

Biblioteka Główna i OINT
Politechniki Wrocławskiej



100100257168

Oficyna Wydawnicza Politechniki Wrocławskiej

Hanna Mazur
Zygmunt Mazur

**PROJEKTOWANIE
RELACYJNYCH
BAZ DANYCH**



PROJEKTOWANIE RELACYJNYCH BAZ DANYCH

Hanna Mazur
Zygmunt Mazur



Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2004

MS 116279

Recenzent

Jacek GRUBER

Redakcja i projekt okładki

Hanna MAZUR

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości
lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione.



319352 W/26

© Copyright by Hanna Mazur & Zygmunt Mazur, Wrocław 2004

OFICyna WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

ISBN 83-7085-837-6

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam. nr 847/2004.

SPIS TREŚCI

Wstęp	5
1. Etapy projektowania systemów bazodanowych	9
2. Podstawy relacyjnych baz danych	12
2.1. Relacyjny model danych	12
2.2. Podstawowe operacje algebry relacji	16
2.3. Normalizacja schematów relacji	36
2.4. Podstawowe pojęcia baz danych	49
3. Projektowanie konceptualne	53
3.1. Faza analizy	53
3.1.1. Analiza wycinka rzeczywistości	54
3.1.2. Słownik pojęć	58
3.1.3. Analiza istniejącej bazy danych	58
3.1.4. Analiza wymagań funkcjonalnych	59
3.1.5. Analiza wymagań niefunkcjonalnych	59
3.1.6. Analiza kosztów	60
3.2. Definicje kategorii	60
3.3. Reguły funkcjonowania	62
3.4. Ograniczenia dziedzinowe	62
3.5. Transakcje	63
3.6. Model konceptualny	65
3.6.1. Identyfikacja encji	65
3.6.2. Związki i typy związków	69
3.6.3. Definicje predykatowe encji i związków	74
3.6.3.1. Definicje predykatowe typów encji	74
3.6.3.2. Definicje predykatowe typów związków	75
3.6.4. Diagram związków encji	76

4. Projektowanie logiczne	85
4.1. Transformacja modelu konceptualnego do modelu logicznego	85
4.1.1. Reguły transformacji	85
4.1.2. Definicje schematów relacji	99
4.1.3. Schemat relacyjnej bazy danych	101
4.1.4. Słownik atrybutów	102
4.2. Definiowanie perspektyw	104
5. Zakończenie	107
6. Zestawienie etapów projektowania relacyjnej bazy danych	109
7. Literatura	110
Dodatek	113

*Uczcie się abecadła nauki,
zanim będziecie próbowali wspiąć się na jej szczyty.
Nigdy nie bierzcie się za rzeczy dalsze,
dopóki nie przyswoiliście wcześniejszych.
Nigdy nie myślcie, że wiecie już wszystko.
Bez względu na to jak Was cenią,
miejcie zawsze odwagę powiedzieć sobie:
nic nie wiem.”
- J. Pawłow*

WSTĘP

Większość firm i przedsiębiorstw, niezależnie od ich wielkości, ma problem z odpowiednim przechowywaniem danych umożliwiającym szybkie wyszukiwanie, dopisywanie, aktualizację oraz usuwanie danych. Z problemem tym spotykają się również osoby prowadzące działalność gospodarczą oraz te, które chciałyby gromadzić i przetwarzać różne dane tematyczne, na przykład o posiadanych i pożyczanych znajomym książkach, o zdjęciach i ważnych wydarzeniach, o utworach muzycznych, autorach i wykonawcach itd. Ogólna dostępność i powszechność komputerów zachęca ludzi do przechowywania danych w postaci elektronicznej. Dzięki nowoczesnym komputerom i wielu różnorodnym systemom zarządzania bazami danych możliwe i opłacalne jest gromadzenie i przetwarzanie zarówno małej jak i bardzo dużej liczby danych, związanych praktycznie z każdą dziedziną życia (firmy, gospodarstwa domowe, szpitale, banki, biblioteki, urzędy, uczelnie i wiele innych).

Nowoczesne systemy zarządzania bazami danych (SZBD, ang. *Database Management System – DBMS*) pozwalają na proste formułowanie zapytań do bazy danych i szybkie wyszukiwanie odpowiednich danych z baz danych. W wielu publikacjach i dokumentacjach firmowych narzędzi, główny nacisk kładzie się nie na projekt bazy danych czy aplikacji, ale na opisy składni:

- języka definiowania (opisu) danych (*DDL* – ang. *Data Definition (Description) Language*),
- języka manipulowania danymi (*DML* – ang. *Data Manipulation Language*),
- języka zapytań (wyszukiwania) danych (np. *QBE* – ang. *Query By Example*),

- języka nadzoru (DCL — ang. *Data Control Language*), który obejmuje nadawanie i odbieranie uprawnień użytkownikom, autoryzację (ang. *authorization*) dostępu do bazy danych – zajmuje się tym administrator bazy danych (*DBA* – ang. *Database Administrator*).

Wynika to przede wszystkim z dużej ilości dialektów języka SQL i możliwych odmian języków do wykonywania operacji relacyjnych.

Niniejsza książka ma celu zwrócenie uwagi Czytelnika na podstawowe problemy związane z wykonaniem prawidłowego projektu bazy danych. Jakość poszczególnych etapów projektowania ma bardzo istotny wpływ na efekt końcowy, czyli na ostateczną postać bazy danych.

W starszych systemach bazodanowych, oprogramowanie obsługujące bazę danych często było uzależnione od danych czyli sposób przechowywania fizycznych danych i metody dostępu były określane przez wymagania danej aplikacji. Wiedza o organizacji danych i metodach dostępu była wbudowywana w logikę i kod aplikacji. Aplikacje były zależne od danych w związku z czym nie można było zmieniać struktury przechowywanych danych jak i metod dostępu. Nowoczesne systemy zarządzania bazami danych pozwalają zapewnić niezależność aplikacji od danych, rozumianą jako odporność aplikacji na zmiany w strukturze przechowywanych danych i sposobie dostępu, co znaczy, że zmiany w strukturze danych czy metodach dostępu nie będą wymagały już zmian w oprogramowaniu aplikacji. Postulat niezależności danych pozwala na oddzielenie wykonania projektu bazy danych od projektu aplikacji.

W książce autorzy zajmują się projektowaniem samej bazy danych a nie aplikacji. Umiejętność właściwego zaprojektowania bazy danych jest równie ważna jak umiejętność programowania. W przyszłości planowane jest wydanie książki na temat projektowania aplikacji bazodanowych.

W aplikacjach wykorzystujących źle zaprojektowane bazy danych napotyka się na spore trudności przy próbie rozbudowy aplikacji oraz z utrzymaniem spójności bazy danych. Niekiedy trzeba wykonywać dodatkowe czynności związane z obsługą anomalii aktualizowania danych, dopisywania danych do bazy oraz ich usuwania. Aby baza danych była poprawnie zaprojektowana należy bardzo dokładnie wyodrębnić i przeanalizować, a następnie uwzględnić w projekcie **reguły biznesowe** (ang. *business rules*). Określają one, w jaki sposób dane są tworzone, modyfikowane i usuwane z bazy oraz jakie warunki te dane muszą spełniać.

Nawet w bardzo prostych bazach danych (źle zaprojektowanych!) można napotkać wielkie problemy. Rozważmy prostą bazę danych „PANORAMA FIRM” składającą się tylko z jednego schematu relacji **Panorama** (*Firma, Adres, Towar, Cena*). W bazie mają być gromadzone dane o unikalnych nazwach firm, ich adresach, dostarczanych towarach i aktualnych cenach towarów. W tak źle zaprojektowanej bazie danych wystąpią anomalie dopisywania danych, usuwania i aktualizacji oraz redundancja danych. Zauważmy, że adres firmy będzie się powtarzał tyle razy, ile różnych towarów dostarcza dana firma. Zmiana adresu firmy będzie wymagała aktualizowania wielu wpisów, w zależności od liczby oferowanych produktów. Kluczem relacji jest zbiór atrybutów {*Firma, Towar*}, w związku z czym, nie będzie można wprowadzić

danych o firmie dopóki firma nie będzie miała w swojej ofercie co najmniej jednego towaru. Ponadto, gdyby dana firma przestała dostarczać towary, to jej dane trzeba usunąć z bazy i traciemy w ten sposób informacje, które być może należałoby nadal przechowywać itd. Wszystkie te anomalie mogą być wykluczone po wykonaniu poprawnego projektu bazy danych.

Podstawą tworzenia każdej poprawnie zaprojektowanej bazy danych jest **modelowanie biznesowe** (ang. *business modelling*), czyli proces identyfikacji i opisywania codziennych czynności wykonywanych w przedsiębiorstwie. Konieczne jest uwzględnienie wszystkich wymagań i potrzeb przyszłego użytkownika. Semantyka świata rzeczywistego musi wystąpić w bazie danych. Zasadniczą częścią modelowania biznesowego jest przeprowadzanie wywiadów z odpowiednimi osobami, w tym z przyszłymi potencjalnymi użytkownikami bazy danych. Projektując bazę danych należy mieć na uwadze funkcje, jakie będzie się na tych danych wykonywać. Zły projekt bazy danych bardzo komplikuje zaprojektowanie poprawnie działającej aplikacji spełniającej oczekiwania użytkownika.

Wiele faktów świata rzeczywistego, niejednokrotnie pozornie nieistotnych z punktu widzenia projektanta (np. dostawca musi czy może dostarczać części, część jest dostarczana przez jednego lub wielu dostawców albo nie dostarczana przez nikogo) ma bardzo istotny wpływ na ostateczną postać bazy danych. Prawidłowe wykonanie wszystkich etapów projektowania, opisanych w kolejnych rozdziałach niniejszej pracy, pozwoli na sporządzenie dobrego i w pełni udokumentowanego projektu bazy danych. Zaprojektowana baza danych jest tak dobra jak jej najsłabszy element.

Niniejsza książka dotyczy strukturalnej metodyki projektowania baz danych (jednej z wielu możliwych) z elementami niektórych strukturalnych metodologii projektowych. Jest przeznaczona dla osób zapoznających się z tajnikami projektowania baz danych (dla początkującego projektanta). Może być wykorzystana jako podręcznik dydaktyczny w ramach zajęć związanych z tematyką projektowania relacyjnych baz danych. Stanowi wprowadzenie do zaawansowanego projektowania strukturalnego relacyjnych baz danych narzędziami programowymi CASE (ang. *Computer Aided Software Engineering* – komputerowe wspomaganie inżynierii oprogramowania, *Computer Aided Systems Engineering* – komputerowe wspomaganie inżynierii systemów), wspomagającymi analizę i projektowanie systemów informatycznych wysokiej jakości. Należy podkreślić, że zaprezentowana metodyka ma charakter ręcznego projektowania metodą tworzenia dokumentów projektowych.

Treść książki oparto na notatkach i materiałach do kursów z baz danych prowadzonych dla studentów wyższych uczelni.

Relacyjny model danych dotyczy zagadnień logicznych czyli definicji (struktury) danych, operowania danymi (podstawowe operatory algebry relacji to operatory selekcji, rzutu i złączenia) oraz zapewnienia integralności danych (poprzez określenie reguł, które stany bazy danych są poprawne). Tematyka niniejszej książki jest związana z tymi zagadnieniami.

W rozdziale pierwszym podano krótką charakterystykę etapów projektowania relacyjnych baz danych oraz aplikacji bazodanowych. Rozdział drugi omawia podstawowe pojęcia relacyjnych baz danych takie jak relacja, schemat relacji, operatory algebry relacji, proces normalizacji schematów relacji. W rozdziale trzecim omówiono etapy projektowania conceptualnego od fazy analizy do diagramów związków encji. Rozdział czwarty zawiera opis podstawowych zasad transformacji modelu conceptualnego do relacyjnego modelu logicznego oraz omówienie zagadnień związanych z definiowaniem perspektyw. W dodatku podano wybrane definicje bazy danych dostępne w literaturze.

Autorzy dziękują recenzentowi za komentarze, sugestie i rady.

1. ETAPY PROJEKTOWANIA SYSTEMÓW BAZODANOWYCH

W pracy zostaną omówione najważniejsze pojęcia z teorii relacyjnych baz danych oraz przedstawione kolejne etapy projektowania relacyjnych baz danych. Obecnie jest dostępnych wiele publikacji dotyczących tematyki baz danych, ale są to na ogół pozycje bardzo obszerne (kilkaset stron), poruszające wiele zagadnień (nie zawsze bezpośrednio dotyczących projektowania baz danych), co utrudnia wydobycie informacji istotnych z punktu widzenia projektanta bazy danych.

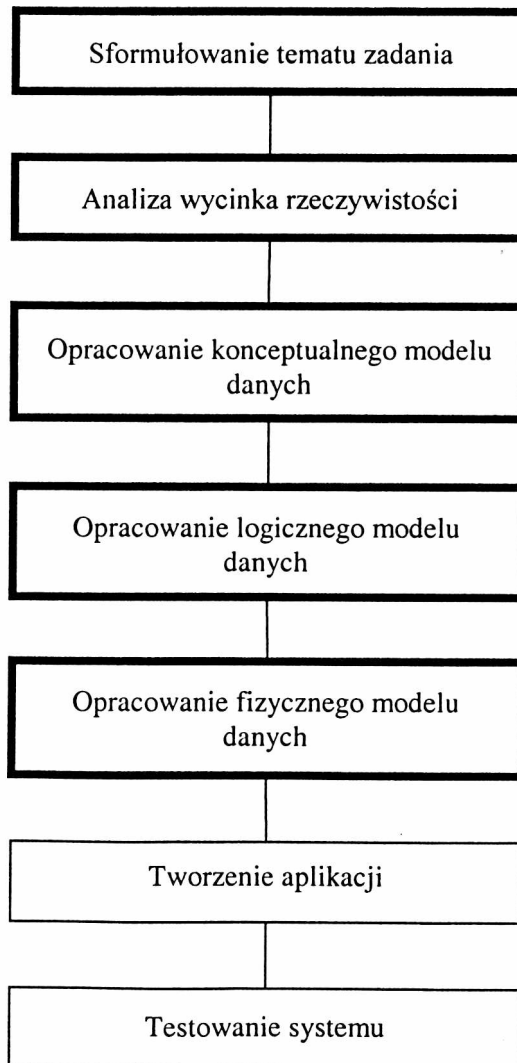
Poprawnie zaprojektowana relacyjna baza danych, musi spełniać wszystkie oczekiwania przyszłego użytkownika a jednocześnie musi gwarantować bezbłędne, wydajne i przyjazne w obsłudze funkcjonowanie. Takie własności można zapewnić, jeśli poprawnie przeprowadzi się wszystkie etapy projektowania bazy danych.

Celem pracy jest przedstawienie w sposób systematyczny i uporządkowany, a zarazem prosty i zrozumiały, kolejnych etapów projektowania relacyjnych baz danych, przy czym autorzy dużą wagę przywiązują do znormalizowanego sposobu dokumentowania poszczególnych etapów w taki sposób, aby na każdym etapie przedstawić używaną notację i pojęcia.

Prawidłowo zaprojektowana baza danych nie powinna zawierać żadnych zbędnych danych, powinna natomiast umożliwiać efektywną pracę zaimplementowanego systemu oraz możliwości jego rozbudowy. W projektowaniu relacyjnej bazy danych można wyróżnić następujące etapy:

- projektowanie koncepcyjne (konceptualne, pojęciowe), polegające na zdefiniowaniu wymagań niezależnych od implementacji. Proces zbierania wymagań powinien doprowadzić do powstania konceptualnego modelu danych (CDM – ang. *Conceptual Data Model*), zwykle przedstawianego w postaci graficznej za pomocą tzw. *diagramów związków encji* (ERD – ang. *Entity Relationship Diagram*). Na tym etapie często przeprowadzany jest również **proces normalizacji**. W fazie tej nie bierze się pod uwagę docelowej, implementacyjnej platformy sprzętowej przyszłego systemu bazy danych.
- projektowanie logiczne, polegające na transformacji modelu konceptualnego do modelu logicznego (w tym przypadku relacyjnego), w którym określamy strukturę przechowywanych danych, czyli tzw. schematy relacji. Na tym etapie często przeprowadza się normalizację schematów relacji. Normalizacja danych jest sformalizowaną procedurą, w wyniku której eliminujemy redundancję (powtarzanie się) danych, ich niespójności i anomalie, czyli nieprawidłowości związane z aktualizacją, usuwaniem i dopisywaniem danych.

- projektowanie fizyczne, w wyniku którego powstaje fizyczny model danych (PDM – ang. *Physical Data Model*). Na tym etapie należy określić narzędzie, jakie będzie wykorzystane do implementacji systemu oraz sprzęt, na którym ma działać docelowy system informatyczny. Elementy logicznego projektu bazy danych zamieniamy na rzeczywiste obiekty bazy danych. Etap ten jest ściśle związany z budową aplikacji bazodanowej.



Rys. 1.1. Etapy projektowania bazy danych oraz tworzenia systemu bazy danych

Kolejne etapy tworzenia systemu bazodanowego, obejmujące etapy projektowania bazy danych jak i projektowania części aplikacyjnej systemu bazy danych można przedstawić jako kaskadę działań projektowych (rysunek 1.1).

Zaprojektowana baza danych może być zaimplementowana w wybranym systemie zarządzania bazą danych (SZBD). W różnych SZBD, bazy danych są przechowywane w odmienny sposób, np. tej samej dziedzinie atrybutów mogą odpowiadać różne typy itd. Ten poziom szczegółowości na ogół jest nieistotny dla projektanta bazy danych, którego interesuje jedynie opracowanie modelu danych oraz określenie sposobu dostępu do danych.

Po prawidłowo przeprowadzonym procesie projektowania bazy danych można **zaprojektować aplikację** obsługującą bazę danych w sposób poprawny, efektywny i przyjazny. Wiedząc już, *co aplikacja ma robić*, należy zastanowić się nad tym *jak to zrobić*. Po całościowym zaprojektowaniu aplikacji można przejść do **zaimplementowania systemu**. W tym celu wykorzystuje się odpowiednio dobrane środowisko. Kolejnym etapem powinno być **testowanie** systemu i uporządkowanie **dokumentacji** obejmującej wszystkie etapy projektowania schematu bazy danych, opis zbudowanego systemu bazodanowego, omówienie sposobu zainstalowania programu, rozpoczęcia pracy z systemem bazy danych oraz instrukcja posługiwania się aplikacją przez przyszłego użytkownika.

Elementy rysunku 1.1, narysowane grubą linią, odpowiadają etapom projektowania schematu bazy danych.

W kolejnych rozdziałach przedstawimy etapy projektowania bazy danych. Najpierw jednak omówimy podstawowe pojęcia z teorii relacyjnych baz danych.

2. PODSTAWY RELACYJNYCH BAZ DANYCH

Baza danych (ang. *database*) jest trwałym zbiorem tematycznie ze sobą powiązanych danych (zob. też Dodatek). Pojęcie bazy danych zostało sformułowane w latach 1962-63. Najbardziej abstrakcyjnym poziomem projektu bazy danych jest model danych, czyli opis pojęciowy przestrzeni zagadnienia [Riordan2000]. W modelu danych nie ma żadnych odniesień do fizycznej struktury systemu.

Wyróżnia się następujące podstawowe modele baz danych:

- hierarchiczny,
- sieciowy,
- relacyjny,
- obiektowy.

Obszerne informacje na temat tych podstawowych modeli można znaleźć w literaturze np. [Date1981, Hernan1998, Pankow1992, UIIWid2000]. W niniejszej pracy skupimy się tylko na **modelu relacyjnym**. Twórcą relacyjnego modelu bazy danych jest E.F.Codd, który w 1970 r. opublikował fundamentalną pracę na temat relacyjnych baz danych [Codd1970].

2.1. RELACYJNY MODEL DANYCH

Modele danych są opisywane w kategoriach encji, atrybutów, dziedzin i powiązań, których definicje podamy w dalszej części pracy.

Ze względu na bogatą dostępną literaturę na temat relacyjnych baz danych (np. [Date2000, DelAdi1989]), ograniczymy się tutaj jedynie do podania podstawowych pojęć teorii relacyjnych baz danych.

Relacja R (ang. *relation*) w sensie matematycznym jest podzbiorem iloczynu kartezjańskiego n ($n > 0$) dowolnych zbiorów wartości co zapisujemy jako:

$$R \subseteq D_1 \times D_2 \times \dots \times D_n$$

gdzie D_i ($i = 1, \dots, n$) jest dowolnym zbiorem wartości (**dziedziną** – ang. *domain*).

Elementami iloczynu kartezjańskiego są n -krotki (zwane też po prostu **krotkami**) wartości $(d_1, d_2, \dots, d_n)$, przy czym

$$d_i \in D_i, \text{ dla } i = 1, \dots, n.$$

Aby uniezależnić relacje od uporządkowania dziedzin, w relacyjnym modelu danych przyjęto trochę inną konwencję notacyjną. Niech $U = \{A_1, \dots, A_n\}$ będzie zbiorem elementów zwanych atrybutami i dla każdego $A \in U$ przyporządkowany jest pewien zbiór wartości $\text{dom}(A)$ zwany dziedziną (domeną) atrybutu.

Atrybut (ang. *attribute*) – opisuje użycie pewnej dziedziny w danej relacji. Atrybuty mają nazwy i mogą przyjmować wartości z ustalonego zbioru – dziedziny atrybutu. Aby zaznaczyć różnicę pomiędzy pojęciem *atrybut* a pojęciem *dziedzina*, często nadaje się atrybutom inne nazwy niż dziedzinom, z których się wywodzą [Date1981].

Krotka (ang. *tuple*) – zbiór par postaci (nazwa atrybutu : wartość atrybutu), czyli

$$t = \{A_1 : d_1, \dots, A_n : d_n\}, \text{ gdzie } \text{dom}(A_i) = D_i \text{ oraz } d_i \in D_i \text{ dla } i = 1, \dots, n.$$

Kolejność krotek w relacji nie ma znaczenia.

Składowa krotki (ang. *component of tuple*) – wartość atrybutu w krotce.

Schemat relacji (ang. *relation scheme*) – specyfikacja relacji, obejmuje nazwę schematu relacji oraz zbiór atrybutów relacji o określonych (niekoniecznie różnych) dziedzinach, co zapisujemy

$$\mathbf{R}(\{A_1 : D_1, \dots, A_n : D_n\}).$$

Z powyższej definicji wynika, że kolejność atrybutów w schemacie relacji jest dowolna (nie ma znaczenia). Nazwy atrybutów w schemacie relacji muszą być różne.

Relacja (instancja relacji – ang. *relation instance*) o schemacie $\mathbf{R}$, w relacyjnym modelu danych, jest zbiorem krotek $\{t_1, \dots, t_s\}$. Schemat relacji $\mathbf{R}$ wyznacza relacje postaci

$$(A_{i_1} : D_{i_1}) \times \dots \times (A_{i_n} : D_{i_n}),$$

gdzie $i_1, \dots, i_n \in [1, n]$, $\text{dom}(A_i) = D_i$ (dla $i = 1, \dots, n$) oraz dla każdego $k \neq j$ (gdzie $k, j \in [1, n]$) $i_k \neq i_j$, czyli ciąg $i_1, \dots, i_n$ jest dowolną permutacją liczb $1, \dots, n$.

Relacja może być nazwana i wówczas jest identyfikowana przez **nazwę relacji**.

Dla graficznego rozróżnienia nazwy relacji i nazwy schematu relacji (bardzo często nazwa jest taka sama) przyjmujemy, że nazwy relacji będziemy pisać czcionką pochyłą (np. *R*) a nazwy schematów relacji będziemy pisać czcionką pogrubioną (np. $\mathbf{R}$).

Przykład 2.1

Rozważmy trzy relacje o schemacie

$$\mathbf{Osoby}(\{Id : Integer, Nazwisko : String, Imię : String\})$$

zawierające krotki:

relacja pierwsza:

$$\{ \{Id : 1, Nazwisko : "Kowalski", Imię : "Jan" \},$$

$$\{Id : 2, Nazwisko : "Nowak", Imię : "Adam" \} \},$$

relacja druga:

$\{\{Id : 2, Imię : "Adam", Nazwisko : "Nowak"\},$
 $\{Id : 1, Imię : "Jan", Nazwisko : "Kowalski"\}\},$

relacja trzecia:

$\{Id : 2, Nazwisko : "Nowak", Imię : "Adam"\},$
 $\{Id : 1, Nazwisko : "Kowalski", Imię : "Jan"\}\}.$

Z definicji relacji i schematu relacji wynika, że te trzy relacje, w relacyjnym modelu danych, są takie same (mają ten sam schemat relacji i takie same składowe krotek). $\square$

Jeśli z kontekstu i z nazw atrybutów wynika dziedzina atrybutów, to zamiast zapisu $\mathbf{R}(\{A_1: D_1, \dots, A_n: D_n\})$ będziemy stosowali zapis $\mathbf{R}(\{A_1, \dots, A_n\})$.

Pamiętając, że kolejność atrybutów w schemacie relacji $\mathbf{R}(\{A_1, \dots, A_n\})$ nie ma znaczenia, stosując zapis $\mathbf{R}(A_1, \dots, A_n)$ przyjmujemy domyślne uporządkowanie atrybutów w schemacie relacji i takie samo uporządkowanie wartości w krotkach relacji R o schemacie $\mathbf{R}(A_1, \dots, A_n)$.

Zapis $\text{sch}(R) = \mathbf{R}(A_1, \dots, A_n)$ oznacza, że relacja R ma schemat $\mathbf{R}(A_1, \dots, A_n)$.

Uwzględniając powyższe uwagi, zamiast zapisu schematu relacji

Osoby($\{Id : Integer, Nazwisko : String, Imię : String\}$)

możemy napisać

Osoby($Id, Nazwisko, Imię$)

a przykładową krotkę zamiast $\{Id : 1, Nazwisko : "Kowalski", Imię : "Jan"\}$ napiszemy jako $(1, "Kowalski", "Jan")$.

Stopień relacji (ang. *degree*) – liczba atrybutów relacji. Relację stopnia pierwszego (o jednym atrybucie) nazywamy relacją unarną, drugiego – binarną, trzeciego – ternarną, relację stopnia n -tego nazywamy relacją n -arną.

Liczebność relacji (ang. *cardinality*) – liczba krotek w relacji. Liczebność relacji R zapisujemy jako $\text{card}(R)$.

Instancja relacji (ang. *instance of relation*) – zbiór krotek relacji.

Relacja jest w zasadzie zmienną, czyli obiektem, który w czasie może zmieniać swoją wartość (bez zmiany schematu relacji). Naturalnym sposobem reprezentowania relacji jest tabela dwuwymiarowa. Abstrakcyjny obiekt relację możemy reprezentować więc jako konkretny obiekt – tabelę dwuwymiarową, w której kolumny odpowiadają atrybutom relacji a wiersze – krotkom. Kolejność wierszy w tabeli jest

dowolna. Reprezentując relację w formie tabeli przyjmujemy, że nazwy i kolejność kolumn odpowiadają nazwom i kolejności atrybutów w relacji.

Przykład 2.2

Założmy, że interesują nas cechy samochodów takie jak:

marka, kolor, rok produkcji, zużycie paliwa, numer rejestracyjny,
data rejestracji,

gdzie:

marka – przyjmuje wartości ze zbioru nazw marek, np. opel, BMW, Fiat 126p;

kolor – przyjmuje wartości ze zbioru kolorów, np. czerwony, biało-zielony;

rok produkcji – jest liczbą całkowitą większą od 1900, np. 1995;

zużycie paliwa – jest liczbą rzeczywistą dodatnią z jednym miejscem dziesiętnym,
oznaczającą zużycie paliwa w litrach na 100 km, np. 8.5;

numer rejestracyjny – jest ciągiem 5 liter i 3 cyfr, np. NUMER123;

data rejestracji – jest datą postaci dd-mm-rrrr, np. 02-02-1999.

Dane o samochodach można zapisać w postaci krotek relacji *Rejestr* o schemacie **Rejestr** (*Marka, Kolor, RokProd, Zużycie, NrRej, DataRej*):

{("Opel", "Biały", 1990, 6, "ABCDE123", 10-02-1998),
("Fiat 126p", "Czerwony", 1995, 7.4, "TYCIK543", 01-03-1995),
("Opel", "Czerwony", 1995, 5.6, "ZDREW876", 15-02-1998)}

lub przedstawić w czytelniejszej postaci tabeli dwuwymiarowej (tabela 2.1).

Tabela 2.1

Atrybuty i krotki relacji *Rejestr*

<i>Marka</i>	<i>Kolor</i>	<i>RokProd</i>	<i>Zużycie</i>	<i>NrRej</i>	<i>DataRej</i>
Opel	Biały	1990	6	ABCDE123	10-02-1998
Fiat 126p	Czerwony	1995	7.4	TYCIK543	01-03-1995
Opel	Czerwony	1995	5.6	ZDREW876	15-02-1998

□

Podamy teraz definicję pojęć dla relacji o zadanym schemacie.

Zbiór identyfikujący relacji (ang. *identifying set of relation*) – taki zbiór atrybutów relacji, których kombinacje wartości jednoznacznie identyfikują każdą krotkę relacji [CELL1988].

Klucz (ang. *key*) – jest to minimalny zbiór identyfikujący relacji tzn. taki zbiór identyfikujący relacji, którego żaden podzbiór nie jest zbiorem identyfikującym, czyli jest to jeden atrybut (lub minimalny zbiór atrybutów), którego wartość jednoznacznie identyfikuje każdą krotkę w relacji.

Klucz kandydujący (ang. *candidate key*) – dla jednej relacji może istnieć kilka kluczy, które jednoznacznie identyfikują krotki relacji. Nazywa się je kluczami kandydującymi. Spośród kluczy kandydujących wybiera się jeden tzw. klucz główny. W kluczu kandydującym żaden atrybut nie może przyjmować wartości pustej (NULL).

Klucz główny (ang. *primary key*) – dla jednej relacji może istnieć kilka kluczy kandydujących. Klucz, wybrany spośród kluczy kandydujących, nazywa się kluczem głównym.

Sztuczny klucz główny (ang. *artificial primary key*) – sztucznie stworzony klucz główny w przypadku, gdy klucze kandydujące są bardzo złożone lub, gdy brak jest klucza głównego. Słowo „sztuczny” oznacza, że klucz ten nie pojawił się w sposób naturalny i trzeba go dołączyć do atrybutów relacji, np. *IdWpisu*.

Klucz obcy (ang. *foreign key*) – atrybut lub kilka atrybutów relacji, które są kluczem głównym w innej relacji.

Klucz prosty (ang. *simple key*) – klucz jest prosty, jeśli zbiór identyfikujący relacji jest jednoelementowy, czyli jeśli kluczem jest jeden atrybut.

Klucz złożony (ang. *complex key*) – klucz jest złożony, jeśli zbiór identyfikujący relacji nie jest zbiorem jednoelementowym, czyli jeśli klucz składa się z kilku atrybutów.

Nadklucz (ang. *super key*) – zbiór atrybutów, który zawiera klucz (czyli zbiór atrybutów, z których można utworzyć klucz danej relacji) i który zapewnia rozróżnialność krotek. W szczególności, zbiór wszystkich atrybutów relacji jest nadkluczem.

Powiązanie (ang. *relationship*) – związek zachodzący pomiędzy określonymi atrybutami krotek różnych (lub tej samej) relacji.

2.2. PODSTAWOWE OPERACJE ALGEBRY RELACJI

W teoretycznym opisie modelu relacyjnego operacje na danych definiuje się w terminach algebry relacyjnej ([Codd1972, Beynon2000, Cell1988, Date1981, Pankow1992, Date2000]).

Operatory algebry relacyjnej mają za argumenty jedną lub dwie relacje (nie zbiory), a wynikiem ich działania jest nienazwana relacja. Podstawowe operacje algebry relacyjnej to:

- **Selekcja** (ang. *select*)

jest to operator jednoargumentowy oznaczający wybór z relacji krotek spełniających zadany warunek, dotyczący wartości atrybutów danej relacji, np. można wybrać dane osób o nazwisku „Nowak” z relacji *Pracownicy* o schemacie

Pracownicy(*IdOsoby, Nazwisko, Imię, Pensja, IdOddz, MiastoZam*).

Wynikiem operacji selekcji jest relacja zawierająca te krotki z relacji *Pracownicy*, których odpowiednie atrybuty spełniają zadany warunek, czyli *Nazwisko* = „Nowak”. Schemat relacji wynikowej zawiera takie same atrybuty jak schemat relacji wyjściowej (w tym przypadku takie same jak schemat relacji *Pracownicy*).

Date w pracy [Date2000] operację wyboru nazywa operacją restrykcji (ang. *restrict*), dla odróżnienia operatora selekcji od polecenia SELECT występującego w SQL.

- **Projekcja** (ang. *project*)

czyli rzutowanie, jest to operator jednoargumentowy. Operacja rzutowania polega na wyborze z danej relacji *R* określonego zbioru atrybutów *Z*, np. z relacji *Pracownicy* można wybrać nazwisko, imię i pensję wszystkich pracowników. Schemat relacji wynikowej zawiera określone atrybuty schematu relacji wyjściowej *R* (należące do zbioru *Z*). Liczba krotek relacji wynikowej może być mniejsza lub równa liczbie krotek relacji *R*. W szczególnym przypadku zbiór *Z* może zawierać wszystkie atrybuty relacji *R*, wówczas schemat relacji wynikowej jest taki sam jak schemat relacji wyjściowej.

- **Złączenie** (ang. *join*)

jest to operator dwuargumentowy. Operacja złączenia polega na połączeniu krotek z dwóch relacji, posiadających atrybuty o takich samych dziedzinach. Z relacji wynikowej usuwane są powtarzające się krotki. Bardzo często warunkiem złączenia jest równość wartości klucza głównego z jednej relacji i odpowiadającego mu klucza obcego w drugiej relacji. Operację złączenia można wykorzystać, na przykład, w celu uzyskania danych o pracownikach i oddziałach, w których pracują. Dla relacji *Pracownicy* o schemacie

Pracownicy(*IdOsoby, Nazwisko, Imię, Pensja, IdOddziału, MiastoZam*)

zawierającej dane o pracownikach oraz relacji *Oddziały* o schemacie

Oddziały(*IdOddziału, Nazwa, Adres*),

zawierającej dane o oddziałach firmy, należy wykonać operację złączenia relacji *Pracownicy* i *Oddziały*, przy warunku złączenia

Pracownicy.IdOddziału=Oddziały.IdOddziału.

- **Iloraz** (ang. *divide*)

jest to operator dwuargumentowy. Na przykład, jeśli jedna relacja (dzielnia) jest binarna czyli o schemacie **R1**(*A, B*), druga (dzielnik) – unarna, o schemacie **R2**(*B*), to wynikiem jest relacja zawierająca wszystkie wartości atrybutu *A* z relacji *R1*, dla których odpowiednia wartość *R1.B* atrybutu *B* jest równa *R2.B* (dla wszystkich wartości atrybutu *B* w relacji *R2*).

