

Przykład 4.73.

Ustal, z jakich miast pochodzą pracownicy poszczególnych działów — grupowanie z frazą ROLLUP.

Grupujemy wiersze ze względu na DZIAŁ i MIASTO. Jeśli w klauzuli GROUP BY używamy wielu wyrażeń (w naszym przykładzie to: P.”KOD DZIAŁU”,MIASTO), to zbiór wierszy dzielimy na grupy wyznaczone pierwszym wyrażeniem (P.”KOD DZIAŁU”), a w ramach grupy tworzymy podgrupy na podstawie wartości drugiego wyrażenia (MIASTO). Pytanie z grupowaniem bez frazy ROLLUP udostępni wartości funkcji agregujących wyliczone dla każdej podgrupy, w przykładzie będzie to informacja o osobach z poszczególnych działów i pochodzących z określonych miast. Dodanie do pytania frazy ROLLUP spowoduje wprowadzenie do wyniku dodatkowego wiersza z podsumowaniem dotyczącym grupy. W przykładowym pytaniu będzie to informacja dotycząca liczby zatrudnionych pracowników w każdym dziale niezależnie od miejsca zamieszkania.

```
SELECT MIASTO, P."KOD DZIAŁU", COUNT(*) FROM
PRACOWNICY P, ADRESY A
WHERE P.IDENTYFIKATOR = A.IDENTYFIKATOR
```

```
GROUP BY ROLLUP (P."KOD DZIAŁU", MIASTO);
```

Wynik tego pytania różni się od zwykłego grupowania dodatkowym wierszem podsumowującym grupy, w którym jest wyliczona funkcja agregująca dla podzbioru atrybutów definiujących grupy (rysunek 4.7). W naszym przykładzie jest to podsumowanie dotyczące działów. Mamy więc dodatkową informację, że w dziale AD jest tylko jeden pracownik mieszkający w Tarchominie, a w dziale CH jest ośmiu pracowników, po jednym z Wesołej, Pruszkowa, Legionowa i Piaseczna oraz czterech z Warszawy itd.

Rysunek 4.7.
Wynik użycia frazy ROLLUP w GROUP BY

	MIASTO	DZIAL	ILOSC_OSOB
1	Tarchomin	AD	1
2	(null)	AD	1
3	Wesoła	CH	1
4	Pruszków	CH	1
5	Warszawa	CH	4
6	Legionowo	CH	1
7	Piaseczno	CH	1
8	(null)	CH	8
9	Warszawa	EE	8
10	Zielonka	EE	1
11	Legionowo	EE	1
12	(null)	EE	10

Przykład 4.74. _____

Ustal, z jakich miast pochodzą pracownicy poszczególnych działów oraz ilu jest pracowników z poszczególnych miast — dodanie CUBE do GROUP BY.

Otrzymujemy wynik taki, jak poprzednio w wersji z ROLLUP oraz dodatkowe wiersze podsumowań dla kolejnego podzbioru atrybutów definiujących grupy i podsumowanie całości. Otrzymamy więc wynik zaprezentowany na rysunku 4.8.

Rysunek 4.8.
Wynik użycia frazy CUBE do GROUP BY

	MIASTO	DZIAL	ILOSC_OSOB
29	Warszawa	ZA	6
30	Wołomin	ZA	1
31	(null)	ZA	8
32	Warszawa	(null)	31
33	Pruszków	(null)	3
34	Wołomin	(null)	3
35	Wesoła	(null)	3
36	Tarchomin	(null)	2
37	Zielonka	(null)	1
38	Legionowo	(null)	3
39	Piaseczno	(null)	4
40	(null)	(null)	50

W wyniku pojawiły się dodatkowe wiersze podsumowujące grupy zdefiniowane na podstawie wartości drugiego wyrażenia w klauzuli GROUP BY, w przykładzie to atrybut MIASTO. Mamy więc dodatkową informację, ilu pracowników pochodzi z każdego miasta, niezależnie od miejsca zatrudnienia, oraz ostatni wiersz wskazujący, ilu jest wszystkich pracowników.

4.6. Podzapytania

Podzapytania mają taką samą składnię, jak pytania — są ujęte w nawias i umieszczone wewnątrz innego polecenia SQL. Stosuje się je wtedy, gdy w pytaniu chcemy odwołać się do wyników innego pytania. Podzapytanie może być umieszczone w klauzulach WHERE i HAVING (najczęstsze wypadki), a także w klauzulach SELECT i FROM (w specyficznych rozwiązaniach). W Oracle, w przeciwieństwie do Standardu SQL, nie zawsze podzapytanie jest umieszczone w nawiasie.

Ogólny schemat prostego pytania z podzapytaniem wygląda następująco (choć podczas szerszego omawiania tego tematu, przekonamy się, że może on podlegać znacznym modyfikacjom):

```
Zapyt.gł.      SELECT nazwy(a)_kolumn(y)
          FROM nazwa_tabeli              WHERE
nazwa_kolumny
          operator_porównania|operator_porównania_zbioru
Podzapytanie              (SELECT  nazwa_kolumny
FROM nazwa_tabeli
            [WHERE warunek])
```

Jeśli podzapytanie będzie zwracało pojedynczą wartość, może występować z operatorami arytmetycznymi: =, <, >, >=, <=, <>, natomiast jeśli będzie zwracało wiele wartości, musi wystąpić z operatorami porównania zbioru: IN lub NOT IN.

Wyróżniamy dwa rodzaje podzapytań: zagnieżdżone i skorelowane. Schemat zamieszczony powyżej uwzględnia oba typy podzapytań.

Podzapytania zagnieżdżone

W podzapytaniu zagnieżdżonym (zwanym zwykłym lub wewnętrznym) nie ma odwołań do atrybutów z zapytania głównego. Wynik takiego pytania nie zależy od zapytania głównego. Każde z pytań tak powiązanych jest wykonywane jednokrotnie, począwszy od najbardziej zagnieżdżonego. Wynik pytania jest uwzględniony w warunku kolejnego zapytania zagnieżdżonego lub zapytania głównego, które jest wykonywane na końcu.

Należy pamiętać, że liczba wartości ustalanych przez podzapytanie oraz ich typ muszą być zgodne z liczbą i typem atrybutów wymienionych w warunku selekcji zapytania zewnętrznego. Zależnie od liczby wartości zwracanych przez podzapytanie, stosujemy różne operatory w warunku selekcji zapytania zewnętrznego.

Przykład 4.75. _____

Znajdź osoby, które mają najwyższe stawki — pytanie z podzapytaniem

wewnętrznym.

Najpierw należy wyliczyć najwyższą stawkę dla wszystkich pracowników. Ustalimy ją prostym pytaniem:

```
SELECT MAX(STAWKA) FROM PRACOWNICY.
```

To pytanie zwraca jedną określoną wartość i jest uruchamiane na początku — nazywa się podzapytaniem zagnieżdżonym. Następnie wyliczoną maksymalną stawkę uwzględniamy w warunku wyszukiwania zapytania głównego.

Całość polecenia będzie miała następującą składnię:

```
SELECT *  FROM PRACOWNICY
WHERE STAWKA =
  (SELECT MAX(STAWKA)
  FROM PRACOWNICY);
```

Ten rodzaj podzapytań można zagnieżdżać na wielu poziomach — prezentują to poniższe przykłady.

Przykład 4.76.

Znajdź adres kierownika działu o nazwie Administracja.

```
SELECT * FROM ADRESY WHERE
IDENTYFIKATOR =
  (SELECT IDENTYFIKATOR FROM KIEROWNICY   WHERE
"KOD DZIAŁU" =
        (SELECT "KOD DZIAŁU" FROM DZIAŁY
        WHERE "NAZWA DZIAŁU" = 'Administracja'));
```

Najpierw jest wykonywane najbardziej zagnieżdżone podzapytanie, które ustala "Kod działu" związany z podaną nazwą. Następne pytanie znajduje Identyfikator kierownika tego działu. Na końcu pytanie główne wyszukuje adres związany z Identyfikatorem konkretnego kierownika. Każde pytanie zwraca jedną wartość, dlatego w każdym warunku jest znak równości.

Przykład 4.77.

Podaj nazwy działów, w których pracują mieszkańcy Warszawy.

```
SELECT  "NAZWA DZIALU" FROM DZIAŁY
WHERE "KOD DZIALU" IN
  (SELECT "KOD DZIALU" FROM PRACOWNICY   WHERE
IDENTYFIKATOR IN
        (SELECT IDENTYFIKATOR FROM ADRESY   WHERE
       MIASTO = 'Warszawa'));
```

Na początku są ustalane identyfikatory adresów związanych z miastem Warszawa. Każde pytanie zwróci zbiór wartości, dlatego w warunkach jest użyty operator IN. Drugie pytanie wyszuka działy, w których są zatrudnione osoby z podanymi identyfikatorami.
Na końcu pytanie główne zwróci nazwy działów, w których są zatrudnieni mieszkańcy Warszawy.

Przykład 4.78.

Wybierz działy, w których średnia stawka jest wyższa od przeciętnej, tzn. większa niż średnia stawka wszystkich pracowników — zastosowanie podzapytania w klauzuli HAVING.

```
SELECT "KOD DZIAŁU"
FROM PRACOWNICY
GROUP BY "KOD DZIAŁU" HAVING
AVG(STAWKA)>
  (SELECT AVG(STAWKA)
  FROM PRACOWNICY);
```

Zależnie od wartości zwracanych przez podzapytanie zagnieżdżone, rozróżniamy podzapytania wierszowe i tablicowe.

Podzapytanie, które zwraca pojedynczą wartość, jest nazywane **podzapytaniem skalarnym**. W warunkach klauzuli WHERE lub HAVING w zapytaniach zewnętrznych są stosowane operatory: =, <, >, >=, <=, <>.

W **podzapytaniach tablicowych**, zwanych również wielowierszowymi, wynik może dać jeden lub wiele wierszy, zawierających wartości jednego lub wielu atrybutów.