- **Suma** (ang. *union*)
operator określony dla relacji o takich samych atrybutach. Wynikiem jest relacja, której krotki są sumą teoriomnogościową krotek relacji wyjściowych.
- **Różnica** (ang. *difference*)
operator określony dla relacji o takich samych atrybutach. Wynikiem jest relacja, której krotki są różnicą teoriomnogościową krotek relacji wyjściowych.
- **Przecięcie** (ang. *intersect*)
czyli część wspólna; operator określony dla relacji o takich samych atrybutach. Wynikiem jest relacja, której krotki należą do obydwu relacji wyjściowych.
- **Iloczyn kartezjański** (ang. *product*)
operator dwuargumentowy określony dla relacji o różnych nazwach atrybutów. Relacja wynikowa zawiera atrybuty jednej i drugiej relacji. Zbiór krotek relacji wyjściowej powstaje przez utworzenie iloczynu kartezjańskiego krotek obu relacji wyjściowych a następnie każda krotka, która jest parą krotek z relacji wyjściowych zostaje zastąpiona konkatencją tych krotek, tzn. jeśli krotka $k1 \in R1$, $k2 \in R2$, to do iloczynu kartezjańskiego $R1 \times R2$ należy krotka utworzona ze składowych krotek $k1$ i $k2$.

Ze względu na brak standardowej składni operatorów relacyjnych w literaturze do opisu operacji relacyjnych wykorzystuje się wysokiego poziomu subjęzyki danych, które są zbiorem operatorów wykonywanych na danych zapisanych w postaci relacji. Ponadto można definiować dodatkowe operatory algebry relacji, dla których argumentami i wynikami będą relacje, np.:

- **złączenie naturalne** (złączenie, z którego usuwane są powtarzające się atrybuty),
- **Θ -złączenie** (złączenie krotek spełniających warunek Θ),
- **złączenie zewnętrzne** (złączenie zewnętrzne relacji L i P jest to złączenie, w którym są zachowywane wszystkie krotki relacji L i P ; złączenie, w którym są zachowane wszystkie krotki relacji L – **złączenie lewostronne**, relacji P – **złączenie prawostronne**).
- **nadanie nazwy relacji** – operator oznaczany symbolem $:=$, np. zapis $S := R1 \times R2$ oznacza wykonanie operacji iloczynu kartezjańskiego relacji $R1$ i $R2$ a następnie nadanie nazwy S otrzymanej relacji wynikowej.

Czytelnika zachęcamy do zdefiniowania własnych operatorów dla operacji wykonywanych na relacjach (np. usuwania danych, modyfikacji danych, zmiany nazwy atrybutu, dopisania nowego atrybutu, wyliczenia sumy dla atrybutu liczbowego itd.), zdefiniowania ich argumentów, wyników, określenia schematów relacji wyjściowych i wynikowych, sposobu działania i podania przykładów ich użycia.

Operacje sumy, przecięcia, iloczynu kartezjańskiego i złączenia naturalnego w algebrze relacji są łączne i przemienne.

Omówimy teraz podstawowe operatory algebry relacji ilustrując ich użycie przykładami.

Niech F oznacza formułę zawierającą jako argumenty nazwy atrybutów lub stałe, operatory porównania ($=, >, \geq, <, \leq, \neq$) oraz operatory logiczne ($\neg, \wedge, \vee$).

Dana jest relacja *Pracownicy* (tabela 2.2) o schemacie:

Pracownicy(*IdOsoby*, *Nazwisko*, *Imię*, *Pensja*, *IdOddziału*, *MiastoZam*)

oraz relacja *Oddziały* (tabela 2.3) o schemacie:

Oddziały(*IdOddziału*, *Nazwa*, *Adres*),

przy czym atrybut podkreślony oznacza klucz główny relacji.

Tabela 2.2

Relacja *Pracownicy*

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>Pensja</i>	<i>IdOddziału</i>	<i>MiastoZam</i>
0001	Wabacki	Jan	2300	2	Opole
0002	Jarecki	Jan	1500	2	Opole
0004	Kotecki	Adam	1500	4	Nysa
0005	Abacka	Ewa	1100	4	Warszawa
0006	Wabacki	Marek	2300	2	Opole

Tabela 2.3

Relacja *Oddziały*

<i>IdOddziału</i>	<i>Nazwa</i>	<i>Adres</i>
1	Wrocław	50-370 Wrocław, Polna 1
2	Opole	48-100 Opole, Lipowa 2
3	Poznań	60-965 Poznań, Wiosenna 3
4	Warszawa	01-003 Warszawa, Dworcowa 4

Operacja **selekcji** (wyboru) zapisywana przy użyciu operatora σ (SELECT) jako

$$\sigma_F(R)$$

oznacza wybór tych krotek z relacji R , które spełniają formułę F .

Jeśli relacja R ma schemat $\mathbf{R}(A_1, \dots, A_n)$, to relacja $\sigma_F(R)$ ma również schemat $\mathbf{R}(A_1, \dots, A_n)$. Stopień relacji wynikowej jest więc taki sam jak relacji R , natomiast $\text{card}(\sigma_F(R)) \leq \text{card}(R)$.

Przykład 2.3

Zdefiniujmy dla relacji *Pracownicy* (tabela 2.2) operację selekcji

$$\sigma_{\text{MiastoZam} = \text{"Opole"}}(\text{Pracownicy}),$$

którą czytamy jako

„Wybierz z relacji *Pracownicy* krotki, dla których atrybut *MiastoZam* ma wartość *Opole*”

lub, w sposób bardziej naturalny,

„Podaj dane pracowników zamieszkałych w Opolu”.

Wynik operacji selekcji przedstawiono w tabeli 2.4.

Zauważmy, że $\text{sch}(\sigma_{\text{MiastoZam} = \text{„Opole”}}(\text{Pracownicy})) = \text{sch}(\text{Pracownicy})$.

Tabela 2.4

Relacja $\sigma_{\text{MiastoZam} = \text{„Opole”}}(\text{Pracownicy})$

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>Pensja</i>	<i>IdOddziału</i>	<i>MiastoZam</i>
0001	Wabacki	Jan	2300	2	Opole
0002	Jarecki	Jan	1500	2	Opole
0006	Wabacki	Marek	2300	2	Opole

Z tabeli 2.4 wynika, że $\text{card}(\sigma_{\text{MiastoZam} = \text{„Opole”}}(\text{Pracownicy})) = 3$.

□

Operacja **projekcji** (rzutu) zapisywana przy pomocy operatora π (PROJECT) jako $\pi_Z(R)$ powoduje wybór tych atrybutów z relacji R , które należą do zbioru Z . Zatem, jeśli relacja R ma schemat $\mathbf{R}(A_1, \dots, A_n)$ i $Z = \{ A_{i_1}, \dots, A_{i_j} \}$, gdzie $j \in [1, n]$, wówczas wynikiem operacji

$$\pi_Z(R)$$

jest relacja o atrybutach $A_{i_1}, \dots, A_{i_j}$. Stopień relacji wynikowej równa się liczbie atrybutów w zbiorze Z , liczebność relacji wynikowej jest mniejsza lub równa liczebności relacji wyjściowej R .

Przykład 2.4

Dla relacji *Pracownicy* (tabela 2.2) zdefiniujmy operację projekcji

$$\pi_{\{Id, Nazwisko, Imię\}}(\text{Pracownicy}),$$

którą czytamy jako

„Wybierz z relacji *Pracownicy* atrybuty *Id*, *Nazwisko*, *Imię*”

lub w sposób bardziej naturalny:

„Podaj identyfikator, nazwisko i imię wszystkich pracowników firmy”.

Relację otrzymaną po wykonaniu projekcji

$$\text{Wyniki_PROJECT} := \pi_{\{Id, Nazwisko, Imię\}}(\text{Pracownicy}),$$

przedstawiono w tabeli 2.5.

Tabela 2.5

Relacja Wyniki_PROJECT

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>
0001	Wabacki	Jan
0002	Jarecki	Jan
0004	Kotecki	Adam
0005	Abacka	Ewa
0006	Wabacki	Marek

Relacja *Pracownicy* jest stopnia 6. a relacja *Wyniki_PROJECT* jest stopnia 3. Liczebność relacji *Wyniki_PROJECT* jest taka sama jak relacji *Pracownicy* i wynosi 5. Natomiast $\text{card}(\pi_{\{Nazwisko\}}(\text{Pracownicy})) = 4 \leq \text{card}(\text{Pracownicy})$.

□

Przykład 2.5

Z relacji *Pracownicy* (tabela 2.2) wyszukać identyfikatory, nazwiska i imiona osób zamieszkałych w Opolu. W tym celu zapisujemy:

$\text{Wyniki_SELECT_PROJECT} := \pi_{\{Id, Nazwisko, Imię\}}(\sigma_{\text{MiastoZam} = \text{"Opole"}}(\text{Pracownicy}))$

Otrzymana relacja *Wyniki_SELECT_PROJECT* jest przedstawiona w tabeli 2.6.

Tabela 2.6

Relacja Wyniki_SELECT_PROJECT

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>
0001	Wabacki	Jan
0002	Jarecki	Jan
0006	Wabacki	Marek

□

Definicje podstawowych operatorów dla relacji różnią się nieco od definicji tych operatorów dla zbiorów, np. definicja operatora iloczynu kartezjańskiego.

Wynikiem operacji **iloczynu kartezjańskiego** relacji *R1* i *R2* o różnych nazwach atrybutów jest relacja, która zawiera atrybuty jednej i drugiej relacji a zbiór krotek relacji wynikowej powstaje przez utworzenie krotek jako iloczynu kartezjańskiego krotek relacji wyjściowych *R1* i *R2*. Iloczyn kartezjański definiowany dla zbiorów jest zbiorem uporządkowanych par wartości z tych zbiorów, iloczyn kartezjański definiowany w algebrze relacji zastępuje parę krotek konkatencją krotek. Ponadto iloczyn kartezjański w algebrze zbiorów nie jest przemienny a w algebrze relacji jest przemienny. Operację iloczynu kartezjańskiego zapisujemy za pomocą operatora $\times$, np. $R1 \times R2$.

Często zdarza się, że operatory algebry relacji wymagają, aby relacje stanowiące ich argumenty miały różne nazwy atrybutów, tymczasem relacje mają atrybuty o takich samych nazwach. Stosuje się wtedy nazwy kwalifikowane postaci

nazwa_relacji.nazwa_atrybutu

co pozwala na jednoznaczne identyfikowanie nazw atrybutów, np.

Pracownicy.IdOddziału, Oddziały.IdOddziału.

Sposób ten nie zawsze jest skuteczny (np. jeśli chcemy wykonać operację $R \times R$). Należy wówczas dokonać przemianowania nazwy relacji lub atrybutów. W dalszej części książki wszędzie tam, gdzie jest wymagana unikalność nazw atrybutów zakładamy, że warunek ten będzie spełniony przez wykonanie wcześniejszej operacji przemianowania nazwy relacji bądź atrybutów.

Przykład 2.6

Aby obliczyć iloczyn kartezjański relacji *Pracownicy* (tabela 2.2) i relacji *Oddziały* (tabela 2.3) należy najpierw zapewnić unikalność nazw atrybutów w obu relacjach. W tym celu wprowadzimy dodatkowy operator

RENAME($R[: A_1 \text{ AS } NewA_1, \dots, A_j \text{ AS } NewA_j]$) [**AS** *NewR*],

gdzie relacja R ma schemat $\mathbf{R} = (A_1, \dots, A_n)$ oraz $1 \leq j \leq n$. Operator **RENAME** umożliwia:

- 1) zmianę nazwy atrybutów w relacji,
- 2) zapisanie relacji pod nową nazwą.

Nawiasy kwadratowe w definicji operatora **RENAME** oznaczają wyrażenie opcjonalne. Operacja zmiany nazwy jest realizowana w ten sposób, że jeśli jest parametr określający zmianę nazwy relacji, to jest tworzona kopia relacji R pod nazwą *NewR* i w relacji *NewR* dokonywane są ewentualne zmiany nazw atrybutów, w przeciwnym razie zmiany nazw atrybutów są dokonywane w relacji R .

Wykorzystując operator **RENAME** utworzymy kopię relacji *Pracownicy*, którą nazwiemy *Kadry*, zmieniając również nazwę atrybutu *IdOddziału* z relacji *Pracownicy* na *IdOddz*:

RENAME(*Pracownicy: IdOddziału AS IdOddz*) **AS** *Kadry*.

Dane relacji *Kadry* przedstawia tabela 2.7.

Tabela 2.7

Relacja *Kadry*

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>Pensja</i>	<i>IdOddz</i>	<i>MiastoZam</i>
0001	Wabacki	Jan	2300	2	Opole
0002	Jarecki	Jan	1500	2	Opole
0004	Kotecki	Adam	1500	4	Nysa
0005	Abacka	Ewa	1100	4	Warszawa
0006	Wabacki	Marek	2300	2	Opole

Teraz możemy już obliczyć iloczyn kartezjański $Kadry \times Oddziały$, którego wynik przedstawia tabela 2.8.

Relacja wynikowa zawiera 20 krotek (liczba krotek relacji $Kadry$ pomnożona przez liczbę krotek relacji $Oddziały$). Iloczyn kartezjański $Kadry \times Oddziały$, zawiera wszystkie możliwe połączenia każdego pracownika z każdym oddziałem.

Zwróćmy uwagę, że relacja $Kadry \times Oddziały$ zawiera np. krotkę
 (0001, "Wabacki", "Jan", 2300, 2, "Opole", 2, "Opole", "48-100 Opole, Lipowa 2"),
 natomiast w algebrze zbiorów byłaby to para krotek

((0001, "Wabacki", "Jan", 2300, 2, "Opole"),
 (2, "Opole", "48-100 Opole, Lipowa 2")).

Tabela 2.8

Iloczyn kartezjański $Kadry \times Oddziały$

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>Pensja</i>	<i>IdOddz</i>	<i>MiastoZam</i>	<i>IdOddzialu</i>	<i>Nazwa</i>	<i>Adres</i>
1	2	3	4	5	6	7	8	9
0001	Wabacki	Jan	2300	2	Opole	1	Wrocław	50-370 Wrocław, Polna 1
0001	Wabacki	Jan	2300	2	Opole	2	Opole	48-100 Opole, Lipowa 2
0001	Wabacki	Jan	2300	2	Opole	3	Poznań	60-965 Poznań, Wiosenna 3
0001	Wabacki	Jan	2300	2	Opole	4	Warszawa	01-003 Warszawa, Dworcowa 4
0002	Jarecki	Jan	1500	2	Opole	1	Wrocław	50-370 Wrocław, Polna 1
0002	Jarecki	Jan	1500	2	Opole	2	Opole	48-100 Opole, Lipowa 2
0002	Jarecki	Jan	1500	2	Opole	3	Poznań	60-965 Poznań, Wiosenna 3

2. Podstawy relacyjnych baz danych

1	2	3	4	5	6	7	8	9
0002	Jarecki	Jan	1500	2	Opole	4	Warszawa	01-003 Warszawa, Dworcowa 4
0004	Kotecki	Adam	1500	4	Nysa	1	Wrocław	50-370 Wrocław, Polna 1
0004	Kotecki	Adam	1500	4	Nysa	2	Opole	48-100 Opole, Lipowa 2
0004	Kotecki	Adam	1500	4	Nysa	3	Poznań	60-965 Poznań, Wiosenna 3
0004	Kotecki	Adam	1500	4	Nysa	4	Warszawa	01-003 Warszawa, Dworcowa 4
0005	Abacka	Ewa	1100	4	Warszawa	1	Wrocław	50-370 Wrocław, Polna 1
0005	Abacka	Ewa	1100	4	Warszawa	2	Opole	48-100 Opole, Lipowa 2
0005	Abacka	Ewa	1100	4	Warszawa	3	Poznań	60-965 Poznań, Wiosenna 3
0005	Abacka	Ewa	1100	4	Warszawa	4	Warszawa	01-003 Warszawa, Dworcowa 4
0006	Wabacki	Marek	2300	2	Opole	1	Wrocław	50-370 Wrocław, Polna 1
0006	Wabacki	Marek	2300	2	Opole	2	Opole	48-100 Opole, Lipowa 2
0006	Wabacki	Marek	2300	2	Opole	3	Poznań	60-965 Poznań, Wiosenna 3
0006	Wabacki	Marek	2300	2	Opole	4	Warszawa	01-003 Warszawa, Dworcowa 4



Operacja **złączenia** relacji $R1$ i $R2$ zapisywana za pomocą operatora złączenia oznaczanego symbolem $\bowtie$ jako

$$R1 \bowtie R2$$

polega na utworzeniu relacji, w której krotki z relacji $R1$ będą połączone z odpowiednimi krotkami relacji $R2$, na podstawie tych samych wartości dla wspólnych atrybutów występujących w relacji $R1$ i $R2$. Stopień relacji wynikowej jest równy sumie stopni relacji wyjściowych $R1$ i $R2$.

Złączenie relacji $R1$ i $R2$ wykonuje się tworząc iloczyn kartezjański relacji $R1$ i $R2$ (wraz z odpowiednimi operacjami zmian powtarzających się nazw atrybutów) a następnie wybierając te krotki, dla których dla każdego zbioru atrybutów łączących $A = (A_{i_1}, \dots, A_{i_j})$, gdzie $j \in [1, n]$, występującego w relacji $R1$, w relacji $R2$ odpowiednie atrybuty mają te same wartości.

Przykład 2.7

Chcemy utworzyć relację, zawierającą dane o pracownikach i oddziałach, w których pracują.

Tabela 2.9

Relacja $Pracownicy \bowtie Oddziały$

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>Pensja</i>	<i>Pracownicy. IdOddziału</i>	<i>MiastoZam</i>	<i>Oddziały. IdOddziału</i>	<i>Nazwa</i>	<i>Adres</i>
0001	Wabacki	Jan	2300	2	Opole	2	Opole	48-100 Opole, Lipowa 2
0002	Jarecki	Jan	1500	2	Opole	2	Opole	48-100 Opole, Lipowa 2
0004	Kotecki	Adam	1500	4	Nysa	4	Warszawa	01-003 Warszawa, Dworcowa 4
0005	Abacka	Ewa	1100	4	Warszawa	4	Warszawa	01-003 Warszawa, Dworcowa 4
0006	Wabacki	Marek	2300	2	Opole	2	Opole	48-100 Opole, Lipowa 2

W tym celu wykonujemy złączenie relacji *Pracownicy* (tabela 2.2) i *Oddziały* (tabela 2.3):

$$Pracownicy \bowtie Oddziały,$$

którego efektem jest relacja o atrybutach

$\{IdOsoby, Nazwisko, Imię, Pensja, Pracownicy.IdOddziału, MiastoZam, Oddziały.IdOddziału, Nazwa, Adres\}$,

reprezentowana jako tabela 2.9 (atrybut złączenia $IdOddziału$ występujący w obu relacjach został przemianowany na $Pracownicy.IdOddziału$ i $Oddziały.IdOddziału$).

W relacji $Pracownicy \triangleright \triangleleft Oddziały$ wartości atrybutu łączącego ($IdOddziału$) występują dwukrotnie.

Stopień relacji wynikowej jest równy sumie stopni relacji $Pracownicy$ i $Oddziały$. □

W celu zdefiniowania operacji złączenia naturalnego relacji $R1$ i $R2$ wprowadzimy kilka oznaczeń. Niech

$$sch(R1) = \mathbf{R1}(A_1, \dots, A_n, Z_1, \dots, Z_k),$$

$$sch(R2) = \mathbf{R2}(Z_1, \dots, Z_k, B_1, \dots, B_m),$$

$$U1 = \{A_1, \dots, A_n\},$$

$$U2 = \{B_1, \dots, B_m\},$$

$$Z = \{Z_1, \dots, Z_k\} \text{ – zbiór atrybutów złączenia, występujących w } \mathbf{R1} \text{ i w } \mathbf{R2},$$

$$W = U1 \cup Z \cup U2 = \{A_1, \dots, A_n, Z_1, \dots, Z_k, B_1, \dots, B_m\}.$$

Zakładamy, że atrybuty $Z_1, \dots, Z_k$ (występujące w obu schematach relacji) o takich samych nazwach mają takie same dziedziny.

Złączenie naturalne relacji $R1$ i $R2$, polega na wykonaniu operacji złączenia relacji $R1$ i $R2$ względem zbioru atrybutów Z , a następnie na rzutowaniu na zbiór atrybutów W , czyli na wykonaniu operacji rzutowania usuwającej powtarzające się atrybuty złączenia

$$\pi(R1 \triangleright \triangleleft R2).$$

Operację złączenia naturalnego zapisujemy za pomocą operatora **JOIN**.

Schemat relacji

$$R1 \text{ JOIN } R2$$

zawiera atrybuty

$$A_1, \dots, A_n, Z_1, \dots, Z_k, B_1, \dots, B_m.$$

Jeśli $Z = \emptyset$, to $R1 \text{ JOIN } R2 = R1 \times R2$.

Jeśli $sch(R1) = sch(R2)$, to $R1 \text{ JOIN } R2 = R1 \cap R2$.

Tak zdefiniowana operacja złączenia naturalnego jest łączna i przemienne.

Jeśli $F = (R1.Z_1 = R2.Z_1 \wedge \dots \wedge R1.Z_k = R2.Z_k)$, to operator $R1 \text{ JOIN } R2$ można zapisać jako

$$\pi_W(\sigma_F(R1 \times R2)).$$

Przykład 2.8 ilustruje operację złączenia naturalnego.

Przykład 2.8

Po wykonaniu operacji złączenia naturalnego relacji *Pracownicy* (tabela 2.2) i *Oddziały* (tabela 2.3) względem atrybutu *Pracownicy.IdOddziału* i odpowiadającego mu atrybutu (o tej samej dziedzinie) *Oddziały.IdOddziału*, otrzymamy relację

$$\text{Pracownicy JOIN Oddziały} = \pi_{U-Z}(\text{Pracownicy} \bowtie \text{Oddziały})$$

o danych zawartych w tabeli 2.10 (zbiór *U* jest zbiorem wszystkich atrybutów relacji *Pracownicy* i *Oddziały*, zbiór *Z* = {*Oddziały.IdOddziału*}).

Tabela 2.10

Relacja *Pracownicy JOIN Oddziały*

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>Pensja</i>	<i>IdOddziału</i>	<i>MiastoZam</i>	<i>Nazwa</i>	<i>Adres</i>
0001	Wabacki	Jan	2300	2	Opole	Opole	48-100 Opole, Lipowa 2
0002	Jarecki	Jan	1500	2	Opole	Opole	48-100 Opole, Lipowa 2
0004	Kotecki	Adam	1500	4	Nysa	Warszawa	01-003 Warszawa, Dworcowa 4
0005	Abacka	Ewa	1100	4	Warszawa	Warszawa	01-003 Warszawa, Dworcowa 4
0006	Wabacki	Marek	2300	2	Opole	Opole	48-100 Opole, Lipowa 2

□

Θ-złączenie (teta-złączenie) relacji *R1* i *R2*, które zapisujemy jako

$$R1 \bowtie_{\Theta} R2,$$

polega na utworzeniu relacji, w której będą krotki z relacji *R1* połączone z krotkami relacji *R2* spełniające warunek logiczny Θ dotyczący atrybutów z relacji *R1* i *R2* o tej samej dziedzinie w obu relacjach (warunek Θ musi mieć sens).

Jeśli warunek Θ jest równością, czyli ma postać *wyrażenie1* = *wyrażenie2*, to mówimy o równozłączeniu (ang. *equijoin*), w przeciwnym razie mówimy o teta-złączeniu.

Operację teta-złączenia wykonuje się tworząc iloczyn kartezjański relacji $R1$ i $R2$ a następnie wybierając te krotki, dla których jest spełniony warunek Θ dotyczący atrybutów o dziedzinie D , występujących w relacji $R1$ i w relacji $R2$, np.

$$R1.A1 < R2.A2, \text{ gdzie } \text{dom}(A1) = \text{dom}(A2) = D.$$

Podobnie jak w przypadku iloczynu kartezjańskiego tak i w przypadku teta-złączenia, obie relacje wyjściowe muszą mieć atrybuty o różnych nazwach.

Przykład 2.9

Wynikiem Θ -złączenia relacji *Kadry* (tabela 2.7) i *Oddziały* (tabela 2.3):

$$Kadry \triangleright \triangleleft IdOddz < IdOddzialu \text{ Oddziały}$$

jest relacja przedstawiona w tabeli 2.11.

Tabela 2.11

Relacja $Kadry \triangleright \triangleleft IdOddz < IdOddzialu \text{ Oddziały}$

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>Pensja</i>	<i>IdOddz</i>	<i>MiastoZam</i>	<i>IdOddzialu</i>	<i>Nazwa</i>	<i>Adres</i>
0001	Wabacki	Jan	2300	2	Opole	3	Poznań	60-965 Poznań, Wiosenna 3
0001	Wabacki	Jan	2300	2	Opole	4	Warszawa	01-003 Warszawa, Dworcowa 4
0002	Jarecki	Jan	1500	2	Opole	3	Poznań	60-965 Poznań, Wiosenna 3
0002	Jarecki	Jan	1500	2	Opole	4	Warszawa	01-003 Warszawa, Dworcowa 4
0006	Wabacki	Marek	2300	2	Opole	3	Poznań	60-965 Poznań, Wiosenna 3
0006	Wabacki	Marek	2300	2	Opole	4	Warszawa	01-003 Warszawa, Dworcowa 4

Relacja wynikowa zawiera dane tylko tych pracowników, dla których istnieje oddział o numerze większym od numeru oddziału, w którym aktualnie pracuje dany pracownik. Jeśli przejście do oddziału o numerze większym oznacza awans dla pracownika, to relacja zawiera dane pracowników, którzy mogą być awansowani, oraz dane oddziałów, do których mogą być awansowani.

Aby uzyskać dane o pracownikach i oddziałach, w których pracują, należy wykonać równozłączenie relacji *Kadry* i *Pracownicy*

$$Kadry \triangleright \triangleleft IdOddz = IdOddzialu \text{ Oddziały}$$

□

Dla oznaczenia dwuargumentowego **złączenia zewnętrznego** przyjmujemy operator $\lt\gt$, dla złączenia **lewostronnego** przyjmujemy operator $\lt$, a dla złączenia **prawostronnego** operator $\gt$. Złączenie lewostronne jest to złączenie, które zachowuje wszystkie krotki z relacji po lewej stronie operatora $\lt$, złączenie prawostronne zachowuje wszystkie krotki z relacji po prawej stronie operatora $\gt$. Złączenie zewnętrzne jest to złączenie zachowujące wszystkie krotki z obu relacji. Brakujące wartości składowych krotek w relacjach wynikowych w złączeniu zewnętrznym, lewostronnym i prawostronnym są wypełniane wartościami pustymi NULL (więcej informacji na temat wartości NULL zawiera rozdz. 3.6.1).

Przykład 2.10

Dane są dwie relacje: *Kadry* (tabela 2.12) i *Oddziały* (tabela 2.13).

Tabela 2.12

Relacja *Kadry*

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>Pensja</i>	<i>IdOddz</i>	<i>MiastoZam</i>
0001	Wabacki	Jan	2300	2	Opole
0002	Jarecki	Jan	1500		Opole
0004	Kotecki	Adam	1500		Nysa
0005	Abacka	Ewa	1100	4	Warszawa
0006	Wabacki	Marek	2300	2	Opole

Tabela 2.13

Relacja *Oddziały*

<i>IdOddziału</i>	<i>Nazwa</i>	<i>Adres</i>
1	Wrocław	50-370 Wrocław, Polna 1
2	Opole	48-100 Opole, Lipowa 2
3	Poznań	60-965 Poznań, Wiosenna 3
4	Warszawa	01-003 Warszawa, Dworcowa 4

Wynikiem złączenia lewostronnego

Kadry $\lt$ *Oddziały*

jest relacja reprezentowana przez tabelę 2.14.

W złączeniu lewostronnym *Kadry* $\lt$ *Oddziały* występują dane wszystkich pracowników i oddziałów, w których pracują. Jeśli pracownik nie jest przypisany do żadnego oddziału, to dane o oddziale mają wartości NULL.

Aby uzyskać dane o wszystkich oddziałach i zatrudnionych w nich pracownikach wykonamy złączenie prawostronne

Kadry $\gt$ *Oddziały*,

którego wynikiem jest relacja reprezentowana przez tabelę 2.15.

Tabela 2.14

Relacja *Kadry* $\triangleleft$ *Oddziały*

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>Pensja</i>	<i>IdOddz</i>	<i>MiastoZam</i>	<i>IdOddziału</i>	<i>Nazwa</i>	<i>Adres</i>
0001	Wabacki	Jan	2300	2	Opole	2	Opole	48-100 Opole, Lipowa 2
0002	Jarecki	Jan	1500	null	Opole	null	null	null
0004	Kotecki	Adam	1500	null	Nysa	null	null	null
0005	Abacka	Ewa	1100	4	Warszawa	4	Warszawa	01-003 Warszawa, Dworcowa 4
0006	Wabacki	Marek	2300	2	Opole	2	Opole	48-100 Opole, Lipowa 2

Tabela 2.15

Relacja *Kadry* $\triangleright$ *Oddziały*

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>Pensja</i>	<i>IdOddz</i>	<i>MiastoZam</i>	<i>IdOddziału</i>	<i>Nazwa</i>	<i>Adres</i>
null	null	null	null	null	null	1	Wrocław	50-370 Wrocław, Polna 1
0001	Wabacki	Jan	2300	2	Opole	2	Opole	48-100 Opole, Lipowa 2
0006	Wabacki	Marek	2300	2	Opole	2	Opole	48-100 Opole, Lipowa 2
null	null	null	null	null	null	3	Poznań	60-965 Poznań, Wiosenna 3
0005	Abacka	Ewa	1100	4	Warszawa	4	Warszawa	01-003 Warszawa, Dworcowa 4

Relacja *Kadry* $\triangleright$ *Oddziały* zawiera dane o wszystkich oddziałach, nawet jeśli do oddziału nie jest przypisany żaden pracownik.

Wynikiem złączenia zewnętrznego (obustronnego)

Kadry $\Leftarrow \Rightarrow$ *Oddziały*

jest relacja reprezentowana przez tabelę 2.16, która zachowuje dane o wszystkich pracownikach i wszystkich oddziałach.

Stopień relacji *Kadry* $\Leftarrow \Rightarrow$ *Oddziały*, będącej wynikiem złączenia zewnętrznego jest równy sumie stopni relacji wyjściowych *Kadry* i *Oddziały*, czyli wynosi 9.

Liczebność relacji *Kadry* $\Leftarrow \Rightarrow$ *Oddziały*, jest równa sumie liczebności relacji wyjściowych *Kadry* i *Oddziały* minus liczebność relacji *Kadry* JOIN *Oddziały*.

W miejsce brakujących wartości w relacji wynikowej jest wstawiana wartość NULL.

Tabela 2.16

Relacja *Kadry* $\Leftarrow \Rightarrow$ *Oddziały*

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>Pensja</i>	<i>IdOddz</i>	<i>MiastoZam</i>	<i>IdOddziału</i>	<i>Nazwa</i>	<i>Adres</i>
0001	Wabacki	Jan	2300	2	Opole	2	Opole	48-100 Opole, Lipowa 2
0002	Jarecki	Jan	1500		Opole	null	null	null
0004	Kotecki	Adam	1500		Nysa	null	null	null
0005	Abacka	Ewa	1100	4	Warszawa	4	Warszawa	01-003 Warszawa, Dworcowa 4
0006	Wabacki	Marek	2300	2	Opole	2	Opole	48-100 Opole, Lipowa 2
null	null	null	null	null	null	1	Wrocław	50-370 Wrocław, Polna 1
null	null	null	null	null	null	3	Poznań	60-965 Poznań, Wiosenna 3



Omówimy teraz kolejne operatory algebry relacji takie jak suma, różnica i przecięcie dwóch relacji, które mają ten sam stopień i atrybuty o tych samych nazwach i dziedzinach.

Suma relacji *R1* i *R2*, o takich samych atrybutach, zawiera krotki należące do relacji *R1* lub do relacji *R2* lub do obu tych relacji. Operator sumy oznaczamy symbolem $\cup$. Relacja wynikowa ma atrybuty takie same jak relacje wyjściowe, liczba krotek jest mniejsza lub równa od sumy liczby krotek w relacjach wyjściowych (w wyniku nie występują powtarzające się krotki), czyli

$$\text{card}(R1 \cup R2) = \text{card}(R1) + \text{card}(R2) - \text{card}(R1 \cap R2),$$

gdzie $R1 \cap R2$ jest przecięciem relacji $R1$ i $R2$.

Przykład 2.11

Rozważmy relacje: *Nauczyciele* (tabela 2.17) i *Studenci* (tabela 2.18) o schemacie relacji **Osoby**(*IdOsoby*, *Nazwisko*, *Imię*, *MiastoZam*).

Relacja $\text{Nauczyciele} \cup \text{Studenci}$ zawiera dane nauczycieli i studentów, przedstawione w tabeli 2.19.

Tabela 2.17

Relacja *Nauczyciele*

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>MiastoZam</i>
0001	Wabacki	Jan	Opole
0002	Jarecki	Jan	Opole
0004	Kotecki	Adam	Nysa
0005	Abacka	Ewa	Warszawa
0006	Wabacki	Marek	Opole

Tabela 2.18

Relacja *Studenci*

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>MiastoZam</i>
0001	Wabacki	Jan	Opole
0002	Abacki	Jan	Warszawa
0003	Nowicki	Adam	Nysa
0004	Abacka	Ewa	Warszawa

Tabela 2.19

Relacja $\text{Nauczyciele} \cup \text{Studenci}$

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>MiastoZam</i>
0001	Wabacki	Jan	Opole
0002	Jarecki	Jan	Opole
0004	Kotecki	Adam	Nysa
0005	Abacka	Ewa	Warszawa
0006	Wabacki	Marek	Opole
0002	Abacki	Jan	Warszawa
0003	Nowicki	Adam	Nysa
0004	Abacka	Ewa	Warszawa

W relacjach wyjściowych *Nauczyciele* i *Studenci* występuje identyczna krotka z danymi Jana Wabackiego ale w relacji wynikowej $Nauczyciele \cup Studenci$ krotka ta występuje tylko jeden raz. Liczebność relacji wynikowej jest więc mniejsza od sumy liczebności relacji wyjściowych. Schemat relacji wynikowej jest taki sam jak relacji wyjściowych, czyli relacja $Nauczyciele \cup Studenci$ ma schemat **OSOBY**.

□

Różnicę relacji (podobnie jak sumę) tworzymy dla dwóch relacji o takich samych atrybutach. Jako operator różnicy przyjmujemy symbol „-”. Do różnicy dwóch relacji $R1 - R2$ należą te krotki, które występują w relacji $R1$ ale nie występują w relacji $R2$.

Schemat relacji wynikowej jest taki sam jak relacji wyjściowych.

Dla różnicy relacji zachodzi zależność $card(R1 - R2) = card(R1) - card(R1 \cap R2)$.

Przykład 2.12

Utwórzmy różnicę relacji *Nauczyciele* (tabela 2.17) i *Studenci* (tabela 2.18) z przykładu 2.11. Relacja

Nauczyciele – Studenci

zawiera krotki przedstawione w tabeli 2.20.

Tabela 2.20

Relacja *Nauczyciele – Studenci*

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>MiastoZam</i>
0002	Jarecki	Jan	Opole
0004	Kotecki	Adam	Nysa
0005	Abacka	Ewa	Warszawa
0006	Wabacki	Marek	Opole

Relacja będąca różnicą relacji $R1$ i $R2$ zawiera te krotki z relacji $R1$, które nie występują w relacji $R2$. Interpretacja różnicy relacji *Nauczyciele – Studenci* w języku naturalnym brzmi:

„Podać dane nauczycieli, którzy nie są studentami”.

Przecięcie (część wspólną) dwóch relacji $R1$ i $R2$ (podobnie jak sumę i różnicę) tworzymy dla relacji o takich samych atrybutach. Dwuargumentowy operator przecięcia oznaczamy symbolem $\cap$. Do przecięcia relacji należą krotki, które występują w obu relacjach wyjściowych $R1$ i $R2$.

Schemat relacji wynikowej jest taki sam jak relacji wyjściowych.

Dla przecięcia relacji $R1$ i $R2$ zachodzi zależność

$$card(R1 \cap R2) \leq \min(card(R1), card(R2)).$$

Przykład 2.13

Dla relacji *Nauczyciele* (tabela 2.17) i *Studenci* (tabela 2.18) z przykładu 2.11, relacja będąca przecięciem $Nauczyciele \cap Studenci$ zawiera krotki, które należą do obydwu relacji, czyli tylko dane Jana Wabackiego (tabela 2.21).

Tabela 2.21Relacja $Nauczyciele \cap Studenci$

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>MiastoZam</i>
0001	Wabacki	Jan	Opole

Do przecięcia relacji $R1$ i $R2$ należą te krotki, które należą zarówno do relacji $R1$, jak i do relacji $R2$. W języku naturalnym można powyższą operację sformułować następująco: „Podać dane nauczycieli, którzy są studentami”.

□

Iloraz relacji jest operatorem dwuargumentowym i oznaczamy go symbolem „÷”. Niech A będzie zbiorem atrybutów występujących tylko w relacji $R1$, B – zbiorem atrybutów występujących w relacji $R1$ i w $R2$, natomiast relacja $R2$ ma tylko zbiór atrybutów B . Relacja będąca wynikiem ilorazu $R1 \div R2$ zawiera atrybuty A oraz krotki o wartościach z dziedzin atrybutów z A takie, że krotka o wartościach z $\{A, B\}$ występuje w $R1$ dla wszystkich wartości z B występujących w relacji $R2$.

Przykład 2.14

Dane są relacje *Nauczyciele* (tabela 2.22) i *Miasta* (tabela 2.23).

Tabela 2.22Relacja *Nauczyciele*

<i>Id</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>IdMiasta</i>
0001	Wabacki	Jan	1
0002	Jarecki	Jan	1
0004	Kotecki	Adam	2
0005	Abacka	Ewa	3
0006	Wabacki	Marek	1
0007	Wabacki	Kornel	2
0008	Wabacki	Adam	3

Tabela 2.23Relacja *Miasta*

<i>IdMiasta</i>	<i>Nazwa</i>
1	Opole
2	Nysa
3	Warszawa

Chcemy wyszukać nazwiska nauczycieli, takie, że w każdym mieście, którego nazwa jest w relacji *Miasta*, jest nauczyciel o tym nazwisku. W tym celu najpierw utworzymy rzut relacji *Nauczyciele* na atrybuty {*Nazwisko*, *IdMiasta*} (tabela 2.24)

$$Nauczyciele_rzut := \pi_{\{Nazwisko, IdMiasta\}}(Nauczyciele)$$

Tabela 2.24

Relacja *Nauczyciele_rzut*

<i>Nazwisko</i>	<i>IdMiasta</i>
Wabacki	1
Jarecki	1
Kotecki	2
Abacka	3
Wabacki	2
Wabacki	3

i rzut tabeli *Miasta* na atrybut {*IdMiasta*}

$$Miasta_rzut := \pi_{\{IdMiasta\}}(Miasta)$$

o danych zawartych w tabeli 2.25.

Tabela 2.25

Relacja *Miasta_rzut*

<i>IdMiasta</i>
1
2
3

Relacja

$$Wyniki_iloraz := Nauczyciele_rzut \div Miasta_rzut$$

zawiera krotki przedstawione w tabeli 2.26, z której wynika, że w każdym mieście z relacji *Miasta* jest nauczyciel o nazwisku "Wabacki".

Tabela 2.26

Relacja *Wyniki_iloraz*

<i>Nazwisko</i>
Wabacki

□

Na zakończenie tego rozdziału podkreślimy ważną własność operatorów algebry relacji: działają one na zbiorach krotek a nie na pojedynczych krotkach. Jest to bardzo ważna cecha relacyjnych systemów baz danych (odróżniająca je od wielu innych systemów nierelacyjnych, przetwarzających pojedyncze pakiety danych np. rekordy).

2.3. NORMALIZACJA SCHEMATÓW RELACJI

Normalizacja schematu relacji jest sformalizowaną procedurą, w wyniku której eliminujemy redundancję (powtarzanie się) danych zgromadzonych w bazie danych, ich niespójności i anomalie.

Normalizacja opiera się na idei funkcyjnej zależności atrybutów i polega na dekompozycji początkowego schematu relacji na kilka innych schematów relacji. Postacie normalne są numerowane kolejno. Im wyższy numer, tym projekt bazy danych powinien być coraz lepszy w tym sensie, że mniej podatny na anomalie, czyli nieprawidłowości związane z aktualizowaniem, usuwaniem i dopisywaniem danych.

Głównym celem projektowania relacyjnej bazy danych jest przedstawienie jej w postaci odpowiedniego zbioru schematów relacji.

Przeprowadzenie procesu **normalizacji** powoduje, że z początkowych schematów relacji wyłaniają się nowe schematy, wcześniej niedostrzegane lub pominięte. Na ogół normalizacja powoduje powstanie większej liczby schematów relacji i odpowiednio większej liczby tabel w relacyjnym logicznym modelu bazy danych (rozdz. 4). Dobrze zaprojektowana baza danych powinna zawierać schematy relacji przynajmniej w trzeciej postaci normalnej a ich liczba powinna być jak najmniejsza. Opis zagadnień związanych z normalizacją można znaleźć w wielu zamieszczonych na końcu pracy pozycjach literatury (np. [Beynon1999, Beynon2000, Date2000, Harris1994, Pankow1992, Riordan2000, Roszko1998]).

Omówimy teraz pojęcia związane z normalizacją schematów relacji. Podstawowym pojęciem jest zależność funkcyjna atrybutów.

Mówimy, że atrybut B relacji R jest *funkcyjnie zależny* od atrybutu A tej relacji, co zapisujemy

$$A \rightarrow B$$

jeśli zawsze każdej wartości a atrybutu A odpowiada nie więcej niż jedna wartość b atrybutu B .

Stwierdzenie, że atrybut B jest funkcyjnie zależny od atrybutu A jest równoważne stwierdzeniu, że atrybut A jednoznacznie wyznacza (identyfikuje) atrybut B .

Przykład 2.15

W schemacie relacji

Osoby (*PESEL*, *Nazwisko*)

atrybut *PESEL* jednoznacznie określa nazwisko osoby (atrybut *Nazwisko* jest funkcyjnie zależny od atrybutu *PESEL*), czyli

PESEL $\rightarrow$ *Nazwisko*

przy czym dane nazwisko może posiadać więcej niż jedna osoba (a więc *PESEL* nie jest funkcyjnie zależny od nazwiska).

□

Pojęcie funkcyjnej zależności stosuje się odpowiednio do zbiorów atrybutów *Z1* i *Z2*, które są dowolnymi podzbiórami zbioru atrybutów relacji *R*.

Mówimy, że zbiór atrybutów *Z2* relacji *R* jest **funkcyjnie zależny** od zbioru atrybutów *Z1* tej relacji, co zapisujemy

$$Z1 \rightarrow Z2$$

jeśli zawsze każdej krotce wartości atrybutów zbioru *Z1* odpowiada nie więcej niż jeden krotka wartości atrybutów zbioru *Z2*.

Zbiór *Z1* nazywa się **wyznacznikiem** zależności funkcyjnej.

Czasami zamiast określenia „zależności funkcyjne” będziemy pisać po prostu „zależności”.

Zależność funkcyjna $\{A_1, A_2, \dots, A_n\} \rightarrow \{B_1, B_2, \dots, B_m\}$ jest **trywialna** wtedy i tylko wtedy, gdy prawa strona zależności jest podzbiorem lewej strony czyli

$$\{B_1, B_2, \dots, B_m\} \subseteq \{A_1, A_2, \dots, A_n\},$$

np. $\{\textit{Student}, \textit{Kurs}\} \rightarrow \textit{Student}$.

Jeśli zbiór atrybutów w zależności funkcyjnej jest jednoelementowy, to w zapisie zależności będziemy pomijali nawiasy klamrowe oznaczające zbiór.

Przykład 2.16

W schemacie relacji

Egzaminy (*NumerIndeksu*, *Kurs*, *Ocena*, *Data*)

zbiór $\{\textit{NumerIndeksu}, \textit{Kurs}\}$ jednoznacznie wyznacza zbiór $\{\textit{Ocena}, \textit{Data}\}$, przy założeniu, że każdy student może mieć tylko jedną ocenę z danego kursu z daną datą czyli zbiór atrybutów $\{\textit{Ocena}, \textit{Data}\}$ jest funkcyjnie zależny od zbioru atrybutów $\{\textit{NumerIndeksu}, \textit{Kurs}\}$

$$\{\textit{NumerIndeksu}, \textit{Kurs}\} \rightarrow \{\textit{Ocena}, \textit{Data}\}.$$

Zbiór $\{\textit{NumerIndeksu}, \textit{Kurs}\}$ nie wyznacza jednoznacznie atrybutu *Ocena* (student mógł najpierw dostać ocenę niedostateczną). Również zbiór $\{\textit{Ocena}, \textit{Data}\}$ nie wyznacza jednoznacznie zbioru $\{\textit{NumerIndeksu}, \textit{Kurs}\}$, czyli

$$\{\textit{NumerIndeksu}, \textit{Kurs}\} \nrightarrow \textit{Ocena}$$

$$\{Ocena, Data\} \rightarrow \{NumerIndeksu, Kurs\}$$

□

Kolejnym ważnym pojęciem jest pełna funkcyjna zależność atrybutów oznaczana symbolem $\Rightarrow$.

Zbiór atrybutów $Z2$ relacji R jest w **pełni funkcyjnie zależny** od zbioru atrybutów $Z1$ tej relacji, co zapisujemy

$$Z1 \Rightarrow Z2,$$

jeśli zbiór $Z2$ jest funkcyjnie zależny od $Z1$, ale nie jest funkcyjnie zależny od żadnego właściwego podzbioru zbioru $Z1$.

Zbiór $Z1$ nazywa się **wyznacznikiem** pełnej zależności funkcyjnej.

Przykład 2.17

a) W schemacie relacji z poprzedniego przykładu (student może zdawać kilka razy egzamin z danego kursu ale w danym dniu tylko jeden raz)

Egzaminy ($NumerIndeksu, Kurs, Ocena, Data$)

zachodzi zależność

$$\{NumerIndeksu, Kurs\} \Rightarrow \{Ocena, Data\}$$

czyli zbiór $\{Ocena, Data\}$ jest w pełni funkcyjnie zależny od zbioru atrybutów $\{NumerIndeksu, Kurs\}$.

b) Dany jest schemat relacji

Oceny_końcowe ($Student, Przedmiot, Data, Ocena$),

do przechowywania danych o ocenach końcowych studentów z różnych przedmiotów.

W schemacie relacji **Oceny_końcowe** zachodzą zależności:

$$\{Student, Przedmiot\} \Rightarrow Data$$

$$\{Student, Przedmiot\} \Rightarrow Ocena$$

czyli atrybut $Data$ (podobnie jak atrybut $Ocena$) jest w pełni funkcyjnie zależny od zbioru atrybutów (klucza złożonego) $\{Student, Przedmiot\}$ ponieważ jest funkcyjnie zależny od zbioru atrybutów $\{Student, Przedmiot\}$ a nie jest funkcyjnie zależny ani od atrybutu $Student$ ani od atrybutu $Przedmiot$.

□

W rozdziale 2.1 zdefiniowaliśmy klucz relacji jako zbiór identyfikujący relacji, którego żaden podzbiór nie jest zbiorem identyfikującym, czyli jest to atrybut (lub zbiór atrybutów), którego wartość jednoznacznie identyfikuje każdą krotkę w relacji. Kluczem relacji jest więc minimalny zbiór identyfikujący relacji.

Zauważmy, że jeśli klucz główny relacji jest kluczem prostym, to wszystkie zależności funkcyjne względem klucza są zależnościami pełnymi.

Zanim omówimy postaci normalne schematów relacji zauważmy, że dane mogą być przedstawione w postaci tabeli nieznormalizowanej czyli dane w polach tabeli mogą być zbiorami wartości a nie wartościami elementarnymi. Tabela, w której wszystkie wartości są elementarne (atomowe) i nie ma powtarzających się wierszy jest reprezentacją relacji o schemacie w pierwszej postaci normalnej.

Podamy teraz definicje postaci normalnych schematów relacji.

Pierwsza postać normalna (1PN) – (ang. *first normal form*) schemat relacji jest w pierwszej postaci normalnej wtedy i tylko wtedy, gdy każda wartość z dziedziny atrybutów jest elementarna (atomowa, nierozkładalna).

Inaczej można to wyrazić, że nie ma powtarzających się grup wartości tzn. że każda składowa krotki jest pojedynczą (elementarną) wartością a nie zbiorem wartości.

Przykład 2.18

a) schematy relacji w pierwszej postaci normalnej:

Klienci (*NrKlienta*, *Nazwisko*, *Imię*, *Miasto*, *UlicaNr*)

Kasety (*NrKasety*, *Opis*)

Filmy (*KodFilmu*, *Tytuł*, *CzasTrwania*)

Gatunki (*KodGatunku*, *Rodzaj*)

Dowolnie wybrana składowa krotki z relacji o powyższych schematach ma zawsze dokładnie jedną wartość atomową z dziedziny atrybutu.

Relację *Klienci* o schemacie **Klienci** przedstawia tabela 2.27.

Tabela 2.27

Relacja *Klienci* w pierwszej postaci normalnej

<i>NrKlienta</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>Miasto</i>	<i>UlicaNr</i>
0001	Batycki	Jerzy	Opole	Polna 34
0002	Batycki	Marek	Opole	Polna 34
0003	Widecka	Barbara	Sopot	Rynek 1/1
0004	Artecka	Barbara	Opole	Rynek 2

b) rozważmy tabelę *Osoby* (tabela 2.28) zawierającą kolumny:

NrOsoby, *Nazwisko*, *Imię*, *ImionaDzieci*, *DatyUrDzieci*

Tabela 2.28 nie reprezentuje żadnej relacji, jest nieznormalizowana czyli nie jest w pierwszej postaci normalnej, ponieważ np. w kolumnie *ImionaDzieci* dla Padurskiej Marii jest zbiór wartości {Adam, Ewa, Hanna} a nie pojedyncza wartość.

Tabela 2.28

Nieznormalizowana tabela *Osoby*

<i>NrOsoby</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>ImionaDzieci</i>	<i>DatyUrDzieci</i>
1	Adurska	Janina		
2	Padurska	Genowefa	Adam	10-03-1960
3	Padurska	Maria	Adam	11-05-1950
			Ewa	01-01-1954
			Hanna	04-02-1960
4	Wadurska	Teresa	Monika	03-08-1962
			Roland	03-08-1962

c) schemat relacji **Osoby**:

Osoby (*NrOsoby*, *Nazwisko*, *Imię*, *ImięDziecka*, *DataUrDziecka*).

jest w pierwszej postaci normalnej.

Relacja *Osoby* o schemacie **Osoby**, reprezentowana przez tabelę 2.29 posiada wiele wad, np. nie występują w niej dane Janiny Adurskiej (zobacz tabela 2.28), która nie posiada dzieci.

Tabela 2.29

Znormalizowana relacja *Osoby* (w pierwszej postaci normalnej)

<i>NrOsoby</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>ImięDziecka</i>	<i>DataUrDziecka</i>
2	Padurska	Genowefa	Adam	10-03-1960
3	Padurska	Maria	Adam	11-05-1950
3	Padurska	Maria	Ewa	01-01-1954
3	Padurska	Maria	Anna	04-02-1960
4	Wadurska	Teresa	Monika	03-08-1962
4	Wadurska	Teresa	Roland	03-08-1962

Analizując relację *Osoby* można zauważyć, że dane osoby powtarzają się dla każdego dziecka tej osoby, co znacznie wydłuża i utrudnia wykonywanie operacji na danych np. aktualizowanie (anomalie przy aktualizacji) oraz bardzo łatwo można utracić spójność danych np. przez zmodyfikowanie nazwiska „Padurska” na „Paderska” tylko dla dziecka o imieniu Ewa lub zmodyfikować wszystkie nazwiska „Padurska” na „Paderska” zamiast tylko Padurskiej Marii. Ponadto występują anomalie przy dopisywaniu danych np. nie można dopisać danych dziecka bez dopisania danych rodzica i nie można dopisać osoby bez dziecka (atrybut *ImięDziecka* wchodzi w skład klucza głównego, więc nie może mieć wartości NULL) oraz anomalie przy usuwaniu danych (usuwając dane o wszystkich dzieciach pracownika trzeba usunąć dane o pracowniku).

□

Druga postać normalna (2PN) – (ang. *second normal form*) schemat relacji jest w drugiej postaci normalnej, jeśli każdy atrybut schematu relacji, nie wchodzący w skład żadnego klucza kandydującego, jest w pełni funkcyjnie zależny od każdego klucza kandydującego.

Zauważmy, że schematy relacji, które są w pierwszej postaci normalnej, i których wszystkie klucze kandydujące są kluczami prostymi, są również w drugiej postaci normalnej.

W celu uzyskania drugiej postaci normalnej należy z danego schematu relacji utworzyć kilka schematów relacji (na ogół o mniejszej liczbie atrybutów) tak, aby wszystkie atrybuty w poszczególnych schematach relacji były w pełni funkcyjnie zależne od klucza.

Symbolicznie sprowadzanie schematu do drugiej postaci normalnej możemy zapisać następująco:

schemat relacji $R(\underline{A}, B, C, D)$, w którym zachodzą zależności:

$$\{\underline{A}, B\} \rightarrow C,$$

$$A \rightarrow D,$$

należy zastąpić dwoma schematami (wykonać odpowiednie projekcje):

$$R1(\underline{A}, B, C), \text{ w którym } \{\underline{A}, B\} \rightarrow C,$$

oraz

$$R2(\underline{A}, D), \text{ w którym } \underline{A} \rightarrow D.$$

Przykład 2.19

Dany jest schemat relacji

Osoby (*PESEL, Nazwisko, Imię, ImięDziecka, DataUrDziecka*)

do przechowywania danych o pracownikach i ich dzieciach. Każda osoba, której dane przechowujemy, ma co najmniej jedno dziecko.

W schemacie relacji **Osoby** istnieją następujące zależności funkcyjne:

$$PESEL \rightarrow Nazwisko$$

$$PESEL \rightarrow Imię,$$

$$\{PESEL, ImięDziecka\} \rightarrow DataUrDziecka$$

Jedynym kluczem schematu relacji **Osoby** jest zbiór $\{PESEL, ImięDziecka\}$. Atrybuty niekluczowe *Nazwisko, Imię* nie są w pełni funkcyjnie zależne od klucza (bo zależą funkcyjnie od atrybutu *PESEL*) czyli schemat relacji **Osoby** nie jest w drugiej postaci normalnej. Dlatego należy zdekomponować go na dwa schematy relacji:

Dorośli (*PESEL, Nazwisko, Imię*)

Dzieci (*PESEL, ImięDziecka, DataUrDziecka*)

które są w drugiej postaci normalnej (dlaczego?).

□

Zanim przejdziemy do omówienia trzeciej postaci normalnej wprowadzimy pojęcie przechodniej zależności funkcyjnej.

Niech A, B, C będą trzema rozłącznymi podzbiorami atrybutów danego schematu relacji. Podzbiór atrybutów C jest **przechodnio (tranzytywnie) funkcyjnie zależny** od podzbioru atrybutów A (ang. *transitively dependent*), co zapisujemy

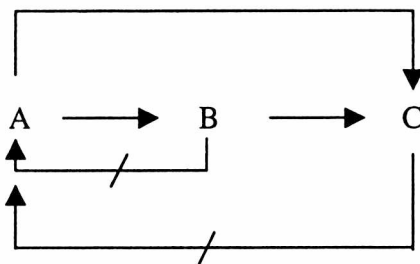
$$A \rightarrow C$$

jeśli

$$A \rightarrow B, B \rightarrow C, B \nrightarrow A, C \nrightarrow A$$

(B zależy funkcyjnie od A, C zależy funkcyjnie od B, A nie zależy funkcyjnie od B i A nie zależy funkcyjnie od C).

Graficznie przechodnią zależność funkcyjną przedstawia rysunek 2.1.



Rys. 2.1. Przechodnia zależność funkcyjna podzbioru atrybutów C od A

Przykład 2.20

W schemacie relacji

Faktury (NrFaktury, NazwaKlienta, AdresKlienta)

istnieją następujące zależności funkcyjne:

$$NrFaktury \rightarrow NazwaKlienta$$

$$NazwaKlienta \rightarrow AdresKlienta$$

Atrybut *AdresKlienta* jest przechodnio funkcyjnie zależny od *NrFaktury* ponieważ zależy od atrybutu *NazwaKlienta*, który jest funkcyjnie zależny od *NrFaktury*. Z kolei *NrFaktury* nie jest funkcyjnie zależny od atrybutu *NazwaKlienta* ponieważ jeden klient może mieć wiele faktur. Również *NrFaktury* nie zależy funkcyjnie od atrybutu *AdresKlienta* ponieważ dany adres może być na wielu fakturach.

□

Trzecia postać normalna (3PN) – (ang. *third normal form*) schemat relacji jest w trzeciej postaci normalnej, jeśli jest w drugiej postaci normalnej i każdy **atrybut niekluczowy** (nie wchodzący w skład żadnego klucza kandydującego) nie jest przechodnio funkcyjnie zależny od żadnego klucza kandydującego (brak zależności funkcyjnych między atrybutami niekluczowymi).

Jeśli dla danego schematu relacji zachodzi przechodnia zależność funkcyjna taka jak na rysunku 2.1, to schemat nie jest w trzeciej postaci normalnej.

Aby schemat relacji doprowadzić do trzeciej postaci normalnej, w którym atrybuty są w przechodniej zależności funkcyjnej od klucza, należy zdekomponować go na nowe schematy relacji, np. ze schematu relacji $R(\underline{A}, B, C)$, gdzie

$$\begin{aligned}\underline{A} &\rightarrow B, \\ B &\rightarrow C, \\ B &\not\rightarrow A, C \not\rightarrow A\end{aligned}$$

czyli

$$\underline{A} \rightarrow C,$$

należy utworzyć (poprzez odpowiednie operacje rzutowania) dwa schematy relacji $R1(\underline{A}, B)$ oraz $R2(\underline{B}, C)$.

Przykład 2.21

Schemat relacji

Faktury (NrFaktury, NazwaKlienta, AdresKlienta),

w którym jest przechodnia zależność funkcyjna (zobacz przykład 2.20), należy zastąpić dwoma schematami relacji:

Wykazy (NrFaktury, NazwaKlienta)

Klienci (NazwaKlienta, AdresKlienta),

które są w 3PN.

□

Przykład 2.22

Założmy, że w schemacie relacji

Zajęcia (Student, Kurs, Wykładowca)

zachodzą zależności funkcyjne:

$$\{\underline{Student}, \underline{Kurs}\} \rightarrow Wykładowca$$

(student może uczęszczać na dany kurs tylko do jednego wykładowcy);

$$Wykładowca \rightarrow Kurs$$

(każdy wykładowca prowadzi dokładnie jeden kurs).

W schemacie relacji **Zajęcia**, który jest już w trzeciej postaci normalnej, nadal występują anomalie związane z usuwaniem danych (po usunięciu danych o ostatnim studencie uczęszczającym na dany kurs tracimy dane o tym kursie) oraz anomalie dopisywania danych (nie można dopisać kursu i wykładowcy, zanim na kurs nie zapisze się co najmniej jeden student).

□

Poszczególne etapy normalizacji przedstawia tabela 2.30.

Tabela 2.30

Etapy normalizacji schematu relacji

Tabela nieznormalizowana	Wartości w tabeli nie są atomowe
Pierwsza postać normalna	Wszystkie wartości atrybutów relacji o danym schemacie są atomowe
Druga postać normalna	Schemat jest w pierwszej postaci normalnej i atrybuty niekluczowe są w pełni funkcyjnie zależne od klucza
Trzecia postać normalna	Schemat relacji jest w drugiej postaci normalnej i każdy atrybut niekluczowy nie jest przechodnio funkcyjnie zależny od żadnego klucza kandydującego

W praktyce normalizację schematu relacji kończy się najczęściej na trzeciej postaci normalnej. W teorii relacyjnych baz danych definiuje się kolejne postaci, które krótko przedstawimy poniżej.

Postać normalna Boyce’a–Codda (PNBC) – (ang. *Boyce–Codd normal form*) jest to poprawiona wersja trzeciej postaci normalnej. Schemat relacji **R** jest w postaci normalnej Boyce’a–Codda, jeśli wszystkie zależności funkcyjne w schemacie relacji determinowane są przez klucze (**wszystkie atrybuty zależą tylko i wyłącznie od kluczy**).

Z definicji PNBC wynika, że w schemacie relacji **R**, będącym w PNBC, jeśli zbiór atrybutów **A** jest wyznacznikiem jakiegokolwiek atrybutu spoza **A**, to zbiór **A** musi być wyznacznikiem każdego atrybutu w tym schemacie.

Schemat, który jest w postaci normalnej Boyce’a–Codda nie zawiera redundancji, niespójności i anomalii.

Trzecia postać normalna nie dopuszcza zależności funkcyjnych między atrybutami niekluczowymi ale dopuszcza:

- zależności między atrybutami kluczowymi (atrybutami wchodzącymi w skład klucza)
- zależności atrybutów kluczowych od atrybutów niekluczowych,
- zależności atrybutów kluczowych od części klucza, do którego nie należą.

3PN dopuszcza więc zależność postaci $A \rightarrow B$, gdzie **A** nie jest kluczem kandydującym a **B** jest atrybutem klucza głównego (zobacz przykład 2.22). Zależność ta jest przyczyną redundancji, którą może wyeliminować silniejsza postać normalna czyli postać normalna Boyce’a–Codda. W schemacie relacji w PNBC każdy wyznacznik zależności funkcyjnej jest kluczem kandydującym.

Postać Boyce’a–Codda określa, że nie mogą istnieć zależności funkcyjne między kluczami kandydującymi.

Przykład 2.23

Schemat relacji

Zajęcia (*Student*, *Kurs*, *Wykładowca*),

w którym zachodzą zależności funkcyjne:

$\{Student, Kurs\} \rightarrow Wykładowca$ (student może uczęszczać na dany kurs tylko do jednego wykładowcy);

$Wykładowca \rightarrow Kurs$ (każdy wykładowca prowadzi dokładnie jeden kurs),

jest w 3PN ale nie jest w postaci normalnej Boyce'a–Codda (pomimo, że *Wykładowca* $\rightarrow$ *Kurs*, to nie zachodzi zależność *Wykładowca* $\rightarrow$ *Student*, czyli atrybut *Kurs* zależy od atrybutu niekluczowego *Wykładowca*) i dlatego występują anomalie usuwania i dopisywania danych.

Schemat relacji **Zajęcia** powinien zostać zastąpiony następującymi dwoma schematami, które są w PNBC:

Przydział (*Wykładowca*, *Kurs*),

Zapisy (*Student*, *Kurs*).

W praktyce okazuje się jednak, że silniejsza PNBC nie zawsze jest pożądana, gdyż może doprowadzić do utraty pewnych zależności funkcyjnych, i często jest lepiej poprzestać na 3PN. Zastąpienie schematu relacji **Zajęcia** (3PN) przez schematy **Przydział** i **Zapisy** (PNBC), spowodował utratę zależności

$\{Student, Kurs\} \rightarrow Wykładowca$.

□

W schematach relacji oprócz zależności funkcyjnych często występują zależności wielowartościowe.

Zależność wielowartościowa (ang. *multivalued dependency*, *MVD*) zbioru atrybutów B od zbioru atrybutów A, co zapisujemy jako

$A \twoheadrightarrow B$

oznacza, że w schemacie relacji, który zawiera dowolne podzbiory atrybutów A, B, C, zbiór wartości B odpowiadający danej parze (wartość A, wartość C) zależy jedynie od wartości A i jest niezależny od C.

Zależność funkcyjna $A \rightarrow B$ oznacza, że krotki wartości atrybutów zbioru A wyznaczają jednoznacznie krotkę wartości atrybutów zbioru B, natomiast zależność wielowartościowa $A \twoheadrightarrow B$ oznacza, że krotki wartości atrybutów zbioru A wyznaczają **zbiór** krotek wartości atrybutów zbioru B.

Każda zależność funkcyjna jest trywialną zależnością wielowartościową.

Przykład 2.24**Schemat relacji****Osoby** (*NrPrac*, *Stanowisko*, *ImięDziecka*)

jest w pierwszej postaci normalnej i zawiera dwie wielowartościowe zależności:

- (1) $NrPrac \twoheadrightarrow Stanowisko$

czyli dla dowolnej pary wartości atrybutów (*NrPrac*, *ImięDziecka*) atrybut *Stanowisko* zależy tylko od atrybutu *NrPrac* a jest niezależny od atrybutu *ImięDziecka*,

- (2) $NrPrac \twoheadrightarrow ImięDziecka$

czyli dla dowolnej pary wartości atrybutów (*NrPrac*, *Stanowisko*) atrybut *ImięDziecka* zależy tylko od atrybutu *NrPrac* a jest niezależny od atrybutu *Stanowisko*.

Atrybutowi *NrPrac* w relacji *Osoby* o schemacie **Osoby** może odpowiadać wiele wartości atrybutu *Stanowisko* (dotychczas zajmowane stanowiska w firmie) i wiele wartości atrybutu *ImięDziecka* (tabela 2.31).

Tabela 2.31Relacja *Osoby*

<i>NrPrac</i>	<i>Stanowisko</i>	<i>ImięDziecka</i>
1	dyrektor	Adam
2	programista	Adam
2	projektant	Adam
2	programista	Ewa
2	projektant	Ewa
2	programista	Hanna
2	projektant	Hanna
3	programista	Monika
3	analityk	Monika
3	programista	Ewa
3	analityk	Ewa

Relacja *Osoby* zawiera dużo powtarzających się danych. Schemat relacji **Osoby** pomimo, że jest w postaci normalnej Boyce'a–Codda (ponieważ zawiera same atrybuty kluczowe) narażony jest na anomalie usuwania, modyfikacji i dopisywania danych.

□

Czwarta postać normalna (4PN) – schemat relacji jest w czwartej postaci normalnej wtedy i tylko wtedy, gdy jest w PNBC i nie zawiera wielowartościowej nietrywialnej zależności atrybutów.

Zależność jest trywialna (oczywista), jeżeli zawsze jest spełniona (nie może być nie spełniona).

Sprowadzenie schematu relacji do czwartej postaci normalnej polega na usunięciu wielowartościowych zależności funkcyjnych.

Sprowadzenie dowolnego schematu relacji do PNBC jest zawsze możliwe.

Schemat relacji **Osoby** (przykład 2.24) nie jest w 4PN ponieważ zawiera nietrywialne wielowartościowe zależności funkcyjnych.

Przykład 2.25

Aby poprawić schemat relacji, będący w PNBC (z przykładu 2.24)

Osoby (*NrPrac*, *Stanowisko*, *ImięDziecka*)

należy zdekomponować go na dwa schematy, które są również w PNBC:

Dorośli (*NrPrac*, *Stanowisko*)

Dzieci (*NrPrac*, *ImięDziecka*)

Schemat relacji **Osoby** nie jest w 4PN, natomiast schematy relacji **Dorośli** i **Dzieci** są w 4PN.

Relację *Osoby* przedstawioną w tabeli 2.31, o schemacie **Osoby**, zastępujemy relacjami:

Dorośli (tabela 2.32) o schemacie **Dorośli**,

Dzieci (tabela 2.33) o schemacie **Dzieci**.

Relację *Osoby* można otrzymać ponownie poprzez złączenie naturalne relacji *Dorośli* i *Dzieci*.

Tabela 2.32

Relacja *Dorośli*

<i>NrPrac</i>	<i>Stanowisko</i>
1	dyrektor
2	programista
2	projektant
3	programista
3	analityk

Tabela 2.33

Relacja *Dzieci*

<i>NrPrac</i>	<i>ImięDziecka</i>
1	Adam
2	Adam
2	Ewa
2	Hanna
3	Monika
3	Ewa

□

Aby zdefiniować piątą postać normalną wprowadzimy pojęcie zależności złączenia.

Niech $A, B, \dots, Z$ będą dowolnymi podzbiorami zbioru atrybutów schematu relacji R . Mówimy, że relacja R o schemacie R spełnia **zależność złączenia** ($A, B, \dots, Z$) wtedy i tylko wtedy, gdy relacja R jest równa złączeniu swoich rzutów na $A, B, \dots, Z$.

Piąta postać normalna (5PN – postać normalna rzutu-złączenia) – schemat relacji R jest w piątej postaci normalnej wtedy i tylko wtedy, gdy każda zależność złączenia w R jest implikowana kluczami kandydującymi schematu relacji. Inaczej można to wyrazić, że schemat jest w piątej postaci normalnej, jeżeli jest w czwartej postaci normalnej i nie istnieje odwracalny rozkład na schematy relacji.

Schemat relacji, który jest w 5PN nie zawiera anomalii, które by można usunąć za pomocą operacji projekcji.

Piąta postać normalna w praktyce ma zastosowanie w wyjątkowo rzadkich przypadkach.

Przykład 2.26

Schemat relacji

Dostawy (Dostawca, Towar, Odbiorca),

który jest w 4PN ale nie jest w 5PN, zastępujemy trzema schematami relacji, które są w 5PN:

Kto_Co (Dostawca, Towar)

Kto_Komu (Dostawca, Odbiorca)

Co_Komu (Towar, Odbiorca)

□

W niniejszym rozdziale przedstawiono sposób tworzenia poprawnego schematu bazy danych czyli zbioru schematów relacji (rozdz. 4.1.3) poprzez wyjście od jednego globalnego schematu relacji i określenie zależności funkcyjnych, a następnie

przeprowadzenie procesu normalizacji i sprowadzenie schematów relacji co najmniej do 3PN.

Na zakończenie tego rozdziału pragniemy podkreślić, że normalizacja jest procesem tworzenia schematów relacji, które nie podlegają anomalii. Dane przechowywane w relacjach znormalizowanych zajmują mniej miejsca, nie powtarzają się (brak redundancji danych), łatwiejsza jest ich modyfikacja, dopisywanie oraz ich usuwanie (nie ma anomalii przy dodawaniu, usuwaniu, aktualizowaniu danych).

Głównym celem normalizacji jest usunięcie nadmiarowości z bazy danych. Date definiuje redundancję następująco: jeżeli w relacji R występuje zależność funkcyjna $A \rightarrow B$ oraz A nie jest kluczem kandydującym i relacja ma co najmniej dwie krotki, to w relacji R występuje **redundancja** [Date2000].

Ideę procesu normalizacji można zawrzeć w punktach:

- wartości atrybutów są elementarne (bez powtórzeń),
- atrybuty zależą tylko od całego klucza (a nie od podzbioru klucza),
- i od niczego innego, tylko od klucza.