W warunkach klauzuli WHERE lub HAVING w zapytaniach zewnętrznych są stosowane operatory: IN, ANY, SOME, ALL. Warunek z operatorem ANY lub SOME (operatory są równoważne) jest prawdziwy, jeśli jest spełniony dla chociaż jednej wartości zwróconej przez podzapytanie. Warunek z operatorem ALL jest prawdziwy, jeśli jest spełniony dla każdej wartości zwróconej przez podzapytanie.

Przykład 4.79.

Ustal listę pracowników, którzy mają najniższe stawki w swoich działach — zastosowanie podzapytań tablicowych.

```
SELECT  * FROM PRACOWNICY
WHERE ("KOD DZIAŁU", STAWKA) IN
  (SELECT "KOD DZIAŁU", MIN(STAWKA)
  FROM PRACOWNICY
  GROUP BY "KOD DZIAŁU" );
```

Przykład 4.80.

Znajdź pracowników, którzy mają pensję wyższą niż przynajmniej jeden pracownik administracji — zastosowanie operatora ANY/SOME.

Podzapytanie zwróci wiele wartości jednego atrybutu.

```
SELECT *
FROM PRACOWNICY
WHERE STAWKA > SOME(SELECT STAWKA
FROM PRACOWNICY
WHERE "KOD DZIAŁU" = 'AD');
```

Podzapytania skorelowane

Różnią się od poprzednich sposobem powiązania podzapytania z zapytaniem głównym oraz kolejnością i sposobem przetwarzania. Warunek powiązania, umieszczony w podzapytaniu, jest zwany warunkiem korelacji, bo zawiera odwołanie do zapytania zewnętrznego. Ten warunek jest ustalany przez zapytanie główne dla każdego wiersza analizowanego w tym zapytaniu. Podzapytanie skorelowane jest uruchamiane wielokrotnie.

Przykład 4.81.

Ustal, którzy pracownicy mają najniższe stawki w swoich działach. Składnia tego pytania będzie następująca:

```
SELECT *
FROM PRACOWNICY G
WHERE STAWKA =
  (SELECT MIN(STAWKA)
  FROM PRACOWNICY P
  WHERE P. "KOD DZIAŁU" = G. "KOD DZIAŁU");
```

W obu powiązanych pytaniach jest zadeklarowana ta sama tablica Pracownicy, jest więc konieczne rozróżnienie tablic przez aliasy. W pytaniu głównym tablica Pracownicy ma zadeklarowany alias i nazywa się teraz tablicą G. Alias w podzapytaniu jest zbędny, bo występuje w nim tylko jedna tablica, ale dla ujednolicenia zapisu została zadeklarowana nowa nazwa P.

Na początku pytanie główne pobierze pierwszy wiersz z tablicy Pracownicy i ustali "Kod działu", w którym jest zatrudniony pierwszy pracownik. Następnie zostanie uruchomione podzapytanie, które dla tego konkretnego działu (zgodnie z warunkiem korelacji P. "KOD DZIAŁU" = G. "KOD DZIAŁU") znajdzie najniższą stawkę. Zwrócony wynik będzie teraz uwzględniony w warunku zapytania głównego — WHERE STAWKA = (*wynik podzapytania*). Jeśli warunek będzie miał wartość PRAWDA, osoba będzie włączona do zbioru wyników, zostanie pobrany kolejny wiersz z tabeli G, ustalone miejsce zatrudnienia i ponownie uruchomione podzapytanie.

Przykład 4.82.

Wyszukaj działy, w których pracuje poniżej 10 osób.

```
SELECT * FROM DZIAŁY D
WHERE 10>
(SELECT COUNT(*)
FROM PRACOWNICY P
WHERE P. "KOD DZIAŁU" = D."KOD DZIAŁU");
```

W tym poleceniu dla każdego działu zostanie uruchomione podzapytanie, które zwróci liczbę zatrudnionych osób w dziale, zgodnie z warunkiem korelacji. Jeśli zwrócona wartość będzie mniejsza od 10, wówczas opis działu — Kod i Nazwa działu — zostaną włączone do zbioru wynikowego.

Przykład 4.83.

Wyszukaj trzech pracowników najwięcej zarabiających.

```
SELECT *
FROM PRACOWNICY G
WHERE 3>
  (SELECT COUNT(*)
  FROM PRACOWNICY P
  WHERE P.STAWKA * P. "CZAS PRACY"  >
  G.STAWKA * G."CZAS PRACY")
ORDER BY 2 DESC;
```

W podzapytaniu funkcja COUNT zlicza, w ilu wypadkach pensja poszczególnych pracowników jest większa od tej, jaką ma pracownik w wierszu analizowanym w zapytaniu głównym. Jeśli wartość zwrócona jest mniejsza od 3, osoba z tablicy G jest włączona do zbioru wynikowego.

Przykład. 4.84.

Wypisz nazwy działów, w których pracuje najwięcej osób.

Takie wyszukiwanie realizuje następujące polecenie:

```
SELECT "NAZWA DZIALU", COUNT(P.IDENTYFIKATOR) OBSADA
FROM PRACOWNICY P , DZIAŁY D
WHERE UPPER(P."KOD DZIALU") = UPPER(D."KOD DZIALU")
GROUP BY "NAZWA DZIALU"
HAVING COUNT(P.IDENTYFIKATOR)=
  (SELECT MAX(ZATRUDNIENI) FROM
    (SELECT COUNT(IDENTYFIKATOR) ZATRUDNIENI
    FROM PRACOWNICY
    GROUP BY UPPER("KOD DZIALU") ) WYNIK1);
```

Prześledźmy pytania w tej kolejności, w jakiej będą uruchamiane. Pierwsze pytanie, zamieszczone poniżej, zliczy osoby zatrudnione w każdym dziale.

```
SELECT COUNT(IDENTYFIKATOR) ZATRUDNIENI
FROM PRACOWNICY
GROUP BY UPPER("KOD DZIALU");
```

Wynik tego pytania (w przykładzie ma nadany alias — ZATRUDNIENI) będzie źródłem danych dla następnego pytania, które wybierze największą wartość:

```
SELECT MAX(ZATRUDNIENI) FROM
  (SELECT COUNT(IDENTYFIKATOR) ZATRUDNIENI
  FROM PRACOWNICY
  GROUP BY UPPER("KOD DZIALU")) WYNIK1;
```

Ustalona wartość (tablica dynamiczna również ma zadeklarowany alias — WYNIK1) posłuży do zbudowania warunku w klauzuli HAVING kolejnego polecenia, które ma wygenerować wynik. Pozostaje tylko zbudować pytanie z grupowaniem na złączonych tablicach (aby móc wyświetlić nazwy działów, musi być użyta tablica DZIAŁY, natomiast aby ustalić liczbę zatrudnionych w dziale, jest konieczna tablica PRACOWNICY).

W pytaniach z podzapytaniami skorelowanymi jest często stosowany operator EXISTS. Operator ten ma wartość prawdy, gdy następujące po nim podzapytanie zwróci przynajmniej jeden rekord. Nie ma znaczenia, czy podzapytanie zwróci pełny wiersz wybrany poleceniem SELECT *, czy określony literał, np. SELECT 1.

Przykład 4.85.

Ustal nazwy działów, w których nie ma zatrudnionych pracowników — zastosowanie operatora EXISTS w zapytaniu.

```
SELECT "NAZWA DZIAŁU"
FROM DZIAŁY K
WHERE  NOT EXISTS (SELECT 1
  FROM PRACOWNICY P
  WHERE  UPPER(P."KOD  DZIAŁU") =   UPPER(K."KOD
            DZIAŁU"));
```

Podzapytanie w klauzuli SELECT

W niektórych SZBD, również w Oracle, można umieszczać podzapytania w klauzuli SELECT, co przedstawia poniższy przykład.

Przykład 4.86.

Wyświetl nazwy działów i maksymalne stawki w działach.

W pytaniu zewnętrznym operujacym na tablicy DZIAŁY dla każdego działu po podaniu Nazwy działu podzapytanie wyświetla wyliczoną maksymalną stawkę w danym dziale.

```
SELECT "NAZWA DZIAŁU",
  (SELECT MAX(STAWKA) FROM PRACOWNICY P
  WHERE P."KOD DZIAŁU" = D."KOD DZIAŁU") MAKSY
FROM  DZIAŁY D;
```

Podzapytanie umieszczone w klauzuli SELECT musi zwracać co najwyżej jedną wartość dla wiersza analizowanego przez zapytanie zewnętrzne. Należy zdefiniować alias dla atrybutu zwracanego przez podzapytanie (w przykładzie jest to MAKSY).

Podzapytania w klauzuli FROM

Podzapytania mogą być umieszczone w klauzuli FROM zapytania zewnętrznego. Dane zwracane przez podzapytanie są zbiorem wejściowym dla zapytania zewnętrznego. Podzapytania w klauzuli FROM noszą nazwę *inline views* i są traktowane jak dynamiczne

widoki.

Przykład 4.87.

Wybierz pracowników (podając nazwisko, stawkę pracownika i wyliczoną średnią stawkę w dziale), którzy mają stawkę większą niż średnia stawka w ich dziale.

```
SELECT NAZWISKO, STAWKA, SREDNIA
FROM
```

```
(SELECT "KOD DZIALU", ROUND(AVG(STAWKA),0) AS SREDNIA
FROM PRACOWNICY
GROUP BY "KOD DZIALU") Z
JOIN PRACOWNICY P ON Z."KOD DZIALU" = P."KOD DZIALU" WHERE
STAWKA > SREDNIA;
```

W podzapytaniu stosowanym w klauzuli FROM muszą być zdefiniowane aliasy — w przykładzie całe podzapytanie ma alias Z, kolumna uwzględniona w wyniku zapytania głównego to SREDNIA.