W praktyce często po przeprowadzonej normalizacji danych przeprowadza się denormalizację danych [Barker1996, Harris1994] polegającą na łączeniu niektórych znormalizowanych relacji (np. w przypadku gdy redundancje danych nie są zbyt duże) w celu przyspieszenia operacji wykonywanych na danych. Jeśli projektant przewiduje, że w zaimplementowanym systemie często będą wykonywane operacje łączenia relacji, które są bardzo czasochłonne, to musi dokonać wyboru, czy lepiej wykonać denormalizację (pojawia się znowu anomalie, które eliminowała normalizacja), czy pogodzić się z wolniejszym wykonywaniem operacji bazodanowych.

2.4. PODSTAWOWE POJĘCIA BAZ DANYCH

Podamy teraz podstawowe pojęcia związane z relacyjnymi bazami danych. Nieformalnie, abstrakcyjne pojęcie „relacja” zastępuje się pojęciem „tabela”. Przypomnijmy podstawowe własności wynikające z definicji relacji:

1. relacja jest zbiorem krotek więc nie zawiera dwóch identycznych krotek,
2. krotki relacji nie są uporządkowane (elementy zbioru bez określonego porządku),
3. atrybuty w schemacie relacji nie są uporządkowane,
4. składowe krotek są wartościami elementarnymi (atomowymi) należącymi do dziedziny atrybutu.

Reprezentowanie relacji w postaci narysowanych tabel czyni relacyjny model danych czytelnym i zrozumiałym.

Tabela (ang. *table*) jest jedną z najczęstszych reprezentacji relacji. Tabela jest podzielona na wiersze i kolumny.

Kolumna (ang. *column*) zawiera wartości z danej dziedziny, jest odpowiednikiem atrybutu relacji.

Wiersz (ang. *row*) odpowiada krotce relacji, czyli jest wystąpieniem (elementem) relacji.

Pole (ang. *field*) jest to przecięcie wiersza i kolumny, jest odpowiednikiem składowej krotki. Wartość każdego pola powinna być wartością atomową (elementarną). Odwołanie do pola następuje poprzez nazwę kolumny.

Należy pokreślić, że *tabela* i *relacja* to nie jest to samo. Tabela jest jedynie wizualnym sposobem przedstawienia relacji. Reprezentacja relacji w postaci tabeli sugeruje wiele nieprawdziwych skojarzeń, np. że wiersze tabeli (czyli krotki relacji) są uporządkowane w przedstawiony sposób, że jest wiersz pierwszy, następny, poprzedni (czyli w relacji jest krotka pierwsza, następna, poprzednia), że kolumny tabeli (czyli atrybuty relacji) są uporządkowane od lewej do prawej – wszystkie te stwierdzenia są oczywiście nieprawdziwe.

Tabelę można traktować jako obraz relacji, jeśli ustalimy odpowiednie reguły interpretacji takiej reprezentacji, które podajemy poniżej.

Każda tabela ma swoją unikalną nazwę odpowiadającą nazwie relacji, którą reprezentuje. Tabela składa się z **nagłówka**, zawierającego nazwy kolumn (odpowiadające nazwom atrybutów), oraz z wierszy (odpowiedników krotek), stanowiących **treść** tabeli. Nazwa kolumny jest jednoznaczna w obrębie danej tabeli. Kolejność kolumn jest dowolna. Każdy wiersz tabeli odpowiada jednej krotce relacji i musi się różnić od pozostałych wierszy w tabeli chociaż jedną wartością. Kolejność wierszy jest dowolna. Tabela może być wypełniana przez dopisywanie wierszy. Dane w tabeli można zmieniać. Można usuwać wiersze z tabeli. Można wyszukiwać dane. Czynności takie jak dopisywanie kolumn do tabeli, modyfikowanie nazw kolumn tabeli, usuwanie kolumn z tabeli również można wykonywać, ale w praktyce, w zaimplementowanej bazie danych, takie operacje wykonuje się bardzo rzadko i powinno się tego unikać.

Zwróćmy uwagę, że w tabeli może być kilka kolumn o takiej samej dziedzinie, przy czym nazwy kolumn muszą być różne (np. dla dwóch różnych kolumn *Imię Ojca*, *Imię Matki*, może być ta sama dziedzina – np. *Imię* – łańcuch maksymalnie 15. literowy). Kolejność dziedzin w relacji matematycznej jest ściśle ustalona, ale w przypadku tabeli w modelu relacyjnym nie ma ona znaczenia (z kolei – niestety – ma znaczenie np. w strukturalnym języku zapytań do zarządzania bazami danych **SQL** – ang. *Structured Query Language* [DateDar2000], str. 102).

Bardzo często określenie tabela jest używane również w odniesieniu do plików komputerowych odpowiadających tabelom relacyjnym.

Plik (ang. *file*) jest zbiorem (logicznie powiązanych) informacji przechowywanych np. w pamięci dyskowej. Identyfikowany jest poprzez swoją nazwę.

Rekord (ang. *record*) jest podstawową jednostką informacji przechowywania danych w pliku. Odpowiednikiem rekordu jest krotka w relacji i wiersz w tabeli. Rozmiar rekordu w danym pliku jest stały.

Relacyjna baza danych (ang. *relational database*) to baza danych oparta na relacyjnym modelu danych. W praktycznych implementacjach założenia modelu relacyjnego sformułowane przez E.F. Codd są modyfikowane lub nie przestrzegane. Z tego względu, większość baz danych określanych jako relacyjne, tak naprawdę relacyjnymi nie są. Relacyjna baza danych jest postrzegana jako zbiór nazwanych dwuwymiarowych tabel o określonej liczbie kolumn i nieograniczonej (ale skończonej) liczbie wierszy, co nie oznacza, że dane są pamiętane w postaci fizycznych tablic. Wartości przechowywane w relacyjnej bazie danych są atomowe i jest do nich zapewniony dostęp poprzez języki zapytań wysokiego poziomu (np. QBE, SQL). Na danych można wykonywać operacje algebry relacyjnej.

System zarządzania bazą danych – SZBD (ang. *Database Management System – DBMS*) jest to oprogramowanie narzędziowe służące do zakładania, przechowywania i wydajnego przetwarzania dużych zbiorów informacji, np. ORACLE, INGRES, PROGRESS, ACCESS, dBASE, ObjectStore, Jasmine, GemStone, Versant. Podstawowe funkcje SZBD to zapewnienie bezpieczeństwa i integralności danych, umożliwianie autoryzacji użytkowników, kontroli dostępu do danych, przetwarzania transakcji, obsługę wielodostępu, utrzymywanie dzienników systemowych, tworzenie i uaktualnianie słownika danych, odtwarzanie bazy po awariach.

System zarządzania relacyjną bazą danych – SZRBD (ang. *Relational Database Management System – RDBMS*) jest to oprogramowanie narzędziowe służące do zakładania, przechowywania i przetwarzania relacyjnych baz danych, np. ORACLE, SYBASE, INFORMIX, INGRES, PROGRESS, ACCESS, dBASE.

System bazy danych (ang. *database system*) jest to baza danych wraz z jej systemem obsługi wykorzystującym określony SZBD i inne programy narzędziowe, np. arkusze kalkulacyjne, edytory tekstowe itd. Według C.J. Date [Date2000] system bazy danych to użytkownicy, dane, sprzęt i oprogramowanie (m.in. SZBD, programy narzędziowe, narzędzia do budowania aplikacji, pomoce projektowe, edytory, arkusze kalkulacyjne, generatory raportów itd.).

Transakcja (ang. *transaction*) jest to ciąg poleceń (operacji) przeznaczonych do wykonywania w całości albo wcale, np. przelanie pewnej kwoty pieniędzy z jednego konta na drugie wiąże się z wykonaniem transakcji składającej się z kilku czynności takich jak sprawdzenie, czy na pierwszym koncie jest żądana kwota pieniędzy, jeśli tak, to odjęcie danej kwoty z pierwszego konta oraz dodanie kwoty do stanu drugiego konta.

Perspektywa (ang. *view*) – inaczej widok, wirtualna tabela, zdefiniowana za pomocą innych wyjściowych tabel, zawiera dane z wybranych kolumn i wierszy z jednej lub więcej tabel, nie może istnieć samodzielnie [Date2000].

Integralność (ang. *integrity*) bazy danych oznacza, że baza danych jest stale zgodna z rzeczywistością, odzwierciedla aktualny stan danych w rzeczywistości oraz ich zmiany.

Trwałość (ang. *durability*) bazy danych oznacza, że dane są trwale przechowywane w bazie danych.

Spójność (ang. *consistency*) bazy danych oznacza poprawność poszczególnych danych oraz poprawność bazy danych jako całości. Zapewnienie spójności bazy danych jest podstawowym problemem związanym z definiowaniem transakcji. Gdyby w trakcie wykonywania transakcji przelewania pieniędzy z jednego konta na drugie po wykonaniu czynności odjęcia danej kwoty z pierwszego konta nastąpiła awaria systemu, to w celu zapewnienia spójności bazy danych należy przywrócić stan bazy do stanu początkowego (zanim rozpoczęła się transakcja). Transakcja powinna więc być w całości zrealizowana poprawnie albo wcale.

3. PROJEKTOWANIE KONCEPTUALNE

Projektowanie konceptualne (pojęciowe) ma na celu zbudowanie modelu konceptualnego (CDM – ang. *Conceptual Data Model*). Aby opracowywany projekt bazy danych był poprawny i spełniał oczekiwania przyszłego użytkownika musimy uświadomić sobie, w jakim celu projektujemy bazę, kto będzie jej użytkownikiem, jakie są jego potrzeby, ograniczenia i wymagania. Nie wystarczy pobieżnie określić, że projektujemy bazę np. dla szkoły, ponieważ w zależności od tego, kto ma być jej użytkownikiem i do wykonywania jakich zadań dane będą wykorzystywane, projektowana baza może zawierać np. tylko dane pracowników (dział finansowy pracowników szkoły) albo tylko dane uczniów (księga uczniów), albo dane zarówno pracowników jak i uczniów (kompleksowa obsługa szkoły).

Pierwszym i zasadniczym elementem projektowania bazy danych jest więc zwięzłe i czytelne sformułowanie, w jakim celu projektujemy bazę danych. Jasno określony cel umożliwi przystąpienie do fazy analizy projektowania bazy danych.

3.1. FAZA ANALIZY

Dla danej rzeczywistości (przedsiębiorstwa) chcemy zaprojektować bazę danych czyli opracować model pewnych aspektów rozważanej rzeczywistości. Ten wyodrębniony wycinek rzeczywistości nazywamy **obszarem analizy** – OA (ang. *universe of discourse*) lub dziedziną analizy (miniświatem bazy danych). Proces analizowania potrzeb przedsiębiorstwa i zbierania od użytkowników wymagań wobec projektowanej bazy danych nazywamy **analizą wymagań** (ang. *requirements analysis*).

Prawidłowo zaprojektowana baza danych nie powinna zawierać żadnych zbędnych danych, powinna natomiast umożliwiać efektywną pracę zaimplementowanego systemu oraz umożliwiać jego rozbudowę.

Celem fazy analizy jest:

- a) określenie celów przedsięwzięcia i ich związków z projektowaną bazą danych,
- b) przeprowadzenie wnikliwej analizy tematu,
- c) przeprowadzenie z ekspertem dziedzinowym szczegółowego wywiadu, na podstawie którego będzie można precyzyjnie opisać określony wycinek rzeczywistości oraz zaplanować przyszłe funkcje systemu dla przewidywanych użytkowników systemu,
- d) ocena stanu ewentualnie istniejącej bazy danych,

- e) oszacowanie kosztów związanych z budową nowej bazy danych i nowego systemu zarządzania tą bazą.

Aby prawidłowo sformułować cel przedsięwzięcia należy określić przyszłych użytkowników projektowanej bazy danych. Użytkownikami bazy danych mogą być zarówno ludzie (projektanci, administratorzy, operatorzy) jak i programy (systemy komputerowe) lub urządzenia peryferyjne. Użytkowników bazy danych można pogrupować według przypisanych im ról, przy czym jedna osoba może mieć przypisanych wiele ról a do jednej roli można przydzielić wiele osób. Poszczególne role zwykle różnią się funkcjami wykonywanymi na bazie danych oraz przydzielonymi uprawnieniami. Szczególną rolą jest rola administratora bazy danych. Na przykład, dla bazy danych **Wydział** można określić rolę nauczyciela, studenta, pracownika dziekanatu, administratora.

W modelowanym OA należy wyodrębnić dwa aspekty:

- statyczny czyli dane,
- dynamiczny czyli operacje na danych,

oraz reguły funkcjonowania (reguły zachowania spójności tzw. więzy integralności).

W fazie analizy wyodrębniamy więc:

- obiekty (kategorie, encje) czyli rzeczy istotne (i nadajemy im nazwy, które są rzeczownikami),
- związki zachodzące pomiędzy encjami (i nadajemy im nazwy, które są czasownikami).
- reguły funkcjonowania i ograniczenia dziedzinowe.

W encjach wyróżniamy cechy dla nich charakterystyczne, czyli atrybuty encji.

3.1.1. ANALIZA WYCINKA RZECZYWISTOŚCI

Pierwszym etapem projektowania bazy danych jest szczegółowa **analiza wycinka rzeczywistości**, który chcemy modelować w bazie danych [Górski2000, Hernan1998]. W analizowanym wycinku rzeczywistości możemy wyróżnić rzeczywistość fizyczną (rzeczywiście istniejącą np. książka, bilet, faktura, towar, arkusz ocen) i rzeczywistość abstrakcyjną czyli konceptualną, pojęciową, istniejącą jedynie jako pojęcia w wyobraźni ludzkiej (np. wypożyczenie, lot, kupno).

W celu rzetelnego przeprowadzenia analizy wycinka rzeczywistości należy przeprowadzić wnikliwy wywiad z ekspertem dziedzinowym i określić założenia wstępne dotyczące przechowywanych danych.

Dobrze przeprowadzony wywiad pozwoli wyodrębnić:

- a) grupy użytkowników bazy danych (systemu bazy danych),
- b) oczekiwania przyszłych użytkowników bazy,
- c) reguły funkcjonowania i ograniczenia dziedzinowe,

- d) wymagania funkcyjne – należy uwzględnić operacje, jakie na danych użytkownicy będą chcieli wykonywać oraz w jaki sposób będą chcieli dane, i uzyskane informacje, prezentować,
- e) rzeczy istotne czyli obiekty, które trzeba uwzględnić w bazie danych,
- f) atrybuty obiektów i ich dziedziny, czyli zbiory wartości,
- g) powiązania między obiektami.

Pomyślność przeprowadzenia poprawnej analizy zależy zarówno od eksperta, który udziela odpowiedzi na zadawane pytania, od jego wiedzy, jak również od osoby, przeprowadzającej wywiad. Często tworzy się zespoły składające się ze specjalistów z danej dziedziny, kierownika działu, informatyków, przedstawiciela inwestora (sponsora przedsięwzięcia) itd. aby wszechstronnie przeanalizować zagadnienie i wyznaczyć jego zakres.

Dobrze jest zgromadzić wszystkie dostępne materiały i dokumenty związane z analizowaną rzeczywistością, np.:

- przepisy określające funkcjonowanie danego przedsiębiorstwa,
- druki i formularze papierowe używane dotychczas (np. faktury, zamówienia),
- postaci raportów i zestawień itp.

Należy określić również, czy postać dotychczasowych druków jest odpowiednia i ma być zachowana, czy raczej jest przestarzała i należy ją zmodyfikować.

W trakcie wywiadu należy przeanalizować zarówno sytuacje typowe (czynności zwykle wykonywane np. przy rejestrowaniu pacjenta czy wystawianiu faktury VAT) jak również sytuacje wyjątkowe (np. co dzieje się z kartą pacjenta po jego śmierci). Bardzo istotne są informacje na temat postępowania w przypadkach, gdy dane są niekompletne (czy dane niekompletne się przechowuje, odrzuca, uzupełnia itp.).

Projektant bazy danych powinien wiedzieć wszystko na temat pozyskiwania danych – skąd się je pozyskuje, w jaki sposób, jaką mają postać, jak się postępuje, gdy są w innej postaci lub są niekompletne albo nieczytelne, czy dopuszcza się brak danych itp.

Ze względu na złożoność świata rzeczywistego, w wielu projektach ogranicza się modelowaną rzeczywistość do pewnego wycinka, wprowadza się niejednokrotnie pewne uproszczenia i ograniczenia. Opracowując model świata rzeczywistego, czyli projektując bazę danych, musimy jednak uważać, by przyjęte uproszczenia nie zmieniły zasad funkcjonowania rzeczywistości lub nie ograniczyły jej w sposób niedozwolony.

Dane przechowywane w bazie powinny być:

- pełne,
- aktualne,
- trwałe, czyli istnieją przez określony, odpowiednio długi, przedział czasu,
- odpowiednie (jeśli interesuje nas wydobywanie węgla w Polsce a ktoś poda nam pełne i aktualne dane dotyczące budowy dróg, to nie są to dane odpowiednie).

W tym miejscu chcielibyśmy podkreślić różnicę między pojęciami *dana*, *informacja* i *wiedza*. Ujmując to krótko, **dana** (ang. *data*, *datum*) to para (nazwa, wartość), gdzie wartość to np. liczba całkowita, ciąg znaków, obrazek, i sama w sobie nie ma większego znaczenia, nabiera sensu dopiero zinterpretowana *dana* czyli **informacja**. Angielskie słowo *datum*, wywodzące się z języka łacińskiego, oznacza dosłownie fakt, przesłankę.

Dane przechowywane w bazie danych to np. (*LiczbaDzieci*, 5), (*Imię*, „Adam”), (*Nazwisko*, „Nowak”), (*Data*, 12/12/1970). Wartości danych nie ulegają samoistnym zmianom aż do zmodyfikowania ich ręcznego lub przez jakiś zautomatyzowany proces. Same w sobie są niezrozumiałe, można im nadać różne interpretacje. Nawet gdybyśmy wiedzieli, że wartość 12/12/1970 oznacza datę, to i tak dalej nie wiemy, co ta data oznacza, jak ją interpretować.

Informacje to dane zinterpretowane, przetworzone w sposób, który uwidacznia ich znaczenie, są dynamiczne w tym sensie, że podlegają ciągłym zmianom w stosunku do danych przechowywanych w bazie danych i można je przetwarzać na nieograniczoną liczbę sposobów. Informacją jest wyświetlenie na ekranie komputera komunikatu „Pracownik: Adam Nowak, data urodzenia: 12/12/1970”. Informacje można przedstawiać na różne sposoby (jako tekst komunikatu, arkusz kalkulacyjny, formularz, raport, na ekranie komputera lub na papierze, w formie tabeli lub na wykresie itp.).

Dane muszą zostać poddane przetworzeniu, aby stały się informacjami.

Baza danych, dzięki poprawnemu skonstruowaniu, powinna umożliwiać przetwarzanie danych i dostarczanie użytkownikom odpowiednich informacji.

Wiedza jest to zbiór reguł i warunków, które pozwalają podjąć decyzję na podstawie dostępnych informacji. Przechowywana w bazie danych wartość 12 nic nam nie mówi, można ją różnie zinterpretować (godzina, dzień, miesiąc, cena towaru, liczba studentów w laboratorium, numer domu itd.), *dana* (*Staż*, 12) stanie się informacją, jeśli będziemy wiedzieli, że oznacza ona staż pracy pracownika Jana Kowalskiego. Kierownik zakładu, korzystając z informacji dotyczących stażu pracy pracowników i przepisów firmy (np. że pracownik po przepracowaniu 12 lat otrzymuje nagrodę), może przyznać nagrodę pracownikowi. Znając więc przepisy firmy i staż pracy pracownika, wiemy, że dostał nagrodę. Wiedza w węższym znaczeniu, to ogół wiarygodnych informacji wraz z umiejętnością ich wykorzystania. Z wyników badań pacjenta lekarz otrzymuje informacje dotyczące jego zdrowia. Aby postawić diagnozę musi dysponować jednak wiedzą medyczną.

W każdej chwili baza danych znajduje się w pewnym stanie czyli jest konkretnym zbiorem danych. Należy zadbać o to, by baza danych znajdowała się w stanie spójnym i integralnym. Stan bazy nazwiemy integralnym, jeśli wartości zgromadzonych danych są zgodne z wartościami w świecie rzeczywistym (odzwierciedlają aktualny stan danych w rzeczywistości). Stan ten może się zmienić wskutek wykonania pewnych transakcji. Wszelkie zmiany w bazie zachodzą w sposób dyskretny. Spójność (ang. *consistency*) bazy danych oznacza poprawność poszczególnych

danych oraz poprawność bazy danych jako całości. Zapewnienie spójności bazy danych jest podstawowym problemem związanym z definiowaniem transakcji. Transakcja musi być zrealizowana w całości poprawnie albo wcale (musi być wówczas przywrócony stan sprzed rozpoczęcia transakcji).

Baza danych może być wykorzystywana w tym samym czasie nie tylko przez różnych użytkowników tej samej aplikacji ale również może być wykorzystywana przez różne systemy bazodanowe. Istnieje wówczas potrzeba współdzielenia danych przez kilka aplikacji (procesów). Mówimy wówczas o współdzielonych bazach danych. Współbieżny dostęp do danych współdzielonych może powodować ich niespójność. Zagadnienia te wykraczają jednak poza ramy niniejszego opracowania. W literaturze można znaleźć również informacje o bazach danych jeszcze innych typów np. dedukcyjnych, temporalnych, wielowersyjnych, aktywnych, rozproszonych. Zainteresowanych Czytelników zachęcamy do zapoznania się z tymi zagadnieniami we własnym zakresie.

Bazy danych możemy podzielić na bazy analityczne, operacyjne oraz na hurtownie danych. **Operacyjna** baza danych to baza, w której podstawowymi operacjami na danych są operacje dopisywania, usuwania, modyfikowania i wyszukiwania danych. **Analityczna** baza danych, to baza, której dane często pochodzą z bazy analitycznej ale po wprowadzeniu do tej bazy są stałe, nie podlegają modyfikacjom tylko analizie. Główne operacje wykonywane na tych danych to wyszukiwanie danych, sporządzanie zestawień statystycznych, przeprowadzanie analiz i prognoz. Baza analityczna (archiwalna) na ogół jest zupełnie inaczej zaprojektowana niż baza operacyjna. **Hurtownia danych** (ang. *data warehouse*), to zintegrowana, zorientowana tematycznie zmienna w czasie baza danych (analityczna, wykorzystująca dane z wielu operacyjnych baz danych) służąca analitykom do wspomagania podejmowania decyzji, np. dane dotyczące sprzedaży ratalnej, udzielania kredytów. Głównym problemem jest sposób transformacji danych do jednego formatu (przez wykorzystanie metabazy czyli słownika danych), bardzo duża ilość przechowywanych i wielowymiarowo analizowanych danych. Przetwarzanie danych wymaga ogromnych zasobów obliczeniowych.

W niniejszym opracowaniu skupimy się na projektowaniu operacyjnych relacyjnych baz danych.

Wyczerpująco i poprawnie przeprowadzona analiza jest punktem wyjścia do prawidłowego zaprojektowania bazy danych. Brak umiejętności w przeprowadzaniu wywiadu bądź brak kompetencji osoby dostarczającej wiedzy o analizowanej rzeczywistości jest często przyczyną złego projektu, którego wady często są zauważane dopiero w trakcie eksploatacji bazy. Usunięcie wad systemu bazodanowego w trakcie jego eksploatacji jest bardzo kosztowne a często wręcz (z wielu względów) niemożliwe.

3.1.2. SŁOWNIK POJĘĆ

Przeprowadzając wywiad i analizę wycinka rzeczywistości używamy pojęć związanych z danym wycinkiem. Pojęcia istotne, najczęściej występujące w opisie należy bardzo precyzyjnie zdefiniować, aby ich znaczenie było ściśle ustalone i jednoznaczne. Bardzo często terminy nawet powszechnie używane mogą budzić różne skojarzenia i osoby posługujące się nimi mogą zupełnie coś innego mieć na myśli. W trakcie wywiadu na temat funkcjonowania hotelu jeden z pracowników bardzo często używał słowa „gość”. Po dłuższej rozmowie okazało się, że dla niego „gość” nie oznacza „gość hotelowy” tylko „dowolny człowiek” (kierownik hotelu, recepcjonistka, klient hotelu, sprzątaczką itd.). Jeżeli zdefiniujemy, że gość to osoba przebywająca (zameldowana) w hotelu, to mówiąc, że chcemy przechowywać dane o gościach hotelu rzeczywiście mamy na myśli, że będą to dane tylko o osobach aktualnie zameldowanych w hotelu a po opuszczeniu hotelu przez daną osobę zaraz te dane chcemy usuwać z bazy danych, czy jednak chcemy te dane nadal przechowywać a więc gość to osoba, która jest lub była klientem hotelu. Ale czy rzeczywiście dane klientów hotelu nie będą w ogóle usuwane z bazy, czy tylko przez określony czas, np. 5 lat? Tak więc znowu musimy sprecyzować, np. że gość to osoba, która korzystała z usług hotelu co najmniej raz w przeciągu ostatnich 5 lat.

Poniżej podajemy przykład definicji pojęć związanych z działalnością biura podróży:

- Oferta** – pełny zestaw usług oferowanych przez biuro podróży. Oferty gromadzone są w katalogu ofert biura. Zawierają usługi noclegowe, transportowe, żywieniowe, ubezpieczeniowe, rozrywkowe. Określają miejsce i termin trwania usługi.
- Klient** – osoba korzystająca z ofert biura podróży, zawierająca umowę z biurem przy zakupie wybranej imprezy. Klient ma możliwość wyboru oferty z katalogu imprez oferowanych przez biuro oraz otrzymania na piśmie warunków uczestnictwa.
- Umowa** – dokument potwierdzający sprzedaż oferty klientowi przez biuro podróży, spisywany w dwóch jednakowo brzmiących egzemplarzach (po jednym egzemplarzu dla klienta i dla biura podróży).

3.1.3. ANALIZA ISTNIEJĄCEJ BAZY DANYCH

W tej fazie należy przeprowadzić analizę być może istniejącej już bazy danych, związanej z rozważanym obszarem rzeczywistości. Należy wówczas zapoznać się ze sposobem gromadzenia danych i ich wykorzystywania oraz ewentualnie, z funkcjonującym systemem informatycznym obsługującym dotychczasową bazę danych.

Należy przeanalizować zalety i wady istniejącej bazy i systemu informatycznego, uwagi pozytywne i krytyczne użytkowników. Rzetelna ocena dotychczasowej bazy i systemu komputerowego (jakości pracy, funkcjonalności, szybkości itp.) powinna pozwolić na właściwą ocenę, czy rzeczywiście istnieje potrzeba opracowania nowego projektu systemu informatycznego, a czy niezadowolenie użytkownika wynika, na przykład, z nieznajomości zasad obsługi posiadanego programu. Bezkrytyczne wzorowanie się na istniejącej analizie rzeczywistości, bazie danych, formularzach, może doprowadzić do niewłaściwego zaprojektowania bazy danych i powielenia poprzednich błędów. Istniejące bazy danych powinny raczej być wykorzystane do weryfikacji, czy nie zawierają elementów, które nie zostały uwzględnione w nowej bazie danych a są istotne.

3.1.4. ANALIZA WYMAGAŃ FUNKCJONALNYCH

Aby zaprojektowana baza danych umożliwiała wykonywanie określonych operacji należy je ustalić, wyszczególnić a następnie precyzyjnie opisać (rozdz. 3.5). W tym punkcie projektu należy wypisać i opisać funkcje, jakie dla zgromadzonych danych powinny być realizowane, np.

- wprowadzanie, modyfikowanie, usuwanie danych,
- wyszukiwanie danych,
- sporządzanie statystyk,
- przygotowywanie raportów.

3.1.5. ANALIZA WYMAGAŃ NIEFUNKCJONALNYCH

Na etapie analizy wymagań niefunkcjonalnych opisujemy wszystkie pozostałe aspekty związane z opracowywaną bazą, na przykład, czy będziemy używać narzędzi CASE (ang. *Computer-Aided Software Engineering* – Inżynieria oprogramowania wspomagana komputerowo) do projektowania bazy (np. EasyCASE Professional v.4.20), jaka notacja będzie wykorzystana do opracowania diagramów DFD (np. Yourdona) oraz diagramów ERD (np. Chena, Martina).

Dodatkowo, chociaż nie jest to związane z projektowaniem samej bazy danych, ale wiąże się z etapem tworzenia aplikacji, czyli tworzenia systemu bazy danych obsługującego zaprojektowaną bazę danych (rysunek 1.1), określa się możliwości i oczekiwania użytkownika przyszłego systemu dotyczące platformy sprzętowej, sposobu pracy z systemem (klawiatura, mysz), trybu pracy (tryb tekstowy, graficzny), czy program ma pracować w wersji sieciowej czy jednostanowiskowej itp. Szczególnie powinno określić się:

- platformę sprzętową (np. komputer Pentium 3, drukarka),
- platformę systemową (np. Windows 2000),

- środowisko implementacyjne bazy danych (np. Access 2000),
- środowisko programistyczne (np. Borland C++Builder Professional v.4.0),
- rodzaj bazy danych (np. relacyjna),
- oszacowanie liczby danych wejściowych, tempo przyrostu danych,
- sposób archiwizowania danych.

Mimo, że czasami opracowywany system bazy danych jest przeznaczony tylko do jednostanowiskowej pracy (np. na danym komputerze instalowany jest tylko jeden system finansowo-księgowy F/K do pełnej obsługi małej firmy), to zdecydowana większość aplikacji powinna współpracować z istniejącym już systemem i z innymi programami. Dokładna analiza środowiska jest wówczas niezbędną czynnością, aby nowy produkt mógł dobrze działać i współpracować z pozostałym oprogramowaniem.

3.1.6. ANALIZA KOSZTÓW

Zaprojektowanie i implementacja bazy danych oraz systemu obsługującego tę bazę pociągnie za sobą pewne koszty. Pojawia się zapewne również koszty związane z wdrożeniem i eksploatacją oprogramowania, szkoleniem pracowników. Być może konieczne będzie poniesienie kosztów związanych z zakupem lub modernizacją sprzętu komputerowego oraz miejsca pracy użytkownika systemu. Należy więc przeanalizować dotychczasowe koszty związane z wykonywaniem prac (np. liczba pracowników, ich wynagrodzenie itp.) ale również ich skuteczność, szybkość i wydajność pracy, poprawność osiąganych wyników itd. oraz wszelkie koszty jakie będą związane z wykonaniem i eksploatacją nowego systemu bazodanowego oraz korzyści, jakie przyniesie wykorzystywanie nowego systemu.

Po bardzo precyzyjnej analizie wycinka rzeczywistości, która powinna być spisana w języku naturalnym, z uwzględnieniem wszelkich szczegółów rozpatrywanego wycinka i przedstawiona przyszłemu użytkownikowi do zatwierdzenia, mając określone założenia i wymagania użytkownika można przejść do etapu drugiego czyli do określenia kategorii.

3.2. DEFINICJE KATEGORII

Ze szczegółowej analizy rzeczywistości, na podstawie słownika pojęć, należy wyodrębnić rzeczy istotne, o których chcemy przechowywać informacje, np. towar, uczeń, kasetę, lek, faktura, projekt itp., zwane kategoriami (obiektami, pakietami danych). Po wyodrębnieniu kategorii i nazwaniu ich rzeczownikami w liczbie pojedynczej, ustalamy ich atrybuty, czyli ich cechy charakterystyczne.

Następnie sporządzamy listę kategorii opisując każdą następująco:

KAT/x Nazwa kategorii

Opis: Słowny opis kategorii

Atrybuty: Lista atrybutów i ich opisy

gdzie *x* jest kolejnym numerem kategorii.

Należy zwrócić szczególną uwagę na opis kategorii, z którego powinny wynikać powiązania z innymi kategoriami, np. jeśli w bazie mają być gromadzone dane o dostawcach i dostarczanych przez nich towarach, to kategoria **Towar** nie powinna mieć atrybutu *Dostawca* (**Dostawca** powinien być odrębną kategorią), a w opisie powinna być informacja o związku towaru z dostawcą.

Przykład 3.1 (opisy kategorii)

KAT/001 Towar

Opis: Kategoria **Towar** służy do opisu towaru przechowywanego w magazynie, który może być dostarczany przez różnych dostawców.

Atrybuty:

- | | |
|------------------------|--|
| <i>Nazwa</i> | – nazwa grupy towarów np. cukier biały, |
| <i>Symbol towaru</i> | – unikatowy symbol towaru, np. cuk001/99, |
| <i>Jednostka miary</i> | – jednostka miary towaru np. kg, szt., metr, |
| <i>Uwagi</i> | – dowolne uwagi na temat danej grupy towarów, np. towar wprowadzony do sprzedaży w 2000 r. |

KAT/002 Dostawca

Opis: Kategoria **Dostawca** przechowuje dane dostawców dostarczających towary w ciągu ostatnich 5 lat lub planujących dostarczać towary.

Atrybuty:

- | | |
|-------------------------|---|
| <i>NazwaSkrót</i> | – skrócona nazwa dostawcy np. MEGA-HIT, |
| <i>Nazwa</i> | – pełna nazwa dostawcy np.
Hurtownia artykułów spożywczych MEGA-HIT, |
| <i>Adres</i> | – adres dostawcy (zarządu firmy) np.
11-123 Miasteczko, ul. Nowa 2, |
| <i>Adresy Oddziałów</i> | – adresy ewentualnych oddziałów firmy, |
| <i>Telefony</i> | – numery telefonów kontaktowych firmy np. do dyrektora,
sekretariatu, oddziałów, |
| <i>NIP</i> | – Numer Identyfikacji Podatkowej (NIP) firmy, |
| <i>REGON</i> | – numer REGON firmy. |

□

3.3. REGUŁY FUNKCJONOWANIA

Na podstawie sporządzonej wcześniej analizy rzeczywistości należy wyodrębnić **reguły funkcjonowania** (reguły biznesowe) i **ograniczenia dziedzinowe**.

Prawidłowe i wyczerpujące określenie reguł funkcjonowania, a następnie ich przestrzeganie we wszystkich kolejnych etapach, ułatwi modelowanie rzeczywistości, poprawne zaprojektowanie bazy danych i bazodanowego systemu informatycznego.

Reguły należy wypisywać w sposób systematyczny na podstawie dokonanej analizy fragmentu rzeczywistości, oznaczając je np.

REG/ x *sformułowanie reguły funkcjonowania w języku naturalnym*
gdzie x jest kolejnym numerem reguły.

Przykład 3.2 (reguły funkcjonowania)

REG/001	Dane towarów wpisuje kierownik magazynu.
REG/002	Każdy towar ma unikalny symbol nadany przez kierownika magazynu.
REG/003	Towar jest przyjmowany do magazynu na podstawie faktury zakupu.
REG/004	Faktury są przechowywane przez 5 lat.
REG/005	Przy sprzedaży towaru sprzedawca, za zgodą kierownika, może udzielić rabatu w określonej wysokości 3%, 5% lub 8% ceny netto towaru.
REG/006	Do wystawienia faktury niezbędny jest NIP klienta.
REG/007	Towar może być dostarczany przez różnych dostawców.

□

3.4. OGRANICZENIA DZIEDZINOWE

Ograniczenia dziedzinowe są nakładane na wcześniej określone kategorie i ich atrybuty. Wyszczególnione ograniczenia należy uwzględnić w dalszych etapach projektowania bazy danych i tworzonej implementacji. Wszystkie ograniczenia dziedzinowe powinny wynikać z analizy wycinka rzeczywistości (rozdz. 3.1.1).

Pożądane jest wypisywanie ograniczeń w sposób systematyczny, np. dla poszczególnych kategorii.

Ograniczenia dziedzinowe można wypisywać numerując je kolejno, np.

OGR/00 x *sformułowanie ograniczenia w języku naturalnym*
gdzie x jest kolejnym numerem ograniczenia.

Przykład 3.3 (ograniczenia dziedziczne)

OGR/001	Data urodzenia pracownika firmy jest wcześniejsza niż jego data zatrudnienia.
OGR/002	Data zatrudnienia pracownika jest wcześniejsza niż data zwolnienia.
OGR/003	Kod pocztowy w adresie ma postać 99-999, gdzie 9 oznacza dowolną cyfrę.
OGR/004	Numer legitymacji pracownika jest łańcuchem 10 znakowym postaci: Numer pracownika/Rok wystawienia
OGR/005	Liczba pracowników w oddziale firmy jest liczbą dodatnią mniejszą od 25.

□

3.5. TRANSAKCJE

Transakcje są to operacje wykonywane na danych. Wynikają one z przeprowadzonej analizy i są opisywane przy pomocy pojęć języka naturalnego używanych w opisie rzeczywistości.

Transakcje powinny mieć cztery podstawowe własności:

- **niepodzielność, atomowość** (*ang. atomicity*) – transakcja jest wykonywana w całości albo wcale. Po przerwaniu transakcji musi być odtworzony stan bazy danych sprzed rozpoczęcia transakcji,
- **spójność** (*ang. consistency*) – transakcje zachowują spójność bazy danych, chociaż w trakcie działania transakcji stan bazy może być chwilowo niespójny,
- **izolacja** (*ang. isolation*) – transakcje są całkowicie od siebie niezależne, jedna transakcja od drugiej jest odizolowana,
- **trwałość** (*ang. durability*) – zmiany dokonane przez pomyślnie zakończoną transakcję są zachowywane na trwałe.

Te cztery istotne własności są często nazywane w skrócie *własnościami ACID* (*Atomicity, Consistency, Isolation, Durability*).

W przypadku zerwania transakcji (*ang. abort*) jest przywracany stan bazy sprzed rozpoczęcia transakcji na podstawie dziennika transakcji (plik o nazwie *log*), w którym są zapisywane poszczególne stany bazy danych przed i po wykonaniu każdej transakcji. Dziennik taki jest czytany od ostatniego zapisu do początku i przywracany jest stan bazy (*ang. rollback*).

Etap wyodrębniania transakcji jest w zasadzie uszczegółowieniem etapu analizy wymagań funkcjonalnych (rozdz. 3.1.4). Opisując poszczególne transakcje należy określić, które kategorie biorą udział w transakcji, co jest źródłem danych wejściowych oraz jakie informacje otrzymywane są na wyjściu.

W określeniu, skąd pochodzą dane wejściowe i gdzie kierowane są wyniki transakcji, można użyć oznaczeń: U – użytkownik, BD – baza danych.

Przez użytkownika rozumiemy tutaj zarówno osobę korzystającą z bazy danych jak i system obsługujący bazę danych.

W etapie tym należy wymienić wszystkie przewidywane transakcje opisując je np. według schematu:

TRA/x Nazwa transakcji

Opis: Słowny opis transakcji

Uwarunkowania: Specyfikacja warunków dla prawidłowego wykonania transakcji oraz opis ścieżki alternatywnej, do której może dojść, jeśli warunki te nie będą spełnione.

Wejście/Wyjście: Opis wejścia i wyjścia transakcji lub tabela transakcji gdzie *x* jest kolejnym numerem transakcji.

Przykład 3.4 (opis transakcji)

TRA/001 Edycja danych dotyczących towaru

Opis: Zadaniem transakcji jest wyszukanie danych o wybranym towarze i edycja tych danych np. ceny sprzedaży. Edycję może wykonać tylko kierownik magazynu.

Uwarunkowania: Dane towaru muszą istnieć w bazie. Jeśli dane towaru nie zostaną znalezione, zostanie wyświetlony komunikat *Brak danych*. Wyszukanie danych towaru może być wykonane po podaniu jego nazwy lub symbolu towaru. Po wykonaniu edycji danych towaru, użytkownik otrzymuje komunikat *Edycja zakończona*.

Wejście:

- U – Nazwa towaru lub symbol towaru, nowe dane towaru
- BD – Dane towarów

Wyjście:

- U – Komunikat
- BD – Dane towarów

Zamiast podanego wyżej sposobu zdefiniowania wejścia i wyjścia transakcji można sporządzić *tabelę transakcji* (tabela 3.1):

Tabela 3.1

Wejście/Wyjście TRA/001

	WEJŚCIE	WYJŚCIE
Użytkownik	Nazwa lub symbol towaru Nowe dane towaru	Komunikat
Baza danych	Dane towarów	Dane towarów



3.6. MODEL KONCEPTUALNY

Opracowanie modelu konceptualnego (konceptyjnego, pojęciowego, ang. *Conceptual Data Model* – CDM) polega na wyodrębnieniu i zdefiniowaniu encji oraz związków między nimi. Model danych jest pewnym uproszczonym, lecz zgodnym z modelowanym wycinkiem rzeczywistości, opisem. Celem modelowania konceptualnego jest zapewnienia wyższego poziomu abstrakcji widzenia danych z perspektywy użytkownika oraz niezależności danych od implementacji (aplikacji).

Aby zapewnić poprawność modelu danych w sensie zgodności, często podaje się *metamodel danych* czyli model modelu. Metamodel obejmuje definicje pojęć (np. encji, atrybutu) i reguły dotyczące pojęć wykorzystywanych w modelu, np. jeśli model zawiera co najmniej dwie encje, to każda encja musi być co najmniej w jednym związku z dowolną inną encją.

Po wyodrębnieniu encji i związków, przyjmując ustaloną notację (np. Chena [Barker1996, MurRyb1993], Martina [Beynon2000] lub Yourdona [Yourdon1996]) można przedstawić graficznie opracowany model, czyli zbudować diagram związków encji (diagram obiektowo-związkowy).

Diagram związków encji (ang. *Entity Relationship Diagram* – ERD) – pozwala w sposób graficzny przedstawić dane i sposób widzenia ich struktury oraz związki między danymi [Barker1996, Date2000, Rodgers1995]. Aby prawidłowo opracować ERD należy najpierw zdefiniować encje oraz związki pomiędzy nimi a następnie przedstawić je w sposób graficzny używając określonej notacji.

Prawidłowo przeprowadzona analiza rzeczywistości pozwala na poprawne wyodrębnienie i zdefiniowanie encji i związków, co z kolei prowadzi do zdefiniowania znormalizowanych schematów relacji bazy danych (rozdz. 3.6.5). Znormalizowana baza danych zajmuje mniej miejsca, dane się nie powtarzają, łatwiejsze jest dopisywanie, modyfikowanie i usuwanie danych. Dzięki normalizacji baza danych jest lepsza w tym sensie, że nie występują w niej anomalie, czyli nieprawidłowości związane z uaktualnianiem (ang. *update anomalies*), usuwaniem i dopisywaniem danych.

Poniżej podamy definicje podstawowych pojęć wykorzystywanych w tej fazie projektowania bazy danych.

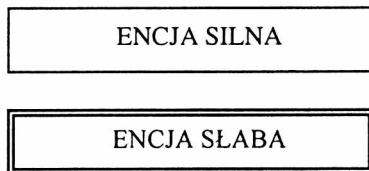
3.6.1. IDENTYFIKACJA ENCJI

W tym etapie należy podać definicje **encji** – w pojęciach naturalnych wynikających z analizy rzeczywistości (opis słowny, atrybuty, klucze kandydujące, klucze główne). Opisywane encje są wyprowadzane z wyodrębnionych wcześniej kategorii (rozdz. 3.2).

Encja – (ang. *entity*) w języku łacińskim *esse* znaczy *być* [Boratyn1995]. Encja jest to pewien obiekt, rzecz, zjawisko, zdarzenie, pewien byt. Encje posiadają atrybuty, czyli cechy je opisujące (charakteryzujące), istotne z punktu widzenia eksperta dziedzinowego, przyjmujące wartości z ustalonego zbioru – **dziedziny**. Encjom nadajemy nazwy, które powinny być rzeczownikami w liczbie pojedynczej. Nazwy encji piszemy wielkimi literami. W diagramie ER encję przedstawiamy w postaci prostokąta rysowanego linią pojedynczą (encja silna) lub podwójną (encja słaba), co ilustruje rysunek 3.1. W niektórych notacjach na diagramie ER wyszczególniane są również atrybuty encji.

Encja silna (regularna, właściwa) – oznacza obiekt, o którym informacja musi być znana i przechowywana. Istnieje niezależnie od innych obiektów. Reprezentowana jest przez prostokąt narysowany linią pojedynczą, ma nazwę, która jest rzeczownikiem w liczbie pojedynczej.

Encja słaba – (ang. *weak entity*) oznacza obiekt, który może istnieć dopiero po zdefiniowaniu innych obiektów, np. dane dzieci pracownika można wprowadzać dopiero wtedy, gdy istnieją dane odpowiedniego pracownika. W tym wypadku encja DZIECKO byłaby encją słabą a encja PRACOWNIK encją silną. Encja słaba reprezentowana jest przez prostokąt narysowany linią podwójną, gdyż na ogół na nią trzeba zwrócić szczególną uwagę. Nazwa encji jest rzeczownikiem w liczbie pojedynczej.



Rys. 3.1. Oznaczanie encji silnych i słabych w ERD

Atrybut – (ang. *attribute*) cecha charakterystyczna dla encji lub związku, szczegół służący do opisu, określania ilości lub wyrażenia stanu encji lub związku; dowolny opis mający znaczenie dla encji lub związku. Atrybut określa dokładnie jedną wartość, która z punktu widzenia projektanta jest niepodzielna (atomowa). Nazwa atrybutu powinna być rzeczownikiem w liczbie pojedynczej, jeśli występuje w liczbie mnogiej to jest to sygnał, że powinniśmy utworzyć nową encję, np. *Imiona dzieci*, *Znane języki obce*.

Atrybut opcjonalny (nieobowiązkowy) – (ang. *optional attribute*) atrybut, którego wartość może być nieokreślona, pusta, nieznana (oznaczana jako NULL).

Atrybut obligatoryjny (wymagany) – atrybut, którego wartość musi być zawsze określona. Nie może przyjmować wartości NULL.

Wartość NULL – wartość pusta interpretowana jako wartość nieznana, pusta, nieokreślona, niepewna, zastrzeżona. Dwa atrybuty o wartości NULL nie są traktowane jako sobie równe w tym sensie, że taka wartość może być różnie interpretowana, np. wartość NULL jako wartość atrybutu Telefon może oznaczać brak telefonu lub nieznaną numer telefonu.

Uwaga: Date w pracy [Date2000], rozdz. 5. dokładnie omawia zagadnienia związane z wprowadzeniem pojęcia NULL do baz danych. Generalnie C.J. Date uważa, że problem jest nie w pełni zrozumiany i nierozwiązany a w związku z tym, pojęcie NULL nie powinno być wprowadzone do modelu relacyjnego ([Date2000], str.152). Stwierdza jednak, że E. Codd ma na ten temat zupełnie przeciwne zdanie.

Encje o takich samych cechach charakterystycznych możemy pogrupować w typy encji. Zwróćmy uwagę, że encja to nie to samo co typ encji. **Typ encji** jest kategorią a encja jest instancją (wystąpieniem) danego typu encji [Beynon2000]. Typem encji jest np. OSOBA o atrybutach {*Nazwisko*, *Imię*}, a encją jest jeden obiekt danego typu np. („Jan”, „Kowalski”). Powszechnie używa się terminu encja zarówno na określenie typu encji jak i samej encji. Tam, gdzie nie prowadzi to do nieporozumień, będziemy używać pojęcia encja zamiast typ encji.

Wyodrębnione typy encji należy wypisać porządkując je np. alfabetycznie według nazw lub w inny uporządkowany logicznie sposób.

Definiując atrybuty typu encji powinno się określić nazwy dla atrybutów, podać opis (znaczenie) każdego atrybutu, jego typ (format, dziedzina), przykładowe wartości lub zakresy wartości, określić, czy jest jednoznacznym identyfikatorem lub wchodzi w skład jednoznacznego identyfikatora, czy jest atrybutem obligatoryjnym (wymaganym) czy opcjonalnym.

Opis typów encji można przedstawić w następujący sposób:

ENC/x NAZWA TYPU ENCJI

Semantyka encji – opis encji danego typu

Wykaz atrybutów – lista atrybutów, ich opis i dziedzina (typ). Listę atrybutów można przedstawić w formie *tabeli atrybutów* (tabela 3.2).

Klucze kandydujące – zbiory atrybutów jednoznacznie identyfikujących encje.

Klucz główny – jeden z kluczy kandydujących wybrany jako główny.

Charakter encji – encja silna lub słaba,

gdzie *x* jest kolejnym numerem typu encji.

Precyzyjny opis typów encji, wszystkich atrybutów i ich dziedzin ułatwi zdefiniowanie modelu logicznego (struktur tabel).

Bardzo często na tym etapie wprowadza się **sztuczne klucze**, takie jak identyfikator encji (symbol, numer), jeśli klucze kandydujące są złożone z wielu atrybutów lub z jednego atrybutu, ale o dużej szerokości pola, lub brak jest klucza kandydującego. Również dane osobowe i dane chronione nie powinny być kluczami

głównymi np. NIP, PESEL, numer konta w banku itp. (szczególnie, gdy nie ma całkowitej pewności co do unikalności ich wartości).

Przykład 3.5 (opisy typów encji)

ENC/001 OSOBA

Semantyka encji – encja zawiera dane dotyczące pracownika szkoły (nauczycieli, pracowników administracyjnych i technicznych, obsługi szkoły).

Wykaz atrybutów:

Tabela 3.2

Wykaz atrybutów encji typu OSOBA

Nazwa atrybutu	Opis atrybutu	Typ	OBL (+) OPC (-)
<i>NrOsoby</i>	Identyfikator osoby	Liczba naturalna	+
<i>Nazwisko</i>	Nazwisko osoby	max. 25 znaków	+
<i>Imię</i>	Imię osoby	max. 15 znaków	+
<i>DataUrodzenia</i>	Data urodzenia osoby	Data	+
<i>NIP</i>	Numer Identyfikacji Podatkowej	Znakowy postaci {999-999-99-99}	+
<i>PESEL</i>	Numer identyfikacyjny PESEL	Ciąg 11 cyfr	+
<i>Wykształcenie</i>	Wykształcenie osoby	{podstawowe, średnie, wyższe}	+

Klucze kandydujące: *NrOsoby*, *NIP*

Klucz główny: *NrOsoby*

Charakter encji: encja silna

ENC/002 PRZEDMIOT

Semantyka encji – encja zawiera dane o przedmiocie nauczonym w szkole

Wykaz atrybutów:

Tabela 3.3

Wykaz atrybutów encji typu PRZEDMIOT

Nazwa atrybutu	Opis atrybutu	Typ	OBL (+) OPC (-)
<i>NrPrzedmiotu</i>	Numer porządkowy	Liczba dwucyfrowa całkowita dodatnia	+
<i>NazwaPrzedmiotu</i>	Nazwa przedmiotu nauczanego w szkole	max. 30 znaków	+

Klucze kandydujące: *NrPrzedmiotu*

Klucz główny: *NrPrzedmiotu*

Charakter encji: encja silna

Aby stwierdzić, czy zdefiniowane encje są poprawne, trzeba odnieść się do analizy rzeczywistości. Gdybyśmy chcieli zdefiniować typ encji KSIĄŻKA, to z analizy rzeczywistości powinno wynikać, czy *wydawnictwo* powinno być postrzegane jako typ encji WYDAWNICTWO, czy raczej jako atrybut *Wydawnictwo* w typie encji KSIĄŻKA (albo jako związek między encją KSIĄŻKA i AUTOR). Definiując *wydawnictwo* jako atrybut, należy podać dziedzinę, czyli zbiór wartości, jakie ten atrybut może przyjmować. W tym wypadku możemy zdefiniować dziedzinę np. jako zbiór łańcuchów co najwyżej 30 znakowych (nazwy wydawnictw) lub jako zbiór symboli (np. WNT, PWN) przypisanych poszczególnym wydawnictwom. Definiując WYDAWNICTWO jako encję można podać np. nazwę wydawnictwa, adres, telefon, e-mail, nazwisko i imię dyrektora, rok założenia wydawnictwa itp.

Atrybut encji powinien opisywać cechy charakterystyczne encji a nie jej związek z innymi encjami. Na przykład, jeśli została wyodrębniona encja DOSTAWCA, to encja TOWAR nie powinna mieć atrybutu *IDDostawcy*, wskazującego na związek z encją DOSTAWCA, gdyż jest to cecha encji DOSTAWCA.

Po zdefiniowaniu encji należy sprawdzić, czy każdy atrybut ma tylko jedną wartość w danej chwili i, czy nie istnieje potrzeba przechowywania kolejnych zmieniających się wartości atrybutu (np. stanowisko pracownika).

W celu zweryfikowania, czy w projektowanej bazie danych nie będzie zbędnych (nigdy nie wykorzystywanych) danych należy sprawdzić, czy każda encja będzie uczestniczyła w co najmniej jednej operacji bazodanowej.

3.6.2. ZWIĄZKI I TYPY ZWIĄZKÓW

Związek (ang. *relationship*) jest to powiązanie (ang. *association*) między encjami (najczęściej między dwiema). Formalnie n -argumentowy związek Z można zapisać jako relację o n argumentach (encjach $E_1, \dots, E_n$):

$$Z(E_1, \dots, E_n)$$

co oznacza, że encje $E_1, \dots, E_n$ uczestniczą w związku Z .

Graficznie związki reprezentujemy jako romby zawierające nazwę związku i połączone krawędziami z encjami uczestniczącymi w związku. Związek binarny (między dwiema encjami) ma dwie krawędzie, związek ternarny zachodzi pomiędzy trzema encjami. Związek może zachodzić również między encjami tego samego typu. Związek musi mieć unikatową nazwę (niektóre notacje np. wprowadzają odrębne nazwy dla każdego końca, reprezentują związki jako linie). Nazwami związków powinny być czasowniki w 3. osobie liczby pojedynczej.

W zasadzie, podobnie jak dla encji, należałoby rozróżniać typ związku od instancji związku. Powszechnie jednak to rozróżnienie jest ignorowane. Na przykład instancja związku binarnego $Z(E_1, E_2)$ jest dwuargumentową relacją na iloczynie kartezjańskim zbiorów instancji encji E_1 i E_2 .

Opisując związki używamy pojęć takich jak **liczebność**, **liczność** oraz **typ uczestnictwa**.

Liczebność lub **stopień związku** (ang. *degree*) – określa liczbę encji uczestniczących w związku, np. liczebność związku binarnego wynosi 2.

Liczność (ang. *cardinality*) związku określamy dla każdego końca związku jako **jeden** lub **wiele**, przy czym jeden należy rozumieć jako jeden i tylko jeden a licznosc wiele należy rozumieć jako jeden lub więcej.

Ze względu na licznosc wyróżniamy następujące typy związków binarnych (stopnia drugiego) $Z(E_1, E_2)$:

1:1 – **jeden do jeden**,

jedna encja typu E_1 może być powiązana tylko z jedną encją typu E_2 i jedna encja typu E_2 może być powiązana tylko z jedną encją typu E_1 , np.

jeden student ma tylko jeden indeks,

jeden indeks należy tylko do jednego studenta

1:N – **jeden do wiele**,

jedna encja typu E_1 może być powiązana z wieloma encjami typu E_2 i jedna encja typu E_2 może być powiązana tylko z jedną encją typu E_1 , np.

jeden pokój jest zamieszkiwany przez wielu studentów,

jeden student mieszka tylko w jednym pokoju.

N:N – **wiele do wiele**,

jedna encja typu E_1 może być powiązana z wieloma encjami typu E_2 i jedna encji typu E_2 może być powiązana z wieloma encjami typu E_1 , np.

student może się zapisać na wiele wykładów,

na wykład może się zapisać wielu studentów.

Uczestnictwo encji w związku określane jako **opcjonalne** (ang. *optional*) lub **obligatoryjne** (ang. *obligatory*) definiuje udział encji w związku. Uczestnictwo **obligatoryjne** (np. w notacji Chena wprowadzonej w 1976 r. – symbol OBL obok specyfikowanego typu encji) oznacza, że wszystkie encje tego typu biorą udział w związku. Natomiast uczestnictwo **opcjonalne** (np. w notacji Chena symbol OPC obok typu encji) oznacza, że nie każda encja musi uczestniczyć w związku czyli, że mogą istnieć encje danego typu nie uczestniczące w związku.

Przykład 3.6

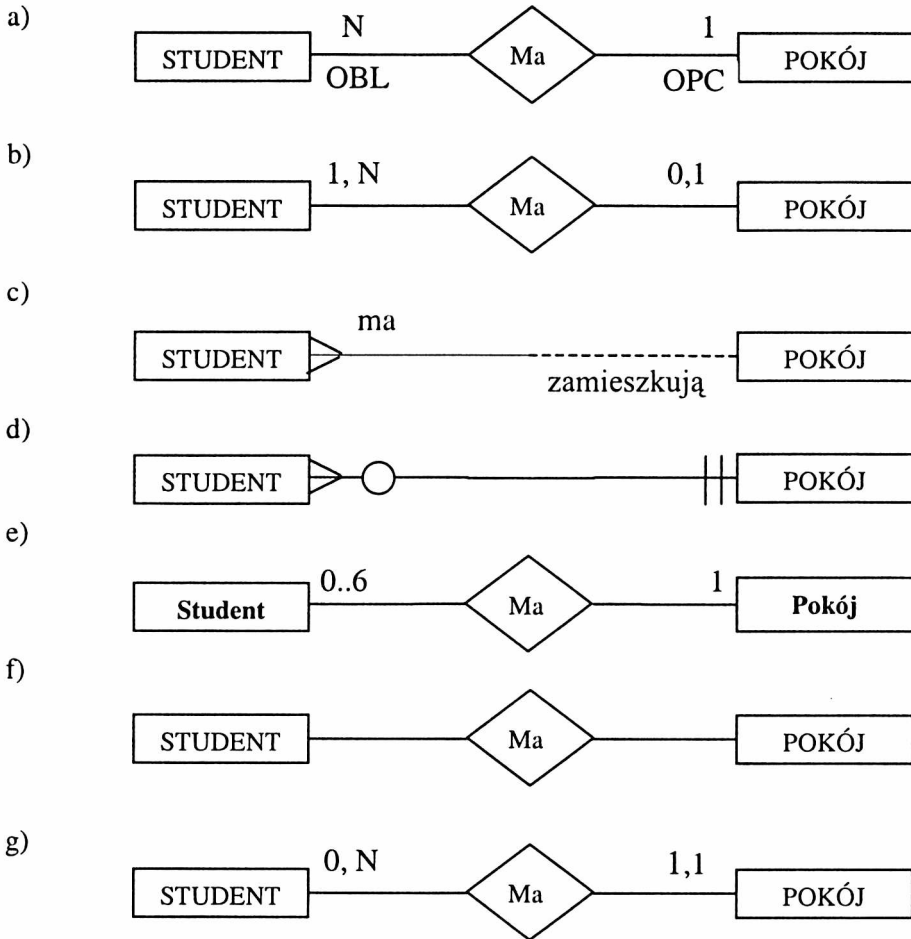
Założmy, że w bazie danych **AKADEMIK** obowiązują następujące reguły:

REG/001 Każdy student musi mieć przypisany dokładnie jeden numer pokoju.

REG/002 Pokój może nie być przypisany do żadnego studenta.

REG/003 W pokoju może zamieszkiwać wielu studentów.

Graficznie reguły te w różnych notacjach przedstawia rysunek 3.2.



Rys. 3.2. Różne sposoby graficznego przedstawienia liczności i uczestnictwa

- a) notacja Chena (Date, Pankowski)
- b) notacja zmodyfikowana Chena
- c) notacja wg Barkera
- d) notacja Martina
- e) notacja UML
- f) notacja Yourdona (bez liczebności i uczestnictwa)
- g) notacja stosowana w niniejszej książce

Przeanalizujmy podane reguły:

REG/001 Każdy student musi mieć przypisany dokładnie jeden numer pokoju, tzn. każdy student bierze udział w związku czyli encja STUDENT uczestniczy obligatoryjnie w związku **Ma** a liczność encji POKÓJ jest 1.

REG/002 Pokój może nie być przypisany do żadnego studenta, tzn. może być pokój, który nie uczestniczy w związku czyli encja POKÓJ uczestniczy opcjonalnie w związku **Ma**.

REG/003 W pokoju może zamieszkiwać wielu studentów, czyli liczność encji STUDENT wynosi N.

Encje STUDENT i POKÓJ oraz związek **Ma** wynikający z podanych reguł przedstawiono na rysunku 3.2.

Sposób zaznaczania i interpretowania liczności i uczestnictwa związków (oraz samych związków) na diagramach nie jest jednoznaczny. Różni autorzy w sposób odmienny definiują te pojęcia. Na przykład:

- a) Date wzorując się na pracy Chena ([Date2000] str. 398–400, [Chen1976]) definiuje uczestnictwo encji E jako obligatoryjne (OBL po stronie encji E, wymagane – ang. *total*) w związku R, jeśli każda instancja encji E bierze udział w co najmniej jednej instancji związku R; w przeciwnym razie jest ono opcjonalne (skrót OPC po stronie encji E, ang. *partial*). Notację Chena stosuje również Pankowski [Pankow1992] (rysunek 3.2a).
- b) Notacja wyprowadzona od Chena przez zastąpienie skrótu OPC oznaczającego opcjonalność przez symbol 0 oraz skrót OBL (obligatoryjność) przez symbol 1 (rysunek 3.2b).
- c) Według Barkera [Barker 1996] związek jest obligatoryjny po stronie encji A, jeśli każda instancja encji A może istnieć tylko w ramach powiązania z encją B (z drugiej strony związku). Jeśli encje A mogą istnieć niezależnie od B, to związek jest opcjonalny (rysunek 3.2c).
- d) Hernandez ([Hernan1998], str. 46, 62), który wykorzystuje metodykę Martina, definiuje uczestnictwo encji w związku jako obowiązkowe (symbol „|” przy encji), jeśli encja musi wystąpić, zanim określi się dane encji po drugiej stronie związku (uczestnictwo opcjonalne zaznacza się symbolem „O” (rysunek 3.2d).
- e) W **UML** (*Unified Modelling Language*) liczność nazywa się krotnością ([Muller2000] str. 184, [Subieta1998] str. 231) i na diagramach opisuje się ją przez podanie zbiorów wartości, np.

0..*	zero lub więcej (opcjonalność, wiele)
0..1	zero lub jeden (opcjonalność, jeden)
1..*	co najmniej jeden (obligatoryjność, wiele)
*	zero lub więcej (opcjonalność, wiele)
2..6	co najmniej 2, co najwyżej 6
1, 5-7	jeden, pięć, sześć lub siedem
brak wartości	1..1 (obligatoryjność, jeden)

(rysunek 3.2e).

- f) Edward Yourdon ([Yourdon1996], rozdz. 12) uważa, że na diagramach związków-encji (E-R) nie powinno umieszczać się żadnych dodatkowych informacji

takich jak liczność, uczestnictwo czy kierunek związku (niektórzy zaznaczają na diagramach dodatkowo kierunek związku) gdyż takie informacje tylko „odwracają uwagę od podstawowego przeznaczenia diagramu E-R, jakim jest przegląd składników i interfejsów między elementami danych w systemie, a można je łatwo opisać w słowniku danych”. Diagram E-R w notacji Yourdona przedstawia rysunek 3.2f.

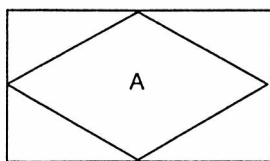
- g) Notacja oparta na notacji Chena, wykorzystywana w przykładach zamieszczonych w niniejszej książce (rysunek 3.2g), która może być interpretowana jako podanie minimalnej i maksymalnej liczności dla każdego końca związku. Według tej interpretacji związek **Ma** z diagramu 3.2g odczytujemy, że student **musi** mieć przypisany co najmniej **jeden** pokój (obligatoryjność występowania encji STUDENT w związku **Ma**), natomiast pokój **może** mieć przypisanych od **0** (opcjonalność występowania encji POKÓJ w związku **Ma**) do **N** studentów.

W pracy [Olle1982] jest podanych 13 różnych sposobów przedstawiania encji i związków na diagramach ERD (oprócz wyżej podanych są jeszcze np. notacje Bachmana, deMarco, Jacksona).

Informacje zawarte na diagramach rysunku 3.2 można przeczytać następująco: każdy student **musi** mieć przydzielony **dokładnie jeden** numer pokoju, natomiast pokój **może** zamieszkiwać **od 0 do n** studentów, czyli pokój może być zamieszkanym przez wielu studentów lub nie zamieszkanym wcale.

□

Uwaga. Aby uprościć transformację modelu konceptualnego do modelu logicznego (rozdz. 4.1), zaleca się zastępowanie związków N:N dwoma związkami 1:N, w wyniku czego pojawia się encja powiązania (asocjacyjna), oznaczana czasami tak jak na rysunku 3.3.



Rys. 3.3. Graficzne przedstawienie encji asocjacyjnej A

W przypadku, gdy związek N:N nie ma dodatkowych atrybutów, można pozostawić go bez zmian i nie zastępować go encją asocjacyjną.

W niniejszej książce encję asocjacyjną będziemy oznaczać za pomocą symbolu prostokąta, tak jak inne encje.

Teoretycznie można wyrysować na kartce wszystkie możliwe kombinacje typów związków biorąc pod uwagę ich liczebność i uczestnictwo. W praktyce niektórych nie spotyka się wcale, niektóre rzadko a niektóre bardzo często (rozdz. 4.1.1).

Nie zawsze zakwalifikowanie danej (jako encje, atrybut czy związek) jest rzeczą jednoznaczną. Niektóre dane można uważać za encję, atrybut lub związek. Rozważmy na przykład dane związane z rozwodem:

- rozwód możemy zdefiniować jako encję opisaną przez datę rozvodu, miejsce rozvodu, nazwisko i imię żony, nazwisko i imię męża,
- rozwód może być atrybutem w encji OSOBA (np. data rozvodu),
- rozwód może być związkiem encji KOBIECI z encją MĘŻCZYZNA.

Wybór odpowiedniej decyzji, jak zaklasyfikować dane, należy do projektanta bazy danych i wynika z jego subiektywnej oceny, intuicji, wiedzy oraz doświadczenia w zakresie projektowania baz danych.

Powinno wyodrębnić się encje, które mają co najmniej dwa atrybuty. Jeśli nie można wskazać dwóch atrybutów dla danej encji należy obiekt przedstawić jako związek lub jako atrybut. Pomocą w znalezieniu właściwych atrybutów mogą być wykorzystywane formularze papierowe lub inne dostępne druki. Należy je dokładnie przeanalizować, gdyż bardzo często oprócz danych wpisywanych w odpowiednie rubryki na formularzach znajduje się wiele danych dopisywanych dodatkowo poza rubrykami, co może świadczyć o potrzebie wprowadzenia dodatkowych atrybutów. Aby określić, do której encji dany atrybut przypisać należy zadać pytanie typu „Co dany atrybut opisuje”. Często w odpowiedzi na to pytanie kryje się nazwa encji.

Przeprowadzając analizę rzeczywistości napotykamy często na dane, które są wyliczane na podstawie zgromadzonych danych, np. średnia ocen studenta, dochód na osobę itp. Powstaje wówczas pytanie, czy daną wielkość lepiej przechowywać w bazie danych jako np. dodatkowy atrybut, czy wyliczać ją za każdym razem kiedy jest potrzebna. Odpowiedź zależy od rodzaju danej wyliczanej, od częstości jej wyliczania i jak często ulega zmianie.

Po wyodrębnieniu encji i związków możemy przejść do kolejnego etapu polegającego na podaniu definicji predykatowych encji i związków.

3.6.3. DEFINICJE PREDYKATOWE ENCJI I ZWIĄZKÓW

W tym rozdziale omówimy sposób zwięzłego przedstawiania wyodrębnionych typów encji oraz typów związków między nimi.

3.6.3.1. Definicje predykatowe typów encji

Definicje predykatowe typów encji pozwalają w zwarty i czytelny sposób opisać encje występujące w diagramie związków encji oraz na szybką, niemalże mechaniczną, transformację modelu konceptualnego do modelu logicznego.

Na podstawie opisu encji sporządzamy zestawienie typów encji. W tym celu wypisujemy kolejne nazwy typów encji (np. uporządkowane alfabetycznie, co ułatwia odszukanie wybranego typu encji) oraz w nawiasach podajemy nazwy atrybutów. Klucz główny encji zaznaczamy przez podkreślenie i umieszczamy na początku listy atrybutów.

Definicje predykatowe typów encji można przedstawić w następujący sposób:

ENC/*x* NAZWA TYPU ENCJI (lista atrybutów)

gdzie *x* jest kolejnym numerem typu encji.

Przykład 3.7 (definicje predykatowe typów encji)

ENC/001 JĘZYK (IdJęzyka, Nazwa)

ENC/002 OSOBA (IdOsoby, Nazwisko, Imię, DataUrodzenia)

□

3.6.3.2. Definicje predykatowe typów związków

Podobnie jak w przypadku definicji predykatowych typów encji, tak i w przypadku definicji predykatowych typów związków, należy wypisać kolejne nazwy związków zachodzących między encjami (np. uporządkowane alfabetycznie według nazw związków, co ułatwia odszukanie wybranego związku) oraz w nawiasach wypisać nazwy encji, między którymi związek zachodzi, uczestnictwo i licznosc związku po stronie danej encji, oraz nazwy atrybutów związku. Nazwa związku wraz z nazwami powiązanych z nią encji ma określone znaczenie semantyczne.