Podzapytania w poleceniu INSERT

Podzapytanie może być również uwzględnione w instrukcji INSERT, umożliwiając automatyczne wstawienie do tablicy wielu wierszy (składnia INSERT INTO...SELECT omówiona przy operacji wstawiania danych w rozdziale 5.1.).

Podzapytania w poleceniu DELETE

Podzapytanie może również być użyte do wyboru wierszy do usunięcia — poprzez połączenie poleceń DELETE z poleceniem SELECT.

Przykład 4.88.

Wybierz do usunięcia adresy, w których brak danych atrybutu Ulica.

```
DELETE FROM
(SELECT * FROM ADRESY WHERE ULICA IS NULL);
```

Identyczny wynik uzyskamy również przy użyciu pojedynczego polecenia usuwania:

```
DELETE FROM ADRESY WHERE ULICA IS NULL;
```

Podzapytania w poleceniu UPDATE

Przykład 4.89.

Przeprowadź dziesięcioprocentową podwyżkę stawki pracownikom, którzy mają stawkę mniejszą niż średnia stawka pracowników w danym dziale — zastosowanie podzapytania w poleceniu UPDATE.

```
UPDATE PRACOWNICY G
SET STAWKA = STAWKA *1.1
  WHERE STAWKA <
  (SELECT AVG(STAWKA)
  FROM PRACOWNICY P
  WHERE P."KOD DZIALU" = G."KOD DZIALU");
```

Podzapytanie w poleceniu CREATE

Tylko w Oracle podzapytanie może być użyte podczas tworzenia tablicy — w klauzuli AS SELECT polecenia CREATE TABLE. Pokazuje to poniższy przykład.

Przykład 4.90.

Utwórz tablicę OSOBY na podstawie istniejącej tablicy PRACOWNICY — użycie podzapytania przy tworzeniu tablicy.

```
CREATE TABLE OSOBY(NAZWISKO,IMIE,WPLATA)
  AS SELECT NAZWISKO,IMIE,STAWKA
  FROM PRACOWNICY;
```

Tworząc tablicę, nie podajemy nazw typów danych — system sam ustala typy i rozmiary danych, uwzględniając wartości zwracane przez podzapytanie. Tym poleceniem możemy kopiować dane z tabeli i zapisywać je oddzielnie w nowo utworzonej tabeli.

Ograniczanie zbioru wynikowego do N-wierszy

W bazie Oracle takie zadanie przysparza trochę problemów. W innych systemach, np. MySQL czy Sybase, do tego celu służą klauzule TOP, LIMIT i OFFSET — użyte w poleceniu ograniczają zbiór uporządkowany lub nieuporządkowany do N-wierszy. W bazie Oracle do tego celu można wykorzystać specjalną pseudokolumnę ROWNUM.

Pseudokolumna ROWNUM służy do numerowania wierszy wynikowych zapytania — jej wartość rośnie o 1 dla kolejnego wiersza zwróconego w wyniku. Numeracja wierszy odbywa się dla każdego wyniku pytania niezależnie — czyli ten sam wiersz, zwracany przez różne pytania, będzie miał różne wartości w kolumnie ROWNUM. Przyporządkowanie numerów odbywa się dynamicznie w trakcie ustalania zbioru wynikowego — zanim nastąpi sortowanie porządkujące zbiór wynikowy. Przyjrzyjmy się zawartości tej kolumny, wyświetlając ją w zapytaniu.

Przykład 4.91.

Wybieranie pseudokolumny ROWNUM w poleceniu.

```
SELECT ROWNUM, P.* FROM PRACOWNICY P;
```

Polecenie powoduje wyświetlenie wszystkich kolumn z tabeli Pracownicy. Jako pierwsza zostanie wyświetlona pseudokolumna ROWNUM, która ponumeruje rekordy wynikowe (rysunek 4.9). Numer jeden w kolumnie ROWNUM jest przyporządkowany do pierwszego wiersza wyniku, dwa do kolejnego itd.

	ROWNUM	IDE...	NAZWISKO	IMIE	KOD DZIALU	CZAS PRACY	STAWKA	UBEZPIE
1	1	EN01	Jacki ...	Tomasz ...	0	38	15	Tak
2	2	EN03	Górska ...	Hanna ...	za	40	21	Tak
3	3	EN04	Padek ...	Paulina ...	MK	35	24	Tak
4	4	EN05	Binder ...	Julia ...	za	25	8	Tak
5	5	EN07	Sanderska ...	Maria ...	za	29	7	Tak
6	6	EN08	Elbaj ...	Klaudia ...	EE	40	15	Nie
7	7	EN10	Martecka ...	Joanna ...	EE	40	9	Nie
8	8	EN11	Binga ...	Alicja ...	EE	15	7	Tak
9	9	EN12	Czapski ...	Bogdan ...	ch	40	19	Tak
10	10	EN13	Obarski ...	Dominik ...	ad	40	22	Nie
11	11	EN14	Biński ...	Sebastian...	sp	35	13	Tak

Rysunek 4.9. *Wynik wykorzystania pseudokolumny ROWNUM*

Analizując powyższy wynik, wydaje się, że prosty warunek dotyczący kolumny ROWNUM pozwoli ograniczyć zbiór wynikowy do N-wierszy.

Przykład 4.92.

Wybierz 5 kolejnych osób z tablicy Pracownicy.

Poniższe polecenie działa poprawnie, zwróci spodziewany wynik — 5 pierwszych wierszy z tablicy Pracownicy (rysunek 4.10).

```
SELECT ROWNUM, P.* FROM PRACOWNICY P
WHERE ROWNUM <=5;
```

	ROWNUM	IDENTYFIKATOR	NAZWISKO	IMIE	KOD DZIALU
1	1	EN01	Jacki	Tomasz	0
2	2	EN03	Górska	Hanna	za
3	3	EN04	Padek	Paulina	MK
4	4	EN05	Binder	Julia	za
5	5	EN07	Sanderska	Maria	za

Rysunek 4.10. *Wynik pytania z ograniczeniem zbioru wyników do n-wierszy*

Problem pojawia się, gdy chcemy wyświetlić kolejne wiersze (próbując warunku ROWNUM >5) lub konkretny wiersz tablicy (mając na myśli np. ROWNUM = 2).

Przykład 4.93.

Wybierz drugi wiersz z tablicy Pracownicy.

Niestety, polecenie poniższe nie zwraca żadnych wyników.

```
SELECT ROWNUM, P.* FROM PRACOWNICY P
WHERE ROWNUM = 2;
```

Dlaczego tak się dzieje, że nie można wyświetlić wskazanego wiersza, choć przecież udostępniało go poprzednie pytanie?

Otóż kluczową sprawą jest to, że pseudokolumna ROWNUM dynamicznie numeruje wiersze w trakcie tworzenia wyniku.

W naszym pytaniu przebiega to następująco. Najpierw jest wykonywane przypisanie ROWNUM, a potem jest sprawdzany warunek selekcji — czyli dla pierwszego rekordu, którego ROWNUM jest = 1, jest sprawdzany warunek z polecenia: WHERE ROWNUM = 2. Ponieważ jest to FAŁSZ, wiersz jest odrzucany. Drugiemu rekordowi zostaje przypisana wartość ROWNUM = 1 (ponieważ poprzedni rekord został odrzucony). Sprawdzamy ponownie warunek WHERE ROWNUM = 2. Kolejny raz jest to FAŁSZ. Itd. Itd. Czyli ten warunek zadziała tylko wyłącznie dla ROWNUM = 1 — gdy wybierzemy jeden wiersz.

Zaskakujące wyniki uzyskamy również, gdy wprowadzimy sortowanie wierszy rosnąco według stawki i będziemy próbowali wybrać kilka pierwszych wierszy ze zbioru wynikowego, np. 5 osób z najniższymi stawkami.

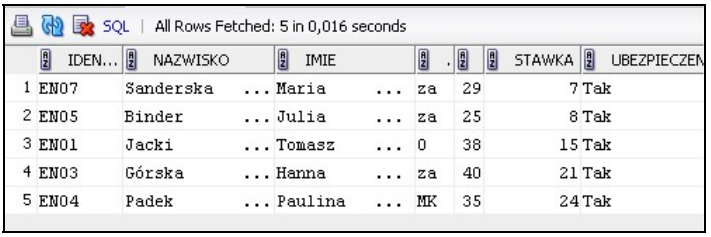
Przykład 4.94. _____

Włączenie do pytania z sortowaniem warunku dla ROWNUM.

```
SELECT * FROM PRACOWNICY
WHERE ROWNUM <= 5
ORDER BY STAWKA;
```

Wynik pytania prezentuje rysunek 4.11. Wiersze wynikowe spełniają warunek ROWNUM<=5, ale dane dotyczące stawki nie odpowiadają wartościom w uporządkowanym zbiorze.

Rysunek 4.11. *Wynik pytania z warunkiem na ROWNUM*


SQL | All Rows Fetched: 5 in 0,016 seconds

	IDEN...	NAZWISKO	IMIE	.		STAWKA	UBEZPIECZEN
1	EN07	Sanderska ...	Maria ...	za	29	7	Tak
2	EN05	Binder ...	Julia ...	za	25	8	Tak
3	EN01	Jacki ...	Tomasz ...	0	38	15	Tak
4	EN03	Górska ...	Hanna ...	za	40	21	Tak
5	EN04	Padek ...	Paulina ...	MK	35	24	Tak

Gdy tymczasem dane o pracownikach posortowane według stawki wyglądają jak na rysunku 4.12.

Rysunek 4.12. *Posortowane wiersze z przykładowej tabeli*

	IDEN...	NAZWISKO	IMIE			STAWKA	UBEZPIE
1	EN50	Cline ...	Darius...	...		7	Nie
2	EN11	Binga ...	Alicja...	...		7	Tak
3	EN07	Sanders...	Maria ...	...		7	Tak
4	EN05	Binder ...	Julia ...	...		8	Tak
5	EN40	Warkows...	Ryszar...	...		8	Nie
6	EN10	Marteck...	Joanna...	...		9	Nie
7	EN20	Kowalcz...	Marlen...	...		9	Tak
8	EN76	Semik ...	Patryc...	...		10	Tak
9	EN15	Aberack...	Maria ...	...		12	Tak

Dlaczego tak się dzieje, że w ostatnim przykładzie nie otrzymujemy pięciu kolejnych wierszy z posortowanego zbioru?