Ogólna postać definicji typu związków binarnych:

ZWI/*x* Nazwa związku(ENCJA1(*min1*, *max1*) : ENCJA2(*min2*, *max2*); lista)

gdzie:

- | | |
|---------------------------|--|
| <i>x</i> | – kolejny numer związku, |
| <i>min1</i> , <i>min2</i> | – minimalna liczba encji uczestniczących w związku, |
| <i>max1</i> , <i>max2</i> | – maksymalna liczba encji uczestniczących w związku, |
| lista | – lista atrybutów związku. |

Przykład 3.8 (definicja predykatowa typu związku)

ZWI/001 Zna(OSOBA(0,N) : JĘZYK(0,N); *StopieńZnajomości*)

□

Przykład 3.9 (definicje predykatowe typów encji i związków)

a)

ENC/001 OSOBA (*IdOsoby, Nazwisko, Imię, DataUrodzenia, NIP, PESEL*)

ENC/002 WYKSZTAŁCENIE (*IdPoziomu, Nazwa*)

ENC/003 PRZEDMIOT (*NumerPrzedmiotu, NazwaPrzedmiotu*)

ZWI/001 Ma (OSOBA(0, N) : WYKSZTAŁCENIE(1, 1))

ZWI/002 Naucza (OSOBA(0, N) : PRZEDMIOT(0, N); *IdWpisu, LiczbaGodzin, Klasa*)

b)

ENC/001 OSOBA (*IdOsoby, Nazwisko, Imię, DataUrodzenia*)

ENC/002 JĘZYK (*IdJęzyka, Nazwa*)

ZWI/001 Zna(OSOBA(0,N) : JĘZYK(0,N); *StopieńZnajomości*)

Jeśli encja typu OSOBA ma dwie instancje:

(1, "Lis", "Alicja", 10-11-1980)

(2, "Kot", "Jan", 15-01-1980)

a encja typu JĘZYK ma trzy instancje:

(1, "Angielski")

(2, "Niemiecki")

(3, "Francuski"),

to ponieważ związek **Zna** określiliśmy jako związek binarny z dodatkowym atrybutem związku *StopieńZnajomości*, więc, ponieważ związek też jest relacją, związek **Zna** może, na przykład, zawierać następujące krotki:

(1, "Lis", "Alicja", 10-11-1980, 1, "Angielski", "bierna")

(1, "Lis", "Alicja", 10-11-1980, 3, "Francuski", "podstawowa").

□

3.6.4. DIAGRAM ZWIĄZKÓW ENCJI

Model konceptualny najlepiej jest wyrazić w sposób graficzny za pomocą **diagramu związków encji** (ERD – ang. *Entity Relationship Diagram*). Dzięki ERD w czytelny i przejrzysty sposób przedstawiamy wyodrębnione we wcześniejszych fazach encje oraz związki między nimi. W niektórych notacjach na diagramach ER (związków encji) są wyszczególniane atrybuty i klucze encji oraz atrybuty związków. Każda encja na diagramie powinna być powiązana z co najmniej jedną inną encją. Bywa jednak tak, że musimy rozstrzygnąć, czy dwie encje, które są już połączone w sposób pośredni, połączyć również w sposób bezpośredni. Aby podjąć właściwą decyzję, należy zwrócić uwagę na spójność danych.

Opracowując graficznie diagram związków encji należy kierować się zasadą, aby uzyskany diagram był poprawny a jednocześnie czytelny. W tym celu należy starać się wyrównywać encje w poziomie, linie związków prowadzić poziomo lub pionowo, ewentualnie załamywać pod kątem prostym lub 45 stopni. Jeśli linie muszą się przecinać, to staramy się to robić pod kątem 30–60 stopni. Nie należy rysować zbyt wielu linii równoległych blisko siebie. Należy zostawiać odpowiednio dużo wolnego miejsca w celu zwiększenia czytelności diagramu. Nazwy encji umieszczane w prostokątach powinny być w obrębie całego diagramu jednakowo sformatowane tzn. umieszczane centralnie, wyrównane lewostronnie lub prawostronnie.

Można poprawić czytelność diagramu umieszczając koniec związku „wiele” z prawej lub dolnej strony prostokąta reprezentującego encję.

Ze względu na zmiany dokonywane w diagramach w trakcie projektowania bazy danych dobrym zwyczajem jest opisywanie diagramu danymi: autor i tytuł diagramu, data opracowania diagramu.

Diagramy można odczytywać zgodnie z przyjętą kolejnością czytania tekstów a więc od lewej do prawej, z góry na dół. Dlatego dobrze jest umieszczać encje o dużym znaczeniu w lewym górnym rogu. Wielkość prostokątów dla oznaczenia encji może być różna.

Dodając nową encję należy zwrócić uwagę, aby miała co najmniej dwa atrybuty, jeden klucz, występowała w co najmniej jednym związku z innymi encjami i była wykorzystywana w co najmniej jednej operacji bazodanowej.

Przykład 3.10

Załóżmy, że projektujemy bazę danych **WYPOŻYCZALNIA_1** dla wypożyczalni kaset video i w wyniku analizy rzeczywistości wyodrębniliśmy cztery encje: KLIENT, KASETA, FILM i GATUNEK oraz trzy związki między tymi encjami: **Zawiera, Jest, Wypożycza**.

Definicje predykatowe wyodrębnionych typów encji i związków mają następującą postać:

ENC/001 KLIENT (*NrKlienta, Nazwisko, Imię, Miasto, UlicaNr*)

ENC/002 KASETA (*NrKasety, Opis*)

ENC/003 FILM (*KodFilmu, Tytuł, CzasTrwania*)

ENC/004 GATUNEK (*KodGatunku, Rodzaj*)

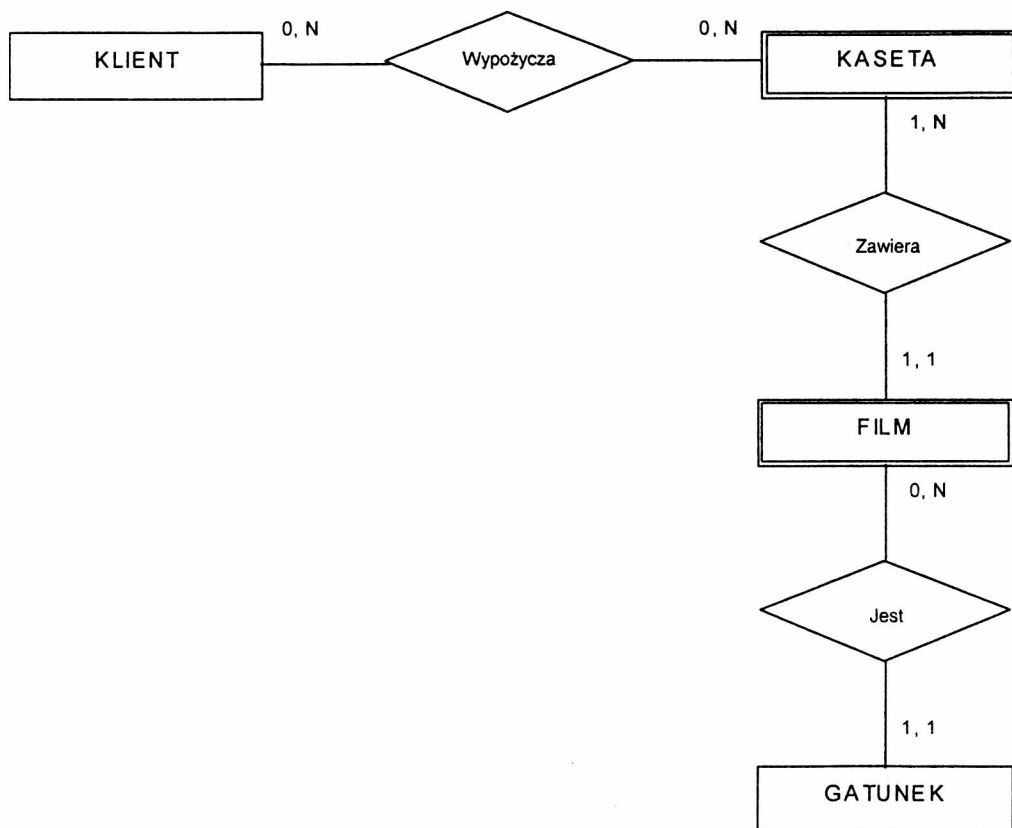
ZWI/001 Zawiera (KASETA(1,N) : FILM(1,1))

ZWI/002 Jest (FILM(0,N) : GATUNEK(1,1))

ZWI/003 Wypożycza (KLIENT(0,N) : KASETA(0,N); *NrWypożyczenia, DataWypożyczenia, DataZwrotu, Oplata*)

Zwróćmy uwagę, że związek **Wypożycza**, jest związkiem opcjonalnym wiele do wiele (jeden klient może wypożyczyć wiele kaset lub żadną, kasecie z wypożyczalni może być nie wypożyczona dotychczas ani razu lub mogła już być wypożyczana przez wielu różnych klientów).

Rysunek 3.4 przedstawia diagram związków encji dla bazy danych **WYPOŻYCZALNIA_1**. Encja **FILM** jest zaznaczona jako słaba, ponieważ jest zależna od encji silnej **GATUNEK** (nie może istnieć bez niej), encja **KASETA** jest również zaznaczona jako słaba, ponieważ jest zależna od encji **FILM**.



Rys. 3.4. Diagram związków encji dla bazy WYPOŻYCZALNIA_1

Zgodnie z uwagą zawartą w rozdz. 3.6.2, zalecającą zastępowanie związków N:N dwoma związkami 1:N, należałoby przekształcić diagram z rysunku 3.4 i zastąpić związek **Wypożycza** encją **WYPOŻYCZENIE** (taką encję nazywamy *encją powiązania* lub *encją asocjacyjną*). Otrzymalibyśmy wówczas bazę danych **WYPOŻYCZALNIA_2** zawierającą następujące encje i związki:

ENC/001 KLIENT (*NrKlienta*, *Nazwisko*, *Imię*, *Miasto*, *UlicaNr*)

ENC/002 KASETA (*NrKasety*, *Opis*)

ENC/003 FILM (*KodFilmu*, *Tytuł*, *CzasTrwania*)

ENC/004 GATUNEK (*KodGatunku*, *Rodzaj*)

ENC/005 WYPOŻYCZENIE (*NrWypożyczenia*, *DataWypożyczenia*,
DataZwrotu, *Oplata*)

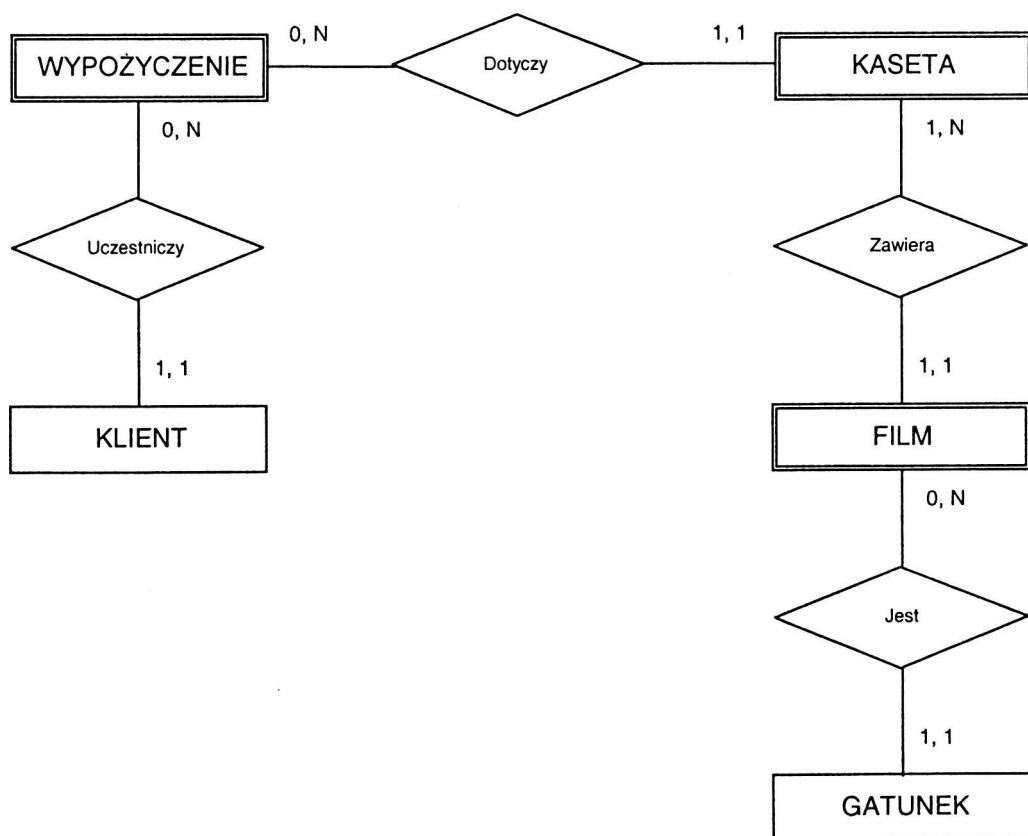
ZWI/001 Zawiera(KASETA(1,N) : FILM(1,1))

ZWI/002 Jest (FILM(0,N) : GATUNEK(1,1))

ZWI/003 Uczestniczy (KLIENT(1,1) : WYPOŻYCZENIA(0,N))

ZWI/004 Dotyczy (WYPOŻYCZENIE (0,N) : KASETA(1,1))

Rysunek 3.5 przedstawia diagram związków encji dla bazy **WYPOŻYCZALNIA_2**.



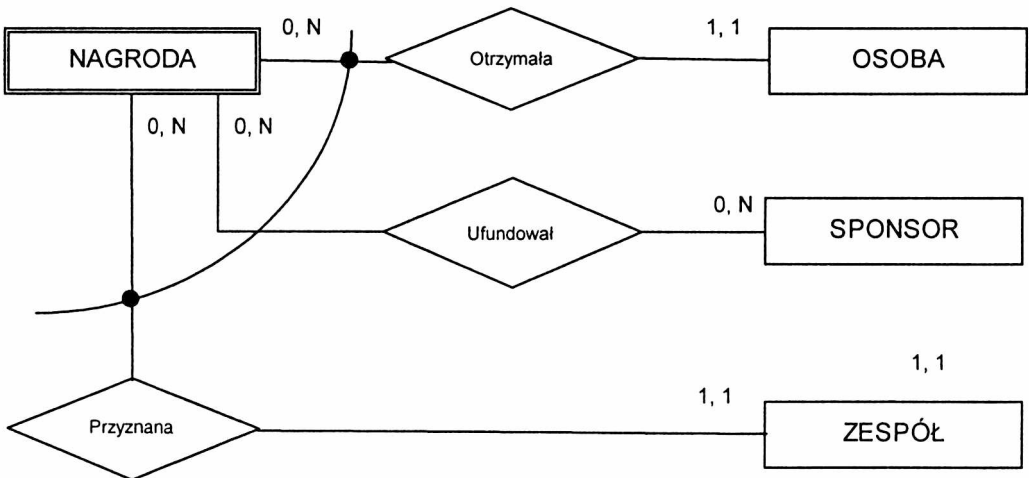
Rys. 3.5. Diagram związków encji dla bazy WYPOŻYCZALNIA_2

□

Czytelników zaintrygowanych związkiem **Zawiera**, który jest obligatoryjny zarówno po stronie encji FILM, jak i encji KASETA, odsyłamy do rozdz. 4.1.1, gdzie jest przeprowadzona jego analiza.

Związki typu wiele do wiele, z wyjątkiem związków nie posiadających dodatkowych atrybutów, powinny być eliminowane z diagramu przez zastąpienie ich encją asocjacyjną o nazwie wyprowadzonej z nazwy związku, np. ze związku **Wypożyczenie** powstała encja WYPOŻYCZENIA.

Często na diagramach jest potrzeba przedstawienia sytuacji powiązania encji A z encją B1 albo z encją B2 (ale tylko z jedną z nich). Można to wyrazić na diagramie prowadząc łuk wykluczający przez linie związków i zaznaczając punkty przecięcia z właściwymi związkami czarnymi kropkami tak jak przedstawiono to na rysunku 3.6.



Rys. 3.6. Diagram związków encji z łukiem wykluczającym po stronie „wiele” związku

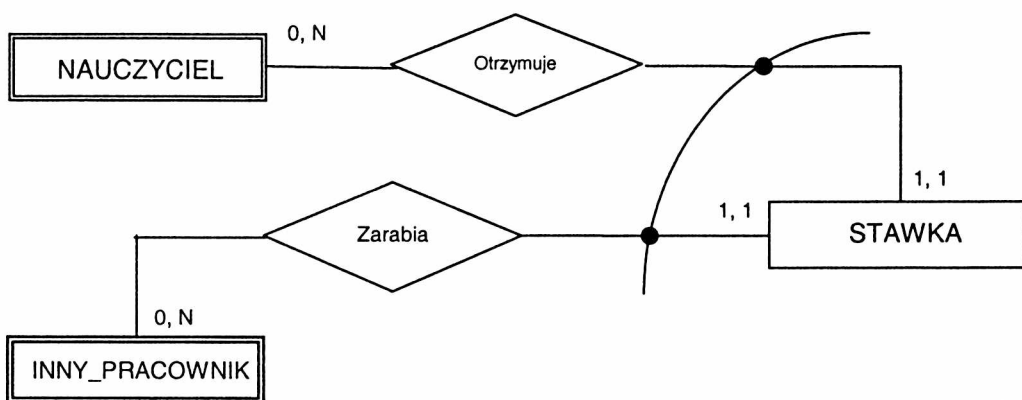
Nagroda (opisywana przez atrybuty *TytułNagrody*, *RokPrzyznania*) jest przyznana **albo** jako indywidualna dla jednej osoby (encja OSOBA o atrybutach *Nazwisko*, *Imię*, *DaneAdresowe*) **albo** dla zespołu (encja ZESPÓŁ o atrybutach *NazwaZespołu*, *LiczbaOsóbWZespole*, *DataPowstania*). Związek **Ufundował** nie jest częścią łuku wykluczającego. Obowiązuje zasada, że dany związek może należeć tylko do jednego łuku wykluczającego. Zwykle łuki rysuje się przy końcach związków po stronie wiele. Końce związków należące do łuku wykluczającego muszą być wszystkie opcjonalne albo wszystkie obligatoryjne.

Na rysunku 3.7 został przedstawiony diagram, w którym łuk wykluczający jest poprowadzony po stronie „jeden” związku. Jest to przypadek poprawny, który umożliwia zmiany w stawkach zarobkowych dla pracowników będących nauczy-

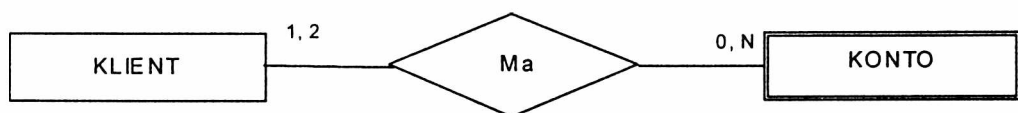
cielami i pozostałego personelu. Dana kategoria zarobkowa jest przypisana albo do nauczycieli albo do pozostałych pracowników.

Czasami chcemy precyzyjniej odzwierciedlić na diagramie ograniczenie dotyczące maksymalnej liczności danego końca związku. Z diagramu przedstawionego na rysunku 3.8 wynika, że klient może założyć w banku wiele kont ale konto może należeć maksymalnie do 2 osób (musi należeć co najmniej do jednej osoby).

Liczność 2 (i każda liczba większa od 1) jest traktowana jak wiele.



Rys. 3.7. Diagram związków encji z luką wykluczającym po stronie „jeden” związku



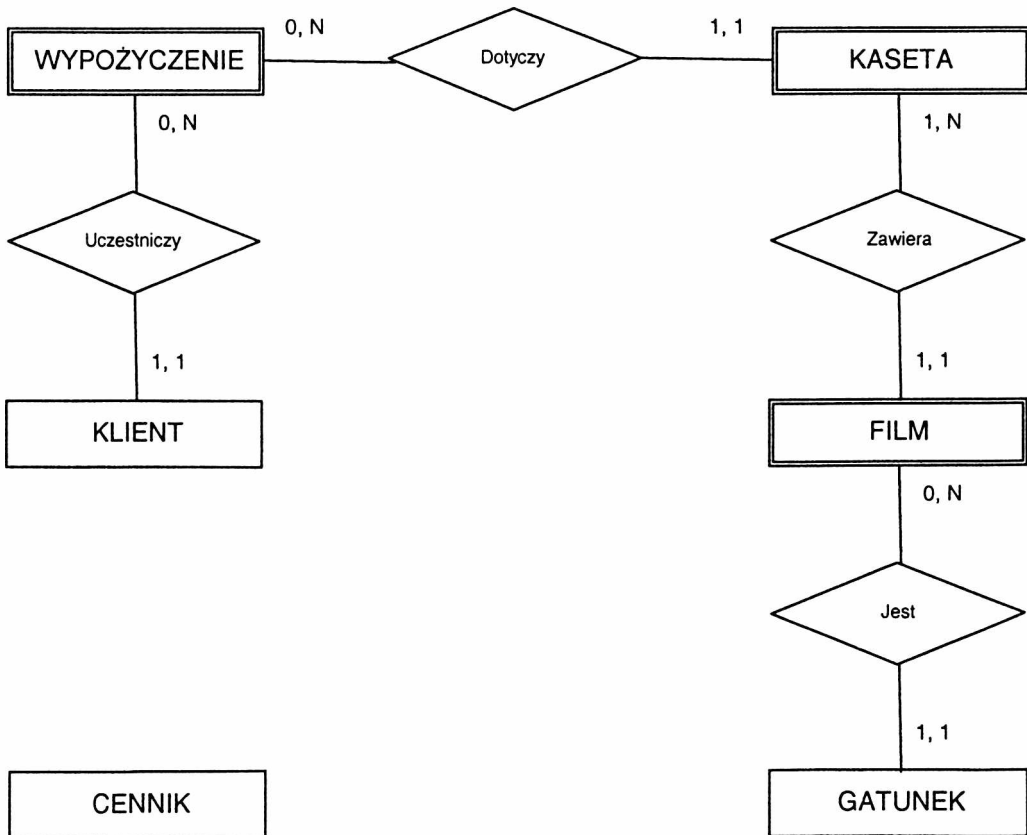
Rys. 3.8. Diagram z ograniczoną (do dwóch) licznością występowania encji KLIENT w związku **Ma**

Zgodnie z notacją przyjętą w tej książce dla oznaczania licznosci związku i uczestnictwa encji w związku, które można interpretować jako podanie minimalnej i maksymalnej licznosci dla każdego końca związku, związek **Ma** z diagramu 3.8 odczytujemy, że klient **może** założyć w banku od **0** (opcjonalność występowania encji KLIENT) do **N** kont, konto **musi** należeć do **1** (obligatoryjność występowania encji KONTO w związku) lub do 2 osób.

Aby sprawdzić poprawność diagramu pod kątem kompletności danych należy wybrać z diagramu kolejno każdą encję i prześledzić jej udział w operacjach bazodanowych czyli prześledzić jej cykl życia. Na przykład klient najpierw jest

bazodanowych czyli prześledzić jej cykl życia. Na przykład klient najpierw jest zapisywany do wypożyczalni, potem może wypożyczać kasety, może je zwracać, można modyfikować jego dane (np. zmiana adresu), można sprawdzić, które kasety dotychczas wypożyczył, których kaset nie oddał, jak długo je przetrzymywał, ile zapłacił za wypożyczenia, czy korzystał z rabatów, ewentualnie po pewnym czasie, można usunąć jego dane z bazy. Na każdym etapie należy sprawdzać, czy dzięki zgromadzonym danym te operacje będą możliwe do wykonania. Być może w opracowanym projekcie np. nie uwzględniono kosztów wypożyczenia i należy dodać odpowiednie atrybuty lub encje.

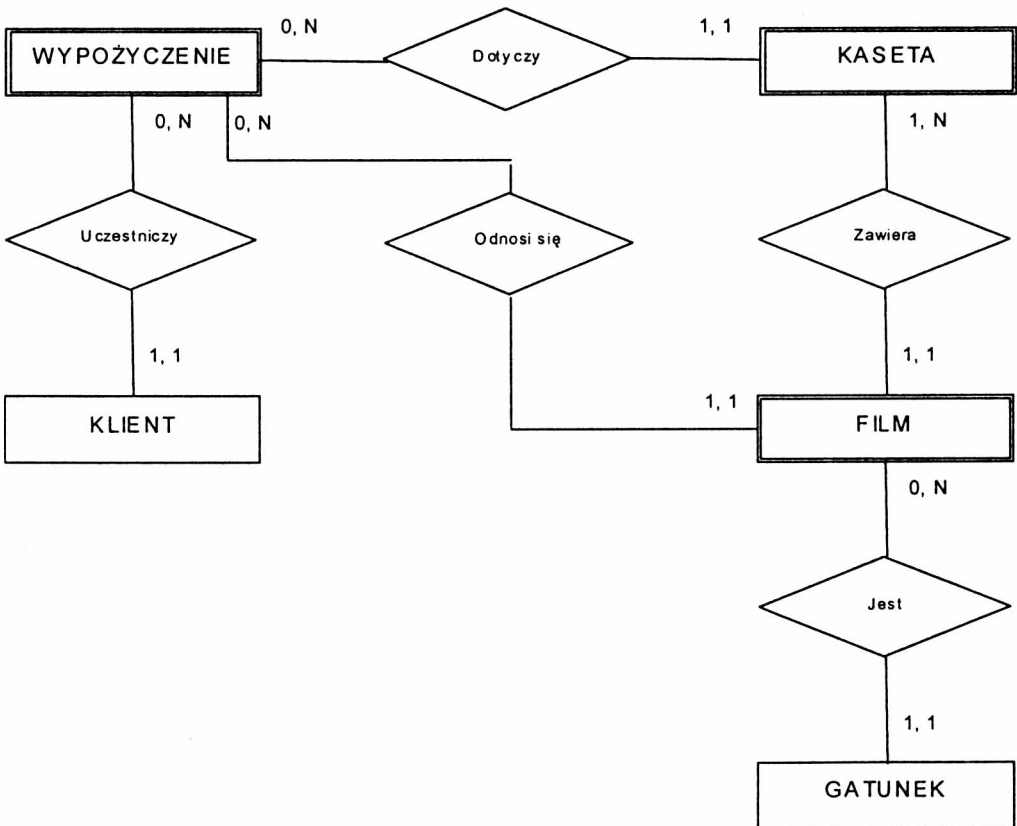
Zgodnie z zasadą, że każda encja występująca na diagramie musi uczestniczyć w co najmniej jednym związku, diagram przedstawiony na rysunku 3.9 należy uznać za niewłaściwy do zaprojektowania bazy danych, ponieważ encja **CENNIK** występuje w związku z żadną inną encją.



Rys. 3.9. Błędny diagram związków encji zawierający encję CENNIK bez związku z żadną inną encją

Poprawny diagram związków encji nie powinien zawierać zbędnych związków, które mogą być wyprowadzone z innych związków, np. na diagramie przedstawionym na rysunku 3.10. związek **Odnosi się** między encjami WYPOŻYCZENIE i FILM jest zbędny ponieważ może być wyprowadzony ze związku **Dotyczy**. Pozostawienie go prowadziłoby do utraty spójności danych (można przypisać do danego wypożyczenia film z innej kasety niż ta, która go zawiera).

Jako ćwiczenie proponujemy opracować diagram przedstawiający przynależność pracowników uczelni wyższej do wydziału, instytutu, zakładu, katedry według reguł obowiązujących w danej uczelni.



Rys. 3.10. Błędny diagram związków encji zawierający zbędny związek **Odnosi się**

Na zakończenie tego rozdziału podkreślmy, że przy projektowaniu bazy danych, prawidłowa identyfikacja encji i związków powinna doprowadzić do utworzenia

schematów relacji będących w trzeciej postaci normalnej (rozdz. 2.3). Aby upewnić się, że w danym modelu konceptualnym prawidłowo wyodrębniliśmy encje i związki należy sprawdzić:

- czy w encjach lub związkach nie ma powtarzających się atrybutów lub grup atrybutów; występowanie atrybutów postaci:

ImięDziecka_1, ImięDziecka_2, ImięDziecka_3,
DataUrDziecka_1, DataUrDziecka_2, DataUrDziecka_3,

jest sygnałem, że należy utworzyć nową encję DZIECKO (1PN),

- czy nie ma atrybutów, które zależą tylko od części klucza głównego, jeśli tak jest, to ten atrybut i ta część klucza, od której atrybut zależy, jest podstawą do utworzenia nowej encji (2PN), np. dla bazy **SZKOŁA** z encji

OCENA(Przedmiot, NrUcznia, Ocena, Nauczyciel),

przy założeniu, że dany przedmiot uczy jeden nauczyciel, należy utworzyć dwie encje

OCENA(Przedmiot, NrUcznia, Ocena),

PRZYDZIAŁ(Przedmiot, Nauczyciel),

- czy nie ma atrybutów zależnych od atrybutów, które nie są częścią klucza głównego, jeśli tak jest, to te atrybuty są podstawą do utworzenia nowej encji pozostającej w związku jeden do wiele z encją wyjściową (3PN), np. encję

PRZYDZIAŁ(Id, Przedmiot, Nauczyciel, AdresNauczyciela),

gdzie nauczyciel może prowadzić wiele przedmiotów i adres nauczyciela jest funkcyjnie zależny od nauczyciela, należy zastąpić encjami:

PRACOWNIK(Nauczyciel, AdresNauczyciela),

PRZYDZIAŁ(Id, Przedmiot, Nauczyciel),

przy czym encja PRACOWNIK będzie w związku 1:N z encją PRZYDZIAŁ.

4. PROJEKTOWANIE LOGICZNE

Projektowanie logiczne bazy danych polega na transformacji opracowanego modelu konceptualnego bazy (rozdz. 3.6) do modelu logicznego (na przykład relacyjnego), w którym określamy schemat bazy danych czyli schematy relacji. Każdy schemat relacji definiuje pewną klasę relacji (rozdz. 2.1), które będziemy reprezentować w postaci tabel (rozdz. 2.3). Na etapie projektowania logicznego powinno się sprawdzić, czy utworzone schematy relacji są przynajmniej w 3PN i jeśli nie, to przeprowadzić normalizację schematów relacji. Jak już wspomnieliśmy wcześniej (rozdz. 2.3) normalizacja danych jest sformalizowaną procedurą, w wyniku której eliminujemy redundancje (powtarzanie się) danych, ich niespójności i anomalie czyli nieprawidłowości występujące w bazie danych związane z wykonywanymi operacjami czyli aktualizacją, usuwaniem i dopisywaniem danych.

Projektowanie logiczne bazy danych ma na celu skonstruowanie bazy, która określi sposób zgrupowania danych i powiązań między nimi (rozdz. 4.1.2). Transformacja modelu konceptualnego bazy danych do modelu logicznego jest wykonywana według pewnych reguł, które są opisane w rozdziale 4.1.1. Model logiczny bazy danych można zwięźle przedstawić podając schemat bazy danych (rozdz. 4.1.3). W kolejnym etapie projektowania logicznego bazy danych należy określić grupy użytkowników bazy danych i wyodrębnić perspektywy (rozdz. 4.2).

4.1. TRANSFORMACJA MODELU KONCEPTUALNEGO DO MODELU LOGICZNEGO

W rozdziale tym omówimy, w jaki sposób przeprowadzamy transformację modelu konceptualnego do relacyjnego modelu logicznego będącego zbiorem struktur danych reprezentowanych przez tabele. Jeśli mamy poprawnie zdefiniowane encje i związki między nimi, to na ogół otrzymujemy schematy relacji w trzeciej postaci normalnej.

4.1.1. REGUŁY TRANSFORMACJI

Transformację modelu konceptualnego do modelu logicznego przeprowadzamy według niżej podanych reguł.

1. Dla każdej encji z diagramu związków encji tworzymy schemat relacji (reprezentacją relacji o zadanym schemacie jest tabela). Najczęściej nazwa schematu

relacji i tabeli (relacji) jest taka sama jak encji tylko w liczbie mnogiej ze względu na to, że relacja zawiera wiele wystąpień obiektu.

2. Atrybuty encji stają się atrybutami w schemacie relacji (o takich samych nazwach jak odpowiadające im atrybuty encji). Dla kolumn tabeli reprezentującej relację dobieramy odpowiednie formaty danych. Atrybuty odpowiadające kluczom głównym encji stają się kluczami głównymi relacji. Atrybuty opcjonalne stają się kolumnami o dopuszczalnych wartościach NULL, atrybuty obligatoryjne (wymagane) stają się kolumnami NOT NULL.
3. Dla każdego związku binarnego *jeden do wiele* (*jeden do jeden*) obligatoryjnego po stronie *wiele*, wstawiamy klucz główny ze strony *jeden* do schematu relacji reprezentującej stronę *wiele* związku. W ten sposób otrzymujemy klucz obcy (ang. *foreign key*) w schemacie relacji ze strony *wiele* związku (NOT NULL).
4. Związek binarny typu *jeden do wiele*, opcjonalny po stronie *wiele*, reprezentujemy zazwyczaj nowym schematem relacji, w którym umieszczamy klucze główne obu encji (ze związku opcjonalnego po stronie *jeden* otrzymujemy kolumny NULL, ze związków obligatoryjnych – NOT NULL). Jeżeli co najmniej 60% instancji encji pozostaje w związku, to uznaje się związek za prawie obligatoryjny i zaleca się zastosować transformację według reguły 3.
5. Dla związku opcjonalnego *wiele do wiele* tworzymy nowy schemat relacji (nową tabelę, ewentualnie ze sztucznym kluczem głównym). Klucze główne z encji po obu stronach związku stają się kluczami obcymi w utworzonym schemacie relacji. Jeśli związek posiadał atrybuty dodatkowe, to stają się one atrybutami w nowo utworzonym schemacie relacji. Kluczem głównym może być klucz złożony z obu kluczy obcych lub nowo wprowadzony klucz sztuczny.
6. Atrybutom, które odpowiadają kluczom kandydującym encji, należy zapewnić unikalność wartości.
7. Jeśli, w wyniku transformacji, dla utworzonego schematu relacji pojawi się klucz naturalny a schemat posiada również klucz sztuczny, to można zbędny klucz sztuczny usunąć.

Powyższe zasady ogólne transformacji na ogół dają dobre efekty, ponieważ uwzględniają dwa typy związków spotykane najczęściej:

$$\mathbf{Zw1(A(1,1) : B(0,N))}$$

oraz

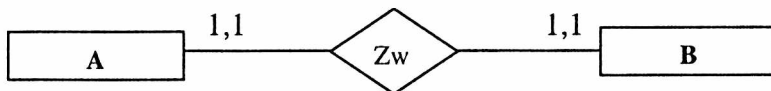
$$\mathbf{Zw2(A(0,N) : B(0,N)).}$$

W szczególnych przypadkach podane zasady transformacji mogą się okazać jednak nie wystarczające, ponieważ nie wszystkie uwzględniają uczestnictwo encji w związku (obligatoryjność/opcjonalność).

Przeanalizujemy teraz dokładnie różne typy związków (rzadko lub często spotykanych) uwzględniając licznosc i uczestnictwo encji w związku.

1. Dla związku binarnego *jeden do jeden* mogą zachodzić trzy przypadki:

a. **Zw(A(1,1) : B(1,1))**



Rys. 4.1. Graficzna reprezentacja związku **Zw(A(1,1) : B(1,1))**

Z encji A i B tworzymy jeden schemat relacji, który zawiera atrybuty z obu encji a kluczem głównym jest klucz główny encji A lub klucz główny encji B, np. studenci danego kierunku studiów mają jeden i tylko jeden indeks, indeks należy dokładnie do jednego studenta. Reguły te odzwierciedla związek **Ma** zachodzący między encjami STUDENT i INDEKS:

STUDENT (PESEL, Nazwisko, Imię, DataUr)

INDEKS (NrIndeksu, Uczelnia, Wydział, Kierunek, DataRozp)

Ma (STUDENT(1,1) : INDEKS(1,1))

W wyniku transformacji powstanie schemat relacji:

Studenci (NrIndeksu, Nazwisko, Imię, DataUr, PESEL,
Uczelnia, Wydział, Kierunek, DataRozp)

W tym wypadku na klucz główny wybrano NrIndeksu a nie PESEL, który powinien być daną chronioną (jako dana osobowa studenta).

b. **Zw(A(1,1) : B(0,1))**



Rys. 4.2. Graficzna reprezentacja związku **Zw(A(1,1) : B(0,1))**

Z encji A i B tworzymy schematy relacji **RA** i **RB**. Do schematu **RB** wstawiamy jako dodatkowy atrybut klucz główny ze schematu **RA**. Kluczem w schemacie relacji **RB** może być atrybut odpowiadający kluczowi encji B lub wstawiony klucz obcy. Omawianą sytuację ilustrują przykładowe encje OSOBA (encja A) i NUMER_NIP (encja B) oraz związek między nimi **Ma** (np. dla bazy Urzędu Skarbowego):

OSOBA (PESEL, Nazwisko, Imię, DataUr)

NUMER_NIP (NIP, NrWpisu, Miasto, DataNadania)

Ma (OSOBA(1,1) : NUMER_NIP(0,1))

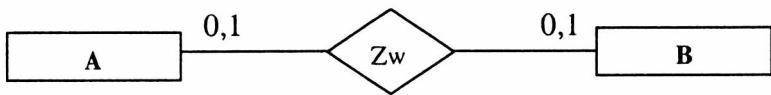
Osoba może mieć (nie musi) co najwyżej jeden numer NIP a dany numer musi mieć przypisaną dokładnie jedną osobę. W wyniku transformacji powstaną następujące schematy relacji:

Osoby (PESEL, Nazwisko, Imię, DataUr)

Numery_NIP (NIP, NrWpisu, Miasto, DataNadania, #PESEL)

Zapis #PESEL w schemacie relacji oznacza klucz obcy (czyli PESEL jest kluczem głównym w innej tabeli). Kluczem głównym schematu Numery_NIP jest atrybut NIP, ale może być również atrybut #PESEL.

c. **Zw(A(0,1) : B(0,1))**



Rys. 4.3. Graficzna reprezentacja związku **Zw(A(0,1) : B(0,1))**

Z encji A i B tworzymy schematy relacji **RA** i **RB**. Związek binarny **Zw** reprezentujemy nowym schematem relacji **RAB**, który zawiera dwa klucze obce: klucz główny ze schematu **RA** i klucz główny ze schematu **RB**. Kluczem głównym nowoutworzonego schematu **RAB** jest dowolny z kluczy obcych.

Rozważmy związek **Wybiera** zachodzący między encjami PANI (encja A) i PAN (encja B):

PANI(NrPani, Nazwisko, Imię, KolorWłosów)

PAN (NrPana, Nazwisko, Imię, Wykształcenie, DataUr)

Wybiera (PANI(0,1) : PAN(0,1); DataWyboru)

Związek **Wybiera** przechowuje informacje o powstałych parach. Pani może (nie musi) być w parze tylko z jednym panem i każdy pan może (nie musi) być w parze tylko z jedną panią.

W wyniku transformacji zostaną utworzone schematy relacji:

Panie(NrPani, Nazwisko, Imię, KolorWłosów)

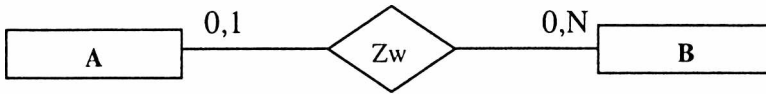
Panowie (NrPana, Nazwisko, Imię, Wykształcenie, DataUr)

Pary (#NrPani, #NrPana, DataWyboru)

Kluczem głównym w schemacie relacji **Pary** może być zarówno atrybut #NrPani jak i atrybut #NrPana.

2. Dla związku binarnego *jeden do wiele* mogą zachodzić cztery przypadki:

a. **Zw**(A(0,1) : B(0,N))



Rys. 4.4. Graficzna reprezentacja związku **Zw**(A(0,1) : B(0,N))

Z encji A i B tworzymy schematy relacji **RA** i **RB**. Związek binarny **Zw** reprezentujemy nowym schematem relacji **RAB**, który zawiera dwa klucze obce: klucz główny ze schematu RA i klucz główny ze schematu RB. Kluczem głównym nowoutworzonego schematu **RAB** jest klucz główny z **RB**. Prześledźmy następujący przykład encji i związku między nimi:

POKÓJ (*NrPokoju*, *Piętro*, *MaxLiczbaOsób*)

STUDENT (*Indeks*, *Nazwisko*, *Imię*)

Jest_przydzielony (**POKÓJ**(0,1) : **STUDENT**(0,N); *DataZamieszkania*)

Związek **Jest_przydzielony** przechowuje informacje o przydzielach studentów (encja B) do pokoi (encja A). Załóżmy, że dany akademik jest przeznaczony dla studentów danego wydziału. Pokój może być zamieszkały przez wielu studentów lub nie zamieszkały, student może zamieszkiwać jakiś pokój lub nie mieszkać w tym akademiku. Po zwolnieniu miejsca w akademiku dane o przydziale studenta do pokoju są usuwane.

W wyniku transformacji powstaną schematy relacji:

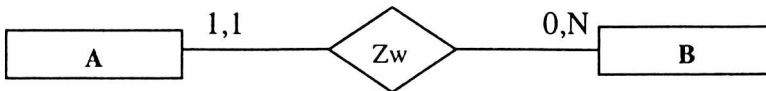
Pokoje (*NrPokoju*, *Piętro*, *MaxLiczbaOsób*)

Studenci (*Indeks*, *Nazwisko*, *Imię*)

Przydziały (*#Indeks*, *#NrPokoju*, *DataZamieszkania*)

Z pewnością czytelnikowi nie sprawi kłopotu uzasadnienie, dlaczego *NrPokoju* nie może być kluczem w schemacie relacji **Przydziały**.

b. **Zw**(A(1,1) : B(0,N))



Rys. 4.5. Graficzna reprezentacja związku **Zw**(A(1,1) : B(0,N))

Jest to bardzo często spotykany typ związku jeden do wiele, obligatoryjny po stronie wiele (każda instancja encji B musi uczestniczyć w związku oraz z jedną

instancją encji A może być związanych wiele instancji encji B) np. pokój (encja A) zamieszkiwany jest przez żadnego lub wielu studentów, student (encja B) zamieszkuje dokładnie jeden pokój.

Z encji A i B tworzymy schematy relacji **RA** i **RB**. Do schematu **RB** wstawiamy jako dodatkowy atrybut klucz główny ze schematu **RA**. Kluczem głównym w schemacie relacji **RA** jest atrybut odpowiadający kluczowi encji B, a kluczem głównym w schemacie **RB** jest atrybut odpowiadający kluczowi encji B. Omawianą sytuację ilustrują przykładowe encje POKÓJ i STUDENT oraz związek **Zamieszkuje**:

POKÓJ (*NrPokoju*, *Piętro*, *MaxLiczbaOsób*)

STUDENT (*Indeks*, *Nazwisko*, *Imię*)

Zamieszkuje (POKÓJ(1,1) : STUDENT(0,N); *DataZamieszkania*)

Związek **Zamieszkuje** wyraża związek studentów zamieszkujących w akademiku z obecnie zajmowanym pokojem. Pokój może być zamieszkiwany przez wielu studentów lub nie zamieszkiwany, student (skoro jest mieszkańcem akademika) musi zamieszkiwać dokładnie jeden pokój. Po zwolnieniu miejsca w akademiku dane o przydziale studenta do pokoju są usuwane.

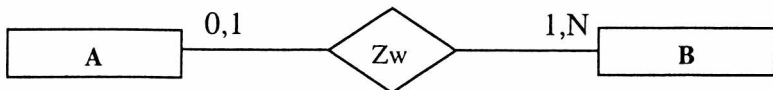
Po przeprowadzeniu transformacji modelu konceptualnego do modelu logicznego otrzymamy schematy relacji:

Pokoje (*NrPokoju*, *Piętro*, *MaxLiczbaOsób*)

Studenci (*Indeks*, *Nazwisko*, *Imię*, *#NrPokoju*, *DataZamieszkania*).

Obligatoryjność związku po stronie wiele (STUDENT) ma bardzo poważne konsekwencje. Dopisanie studenta do bazy nie jest możliwe dopóki nie ma przydzielonego pokoju, usunięcie pokoju z bazy np. przeznaczenie do remontu wiąże się z koniecznością usunięcia studenta lub przydzielenia mu od razu innego pokoju.

c. **Zw(A(0,1) : B(1,N))**



Rys. 4.6. Graficzna reprezentacja związku **Zw(A(0,1) : B(1,N))**

Z encji A i B tworzymy schematy relacji **RA** i **RB**. Jeśli bardzo dużo instancji encji B nie uczestniczy w związku **Zw**, to należy utworzyć nowy schemat **RAB**, do którego wstawiamy klucze główne ze schematu **RA** i **RB**. Kluczem głównym w schemacie **RAB** jest atrybut odpowiadający kluczowi głównemu z **RB**.

Związek tego typu zachodzi bardzo rzadko i często po ponownej analizie rzeczywistości okazuje się, że powinien być to związek N:N. Jeśli jednak odpowiada on modelowanej rzeczywistości, to okazuje się bardzo użytecznym. Rozważmy przykładowe encje PUNKT_SPRZEDAŻY i WYPOSAŻENIE oraz związek **Zakupiony**

PUNKT_SPRZEDAŻY (*NrPunktu*, *Adres*, *Telefon*)

WYPOSAŻENIE (*Symbol*, *Nazwa*, *Opis*)

Zakupiony (PUNKT_SPRZEDAŻY (0,1) : WYPOSAŻENIE (1,N);
DataZakupu, Cena)

W związku **Zakupiony** biorą udział te elementy wyposażenia, o których wiemy, gdzie zostały zakupione (towar mógł być kupiony co najwyżej w jednym punkcie sprzedaży lub nie pamiętamy gdzie go kupiliśmy). Z kolei w danym punkcie sprzedaży mogło być kupionych wiele towarów, przy czym gromadzimy dane tylko o tych punktach, gdzie kupiliśmy chociaż jeden towar.

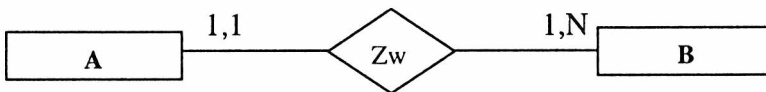
Po transformacji modelu konceptualnego do modelu logicznego otrzymamy schematy relacji:

Punkty_sprzedaży (*NrPunktu*, *Adres*, *Telefon*)

Wyposażenie (*Symbol*, *Nazwa*, *Opis*)

Zakupy (*#Symbol*, *#NrPunktu*, *DataZakupu*, *Cena*).

d. **Zw(A(1,1) : B(1,N))**



Rys. 4.7. Graficzna reprezentacja związku **Zw(A(1,1) : B(1,N))**

Związek tego typu zachodzi bardzo rzadko i często po głębszej analizie rzeczywistości okazuje się, że powinien być to związek opcjonalny po stronie jeden czyli powinien zostać zastąpiony często spotykanym związkiem

Zw1(A(1,1) : B(0,N)).

Pozostawienie związku w niezmienionej postaci będzie powodowało anomalie przy usuwaniu i dopisywaniu danych. Wszystkie encje typu A muszą być w związku **Zw** z co najmniej jedną encją typu B. Wszystkie encje typu B muszą być w związku **Zw** z dokładnie jedną encją typu A.

Transformacja związku polega na utworzeniu schematów relacji **RA** i **RB** z encji A i B. Do schematu **RB** wstawiamy jako dodatkowy atrybut klucz główny ze schematu **RA**. Kluczem głównym w schemacie relacji **RA** jest klucz główny encji A, a kluczem głównym w schemacie **RB** jest atrybut odpowiadający kluczowi encji B.

Prześledźmy następujący przykład encji i związku między nimi:

KLIENT (*NrKlienta*, *Nazwisko*, *Imię*)

BILET(*NrBiletu*, *Lot*, *Data*, *Godzina*)

Ma (KLIENT(1,1) : BILET(1,N))

W związku **Ma** biorą udział wszyscy klienci, którzy muszą mieć wykupiony co najmniej jeden bilet na przelot samolotem, mogą mieć kupionych wiele biletów ale dany bilet jest przypisany dokładnie do jednej osoby.

Po transformacji modelu konceptualnego do modelu logicznego otrzymamy schematy relacji:

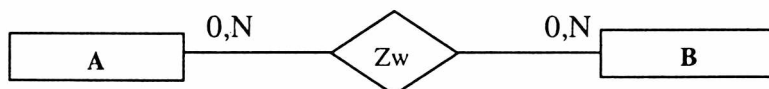
Klienci (*NrKlienta*, *Nazwisko*, *Imię*)

Bilety (*NrBiletu*, *Lot*, *Data*, *Godzina*, *#NrKlienta*).

Tak zaprojektowana baza ma jednak wady: usunięcie danych klienta powoduje konieczność usunięcia danych o biletach przez niego zakupionych, nie ma więc możliwości przechowywania danych o biletach, np. do sporządzania zestawień statystycznych dotyczących sprzedaży biletów, bez jednoczesnego przechowywania danych o klientach. Ponadto trzeba kontrolować usuwanie danych o biletach danego klienta: usunięcie danych o ostatnim bilecie wymaga usunięcia danych klienta.

3. Dla związku binarnego *wiele do wiele* teoretycznie mogą zachodzić trzy przypadki, jednak w praktyce spotykany jest tylko związek opcjonalny wiele do wiele, którego sposób transformacji omawia reguła 5.

a. **Zw**(A(0,N) : B(0,N))



Rys. 4.8. Graficzna reprezentacja związku **Zw**(A(0,N) : B(0,N))

Związek *wiele do wiele* opcjonalny po obu stronach (rysunek 4.8) ilustruje następujący przykład: klient (encja A) mógł dotychczas nie wypożyczyć jeszcze żadnej kasety video (encja B) lub wypożyczyć wiele kaset video; kasetą mogła być wypożyczana przez wielu klientów (po oddaniu kasety danych o wypożyczeniu się nie usuwa) lub ani razu jeszcze nie wypożyczona. Definicje encji i związku **Wypożyczył** są następujące:

KLIENT (*NrKlienta*, *Nazwisko*, *Imię*)

KASETA (*NrKasety*, *Tytuł filmu*)

Ma (KLIENT(0,N) : KASETA(0,N); *DataWypożyczenia*, *DataZwrotu*)

Schematy relacji otrzymane w wyniku transformacji mają postać:

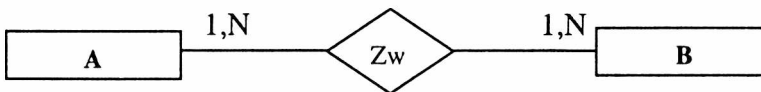
Klienci (*NrKlienta*, *Nazwisko*, *Imię*)

Kasety (*NrKasety*, *Tytuł filmu*)

Wypożyczenia (*NrWypożyczenia*, *#NrKlienta*, *#NrKasety*,
DataWypożyczenia, *DataZwrotu*).

Ponieważ dany klient może wielokrotnie wypożyczyć tę samą kasę (nawet tego samego dnia może ją dwukrotnie wypożyczyć i zwrócić) więc zbiór atrybutów {*#NrKlienta*, *#NrKasety*} nie może być kluczem głównym schematu **Wypożyczenia**, dlatego został wprowadzony klucz sztuczny *NrWypożyczenia*.

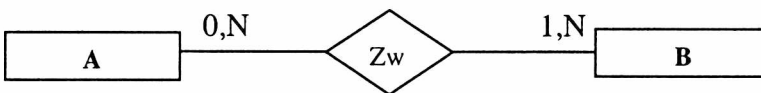
b. $Zw(A(1,N) : B(1,N))$



Rys. 4.9. Graficzna reprezentacja związku $Zw(A(1,N) : B(1,N))$

Związek *wiele do wiele* obligatoryjny po obu stronach, przedstawiony na rysunku 4.9, uważa się za niemożliwy do wystąpienia w praktyce. Związek ten oznacza, że żadna instancja encji po jednej stronie związku nie może istnieć bez instancji encji po drugiej stronie związku. W praktyce, po szczegółowej analizie takiego związku okazuje się, że taka konstrukcja nie oddaje dobrze rzeczywistości, jest przyczyną powstawania anomalii (np. usuwania) i związek ten trzeba zastąpić odpowiednim związkiem innego typu.

c. $Zw(A(0,N) : B(1,N))$



Rys. 4.10. Graficzna reprezentacja związku $Zw(A(0,N) : B(1,N))$

Związek tego typu należy do bardzo rzadko spotykanych i pozostawienie go w takiej postaci jest przyczyną powstawania anomalii w bazie danych przy wykonywaniu podstawowych operacji bazodanowych. Zaleca się ponowną analizę rzeczywistości i zastąpienie związku związkiem obustronnie opcjonalnym.

Przykład ilustrujący podany związek: klient (encja A) musi wypożyczyć co najmniej jedną kasę video, klient może wypożyczyć wiele kaset video; kase (encja B) może być wypożyczana przez wielu klientów (po oddaniu kasety danych o wypożyczeniu się nie usuwa) lub ani razu jeszcze nie wypożyczona. Definicje encji i związku **Wypożyczył** są następujące:

KLIENT (*NrKlienta*, *Nazwisko*, *Imię*)

KASETA (*NrKasety*, *Tytuł filmu*)

Ma (KLIENT(0,N) : KASETA(1,N); *DataWypożyczenia*, *DataZwrotu*)

Schematy relacji otrzymane w wyniku transformacji mają postać:

Klienci (*NrKlienta*, *Nazwisko*, *Imię*)

Kasety (*NrKasety*, *Tytuł filmu*)

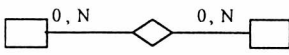
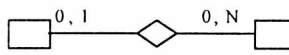
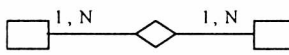
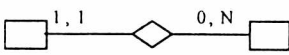
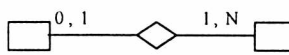
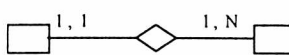
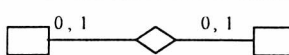
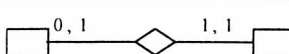
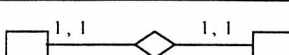
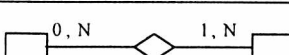
Wypożyczenia (*NrWypożyczenia*, *#NrKlienta*, *#NrKasety*,
DataWypożyczenia, *DataZwrotu*).

Ponieważ zbiór atrybutów {*#NrKlienta*, *#NrKasety*} nie może być kluczem głównym schematu **Wypożyczenia**, dlatego został wprowadzony klucz sztuczny *NrWypożyczenia*. Tak zaprojektowana baza nie pozwala na przechowywanie danych o klientach, którzy nie wypożyczyli żadnej kasety. Przy usuwaniu danych o wypożyczeniach (np. z ubiegłego roku) należy stale kontrolować, czy nie trzeba też usunąć danych klienta, co znacznie wydłuża czas wykonywania transakcji usunięcia.

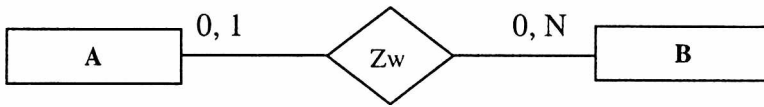
W tabeli 4.1 przedstawiamy związki występujące w praktyce często, rzadko oraz związki niemożliwe [Barker1996].

Tabela 4.1

Występowanie różnych typów związków

Związki występujące często	Związki występujące rzadko	Związki niemożliwe
		
		
		
		
		
		
		

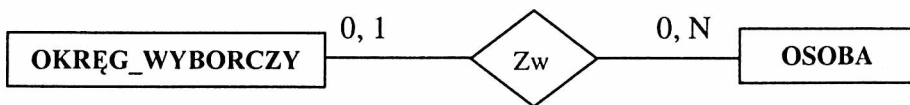
Rozważmy na przykład (rysunek 4.11) rzadko, ale jednak możliwą do wystąpienia, sytuację pierwszą z tabeli 4.1.



Rys. 4.11. Przykład obustronnie opcjonalnego związku 1:N

Na przykład w bibliotece czytelnik (encja A) może wypożyczyć jednorazowo zero lub kilka książek, książka (encja B) może być wypożyczona tylko przez jednego czytelnika lub może aktualnie nie być wypożyczona. Gdybyśmy zastosowali regułę 3. transformacji, to należałoby z encji A utworzyć tabelę **Tab_A**, z encji B utworzyć tabelę **Tab_B** i do **Tab_B** wstawić klucz główny z **Tab_A**. Ponieważ związek po obu stronach ma charakter opcjonalny, więc nie dla każdego wiersza z **Tab_B** musi istnieć odpowiadający mu wiersz z **Tab_A**. Wracając do naszego przykładu z biblioteką, może być taka sytuacja, że w **Tab_B** jest 1 000 000 wierszy opisujących książki a w danej chwili tylko 5 osób wypożyczyło po jednej książce. Tylko 5 książek będzie miało wpisaną wartość identyfikującą czytelnika, pozostałe 999 995 będzie miało wartości nieokreślone. W takim przypadku korzystniej jest wprowadzić trzecią tabelę **Tab_AB** zawierającą klucze główne z obu tabel (kluczem głównym nowoutworzonej tabeli **Tab_AB** będzie klucz obcy z encji B), przechowującą informacje o książkach aktualnie wypożyczonych i o osobach, które je wypożyczyły (czyli zastosować regułę 4). W podobny sposób należy zawsze rozważyć każdą zaistniałą sytuację.

Rozważmy jednak inny przykład związku obustronnie opcjonalnego 1:N, przedstawionego na rysunku 4.12.



Rys. 4.12. Przykład obustronnie opcjonalnego związku 1:N

Osoba jest na czas wyborów przypisana do jednego okręgu wyborczego lub nie przypisana do żadnego okręgu np. osoba bezdomna. Do danego okręgu może być przypisanych wiele osób lub jeszcze nikt nie przypisany. Typ OSOBA zawiera bardzo dużo encji (np. milion) a wśród nich bardzo niewiele nie ma przypisanego okręgu. W tym wypadku korzystniej będzie potraktować związek jako obligatoryjny po stronie wiele i zastosować 3. regułę transformacji czyli utworzyć dwie tabele **Osoby** i **Okręgi**, których klucz główny jest taki jak klucz odpowiednich encji i do tabeli **Osoby** wstawić klucz główny z tabeli **Okręgi**.

W rozdziale 3.6.4 podaliśmy przykład bazy WYPOŻYCZALNIA_1 (rysunek 3.4) i WYPOŻYCZALNIA_2 (rysunek 3.5). Występuje tam związek **Zawiera** obustronnie obligatoryjny

Zawiera (KASETA(1,N) : FILM(1,1))

zachodzący między encjami KASETA i FILM. W tabeli 4.1 związek tego typu występuje w grupie związków zachodzących rzadko. Pozornie może się wydawać, że jest to dobry przykład ilustrujący taki związek – najpierw wpisujemy film, od razu podajemy numery kaset, na których ten film się znajduje (co najmniej jedna kaseeta z filmem musi być, może być ich też więcej). Zakładamy, że dana kaseeta zawiera tylko jeden film i że nie ma kaset pustych. Obligatoryjność po stronie encji KASETA oznacza, że każda kaseeta musi być powiązana z jednym filmem. Podobnie dla encji FILM – każdy film musi być powiązany chociaż z jedną kasetą. Gdybyśmy więc usuwali dane o kasecie, która np. uległa zniszczeniu lub zgubieniu, to musielibyśmy jeszcze kontrolować, czy była to już ostatnia kaseeta z tym filmem. Jeśli tak, to trzeba wówczas, usuwając dane o kasecie, usunąć również dane o filmie. W praktyce na ogół nie usuwa się wtedy danych (jeśli dokupimy kasetę z takim filmem, to ponownie będziemy musieli wprowadzać opis filmu). Dlatego taki związek jest teoretycznie możliwy, ale po ponownej analizie rzeczywistości i ze względu na łatwiejsze utrzymanie spójności w bazie danych, okazuje się, że lepiej byłoby zastąpić go związkiem

Zawiera (KASETA(0,N) : FILM(1,1))

Przy tak określonym związku przy usuwaniu danych kasety z określonym filmem nie musimy kontrolować, czy istnieje jeszcze jakaś kaseeta z danym filmem ponieważ zakładamy, że mogą istnieć dane o filmie nie powiązanym z żadną kasetą znajdującą się na stanie wypożyczalni. Można więc przechowywać opisy filmów nie nagranych na kasetach.

Na diagramach związków encji występują jeszcze inne typy związków np. ternarne (między trzema encjami), rekurencyjne (związek encji z samą sobą). Najczęściej spotykany typ związku ternarnego jest *wiele do wiele do wiele*, opcjonalny z każdej strony

Zw(A(0,N) : B(0,N): C(0,N)).

Wynikiem transformacji związku **Zw** jest nowy schemat relacji **RABC**, do którego wstawiane są klucze główne z encji A, B, C.

W rozdz. 3.6.4 wspomnieliśmy o związkach wzajemnie się wykluczających. Rozważmy związek typu takiego jak na rysunku 4.13. Tworząc model logiczny należy wziąć pod uwagę, czy w encjach wzajemnie wykluczających się klucze główne mają takie same dziedziny czy nie.

Przypadek I – dziedziny kluczy głównych encji wykluczających się wzajemnie są takie same. Wówczas do schematu relacji powstałego z encji powiązanej, wstawiamy dwa dodatkowe (obligatoryjne) atrybuty:

- identyfikator encji (klucz główny encji ze strony jeden),
- identyfikator związku, którego wartość służy do rozróżniania związków.

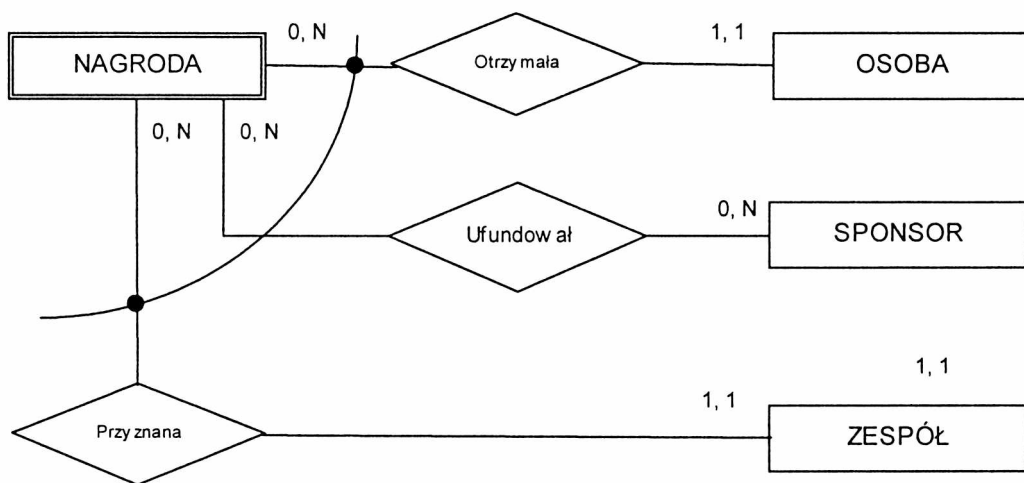
Przykład 4.1

Niech encje z rysunku 4.13 są zdefiniowane następująco:

ENC/001 NAGRODA (*IdNagrody*, *TytulNagrody*)

ENC/002 OSOBA (*IdOsoby*, *Nazwisko*, *Imię*, *PESEL*, *Telefon*)

ENC/003 ZESPÓŁ (*IdZespołu*, *NazwaZespołu*, *LiczbaOsóbWZespole*, *DataPowstania*)



Rys. 4.13. Diagram związków encji z luką wykluczającym po stronie „wiele” związku

Jeśli $\text{dom}(IdOsoby) = \text{dom}(IdZespołu)$, to po transformacji do modelu logicznego otrzymamy schematy relacji:

Nagrody (*IdNagrody*, *TytulNagrody*, *OsobaZespół*, *#IdNagrodzonego*)

Osoby (*IdOsoby*, *Nazwisko*, *Imię*, *PESEL*, *Telefon*)

Zespoły (*IdZespołu*, *NazwaZespołu*, *LiczbaOsóbWZespole*, *DataPowstania*)

Dodatkowy obligatoryjny atrybut *OsobaZespół* w schemacie relacji **Nagrody** może przyjmować np. wartości:

- znak „O” – nagrodzona została osoba. Dodatkowy obligatoryjny atrybut *#IdNagrodzonego* jest kluczem głównym ze schematu relacji **Osoby**,
- znak „Z” – nagrodzony został zespół. Dodatkowy obligatoryjny atrybut *#IdNagrodzonego* jest kluczem głównym ze schematu relacji **Zespoły**.

□

Przypadek II – dziedziny kluczy głównych encji wykluczających się wzajemnie są różne. Wówczas do schematu relacji powstałego z encji powiązanej, wstawiamy

tylko dodatkowych (opcjonalnych) atrybutów, ile encji było połączonych łukiem wykluczającym (odpowiadają one ich kluczom głównym). W każdej krotce utworzonej relacji tylko jeden z tych atrybutów może mieć wartość różną od NULL, a w przypadku związków obligatoryjnych – dokładnie jeden musi mieć wartość różną od NULL.

Przykład 4.2

Jeśli dla encji z przykładu 4.1, zachodzi $\text{dom}(IdOsoby) \neq \text{dom}(IdZespołu)$, np. wartościami atrybutu *IdOsoby* są liczby dodatnie a wartościami atrybutu *IdZespołu* są ciągi 6 znaków, to po transformacji do modelu logicznego otrzymamy następujące schematy relacji:

Nagrody (*IdNagrody*, *TytulNagrody*, #*IdOsoby*, #*IdZespołu*)

Osoby (*IdOsoby*, *Nazwisko*, *Imię*, *PESEL*, *Telefon*)

Zespoły (*IdZespołu*, *NazwaZespołu*, *LiczbaOsóbWZespole*, *DataPowstania*)

W każdej krotce relacji o schemacie **Nagrody** jeden z atrybutów #*IdOsoby* lub #*IdZespołu* musi mieć wartość różną od NULL (wartość klucza głównego z odpowiedniego schematu relacji) a drugi z nich musi mieć wówczas wartość NULL. Kontrola tego typu warunków musi być przeprowadzana przez aplikację bazodanową.

□

Na zakończenie przeglądu reguł transformacji modelu konceptualnego do modelu logicznego rozważmy jeszcze zagadnienie atrybutu wyliczanego. Na ogół wartość atrybutu wyliczanego nie jest przechowywana w bazie danych tylko wyliczana za każdym razem kiedy jest potrzeba. Jeśli jednak wartość ta jest wykorzystywana bardzo często i w zasadzie po wyliczeniu przez bardzo długi okres czasu nie ulega zmianie, to optymalniej byłoby ją co jakiś czas wyliczać i przechowywać jako wartość atrybutu. W takim przypadku należałoby do schematu relacji wprowadzić jeszcze dodatkowy atrybut wyliczany. A więc decyzja o tym, czy przechowywać wartość wyliczaną zależy od rodzaju danej wyliczanej, od częstości jej wykorzystywania (potrzeby wyliczania) i jak często ulega zmianie.

Jeśli np. dla encji OCENA(*IdWpisu*, *IdStudenta*, *IdPrzedmiotu*, *Ocena*, *Data*) na koniec roku jest wyliczana średnia ocen ze wszystkich przedmiotów dla każdego studenta, i średnia ta po zakończonym roku nie ulega zmianie natomiast jest wielokrotnie wykorzystywana np. przy zapisach na kursy, przy przyznawaniu nagród, stypendiów, do sporządzania rankingów itd., to można np. do schematu relacji **Studenci** jako dodatkowe atrybuty dać *ŚredniaOcen*, *DataObliczeniaŚredniej*. Wartością atrybutu *ŚredniaOcen* może być np. średnia ocen za ostatni rok lub dotychczas uzyskana średnia ocen.

Jako ćwiczenie pozostawiamy Czytelnikowi przedstawienie modelu konceptualnego i logicznego odzwierciedlającego reguły: klient wypożyczalni kaset video musi wypożyczyć co najmniej jedną kasetę, kasetą może być wypożyczona maksymalnie przez jednego klienta a po zwróceniu kasety dane o wypożyczeniu są usuwane.

4.1.2. DEFINICJE SCHEMATÓW RELACJI

Otrzymane według powyższych reguł transformacji schematy relacji pozwalają nam utworzyć reprezentacje relacji czyli tabele. Schematy relacji opisujemy np. następująco:

REL/00x Nazwa schematu relacji/NAZWA TYPU ENCJI
lub

REL/00x Nazwa schematu relacji/Nazwa związku, NAZWY TYPÓW ENCJI
gdzie *x* jest kolejnym numerem schematu relacji.

Jeśli schemat relacji powstał z encji, to po nazwie schematu relacji podajemy nazwę typu encji, z którego go tworzymy, np.

REL/001 Klienci/KLIENT

Jeśli schemat relacji tworzymy ze związku i typów encji, to po nazwie schematu relacji podajemy nazwę związku i nazwy typów encji uczestniczących w transformowanym związku, np.

REL/002 Wypożyczenia/Wypożyczza, KLIENT, KASETA

Po utworzeniu niezbędnych schematów relacji podajemy opisy, które można przedstawić w dowolnej czytelnej postaci, np. w postaci tabel, uwzględniając elementy istotne dla tworzonej bazy danych, np.

- Atrybuty relacji (nazwy kolumn tabeli)
- Dziedziny atrybutów
- Maskę do wprowadzania danych
- Obligatoryjność/opcjonalność atrybutu
- Wartość domyślną atrybutu
- Unikalność wartości atrybutu
- Klucz relacji
 - PK – klucz główny – ang. *primary key*
 - FK – klucz obcy – ang. *foreign key*
- Referencje – nazwy relacji, z których pochodzą klucze obce
- Źródło danych

Wszelkie używane w opisach oznaczenia powinny być wyjaśnione i konsekwentnie stosowane.

Po opisie schematu relacji powinniśmy podać opis poszczególnych atrybutów i, jeżeli zachodzi taka potrzeba, wymienić dodatkowe warunki (ograniczenia) nałożone na poszczególne atrybuty.

Następnie należy zamieścić **przykładowe dane**, jakie mogą wystąpić w relacji (tabeli) o danym schemacie, co znacznie ułatwia analizowanie struktury bazy danych. Dane przykładowe powinny być starannie dobrane tak, by ilustrowały sytuacje typowe, wyjątkowe, ograniczenia itp. Umieszczanie np. przykładowych danych osobowych,

w których wszystkie dane są zupełnie różne (czyli nie ma osób np. o takim samym nazwisku, nazwisku i imieniu, dacie urodzenia) nie jest właściwe, nie oddaje rzeczywistości ani możliwych do wystąpienia danych.

Opisy schematów relacji można ułożyć np. alfabetycznie według nazw, co ułatwi ich odszukiwanie.

Przykład 4.1 (opis schematu relacji w modelu logicznym)

REL/001 Klienci/KLIENT

Tabela 4.2

Opis schematu relacji **Klienci**

Nazwa atrybutu	Dziedzina	Maska	OBL + OPC -	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>NrKlienta</i>	Int+		+			+	PK		SZBD
<i>Nazwisko</i>	String[15]		+						USER
<i>Imię</i>	String[15]		+						USER
<i>DataUr</i>	Date	____-____-____	-	1900-01-01	> {1899-01-01}, < DATE()				USER
<i>NrMiasta</i>	Int+		+				FK	Miasta	BD

Znaczenie atrybutów w schemacie relacji **Klienci** przedstawia tabela 4.3.