Ponieważ najpierw nastąpiło przypisanie wartości w ROWNUM kolejnym rekordom, a dopiero później jest wykonane sortowanie.

Obejście tego problemu jest możliwe przy użyciu mechanizmu podzapytań.

Przykład 4.95.

Wracamy do naszego zadania: chcemy znaleźć 5 osób z najniższymi stawkami — wykorzystujemy podzapytanie do przygotowania posortowanego zbioru wierszy. Podzapytanie, umieszczone w klauzuli FROM pytania głównego, jest źródłem danych dla ostatecznego wyniku.

```
SELECT * FROM
  (SELECT  ROWNUM,P.*
  FROM PRACOWNICY P
  ORDER BY STAWKA) WHERE
ROWNUM <=5;
```

Otrzymamy wtedy właściwe wyniki zaprezentowane na rysunku 4.13.

Rysunek 4.13. *Wynik pytania z ograniczeniem do N-wierszy z posortowanego zbioru*

	ROWNUM	IDE...	NAZWISKO	...	IMIE	...	...	STAWKA	UBEZPIE
1	5	EN07	Sanderska	...	Maria ...	za	29	7	Tak
2	30	EN50	Cline	...	Darius...	sp	20	7	Nie
3	8	EN11	Binga	...	Alicja...	EE	15	7	Tak
4	4	EN05	Binder	...	Julia ...	za	25	8	Tak
5	28	EN40	Warkowski	...	Ryszar...	za	25	8	Nie

Przykład 4.96.

Wybierz trzy najwyższe stawki, jakie osiągają pracownicy.

W tym poleceniu również jest konieczne zastosowanie podzapytania.

```
SELECT * FROM
  (SELECT STAWKA
  FROM PRACOWNICY
  GROUP BY STAWKA
  ORDER BY STAWKA DESC)
WHERE ROWNUM < 4;
```

Przykład 4.97.

Wybierz konkretny rekord z tablicy, wskazując jego numer, np. 6 — należy zastosować następujące polecenie:

```
SELECT * FROM
  (SELECT ROWNUM, P.*
  FROM PRACOWNICY P
  WHERE ROWNUM <=6
  ORDER BY ROWNUM DESC)
WHERE ROWNUM =1;
```

Najpierw zostanie wykonane podzapytanie w nawiasie, które wybierze 6 wierszy i posortuje ten zbiór malejąco według ROWNUM. Osoba o numerze ROWNUM=6 będzie występować w wyniku podzapytania jako pierwsza. Pytanie główne z tego uporządkowanego zbioru wybierze pierwszy wiersz.

Pytania parametryczne

Są one definiowane w taki sposób, aby wybrany atrybut z warunku selekcji był parametrem zewnętrznym. Podczas uruchamiania pytania, na żądanie serwera, podajemy wartość tego parametru w okienku dialogowym. Wartości znakowe piszemy w apostrofach.

```
SELECT * FROM PRACOWNICY
WHERE STAWKA = &STAWKA
```

4.7. Operacje na zbiorach

Operacje na zbiorach to suma, przecięcie i różnica. Działania te są wykonywane na zbiorach mających taką samą strukturę. Liczba i typ kolumn zwracanych przez każde pytanie muszą być takie same. Nazwy odpowiadających sobie kolumn w dwóch zapytaniach nie muszą być jednakowe.

W operacjach na zbiorach może być zdefiniowane jawnie sortowanie wyniku za pomocą klauzuli ORDER BY poprzez wskazanie numeru kolumny, według której odbędzie się porządkowanie wierszy.

Suma zbiorów wykonywana dzięki operatorowi UNION to połączenie wyników zapytań w jeden zbiór wynikowy, z wyłączeniem wierszy powtarzających się. W celu wyeliminowania powtórzeń po zsumowaniu zbiorów operatorem UNION następuje sortowanie wierszy wynikowych. Wiersze z różnych instrukcji SELECT są przemieszane. Operator UNION ALL dopuszcza powtarzanie wierszy. Nie ma sortowania wyniku — sumowane zbiory wynikowe pozostają jeden pod drugim.

Różnica zbiorów jest otrzymywana przy użyciu operatora MINUS. Zwraca te wiersze, które występują w wyniku pierwszego zapytania, a nie występują w drugim zbiorze.

Iloczyn zbiorów jest otrzymywany dzięki operatorowi INTERSECT. Zwraca wiersze, które występują w wynikach obu zapytań.

Przykład 4.98.

Wybierz osoby (podając nazwiska i imiona) będące pracownikami lub klientami — suma zbiorów.

```
SELECT NAZWISKO, IMIĘ FROM PRACOWNICY
UNION
SELECT NAZWISKO, IMIĘ FROM  KLIENCI;
```

Każda osoba będzie uwzględniona jednokrotnie, nawet jeśli ktoś jest zarówno pracownikiem, jak i klientem.

Przykład 4.99.

Wyszukaj nazwiska osób, które są pracownikami i klientami jednocześnie — iloczyn zbiorów przy użyciu operatora INTERSECT.

```
SELECT NAZWISKO FROM PRACOWNICY
INTERSECT
SELECT NAZWISKO FROM  KLIENCI;
```

Przykład 4.100.

Znajdź nazwiska pracowników, którzy nie są klientami — różnica zbiorów przy zastosowaniu operatora MINUS.

```
SELECT NAZWISKO FROM PRACOWNICY
MINUS
SELECT NAZWISKO FROM  KLIENCI;
```

Podczas wykonywania sumy i iloczynu zbiorów kolejność zapytań uwzględnianych w tych operacjach nie ma wpływu na wynik. W trakcie definiowania różnicy zbiorów kolejność pytań ma fundamentalne znaczenie.

4.8. Zadania

Projekcja i selekcja:

1. Wypisz alfabetyczną listę płac pracowników, wyświetlając w kolumnie OSOBA — nazwisko i imię pracownika, w kolumnie DZIAŁ — kod działu, a w trzeciej kolumnie PENSJA — pensję wyliczoną na podstawie stawki i czasu pracy.
2. Wypisz pracowników pięciu wybranych działów, mających stawkę z zakresu od 20 do 25 zł.
3. Wypisz pracowników bez ubezpieczenia.
4. Sprawdź, czy wśród pracowników są osoby w wieku emerytalnym.
5. Wypisz kobiety z działu administracji.
6. Znajdź pracowników, którzy zarabiają powyżej 1000 zł i są z dwóch działów (kody działów o wartościach AD i CH), uporządkuj zbiór według daty zatrudnienia.
7. Wyświetl nazwiska i pensję; tam, gdzie nie ma podanej stawki, wpisz 0.
8. Wypisz NAZWISKA i IMIONA osób, a w trzeciej kolumnie Płeć wyświetl literały K lub M wskazujące na kobietę lub mężczyznę.
9. Wypisz w postaci jednego łańcucha nazwisko pracownika i dział, włączając stałe opisy: OSOBA: i Miejsce pracy:.
10. Wyświetl aktualną datę i czas.

Łącząc tablice:

1. Podaj nazwiska pracowników i nazwy działów, w których pracują te osoby.
2. Sprawdź, czy wszystkie działy, w których zatrudnieni są pracownicy, mają w bazie swoją nazwę.
3. Podaj nazwy działów, w których pracują mieszkańcy Warszawy.

4. Podaj pełne informacje o pracownikach, wyświetlając: Nazwisko i imię, Adres, Nazwę działu i Nazwisko kierownika.
5. Wypisz pracowników oraz ich kierowników.
6. Ustaw w pary pracowników z tego samego działu — nazwiska w parze powinny być różne i pary nie mogą się powtarzać.
7. Sprawdź, czy wszyscy pracownicy mają w bazie wpisane adresy.
8. Sprawdź, czy w tablicy są jakieś wiersze do usunięcia oraz czy są adresy nieprzyporządkowane do osób.
9. Wypisz pracowników z Pruszkowa.
10. Sprawdź, czy wśród pracowników są równolatkowie.

Definiując podzapytania:

1. Znajdź, którzy pracownicy nie mają swoich adresów w bazie.
2. Podaj adres kierownika działu o nazwie Administracja.
3. Podaj, w jakich działach pracują mieszkańcy Warszawy.
4. Wypisz osoby mające najwyższe stawki w swoich działach.
5. Wypisz nazwy działów, w których pracuje poniżej 10 osób (jako podzapytanie zagnieżdżone).
6. Wypisz nazwy działów, w których pracuje poniżej 10 osób (jako podzapytanie skorelowane).
7. Podaj miasta, z których dojeżdżają pracownicy działu Administracji.
8. Wypisz trzech pracowników najmniej zarabiających.
9. Wypisz wszystkich pracowników z najniższą stawką.
10. Wypisz najstarszych pracowników z każdego działu.

Grupowanie:

1. Podaj liczbę zatrudnionych i średnie pensje w każdym dziale.
2. Podaj działy, w których najwyższe stawki godzinowe przekraczają 20 zł.
3. Podaj kierowników działów i liczbę podległych im pracowników.
4. Podaj kody działów, nazwy działów oraz najwyższe i najniższe stawki w działach.
5. Podaj, jaki jest fundusz płac każdego działu (suma pensji pracowników w działach).
6. Podaj sumaryczną i średnią płacę w każdym dziale.
7. Zaokrąglij wynik średniej płacy do dwóch miejsc po przecinku.
8. Podaj, ilu pracowników mieszka w poszczególnych miastach.
9. Podaj kierowników działów, nazwy działów i maksymalne stawki w działach.
10. Wypisz kierowników, którzy mają ponad 10 osób w dziale.