Tabela 4.3

Znaczenie atrybutów w schemacie relacji **Klienci**

Nazwa atrybutu	Znaczenie
<i>NrKlienta</i>	Unikalny identyfikator klienta, automatycznie nadawany przez system, klucz główny
<i>Nazwisko</i>	Nazwisko klienta, dowolny ciąg znaków, dozwolone litery polskie
<i>Imię</i>	Pierwsze imię klienta, dowolny ciąg znaków, dozwolone litery polskie
<i>DataUr</i>	Data urodzenia klienta
<i>NrMiasta</i>	Identyfikator miasta, klucz obcy z tabeli Miasta

Przykładowe dane tabeli o schemacie relacji **Klienci** zawiera tabela 4.4.

Tabela 4.4

Przykładowe dane tabeli o schemacie relacji **Klienci**

<i>NrKlienta</i>	<i>Nazwisko</i>	<i>Imię</i>	<i>DataUr</i>	<i>NrMiasta</i>
1	Koperski	Jan	1980-12-21	1
2	Walicki	Łukasz	1979-11-11	1
3	Barycka	Beata	1977-03-18	2
4	Matacka	Maria		5
5	Barycka	Beata	1981-03-18	2
8	Koperski	Jan	1980-12-21	1

□

Należy pamiętać, że kolumny, które odpowiadają kluczom kandydującym encji powinny mieć zapewnione unikalne wartości w tabeli.

4.1.3. SCHEMAT RELACYJNEJ BAZY DANYCH

Schemat bazy danych (ang. *database schema*) to zbiór schematów relacji o różnych nazwach. Zgodnie z definicją podaną w rozdziale 2.1, **schemat relacji** (ang. *relation schema*) to nazwa schematu relacji oraz zbiór atrybutów relacji.

Schemat bazy danych jest konstrukcją logiczną opisującą strukturę danych w pojęciach abstrakcyjnych. Obejmuje opisy schematów relacji, reprezentowane w bazie danych jako tabele (o nazwach takich samych jak schematy relacji), które będą przechowywały dane (i które będą ewentualnie zaimplementowane w przyszłym systemie bazy danych). Według definicji [Subieta1999] schemat bazy danych, to „*obraz zawartości bazy danych, wyrażony w sformalizowanym języku. Schemat bazy danych jest niezbędny dla jej użytkowników i programistów do zrozumienia, co baza danych zawiera i jak dane są zorganizowane.*”

Postać ogólna schematu bazy danych jest następująca:

NAZWA BAZY DANYCH

Nazwa_schematu_relacji_1(lista_atrybutów_schematu_relacji_1)

Nazwa_schematu_relacji_2(lista_atrybutów_schematu_relacji_2)

....

Schemat bazy danych jest więc zbiorem schematów relacji (w praktyce będących najczęściej w trzeciej postaci normalnej). Relacyjna baza danych to taka baza, którą użytkownik postrzega jako zbiór tabel (relacji) o określonej strukturze (schematach).

W schematach relacji podkreślamy atrybuty, które są kluczami głównymi a przed atrybutami, które są kluczami obcymi, wpisujemy symbol #.

Opis schematu bazy danych (metadane) jest przechowywany w tej samej bazie danych co dane właściwe.

Przykład 4.2

Po transformacji modelu konceptualnego bazy danych z rysunku 3.5 (rozdział 3.6.4) do modelu logicznego (zgodnie z regułami podanymi w rozdziale 4.1.1), otrzymalibyśmy następujący schemat relacyjnej bazy danych WYPOŻYCZALNIA_2:

WYPOŻYCZALNIA_2

Klienci (*NrKlienta*, *Nazwisko*, *Imię*, *Miasto*, *UlicaNr*)

Gatunki (*KodGatunku*, *Rodzaj*)

Filmy (*KodFilmu*, *Tytuł*, *CzasTrwania*, #*KodGatunku*)

Kasety (*NrKasety*, *Opis*, #*KodFilmu*)

Wypożyczenia (*NrWypożyczenia*, #*NrKlienta*, #*NrKasety*,
DataWypożyczenia, *DataZwrotu*, *Oplata*)

Jako ćwiczenie pozostawiamy Czytelnikowi sprawdzenie, w której postaci normalnej są powyższe schematy relacji.

□

Relacyjna baza danych jest zbiorem relacji o podanych schematach. Relacje są reprezentowane przez tabele. W danej chwili baza danych składa się z określonego zbioru tabel i ich zawartości, czyli jest w określonym stanie. Stan bazy danych można zmienić przez wykonanie transakcji np. aktualizację, dopisanie czy usunięcie danych. Baza danych w każdym stanie powinna spełniać wszystkie reguły integralności (biznesowe) czyli powinna być poprawna w sensie zgodności z rzeczywistością. Wykonywane transakcje powinny zachowywać stan integralności (ang. *state of integrity*) bazy danych.

4.1.4. SŁOWNIK ATRYBUTÓW

Aby ułatwić wyszukiwanie informacji o wybranym atrybucie sporządzamy *słownik atrybutów*. Słownikiem (ang. *dictionary*) ogólnie nazywa się miejsce przechowywania terminów i ich objaśnień.

Słownik atrybutów lub inaczej słownik danych (ang. *data dictionary*) tworzy się w celu zestawienia nazw kolumn i domen (dziedzin), określających typ danych, jakie

mogą być przechowywane w danej kolumnie oraz ograniczenia, którym podlegają. Jest to pewnego rodzaju baza danych projektanta bazy danych.

Słownik danych można przedstawić np. w postaci 3-kolumnowej tabeli, w której wyszczególniamy nazwy atrybutów relacji reprezentujących odpowiednie schematy relacji (kolumn poszczególnych tabel), dziedziny atrybutów oraz przynależność atrybutu do schematu relacji. Nazwy atrybutów (kolumn) powinny być uporządkowane alfabetycznie. Na bazie tego samego typu można zdefiniować wiele domen poprzez określenie ich cech indywidualnych takich jak np. maksymalna szerokość kolumny, liczba miejsc po przecinku, wzorzec (maska) wprowadzania danych, wartości domyślne, ograniczenie zakresu dozwolonych wartości, czy dopuszczalne są wartości NULL itp. Można określić np. domenę *Nazwisko*, która będzie oznaczała dziedzinę dla atrybutu *NazwiskoKlienta* jako łańcuch co najwyżej 25 znakowy, zawierający litery, myślnik, apostrof, spację.

Przykład 4.3 (słownik atrybutów)

Tabela 4.5

Słownik atrybutów

Nazwa atrybutu	Dziedzina	Przynależność do schematu relacji
<i>Cena</i>	Logical	Towary
<i>DataSprzedaży</i>	Date	Faktury
<i>DataUr</i>	Date	Klienci
<i>IdMiasta</i>	Int+	Miasta
<i>IdTowaru</i>	Int+	Towary DetaleFaktur
<i>Ilość</i>	Int	DetaleFaktur
<i>Jednostka</i>	String[10]	DetaleFaktur
<i>Imię</i>	String[15]	Klienci
<i>NazwaMiasta</i>	String[15]	Miasta
<i>NazwaTowaru</i>	String[25]	Towary
<i>NazwiskoKlienta</i>	Nazwisko	Klienci
<i>NrKlienta</i>	Int+	Klienci Faktury
<i>NrMiasta</i>	Int+	Klienci
<i>Płatność</i>	String[10]	Faktury
<i>Uwagi</i>	Memo	Faktury



4.2. DEFINIOWANIE PERSPEKTYW

Ostatnim etapem projektowania bazy danych jest określenie **perspektyw** dla jej przyszłych użytkowników. W tym celu należy najpierw wyszczególnić grupy użytkowników bazy danych oraz sprecyzować, które dane, w których transakcjach będą przez nich wykorzystywane oraz jakie operacje na tych danych będą mogli wykonywać. Następnie w sposób systematyczny należy wyodrębnić perspektywy opisać.

Perspektywa (ang. *view*) jest to nazwana tabela (wirtualna), która w odróżnieniu od tabeli bazowej (wyjściowej) nie może istnieć samodzielnie i definiowana jest za pomocą jednej lub wielu tabel wyjściowych (bazowych lub perspektyw), ich wszystkich lub niektórych atrybutów [Date2000]. W szerszym znaczeniu jest to pojęciowy obraz danych, który jest przypisany do pojedynczego użytkownika lub do grupy użytkowników i abstrahuje od danych dla nich nieistotnych. Tak rozumiana perspektywa dostosowuje sposób przedstawienia danych do przyzwyczajęń użytkowników (potrzebujących różnych danych) zapewniając jednocześnie wygodę i przyjazność w przetwarzaniu danych. Niezwykle ważne jest zapewnienie przezroczystości (ang. *transparency*), aby użytkownik miał wrażenie, że operuje na danych zapamiętanych a nie wirtualnych [Subieta1999].

W uproszczeniu perspektywę można traktować jak okno (formularz), w którym będą wyświetlone odpowiednie kolumny z jednej lub kilku tabel, i w którym możemy wykonywać odpowiednie operacje na tych danych (przeglądać je, dopisywać, aktualizować, usuwać itd.).

Perspektywa daje możliwość zapewnienia bezpieczeństwa danym poprzez ograniczanie dostępu do danych (np. poprzez nadawanie uprawnień dostępu i wykonywania operacji na określonych tabelach lub wybranych kolumnach tabel). Perspektywy są mocnym narzędziem służącym do ograniczania dostępu do danych.

Największym problemem jest aktualizacja perspektyw i w relacyjnych systemach zarządzania bazami danych na ogół możliwa jest jedynie aktualizacja perspektyw będących wynikiem operacji rzutu lub selekcji tablic bazowych (z zachowaniem klucza głównego).

W etapie definiowania perspektyw należy więc:

- wyszczególnić grupy użytkowników bazy danych,
- wyszczególnić perspektywy dla każdej grupy użytkowników,
- wyszczególnić transakcje dla każdej perspektywy;
- określić dane uczestniczące w transakcji; dane te można opisać podając nazwę relacji i atrybut, gdzie dana jest przechowywana oraz dopuszczalne operacje wykonywane na tej danej. Opisy można przedstawić np. w formie tabeli o następującym nagłówku:

(Nazwa relacji, Atrybut, Zapis, Odczyt, Modyfikacja)

zaznaczając symbolem + w odpowiedniej kolumnie dla odpowiedniego atrybutu możliwość wykonania danej operacji (zapis, odczyt, modyfikacja).

Przykład 4.4

Załóżmy, że użytkownikiem bazy danych jest Administrator, który może dopisywać, edytować i wyświetlać dane pracowników. Opis perspektywy dotyczącej obsługi danych pracownika przez administratora może mieć postać:

PER/001 Dane pracownika

Użytkownik: Administrator

Transakcje: TRA/001, TRA/002, TRA/005

TRA/001 – Wprowadzenie danych pracownika

Tabela 4.6

Tabela transakcji: Wprowadzenie danych pracownika

Nazwa relacji	Atrybut	Zapis	Odczyt	Modyfikacja
Osoby	<i>Id</i>	+		
Osoby	<i>Nazwisko</i>	+		
Osoby	<i>Imię</i>	+		
Osoby	<i>DataUrodzenia</i>	+		
Dzieci	<i>ImięDziecka</i>	+		
Dzieci	<i>DataUrDziecka</i>	+		

TRA/002 – Edycja danych pracownika

Tabela 4.7

Tabela transakcji: Edycja danych pracownika

Nazwa relacji	Atrybut	Zapis	Odczyt	Modyfikacja
Osoby	<i>Id</i>		+	
Osoby	<i>Nazwisko</i>		+	+
Osoby	<i>Imię</i>		+	+
Osoby	<i>DataUrodzenia</i>		+	+
Dzieci	<i>ImięDziecka</i>	+	+	+
Dzieci	<i>DataUrDziecka</i>	+	+	+

Znak "+" w kolumnie **Zapis** powinien występować tylko dla atrybutów opcjonalnych, które mogą mieć nadawane wartości dopiero na etapie edycji danych. Atrybuty obligatoryjne muszą mieć przypisane wartości przy wprowadzaniu danych, podczas edycji mogą być ewentualnie modyfikowane.

TRA/005 – Wyświetlenie danych pracownika

Tabela 4.8

Tabela transakcji: Wyświetlenie danych pracownika

Nazwa relacji	Atrybut	Zapis	Odczyt	Modyfikacja
Osoby	<i>Nazwisko</i>		+	
Osoby	<i>Imię</i>		+	
Osoby	<i>DataUrodzenia</i>		+	
Dzieci	<i>ImięDziecka</i>		+	
Dzieci	<i>DataUrDziecka</i>		+	



5. ZAKOŃCZENIE

W pracy został przedstawiony pełny cykl tworzenia projektu relacyjnej bazy danych. Aby wykonać poprawny projekt bazy danych należy poprawnie wykonać poszczególne etapy, dokładnie je dokumentując. Projekt możemy uznać za poprawny, jeśli jego opis jest zgodny z przyjętą notacją, która musi być na początku projektu przedstawiona i omówiona. Poprawność projektu nie oznacza i nie zapewnia, że system, który na bazie tego projektu zbudujemy, będzie spełniał oczekiwania użytkownika.

Poprawny projekt powinien być [Jaszk1997]:

- **kompletny** – wszystkie składowe projektu są zdefiniowane w sposób odpowiedni i wyczerpujący,
- **niesprzeczny** – każdy element jest zdefiniowany jednoznacznie, nie ma definicji sprzecznych ze sobą,
- **spójny** – informacje zawarte w kolejnych etapach projektu (definicjach, diagramach) są semantycznie zgodne,
- **zgodny z regułami składniowymi notacji** – opis projektu zależy od przyjętej notacji, z którą musi być zgodny.

Aby osiągnąć projekt wysokiej jakości, poszczególne fazy projektowania bazy danych należy nieraz wykonywać wielokrotnie, poprawiać opisy, uzupełniać je i uszczegóławiać. Opracowaną dokumentację należy analizować wielokrotnie zwracając uwagę na cztery wymienionej wyżej cechy (kompletność, niesprzeczność, spójność i zgodność). Aby zapewnić poprawność modelu danych w sensie zgodności często podaje się *metamodel danych* czyli model modelu. Metamodel obejmuje pojęcia i reguły dotyczące pojęć wykorzystywanych w modelu, na przykład metamodel może zawierać następującą regułę: jeśli model zawiera co najmniej dwie encje, to każda encja musi być co najmniej w jednym związku z dowolną inną encją.

Model relacyjny dotyczy zagadnień logicznych czyli struktury danych (tabel), operowania danymi (podstawowe operatory algebry relacji to operatory selekcji, rzutu i złączenia) oraz zapewnienia integralności danych (poprzez odpowiednie klucze główne i obce).

Do projektowania baz danych można wykorzystać narzędzia CASE, które automatyzują i przyspieszają proces projektowania [FugStąTr1995], a których opis wykracza poza przyjęte ramy niniejszej pracy.

W pracy zostało opisane modelowanie danych, przedstawione są kolejne etapy projektowania bazy danych opartego głównie na identyfikacji encji i związków. Do tworzenia schematu bazy danych można podejść również w nieco inny sposób –

metodą normalizacji, która usuwa anomalie występujące w schematach relacji przez odpowiednie operacje projekcji. Na podstawie analizy wycinka rzeczywistości można określić jeden globalny (początkowy, wyjściowy) schemat relacji $R(A_1, A_2, \dots, A_n)$, w którym uwzględnia się wszystkie atrybuty występujące w bazie danych. Baza danych traktowana jest więc jako jedna (być może bardzo duża) relacja. W związku z tym wszystkie nazwy atrybutów muszą być jednoznaczne. Równoległe z wyodrębnianiem atrybutów należy określić zależności funkcyjne między atrybutami. W kolejnych krokach normalizujemy początkowy schemat relacji uzyskując docelowy (końcowy) schemat bazy danych, który jest zbiorem schematów relacji, i schemat ten powinien być lepszy od początkowego schematu relacji. Jako kryterium oceny schematów baz danych przyjmuje się stopień normalizacji wynikowych schematów relacji oraz liczbę schematów relacji. W praktyce wymaga się, aby schematy relacji były co najmniej w trzeciej postaci normalnej a ich liczba była jak najmniejsza. Istnieje kilka metod projektowania schematu bazy danych, np. algorytm dekompozycji, Bernsteina, Niekludowej-Calenki, Rissanena, których opisy można znaleźć np. w [Pankow1992]. Schemat bazy danych otrzymany w wyniku normalizacji powinien być w zasadzie taki sam jak schemat bazy otrzymanej z transformacji diagramu związków encji do modelu logicznego. Wynika to stąd, że wyodrębnianie encji i związków odpowiada intuicyjnie normalizacji schematów relacji. Sprowadzanie związków wieloznacznych do binarnych, ograniczanie się do wartości prostych jako atrybutów, prowadzi do zaprojektowania znormalizowanej bazy danych. W praktyce częściej używa się modelowania danych niż normalizacji a normalizację wykorzystuje się zazwyczaj do sprawdzenia poprawności otrzymanych schematów relacji. C.J. Date uważa, że większa część procesu projektowania bazy danych jest bardziej sztuką niż procesem naukowym [WhMa2002]. W tym bardzo subiektywnym procesie, normalizacja stanowi element naukowy.

Dysponując poprawnym projektem bazy danych można przystąpić do opracowania projektu aplikacji bazodanowej a następnie do jej implementacji systemu informatycznego [Banach1998, CasPal2000, Górski2000, Mazur1997, Riordan2000, Roman2001, W&W1997].

6. ZESTAWIENIE ETAPÓW PROJEKTOWANIA RELACYJNEJ BAZY DANYCH

Model konceptualny

- Etap 1.** Zdefiniowanie celu i założeń wstępnych.
- Etap 2.** Analiza rzeczywistości (wywiad z ekspertem dziedzinowym).
- Etap 3.** Analiza istniejącej bazy danych (wady, zalety).
- Etap 4.** Zdefiniowanie kategorii.
- Etap 5.** Wyodrębnienie reguł funkcjonowania
- Etap 6.** Wyodrębnienie ograniczeń dziedzinowych.
- Etap 7.** Zdefiniowanie transakcji.
- Etap 8.** Identyfikacja encji.
- Etap 9.** Identyfikacja związków.
- Etap 10.** Definicje predykatowe encji i związków.
- Etap 11.** Diagram związków encji.

Model logiczny

- Etap 12.** Transformacja modelu konceptualnego do modelu logicznego
- Etap 13.** Zdefiniowanie schematów relacji.
- Etap 14.** Utworzenie słownika atrybutów.
- Etap 15.** Opracowanie schematu bazy danych.
- Etap 16.** Określenie użytkowników i zdefiniowanie perspektyw.

7. LITERATURA

- [Banach1998] Banachowski Lech, *Bazy danych. Tworzenie aplikacji*, Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1998.
- [Barker1996] Barker Richard, *CASE*Method, Modelowanie związków encji*, WNT, Warszawa, 1996.
- [Beynon1999] Beynon-Davies Paul, *Inżynieria systemów informacyjnych*, WNT, Warszawa, 1999.
- [Beynon2000] Beynon-Davies Paul, *Systemy baz danych*, WNT, Warszawa, 2000
- [Boratyn1995] Boratyn Dariusz, *MS ACCESS 2.0*, Croma&Msoft&DK, Wrocław, 1995.
- [CasPal2000] Cassel Paul, Palmer Pamela, *Access 2000 PL dla każdego*, Helion, Gliwice, 2000.
- [Cell1988] Cellary Wojciech, Królikowski Zbyszko, *Wprowadzenie do projektowania baz danych, dBase III*, WNT, Warszawa, 1988.
- [Chen1976] Chen Peter Pin-Shan, *The Entity-Relationship Model – Toward a Unified View of Data*, ACM TODS 1, No. 1, March 1976. Przedruk w M. Stonebraker (ed.): *Reading In Database Systems*, San Mateo, Calif.: Morgan Kaufmann (1988).
- [Codd1970] Codd E. F., *A Relational Model of Data for Large Shared Data Banks*, Comm. ACM, vol. 13, No. 6, June 1970.
- [Codd1972] Codd E. F., *Relational Completeness of Data Base Sublanguages*, w: *Data Base Systems, Courant Computer Science Symposia Series 6*. Englewood Cliffs, N.J.: Prentice-Hall, 1972.
- [ConBeg2004] Connolly Thomas, Begg Carolyn, *Systemy baz danych: Praktyczne metody projektowania, implementacji i zarządzania*. Tom 1, RM, Warszawa, 2004.
- [Date1981] Date C. J., *Wprowadzenie do baz danych*, WNT, Warszawa, 1981.
- [DateDar2000] Date C. J., Darwen Hugh, *SQL. Omówienie standardu języka*, WNT, Warszawa, 2000.
- [Date2000] Date C. J., *Wprowadzenie do systemów baz danych*, WNT, Warszawa, 2000.
- [DelAdi1989] Delobel Claude, Adiba Michel, *Relacyjne bazy danych*, WNT, Warszawa, 1989.
- [FugStąTr1995] Fuglewicz Piotr, Stąpor Katarzyna, Trojnar Andrzej, *CASE dla ludzi*, LUPUS, Warszawa, 1995.
- [Górski2000] Praca zbiorowa pod redakcją Janusz Górskiego, *Inżynieria oprogramowania w projekcie informatycznym*, MIKOM, Warszawa, 2000.

- [Harrin2001] Harrington Jan L., *Obiektowe bazy danych dla każdego*, MIKOM, Warszawa, 2001.
- [Harris1994] Harris Wayne, *Bazy danych nie tylko dla ludzi biznesu*, WNT, Warszawa, 1994.
- [Hernan1998] Hernandez Michael J. *Bazy danych dla zwykłych śmiertelników*, MIKOM, Warszawa, 1998.
- [Jaszk1997] Jaskiewicz Andrzej, *Inżynieria oprogramowania*, Helion, Gliwice, 1997.
- [Mazur1997] Mazur Hanna, Mazur Zygmunt, *Programowanie w dBASE IV*, PSZI, Wrocław, 1997.
- [Muller2000] Muller Robert J., *Bazy danych. Język UML w modelowaniu danych*, MIKOM, Warszawa, 2000.
- [MurRyb1993] Muraszkiewicz Mieczysław, Rybiński Henryk, *Bazy danych*, Akademicka Oficyna Wydawnicza, Warszawa, 1993.
- [OleńStan1984] Oleński Józef, Staniszkis Witold, *Projektowanie bazy danych*, seria Informatyka w praktyce, Państwowe Wydawnictwo Ekonomiczne, Warszawa, 1984.
- [Olle1982] Olle T.W., Sol H.G., Verrijn-Stuart A.A. (ed), *Information Systems Design Methodologies. A Comparative Review*, Amsterdam, Netherlands: North-Holland/New York, NY: Elsevier Science (1982).
- [Pankow1992] Pankowski Tadeusz, *Podstawy baz danych*, PWN, Warszawa, 1992.
- [Riordan2000] Riordan Rebecca M., *Projektowanie systemów relacyjnych baz danych*, RM, Warszawa, 2000.
- [Rodgers1995] Rodgers Ulka, *ORACLE. Przewodnik projektanta baz danych*, WNT, Warszawa, 1995.
- [Roman2001] Roman Steven, *Access. Baza Danych – Projektowanie i programowanie*, Helion, Gliwice, 2001.
- [Roszko1998] Roszkowski Jerzy, *Analiza i projektowanie strukturalne*, Helion, Gliwice, 1998.
- [Subieta1998] Subieta Kazimierz, *Obiektowość w projektowaniu i bazach danych*, Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1998.
- [Subieta1999] Subieta Kazimierz, *Słownik terminów z zakresu obiektowości*, Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1999.
- [TsiLoch1990] Tschritzis Dionysios C., Lochovsky Frederick H., *Modele danych*, WNT, Warszawa, 1990.
- [UllWid2000] Ullman Jeffrey D., Widom Jennifer, *Podstawowy wykład z systemów baz danych*, WNT, Warszawa, 2000.
- [W&W1997] Strużińska-Walczak Anna, Walczak Krzysztof, *Delphi. Nauka programowania systemów baz danych*, Wydawnictwo W&W, Warszawa, 1997.

- [WhMa2002] Whitehorn Mark, Marklyn Bill, *Relacyjne bazy danych*, Helion, Gliwice, 2002.
- [Yourdon1996] Yourdon Edward, *Współczesna analiza strukturalna*, WNT, Warszawa, 1996.

DODATEK

Niniejsza praca omawia projektowanie bazy danych. Termin „baza danych” jest więc podstawowym pojęciem spotykanym zarówno w tej książce jak i w literaturze związanej z tematyką baz danych. Coraz częściej słyszymy o istniejących lub tworzonych bazach danych w różnych dziedzinach życia, np. baza danych adresowych, telefonicznych, kont bankowych, szkół i uczelni, bezrobotnych, dla Zakładu Ubezpieczeń Społecznych itd. Intuicyjnie każdy rozumie co to jest baza danych lecz w literaturze brak jest jednoznacznej definicji pojęcia bazy danych. Poniżej przytaczamy kilka określeń bazy danych zaczerpniętych z prac dostępnych na polskim rynku. Cytaty przytoczono w kolejności chronologicznej lat, biorąc pod uwagę rok wydania książki, z której pochodzi cytat.

Cytat 1 [OleńStan1984, s. 11]

„Baza danych jest zbiorem zapamiętanych, przechowywanych i aktualizowanych danych, dostępnych dla wielu użytkowników (systemów użytkowych).”

Cytat 2 [Cell1988, s. 7]

„Przez *bazę danych* rozumiemy uporządkowany zbiór danych przechowywanych w pamięci komputera, a przez *system bazy danych* – bazę danych wraz ze środkami programowymi umożliwiającymi operowanie na niej, w szczególności wyszukiwanie i aktualizowanie zawartych w niej informacji.”

Cytat 3 [DelAdi1989, s. 16]

„Bazą danych nazywamy zbiór danych o określonej strukturze, zapisany na zewnętrznym nośniku pamięciowym komputera, mogący zaspokoić potrzeby wielu użytkowników korzystających z niego w sposób selektywny w dogodnym dla siebie czasie.

W bazie danych są zapisywane fakty i zdarzenia zachodzące w pewnym wy-cinku rzeczywistości, po to, by możliwe było ich odtworzenie i analiza w dowolnej chwili.”

Cytat 4 [TsiLoch1990, s. 23]

„W konkretnym zastosowaniu modelu danych nazwy kategorii wraz z ich cechami nazywamy *schematem*. (...)

Kolekcja danych umieszczonych w określony sposób w strukturach odpowiadająca schematowi jest powszechnie nazywana *bazą danych*.”

Cytat 5 [Pankow1992, s. 17]

„Aktywna rola systemu informatycznego, w systemie informacyjno-decyzyjnym wymaga aby:

- był w nim odwzorowany pewien zakres wiedzy o środowisku (miejsce tego odwzorowania nazywamy *bazą danych*, BD, systemu informatycznego);
- zapewnione były środki umożliwiające utrzymanie, aktualizowanie i udostępnianie odwzorowanej wiedzy (oprogramowanie realizujące te funkcje nazywamy systemem zarządzania bazą danych, SZBD).

Zagadnienia odwzorowania wiedzy są rozległą dziedziną badań naukowych i wykraczają poza zakres zagadnień poruszanych w tej książce. Zagadnienia te są przedmiotem badań sztucznej inteligencji, teorii języków i in. My ograniczymy się tylko do takiego zakresu wiedzy, który da się odwzorować w tzw. *sformatowanych bazach danych*, a więc do takiej wiedzy, dla przekazania której można wcześniej ustalić pewien skończony zbiór formatów (wzorców). W dalszym ciągu mówiąc o bazach danych będziemy mieli na myśli sformatowane bazy danych.”

Cytat 6 [MurRyb1993, s. 6]

„Bazy danych, którymi zajmujemy się w tej książce, są komputerowymi reprezentacjami fragmentów istniejącego (niekoniecznie fizycznie) świata rzeczywistego, który jest przedmiotem zainteresowania użytkowników baz.”

Cytat 7 [Barker1996, s. 225]

„**Baza danych** – dowolny zbiór tabel lub plików, będący pod kontrolą systemu zarządzania bazą danych.”

Cytat 8 [Banach1998, s. 5]

„Informacje (dane) są takim samym zasobem firmy jak każdy inny (np. pracownicy, materiały, urządzenia) i wymagającym, tak samo jak one, zarządzania. Informacja może mieć wartość dla firmy tylko wtedy, gdy jest dokładna i dostępna wtedy kiedy trzeba. *Baza danych* stała się standardową metodą strukturalizacji zarządzania informacją w większości organizacji.

Baza danych dotyczy zawsze pewnego fragmentu rzeczywistości związanego z firmą, organizacją lub innym działającym systemem. Stanowi kolekcję danych, których zadaniem jest reprezentowanie tego fragmentu rzeczywistości. Baza danych jest na ogół częścią systemu informacyjnego, obsługującego zapotrzebowania informacyjne pewnego fragmentu rzeczywistości. Bardziej technicznymi terminami używanymi w zastępstwie terminu system informacyjny w celu podkreślenia jego implementacyjnego charakteru są:

- *aplikacja bazy danych*, gdy chcemy zwrócić szczególną uwagę na charakter programistyczny i szczególne miejsce danych w całej aplikacji;
- *system informatyczny*, gdy chcemy zwrócić uwagę na ogólne środki informatyczne a więc także używany sprzęt.

Z pojęciem bazy danych są nierozzerwalnie związane dwie cechy:

1. *trwałość* – dane mają być przechowywane przez pewien okres czasu, na ogół nieokreślony z góry, w odróżnieniu od danych chwilowych – istniejących tylko w momencie działania programu;
2. *zgodność z rzeczywistością* – dane w bazie danych muszą stanowić wierne odzwierciedlenie modelowanego fragmentu rzeczywistości, baza danych też musi się odpowiedni zmieniać. Zapewnienie zgodności z rzeczywistością jest podstawowym procesem we współczesnych systemach informacyjnych korzystających z baz danych.”

Cytat 9 [Subieta1999, s. 23]

„**Baza danych** to zestaw danych, metadanych (katalogów), programów i innych środków pozwalających na utrzymywanie, zabezpieczanie, przetwarzanie i udostępnianie danych dla użytkowników. Bazy danych są podtrzymywane przez systemy zarządzania bazą danych (SZBD). Mogą one być oparte o pewien model danych (np. hierarchiczny, sieciowy, relacyjny, funkcjonalny, obiektowy) lub mogą stosować rozwiązanie własne, nie mieszczące się w ramach zdefiniowanych modeli. Często bazą danych określa się także pewien system plików, pewien zestaw stron WWW, repozytorium dokumentów (np. pod LotusNotes) oraz inne środki przechowywania i udostępniania danych.”

Cytat 10 [Beynon2000, s. 25]

„**Bazę danych** możemy przyrównać do kartoteki z aktami, a ściślej do zbioru kartotek. Baza danych jest magazynem danych z nałożoną na niego wewnętrzną strukturą. Ogólnym celem takiego magazynu jest przechowywanie danych związanych z pewnym zbiorem zadań organizacyjnych. Zwykle takie zadania dotyczą administracji. Celem większości systemów baz danych jest przechowywanie danych, potrzebnych do wykonywania codziennej działalności organizacji. Dlatego na uniwersytecie baza danych jest potrzebna przy takiej czynności, jak rejestrowanie liczby studentów zapisanych na poszczególne moduły.

Struktura powoduje zwykle pewien logiczny podział, zazwyczaj hierarchiczny. Dlatego mówimy o bazie danych jak o zbiorze plików. Zbiór plików zawierających informacje na temat studentów na uniwersytecie stanowiłby bazę danych. Każdy plik w bazie danych jest również strukturalnym zbiorem danych. Odwołując się do porównania, takie pliki byłyby teczkami trzymanymi w kartotece. Każdy plik w kartotece składa się ze zbioru rekordów. Mogą to być karty z informacjami, na przykład, o każdym studencie. Każdy rekord jest z kolei podzielony na zbiór obszarów nazywanymi polami, np. NrStudenta, NazwiskoStudenta, DataUrodzenia itd. W każdym polu jest wpisana konkretna wartość.

Pytanie o strukturę systemu bazy danych jest więc sprawą największej wagi. Z bazą danych są także powiązane inne właściwości: współdzielenie danych, integracja danych, integralność danych, bezpieczeństwo danych, abstrakcja danych i niezależność danych.”

Cytat 11 [Date2000, s. 31]

„**Baza danych** jest to zbiór danych trwałych, które są wykorzystywane przez system aplikacji danej organizacji (lub przedsiębiorstwa).”

Cytat 12 [Górski2000, s. 252]

„Zbiór danych zorganizowany zgodnie z określonym *modelem danych* nazywamy *bazą danych*. Jeśli jeden system DBMS zarządza wieloma takimi zbiorami, mówimy o *systemie baz danych*”.

Cytat 13 [UllWid2000, s. 19]

„Baza danych nie jest w istocie rzeczą niczym więcej niż zbiorem danych istniejącym przez długi czas, często przez wiele lat. W potocznym rozumieniu termin *baza danych* odnosi się do zbioru danych zorganizowanego przez *system zarządzania bazą danych*, który nazywa się również skrótowo DBMS (*database management system*) lub po prostu systemem baz danych.”

Cytat 14 [Harrin2001, s. 253]

„**Baza danych** – kolekcja obiektów zarządzanych w taki sposób, że mogą być dostępne dla wielu użytkowników”.

Cytat 15 [ConBeg2004, s. 14]

„**Baza danych** – Dostępny dla wielu użytkowników zbiór powiązanych logicznie danych wraz z definicją ich struktury, zaprojektowany dla zaspokojenia potrzeb przetwarzania danych przez instytucję”.

W niniejszej pracy bazę danych zdefiniowano jako trwały zbiór tematycznie ze sobą powiązanych danych (rozdz. 2).



Książka w sposób systematyczny przedstawia projektowanie relacyjnych baz danych. Jako projektowania ma bardzo istotny wpływ na baz danych, a nie aplikacji. Umiejętność właściwego zaprojektowania bazy danych jest równie ważna jak umiejętność programowania.

Autorzy omawiają podstawowe pojęcia relacyjnych baz danych, algebrę relacji i proces normalizacji schematów relacji. Opracowanie dotyczy strukturalnej metodyki projektowania baz danych (jednej z wielu możliwych) z elementami niektórych strukturalnych metodologii projektowych. Stanowi wprowadzenie do zaawansowanego projektowania strukturalnego relacyjnych baz danych narzędziami programowymi CASE, wspomagającymi analizę i projektowanie systemów informatycznych wysokiej jakości. Zaprezentowana metodyka ma charakter ręcznego projektowania metodą tworzenia dokumentów projektowych.

Projektowanie relacyjnych baz danych to publikacja przeznaczona dla osób zapoznających się z tajnikami projektowania baz danych (dla początkujących projektantów). Może też być wykorzystana jako podręcznik dydaktyczny do zajęć związanych z tematyką projektowania relacyjnych baz danych.

Wydawnictwa Politechniki Wrocławskiej
są do nabycia w następujących księgarniach:
„Politechnika”

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław
budynek A-1 PWr., tel. (0-71) 320-25-34;
„Tech”

plac Grunwaldzki 13, 50-377 Wrocław
budynek D-1 PWr., tel. (0-71) 320-32-52
Prowadzimy sprzedaż wysyłkową