Rozdział 5. Język manipulowania danymi — DML

Polecenia INSERT, UPDATE i DELETE oddziałują na dane przechowywane w bazie, umożliwiając wprowadzanie nowych wierszy, usuwanie i zmiany wartości w tabelach — dlatego wymagają dużej ostrożności przy ich uruchamianiu. Jeżeli podczas korzystania z tych poleceń zostanie popełniony błąd, należy użyć instrukcji ROLLBACK, aby cofnąć zmiany. Polecenie ROLLBACK wycofuje zmiany w ramach jednej transakcji.

5.1. Polecenie INSERT — wprowadzanie danych do tablicy

Wstawianie jednego rekordu do tablicy, gdy dane są wpisywane z klawiatury. Zadanie to jest wykonywane przy zastosowaniu polecenia:

```
INSERT INTO Nazwa_tabeli[(nazwa_kolumny1, nazwa_kol2 ...)] VALUES
(wartość1, wartość2 , ... )
```

Podczas wstawiania danych muszą być spełnione trzy warunki:

- Użyte wartości muszą być tego samego typu co pola, do których są wpisywane. ♦ Rozmiar poszczególnych danych nie może przekraczać rozmiaru kolumny.
- Położenie dodawanych danych na liście wartości musi odpowiadać położeniu kolumn (tzn. pierwsza wartość musi być wprowadzona do pierwszej kolumny itd.).

W poleceniu INSERT...VALUES... można pominąć listę atrybutów docelowych wymienianych w nawiasie okrągłym po nazwie tablicy tylko w wypadku, gdy wstawiamy wartości do wszystkich kolumn i znamy dobrze strukturę tablicy.

Przykład 5.1. _____

Wstaw wiersz do tablicy DZIAŁY zawierający opis działu prawnego.

```
INSERT INTO DZIAŁY
```

VALUES (‘PR’, ‘PRAWNY’);

Przykład 5.2.

Zarejestruj w tablicy Pracownicy jedną osobę — wstawianie wartości do wybranych kolumn.

```
INSERT INTO PRACOWNICY(IDENTYFIKATOR,IMIĘ,NAZWISKO) VALUES
('EN100','JAN','NOWAK');
```

Operacja wstawiania jest operacją na wierszach. Wpisując wartości do niektórych kolumn, wstawiamy cały wiersz. Pominięte podczas wstawiania atrybuty będą miały wartość NULL.

Polecenie INSERT...SELECT służy do wstawiania do tablicy wielu wierszy będących wynikiem zapytania.

W ten sposób jest możliwe przekopiowanie danych z jednej lub wielu tablic do innej tablicy.

Przykład 5.3.

Przekopiuj do tablicy OSOBY tych pracowników z tablicy PRACOWNICY, którzy mają najniższe stawki w swoich działach.

```
INSERT INTO OSOBY (NAZWISKO,IMIĘ)
  SELECT NAZWISKO,IMIĘ
  FROM PRACOWNICY G
  WHERE STAWKA =
      (SELECT MIN(STAWKA)
      FROM PRACOWNICY P
      WHERE P. "KOD DZIAŁU" = G. "KOD DZIAŁU");
```

Przykład 5.4.

Wpisz do tablicy OSOBY jednego pracownika o nazwisku Nowak, wiedząc, że pole NR w tej tablicy jest kluczem głównym, automatycznie inkrementowanym — wykorzystanie sekwencji podczas wprowadzania danych.

Definiowanie i omówienie sekwencji jest zamieszczone w rozdziale następnym, dotyczącym języka DDL. Jest tam wykorzystana sekwencja o nazwie ID, zdefiniowana w przykładzie 6.4. Podczas wstawiania wiersza wykorzystujemy nową wartość generowaną z użyciem sekwencji, korzystając z pseudokolumny NextVal.

```
INSERT INTO OSOBY (NR, NAZWISKO) VALUES
(ID.NEXTVAL, 'NOWAK');
```

W tym przykładzie, rejestrując Nowaka w tablicy OSOBY, wprowadzamy do kolumny licznikowej NR wartość ID.NextVal. Ta unikalna wartość, aktualnie wygenerowana przez sekwencję, jest identyfikatorem osoby. Ponownie możemy użyć tej wartości sekwencji, np. wpisując adres Nowaka do tablicy Adresy. Dla zachowania spójności danych ten sam identyfikator powinien być przypisany konkretnej osobie i jej adresowi. W takim wypadku, wpisując adres, w kolumnie klucza wstawimy wartość ID.CurrVal.

5.2. Polecenie UPDATE — modyfikacja wartości w tablicy

Polecenie UPDATE umożliwia wprowadzenie modyfikacji danych w jednej tabeli — zmiany mogą dotyczyć jednej lub wielu kolumn w jednym lub wielu wierszach. Polecenie może być również bezskuteczne, kiedy żaden wiersz nie zostanie zmodyfikowany — zależy to od warunku w klauzuli WHERE.

```
UPDATE nazwa_tabeli
SET nazwa_kolumny1 = wartość1 [,nazwa_kolumny2 = wartość2]... WHERE
warunek_wyszukiwania
```

Modyfikacja jest przeprowadzona w wierszach, w których warunek wyszukiwania ma wartość TRUE. Jeżeli fraza WHERE jest opuszczona, wówczas są aktualizowane wszystkie rekordy użytej tabeli.

Przykład 5.5.

Zmień nazwisko wybranej osoby — pani Kowalska aktualnie ma nazywać się Nowak.

```
UPDATE PRACOWNICY
SET NAZWISKO = 'Nowak '
WHERE NAZWISKO = 'Kowalska';
```

Można aktualizować kilka kolumn jednocześnie:

Przykład 5.6.

Przeprowadź dwudziestoprocentową podwyżkę stawki i ustal obowiązujący czasu pracy — 42 godziny dla wszystkich pracowników administracji w dziale AD.

```
UPDATE PRACOWNICY
SET "CZAS PRACY" = 42, STAWKA = STAWKA*1.2
WHERE "KOD DZIAŁU" = 'AD';
```

Wartości aktualizujące mogą być również wynikiem wyrażenia jak w powyższym przykładzie.

5.3. Polecenie DELETE — usuwanie danych w tabeli

Zależnie od składni polecenia DELETE, a dokładnie frazy WHERE w tym poleceniu, można jednokrotnym jego uruchomieniem uzyskać następujące rezultaty: usunąć jeden wiersz, usunąć wiele wierszy, usunąć wszystkie wiersze w tabeli lub niczego nie usuwać. Usuwanie danych jest wykonywane bez ostrzeżenia i dodatkowego pytania o potwierdzenie decyzji.

```
DELETE FROM nazwa_tabeli
WHERE warunek;
```

DELETE usuwa całe rekordy w określonej tabeli. Nie można usunąć wartości z poszczególnych komórek, np. usunąć stawki wybranemu pracownikowi, bo do tego celu należy użyć polecenia UPDATE.

Za pomocą polecenia DELETE można usunąć wszystkie wiersze, ale nie usuniemy tabeli, bo do tego służy m.in. polecenie DROP TABLE.

Polecenie TRUNCATE umożliwia usunięcie wszystkich wierszy w tabeli i zwolnienie miejsca do innych zastosowań, bez usuwania definicji tabeli z bazy danych.

```
TRUNCATE TABLE <nazwa>;
```

Operacji usunięcia wierszy za pomocą polecenia TRUNCATE nie można cofnąć. Polecenie to nie powoduje uruchomienia wyzwalaczy, np. jeśli zdefiniowano triggery, które powodują usunięcie wierszy zależnych od innych wierszy w tabeli.

Opcją domyślną w poleceniu TRUNCATE jest DROP STORAGE — zwolnienie pamięci wykorzystywanej przez tabele. Utrzymanie pamięci wymaga użycia polecenia:

```
TRUNCATE TABLE <nazwa> REUSE STORAGE;
```

Przykłady usuwania wierszy z tabeli są zamieszczone w rozdziale dotyczącym zastosowania podzapytania do instrukcji DELETE.

5.4. Zadania

1. Zarejestruj jednego pracownika, wpisując dane do tablicy.
2. Przekopiuj do tablicy Osoby pracowników, którzy mają najniższe stawki w swoich działach.
3. Zmień nazwisko wybranego pracownika w tablicy Pracownicy.
4. Zaktualizuj dane, przeprowadzając modyfikację nazwiska w tablicy Osoby na podstawie danych w tablicy Pracownicy.
5. Usuń w tablicy Osoby pracownika z najwyższą stawką.
6. Wymień 5 różnić pomiędzy poleceniami DELETE i TRUNCATE.

7. Przeprowadź podwyżkę stawki o 10% dla pracowników działu o kodzie AD.
8. Wpisz do tablicy Osoby należny pracownikom zasiłek, wyliczony według zasady: prawo do zasiłku mają osoby ze stawką mniejszą niż 10 zł, dostają wtedy zasiłek w wysokości 25% pensji, pozostali nie dostają zasiłku.
9. Usuń z tablicy Towar opisy towarów wyczerpanych (nieposiadanych na stanie).
10. Usuń wszystkie dane z tablicy Osoby — omów wszystkie wersje tego zadania.

Rozdział 6. Język definiowania danych — DDL

Grupa poleceń nazywana językiem definiowania danych to CREATE, ALTER, DROP. Służą one do tworzenia, modyfikowania i usuwania w bazie obiektów takich jak tablice i widoki, ale również procedury, funkcje, użytkownicy i inne. W tym opracowaniu ograniczymy się do omówienia poleceń DDL w odniesieniu do tablic fizycznych.

6.1. Polecenie CREATE

Polecenie CREATE ma postać:

```
CREATE TABLE nazwa_tablicy
  (nazwa_kolumny_1 typ_kolumny [DEFAULT wyrażenie]  [ograniczenie
kolumnowe]…[ograniczenia tablicowe]..,  nazwa_kolumny_2
typ_kolumny..[…] );
```

przy czym:

- *nazwa_tablicy* — musi być niepowtarzalna w obrębie schematu, w którym jest tworzona.

Zgodnie z regułami nazewnictwa obiektów baz danych Oracle *nazwa_tablicy* musi zaczynać się od liter A – Z lub a – z. Może zawierać litery, cyfry, znak podkreślenia _ oraz znaki $ i #. Długość nazwy nie może przekraczać 30 znaków.

Ograniczenia kolumnowe są definiowane w następujący sposób:

```
[ CONSTRAINT nazwa ] {NULL | NOT NULL | UNIQUE | PRIMARY KEY}
| CHECK (warunek dla kolumn).
| REFERENCES tablica(kolumna) [ON DELETE CASCADE]
```

Ograniczenia tablicowe mogą odwoływać się do więcej niż jednego atrybutu, np.

```
[ CONSTRAINT nazwa ]
{UNIQUE | PRIMARY KEY} (kolumna [,kolumna]..)
| FOREIGN KEY(kolumna [,kolumna]..)
REFERENCES tablica (kolumna [,kolumna]..) [ON DELETE CASCADE]
| CHECK (warunek_dla_kolumn)}
```

Każde ograniczenie może mieć przypisaną nazwę, zdefiniowaną po słowie kluczowym CONSTRAINT, np. definicja klucza głównego:

```
ID_OSOBY NUMBER(2) constraint id_os_pk PRIMARY KEY
```

Przykład 6.1. _____

Utwórz tablicę WPLATA. Zdefiniuj klucz główny tablicy i wartość domyślną dla wybranego atrybutu.

```
CREATE TABLE WPLATA
```

```
(NR NUMBER(6) constraint kl_gl PRIMARY KEY,
KLASA VARCHAR2(5) NOT NULL,
SZKOLA VARCHAR2(50),
KWOTA NUMBER (10,2) DEFAULT 0);
```

Wartość domyślna zostaje zapamiętana w wypadku pominięcia atrybutu KWOTA podczas wprowadzania danych do tablicy.

Przykład 6.2.

Utwórz tablicę NAGRODY powiązaną z tabelą Pracownicy. Zdefiniuj klucze — główny i obcy — oraz ograniczenia kolumnowe.

```
CREATE TABLE NAGRODY
(ID_NAGRODY NUMBER(3) constraint id_pk PRIMARY KEY,
RODZAJ_NAGRODY VARCHAR2(15) constraint rodzaj_ch CHECK
(RODZAJ_NAGRODY IN ('Dyplom', 'Certyfikat', 'Pochwała', 'Nagr.Pieniężna')),
ID_OSOBY CHAR(5) NOT NULL constraint ad REFERENCES PRACOWNICY(IDENTYFIKATOR) ON
CASCADE);
```

DELETE

RODZAJ_NAGRODY jest atrybutem wyliczeniowym — może on przyjmować tylko wybraną wartość z listy. Klauzula ON DELETE CASCADE w ograniczeniach ostatniej kolumny zadziała w ten sposób, że usunięcie osoby w tablicy Pracownicy spowoduje usuniecie wszystkich jej wpłat.

Przykład 6.3.

Tworzenie tabeli na podstawie innej tabeli (składnia takiego polecenia jest omówiona w rozdziale dotyczącym podzapytań).

```
CREATE TABLE OSOBA  AS SELECT * FROM PRACOWNICY;
```

W nowej tabeli utworzonej przy użyciu powyższego polecenia pojawią się również dane przekopiowane z istniejącej tabeli. Tę technikę można także wykorzystać do tworzenia tabeli z kolumnami o definicjach takich jak w tabeli źródłowej, ale bez wierszy. Wystarczy utworzyć klauzulę WHERE, która nie wybierze żadnych wierszy ze starej tabeli. Ograniczenia z tabeli źródłowej, w tym NOT NULL, nie są automatycznie tworzone w nowej tabeli.

Polecenie create table .. as select... nie zadziała, jeśli jedna z wybieranych kolumn jest typu LONG. Jeśli jedna z kolumn została wygenerowana za pomocą formuły, należy wprowadzić alias kolumny, który system Oracle wykorzysta jako nazwę kolumny podczas tworzenia tabeli.

Jeśli w bazie Oracle jakaś tablica ma mieć kolumnę licznikową, która będzie inkrementowana w trakcie wstawiania wierszy, to musi zostać zdefiniowany dodatkowy obiekt — **sekwencja** (ang. *sequencer*), która automatycznie zwiększa swoją wartość po każdorazowym odczycie.

Sekwencję tworzy się, korzystając z polecenia:

```
CREATE SEQUENCE <nazwa>
[{INCREMENT BY <parametr> }
|{START WITH <wartość> }
|{MAXVALUE<wartość> |NOMAXVALUE }
|{MINVALUE/<wartość>| NOMINVALUE }
|{CYCLE|NOCYCLE }
|{CACHE|NOCACHE }];
```

Poniżej zostały omówione poszczególne parametry:

- INCREMENT BY <*parametr*> — parametr wskazuje, o ile ma zmieniać się kolejna wartość generowana przez sekwencję (!= 0),
- START WITH <*wartość*> — wartość, od której rozpoczyna się sekwencja,
- MAXVALUE<*wartość*> — ewentualne podanie wartości maksymalnej, jaką może wygenerować sekwencja,
- NOMAXVALUE — oznacza dla sekwencji rosnących 10^{27}, a dla malejących –1,
- MINVALUE/<*wartość*> — wartość minimalna, jaką może wygenerować sekwencja,
- NOMINVALUE — dla sekwencji rosnących oznacza, że minimalna wartość wynosi 1,
- CYCLE — dla sekwencji rosnących oznacza, że po dojściu do wartości maksymalnej dalej następuje generowanie od wartości minimalnej; dla sekwencji malejących natomiast: po dojściu do wartości minimalnej dalej są generowane wartości zaczynające się od maksymalnej,
- NOCYCLE — sekwencja nie startuje cyklicznie,
- CACHE — jest podawana liczba wartości, które są wcześniej wygenerowane i przechowywane w pamięci w celu zwiększenia szybkości działania sekwencji,
- NOCACHE — wszystkie wartości są generowane na bieżąco, na żądanie.

Wartości generowane przez sekwencję pobiera się za pomocą dwóch pseudokolumn NextVal i CurrVal podczas wstawiania danych do tablicy — lub za pomocą polecenia SELECT na tablicy DUAL. Zastosowanie sekwencji podczas wykonywania polecenia INSERT w celu zapewnienia unikalnych wartości dla klucza głównego tablicy jest przedstawione w rozdziale 5. w przykładzie 5.4. Utworzenie sekwencji i pobranie kolejnych wygenerowanych wartości prezentuje przykład poniżej.

Przykład 6.4. _____

Utworzenie sekwencji o nazwie ID wykorzystywanej w przykładzie 5.4.

```
CREATE SEQUENCE ID;
```

Jest to sekwencja rosnąca, która wystartuje od 1 — odstęp pomiędzy kolejnymi liczbami jest równy 1, domyślnie nie ma podanych ograniczeń.

Pobranie wartości wygenerowanych przez sekwencję za pomocą tablicy DUAL:

```
SELECT ID.NEXTVAL FROM DUAL;
SELECT ID.CURRVAL FROM DUAL;
```

Odwołanie NextVal dla sekwencji ID jest żądaniem do systemu Oracle o pobranie kolejnego dostępnego numeru sekwencji ID. Dla tego numeru system gwarantuje niepowtarzalność.

Aby wykorzystać dany numer sekwencji wielokrotnie, odwołujemy się do CurrVal, gdzie jest przechowywana wartość z sekwencji do czasu ponownego wywołania NextVal.

Przykład 6.5. Utworzenie sekwencji o nazwie NUM, która, startując od 0, będzie generować kolejne liczby naturalne aż do 9999 — ciąg ten zostanie wygenerowany jednokrotnie.

```
CREATE SEQUENCE NUM
INCREMENT BY 1
START WITH 0
MAXVALUE 9999
MINVALUE 0
NOCYCLE
CACHE 20;
```

Usuwanie sekwencji przy użyciu polecenia:

```
DROP SEQUENCE NUM;
```

6.2. Polecenie ALTER

W poleceniu ALTER TABLE *nazwa_tablicy* należy zdefiniować rodzaj modyfikacji. Modyfikacja może mieć następujące postacie:

- dodanie kolumny:

```
ALTER TABLE nazwa_tablicy
ADD (nazwa_kolumny typ_kolumny [ograniczenia_kolumnowe | ograniczenia_tablicowe);
```

- modyfikacja istniejącej kolumny:

```
ALTER TABLE nazwa_tablicy
MODIFY(nazwa_kolumny nowy_typ_kolumny [nowe_ograniczenia_kolumnowe |
nowe_ograniczenia_tablicowe);
```

- usunięcie zdefiniowanych ograniczeń:

```
ALTER TABLE nazwa_tablicy
 DROP CONSTRAINT nazwa_ograniczenia; ♦
```

usunięcie kolumny:

```
 ALTER TABLE nazwa_tablicy
DROP COLUMN nazwa_kolumny; ♦
```

zmiana nazwy kolumny:

```
ALTER TABLE nazwa_tablicy
 RENAME COLUMN stara_nazwa_kolumny TO nowa_nazwa_kolumny;
```

- zmiana nazwy tabeli:

```
RENAME stara_nazwa_tabeli TO nowa_nazwa_tabeli;
```

- ustawienie tabeli — tylko do odczytu lub do odczytu i zapisu:

```
ALTER TABLE nazwa_tabeli READ ONLY;
ALTER TABLE nazwa_tabeli READ WRITE;
```

Poniżej znajduje się kilka uwag dotyczących usuwania kolumn.

Usunięcie kolumny z listy kolumn tabeli jest proste, natomiast skomplikowane i czasochłonne jest odzyskanie miejsca, które zajmowała kolumna.

- Kolumnę można usunąć natychmiast całkowicie (może to mieć wpływ na wydajność), korzystając z polecenia:
  ```
  …DROP COLUMN
  ```
- lub oznaczyć ją jako nieużywaną, przeznaczoną do usunięcia w późniejszym terminie.
  ```
  ALTER TABLE nazwa_tablicy SET UNUSET COLUMN nazwa_kolumny;
  ```
- Oznaczenie kolumny jako nieużywanej nie zwalnia miejsca poprzednio zajmowanego przez kolumnę. Prowadzi do tego jedynie rzeczywiste usunięcie nieużywanych kolumn:
  ```
  ALTER TABLE nazwa_tablicy DROP UNUSED COLUMNS;
  ```

Do kolumny oznaczonej jako „nieużywana" nie ma dostępu.

Podczas usuwania wielu kolumn nie należy stosować słowa kluczowego COLUMN polecenia ALTER TABLE — jego użycie spowoduje błąd składni. Nazwy kolumn przeznaczonych do usunięcia należy ująć w nawiasy.

Kilka uwag dotyczących modyfikacji kolumn:

- Można zmieniać typ kolumny tylko wtedy, gdy dane w kolumnie mogą być przekonwertowane, np. można zmienić typ numeryczny na znakowy, ale nie odwrotnie. Typ znakowy na numeryczny można zmienić jedynie w wypadku, gdy kolumna zawiera liczbowe dane.
- Kolumna NOT NULL może być dodana tylko do pustej tabeli.
- Kolumna może być zmieniona na NOT NULL tylko wtedy, gdy w żadnym wierszu nie znajduje się wartość NULL lub gdy w czasie modyfikacji została podana wartość domyślna.

Mając na uwadze ograniczenia modyfikacji wymienione powyżej, warto się zastanowić, jak wykonać zadanie polegające na dodaniu do istniejącej już tablicy z danymi nowej kolumny z ograniczeniem NOT NULL, czyli kolumny z wymaganymi obligatoryjnie wartościami. Jak ominąć ograniczenie polegające na tym, że dodanie kolumny NOT NULL działa tylko wtedy, gdy tabela jest pusta? Oto rozwiązania tego zadania prezentowane w następnym przykładzie.

Przykład 6.6. _____

Dodaj do tabeli kolumnę zdefiniowaną jako NOT NULL, jeśli w tabeli były wstawione dane.

Takie zadanie można wykonać dwoma sposobami.

Pierwszy wymaga zrealizowania dwóch etapów.

a) Należy dodać kolumnę bez ograniczenia dotyczącego wymaganych wartości:

```
ALTER TABLE nazwa
ADD (nazwa_kol typ);
```

b) Następnie wstawić do nowej kolumny potrzebne wartości i potem dodać ograniczenie NOT NULL przy użyciu następującego polecenia:

```
ALTER TABLE nazwa
MODIFY (nazwa_kol typ NOT NULL);
```

Drugi sposób rozwiązania tego problemu to dodanie nowej kolumny połączone z podaniem domyślnych wartości:

```
ALTER TABLE nazwa
ADD (nazwa_kol typ  default 'aaa' NOT NULL);
```

Przykład 6.7.

Zdefiniuj w istniejącej tablicy klucz główny.

```
ALTER TABLE PRACOWNICY
ADD CONSTRAINT PK PRIMARY KEY (IDENTYFIKATOR);
```

Przykład 6.8.

Zdefiniuj w istniejącej tablicy klucz obcy.

```
ALTER TABLE NOWA
ADD CONSTRAINT FK FOREIGN KEY (ID)
REFERENCES PRACOWNICY (IDENTYFIKATOR);
```

W tablicy NOWA pole ID jest kluczem obcym, gdyż czerpie wartości z kolumny IDENTYFIKATOR w tablicy PRACOWNICY.

6.3. Polecenie DROP

Usuwanie tablicy realizuje polecenie:

```
DROP TABLE <nazwa>
[CASCADE CONSTRAINS];
```

Klauzula CASCADE CONSTRAINS powoduje usunięcie ograniczeń łączących usuwaną tablicę z tablicami powiązanymi.

Usunięte tabele utrzymują zajmowane miejsce, są tylko logicznie przenoszone do kosza. Przypomina to proces usuwania obiektów z wykorzystaniem kosza w systemach windowsowych.

W Oracle od wersji 10g został wprowadzony mechanizm FLASHBACK TABLE, który umożliwia przywrócenie tabeli do stanu z określonego czasu oraz odzyskanie tabeli usuniętej za pomocą polecenia DROP TABLE. Prześledźmy kolejny przykład.

Przykład 6.9.

Odzyskiwanie usuniętej tabeli z wykorzystaniem mechanizmu FLASHBACK TABLE.

Przy użyciu poniższego polecenia sprawdzamy, jakie tabele dostępne są w naszym schemacie bazy:

```
SELECT * FROM TAB;
```

Otrzymujemy wynik:

TNAME	TABTYPE
PRACOWNICY	TABLE
OSOBA	TABLE
KIEROWNICY	TABLE
DZIALY	TABLE
ADRESY	TABLE

Wybieramy tabelę do usunięcia, niech to będzie np. tabela OSOBA:

```
DROP TABLE OSOBA;
```

Otrzymujemy potwierdzenie wykonania zadania: table

```
OSOBA dropped.
```

Próba dostępu do tej tablicy przy użyciu polecenia np.

```
SELECT * FROM OSOBA;
```

kończy się komunikatem systemu: tabela lub perspektywa nie istnieje.

Po usunięciu tabela jest niedostępna dla użytkownika. Sprawdzamy ponownie dostępne tabele:

```
SELECT * FROM TAB;
```

Otrzymujemy informację:

TNAME	TABTYPE
PRACOWNICY	TABLE
KIEROWNICY	TABLE
DZIALY	TABLE
BIN$CbBig7fZ2cXgUJ6bLXAp0g==$0	TABLE
ADRESY	TABLE

Okazuje się, że usunięta tabela występuje teraz z nową, systemową nazwą. Nadal pozostaje w tej samej przestrzeni tabel. Nie zmieniła fizycznie swojego miejsca, znajduje się w koszu. Można zobaczyć usunięte obiekty dostępne w koszu, korzystając z polecenia: SHOW RECYCLEBIN;

Po uruchomieniu tego polecenia otrzymamy wynik: table OSOBA dropped.

ORIGINAL NAME	RECYCLEBIN NAME	OBJECT TYPE	DROP TIME
OSOBA	BIN$CbBig7fZ2cXgUJ6bLXAp0g==$0	TABLE	2014-12-08:08:29:20

Usunięty obiekt występuje tutaj z oryginalną nazwą oraz nazwą systemową. Spróbujmy odzyskać tabelę:

```
FLASHBACK TABLE OSOBA TO BEFORE DROP;
```

Sprawdzamy teraz dostępne tabele:

```
SELECT * FROM TAB;
```

I otrzymujemy oczekiwany wynik:

TNAME	TABTYPE
PRACOWNICY	TABLE
OSOBA	TABLE
KIEROWNICY	TABLE
DZIALY	TABLE
ADRESY	TABLE

Choć usuwane obiekty nadal przechowywane są w przestrzeni tabel, a tylko „niewidoczne" dla użytkownika przetrzymywane są w koszu, przestrzeń tablic nigdy nie zostanie wypełniona elementami kosza do stanu uniemożliwiającego dalszą pracę z bazą. W przestrzeni tabel zawsze musi być dostępne miejsce na zapisanie nowych obiektów lub nowych danych. W bazie uruchamiana jest samoczynnie procedura stopniowego opróżniania kosza zgodnie z zasadą *first-in-first-out*. Obiekty najstarsze usuwane są jako pierwsze. Wcześniej usuwane są dodatkowe obiekty związane z tabelą takie jak np. indeksy czy triggery, a na końcu tabela. Możliwe jest również selektywne usuwanie obiektów danego użytkownika.

Poniżej znajduje się kilka poleceń służących do częściowego lub całkowitego opróżniania kosza. Polecenia te może uruchamiać użytkownik w odniesieniu do własnych zasobów. Wszystkie opcje administrowania koszem są w gestii administratora bazy.

Usunięcie tabeli z wyrzuceniem jej z kosza możemy osiągnąć, używając polecenia:

```
DROP TABLE <nazwa> PURGE;
```

Wyczyszczenie kosza i wyrzucenie wcześniej usuniętej tabeli:

```
PURGE TABLE <nazwa>;
```

Opróżnienie kosza:

```
PURGE RECYCLEBIN;
```

6.4. Zadania

1. Zdefiniuj tablicę o podanym schemacie:
   ZESPÓŁ
   Id_zespołu NUMBER(2) klucz główny,
   Nazwa Varchar2(20) atrybut z wartościami obligatoryjnymi,
   Adres Varchar(20)
   Nr_kierownika Char(5) klucz obcy, referencja do atrybutu Identyfikator □w tablicy Pracownicy.
2. Do istniejącej tablicy Zespół dodaj atrybut Nr_telefonu typu znakowego z wartościami opcjonalnymi.
3. W tablicy Zespół zmień nazwę kolumny Nazwa na Typ, zdefiniuj ograniczenie, aby była to kolumna z wartością domyślną.
4. Zmień maksymalny rozmiar pola Adres do 30 znaków.
5. Usuń z tablicy Zespół kolumnę Typ.
6. Omów cel definiowania własnej dziedziny i własnego typu danych.
7. Podaj różnice między poleceniami DROP i TRUNCATE.
8. Zdefiniuj tablicę Wypożyczenie, gdzie id jest kluczem głównym, zaś id_czytelnika kluczem obcym wskazującym na tabelę Czytelnik, a pole data_wypożyczenia przyjmuje domyślnie wartość zmiennej SYSDATE, czyli aktualną datę.
9. W tablicy Wypożyczenie zdefiniuj ograniczenie dla kolumny Data_zwrotu, aby w tym polu nie mogła być data wcześniejsza niż data wypożyczenia.
10. Zdefiniuj tablicę PozycjaZamowienia, gdzie id jest kluczem głównym, id_towaru kluczem obcym pochodzącym z tabeli Towar, a pole ilość musi być większe od 0 i przyjmuje domyślnie wartość 1.

Rozdział 7. Rozpoczęcie pracy z bazą

Aby można było nawiązać połączenie z bazą danych Oracle, na komputerze musi być zainstalowane oprogramowanie klienckie, np. SQL*Plus lub Oracle SQL Developer. Najczęściej w trakcie instalowania systemu zarządzania bazą danych jest instalowane oprogramowanie zarówno serwera, jak i klienta.

Poniżej zostaną omówione sposoby logowania się do bazy za pomocą obu tych narzędzi.

7.1. Logowanie za pomocą SQL*Plus

SQL*Plus jest prostym podstawowym narzędziem do komunikowania się z bazą Oracle. Umożliwia definiowanie zapytań do bazy, wyświetlanie wyników, definiowanie nowych i modyfikację istniejących obiektów w bazie. Obsługuje polecenia SQL i posiada też zestaw własnych komend — umożliwia wykonywanie skryptów z zapisanymi poleceniami. Pracuje w trybie tekstowym, wymusza więc na użytkowniku przyswojenie podstawowych komend wpisywanych w linii poleceń. Wielkość liter w poleceniu nie ma znaczenia. Wielu poleceń można używać w formie skróconej do czterech znaków — szczegółów należy szukać w dokumentacji. Polecenie kończy się średnikiem lub ukośnikiem (/). Ukośnik również uruchamia ponownie ostatnio wykonywane polecenie.

7.1.1. Podstawowe polecenia SQL*Plus

Poniżej przedstawiam omówienie podstawowych i najczęściej używanych poleceń SQL*Plus.

CONNECT — polecenie do nawiązania połączenia z bazą. Składnia tego polecenia wymaga podania nazwy użytkownika, hasła i nazwy bazy: connect nazwa_uzytkownika/hasło@nazwa_bazy_według_deskryptora_połączeń

DISCONNECT — polecenie kończy sesję pracy użytkownika. Taki sam efekt uzyskamy, logując się w trakcie pracy na konto innego użytkownika.

EXIT — kończy sesję, zamyka program.

Polecenie HELP INDEX lub HELP <*polecenie*> spowoduje wyświetlenie listy dostępnych poleceń lub składnię konkretnego polecenia.

DESC <*nazwa_tabeli*> — wyświetla strukturę tabeli (DESC to skrót polecenia DESCRIBE).

Polecenia wielokrotnie uruchamiane warto zapisać w oddzielnym pliku, aby w dowolnej chwili uruchomić skrypt SQL-owy. Należy w tym celu uruchomić polecenie EDIT *nazwa_pliku*. To polecenie otworzy edytor tekstu — zwykle jest to Notepad — w którym należy wpisać treść polecenia, zapisać i zamknąć plik.

Polecenie *@nazwa_skryptu* lub START *nazwa_skryptu* spowoduje wykonanie poleceń zapisanych w pliku.

SPOOL — polecenie pozwala zapisać do pliku wszystkie wykonywane komendy i ich wyniki. Wszystkie te dane będą przechowywane w pamięci operacyjnej do czasu wydania polecenia SPOOL OFF.

Oto kilka poleceń przydatnych podczas rozpoczynania pracy z bazą:

1. Polecenie sprawdza, na jakim użytkowniku pracujemy:
   ```
   show user;
   ```
2. Polecenie sprawdza tablice należące do użytkownika:
   ```
   select table_name from user_tables;
   ```
3. Polecenie sprawdzi, jaki jest schemat tabeli OSOBA:
   ```
   desc OSOBA;
   ```
4. Polecenie wypisze prawa dostępu do kolumn, których użytkownik jest właścicielem lub do jakich posiada dostęp: `select table_name from user_col_privs;`

7.2. Logowanie za pomocą Oracle SQL Developer

Program Oracle SQL Developer posiada graficzny interfejs umożliwiający szybki dostęp do wszystkich typów obiektów i łatwe administrowanie bazą. Można z tego programu korzystać, jeśli łączymy się poprzez sieć z serwerem, na którym jest zainstalowana baza danych i jest otwarty port nasłuchu lub oprogramowanie klienckie i serwer są na tym samym komputerze.

Tworzymy nowe połączenie, klikając zielony krzyżyk w lewym górnym rogu zakładki *Connections*. Wszystkie parametry logowania wpisujemy w oknie właściwości połączenia. Należy tam uzupełnić:

- *Connection Name* — nazwa własna połączenia.
- *User Name* — nazwa użytkownika, na koncie którego rozpoczynamy logowanie. Ten użytkownik ma w systemie określone uprawnienia, z których będziemy korzystać.

- *Password* — hasło użytkownika. Poniżej jest opcja *Save Password* — jeśli tego nie zaznaczymy, to hasło należy wprowadzać przy każdym kolejnym logowaniu.
- *Host name* — tu należy wpisać nazwę lub adres IP hosta, na którym znajduje się serwer Oracle. Jeśli baza znajduje się na tym samym komputerze, co klient, wpisujemy — localhost.

- *Port* — port sieciowy, pod którym nasłuchuje serwer bazy danych. Domyślnie jest to 1521.
- *SID* — to systemowy identyfikator bazy danych, nazwa własna.

Jeśli logujemy się jako administrator — użytkownik SYS — należy w okienku *Role* (jest to lista rozwijana) wybrać opcję SYSDBA.

Następnie warto przetestować połączenie przy użyciu przycisku *TEST*, a kiedy w lewym dolnym rogu uzyskamy informację o statusie połączenia — *Success*, wówczas rozpoczynamy sesję przyciskiem *Connect*.

Po uruchomieniu nowego połączenia w zakładce *Connections* mamy widoczną nazwę sesji. Kliknięcie krzyżyka z lewej strony pozwoli na rozwinięcie listy poszczególnych typów obiektów w bazie. Rozwijając kolejne gałęzie, można zobaczyć wszystkie obiekty danej kategorii, np. tablice, struktury tablic, a nawet dane. Klikając prawym przyciskiem myszy nazwę sesji, z otwartego menu możemy wybrać polecenie *Open SQL Worksheet*, które uruchomi okno edytora kodu. Tu wpisujemy i uruchamiamy polecenia SQL. Pojedyncze polecenie uruchamiamy przyciskiem *Run Statement* (*Ctrl + Enter*) — jest to przycisk oznaczony zielonym trójkątem. Jeśli chcemy wykonać rozbudowany program, uruchamiamy cały ciąg poleceń, używając przycisku *Run Script* (*F5*).

Bibliografia

[1] Allen S., *Modelowanie danych*, Wydawnictwo Helion, Gliwice 2006.

[2] Banachowski L., Chądzyńska A., Matejewski K., *Relacyjne bazy danych. Wykłady i ćwiczenia*, Wydawnictwo PJWSTK, Warszawa 2009.

[3] Banachowski L., Mrówka E., Stencel K., *Systemy baz danych. Wykłady i ćwiczenia*, Wydawnictwo PJWSTK, Warszawa 2004.

[4] Barker R., *Case Method. Modelowanie związków encji*, Wydawnictwo WNT, Warszawa 1996.

[5] Beynon-Davies P., *Systemy baz danych*, Wydawnictwo WNT, Warszawa 2003.

[6] Celko J., *SQL. Zaawansowane techniki programowania*, Wydawnictwo PWN, Warszawa 2008.

[7] Coburn R., *SQL dla każdego*, Wydawnictwo Helion, Gliwice 2001.

[8] Elmasri R., Navathe S.B., *Wprowadzenie do systemów baz danych*, Wydawnictwo Helion, Gliwice 2005.

[9] Garcia-Molina H., Ullman J.D., Widom J., *Systemy baz danych. Kompletny podręcznik*, Wydawnictwo Helion, Gliwice 2011.

[10] Garcia-Molina H., Ullman J.D., Widom J., *Systemy baz danych. Pełny wykład*, Wydawnictwo WNT, Warszawa 2006.

[11] Hernandez M. J., *Projektowanie baz danych dla każdego. Przewodnik krok po kroku*, Wydawnictwo Helion, Gliwice 2014.

[12] Kroenke D.M., *Database Processing. Fundamentals, design and implementation*, Pearson Education Inc., New Jersey 2004.

[13] Ladanyi H., *SQL. Księga eksperta*, Wydawnictwo Helion, Gliwice 2000.

[14] Łojewski Z., *Bazy danych — teoria i praktyka*, Wydawnictwo UMCS, Lublin 2011.

[15] Mazur H., Mazur Z., *Projektowanie relacyjnych baz danych*, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2004.

[16] Pelikant A., *Bazy danych: pierwsze starcie*, Wydawnictwo Helion, Gliwice 2009.

[17] Rogulski M., *Bazy danych dla studentów. Podstawy projektowania i języka SQL*, Wydawnictwo WITKOM, Warszawa 2012.

[18] Ullman J.D., Widom J., *Podstawowy kurs systemów baz danych*, Wydawnictwo Helion, Gliwice 2011.

[19] Urbanowicz P., Płonkowski M., Urbanowicz D., *Bazy danych. Teoria i praktyka. Podręcznik dla studentów uczelni wyższych*, Wydawnictwo KUL, Lublin 2000.

[20] Whitehorn M., Marklyn B., *Relacyjne bazy danych. Teoria i praktyka projektowania relacyjnych baz danych*, Wydawnictwo Helion, Gliwice 2003.

Spis rysunków

Spis tabel

Skorowidz

A

B

C

D

E

F

G

I

J

K

L

Ł

M

N

O

P

R

S

T

U

W

Z