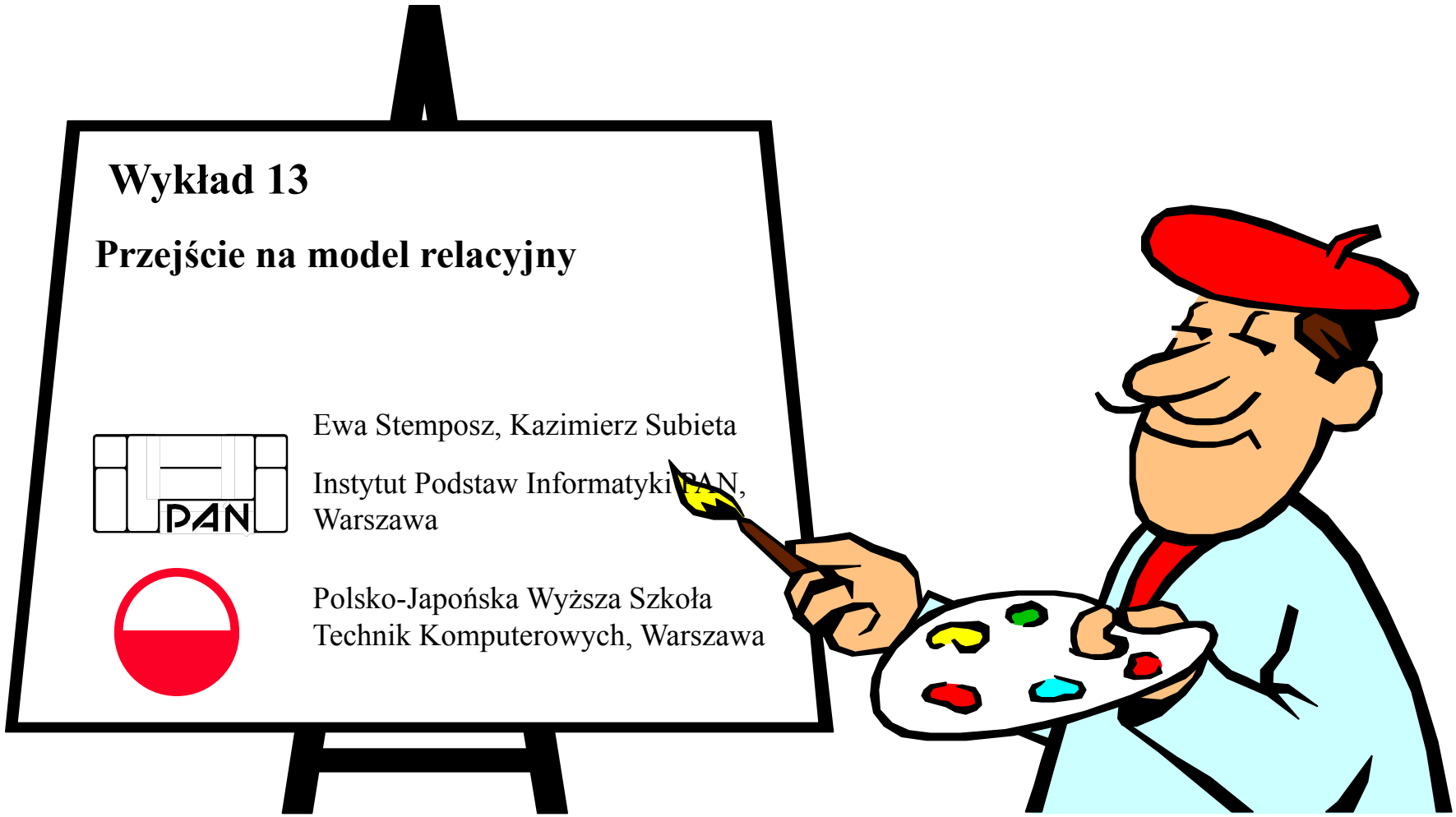
Projektowanie systemów informacyjnych

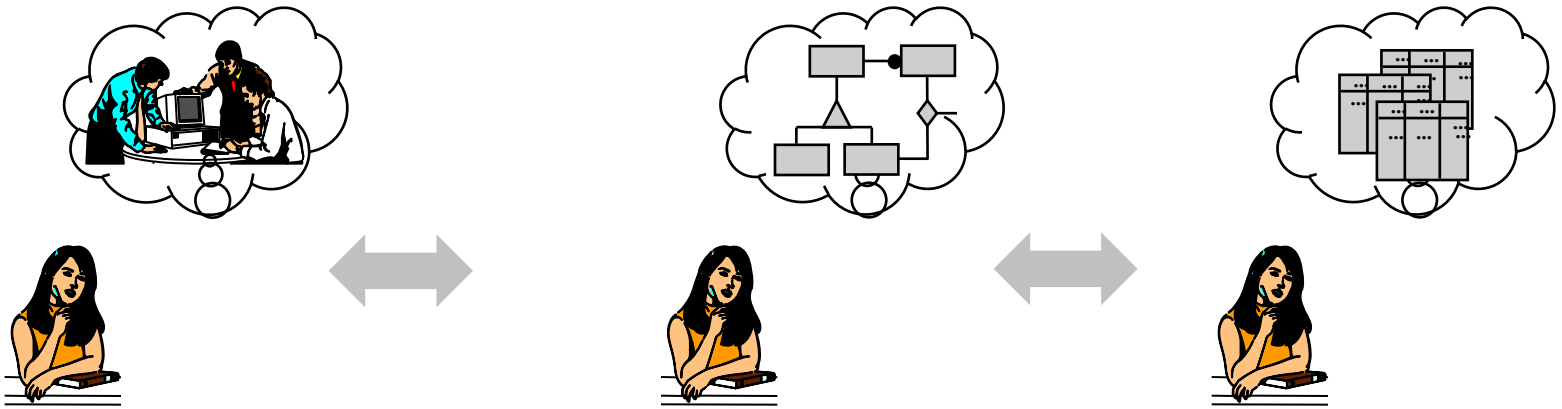


Zagadnienia

Podejście obiektowe kontra relacyjne
Garby modelu relacyjnego
Projektowanie logiczne
Odwzorowanie atrybutów powtarzalnych
Odwzorowanie związków asocjacji
Odwzorowanie złożonych obiektów
Odwzorowanie metod
Obejście braku dziedziczenia
Normalizacja
Analiza wartości zerowych
Analiza wartości długich
Klucze

Dlaczego obiektowość?

Chodzi o uzyskanie jak najmniejszej luki pomiędzy myśleniem o rzeczywistości (percepcją świata), a myśleniem o danych i procesach, które na danych zachodzą.



Percepcja świata | **Model pojęciowy** | **Relacyjny model struktur danych**

W modelu relacyjnym odwzorowanie percepcji świata jest ograniczone środkami implementacyjnymi. W rezultacie, schemat relacyjny gubi część semantyki danych. Model obiektowy podtrzymuje te zgodności, przybliżając semantykę danych do świata rzeczywistego.

Długofalową tendencją w rozwoju SZBD jest uzyskanie zgodności pomiędzy tymi modelami.

Obiektowość kontra model relacyjny

- Model relacyjny przegrał konfrontację z obiektowością w strefie intelektualnej; trwał w tej strefie tylko 10 -15 lat.
- Teorie matematyczne związane z modelem relacyjnym są nieadekwatne do praktyki. Zalety wynikające z matematyzacji dziedziny baz danych okazały się iluzją (nie pierwszą tego typu w informatyce).
- SQL ma zalety, ale jest językiem tworzonym ad hoc, niesystematycznym, nieregularnym, nieortogonalnym, bez istotnego podkładu teoretycznego. Standard SQL3 jest ogromny, eklektyczny, z dość przypadkowymi pomysłami (podobnie SQL1999).
- Powstało szereg systemów relacyjnych, dojrzałych technicznie i użytecznych, ale posiadających zasadnicze odstępstwa od założeń modelu relacyjnego.
- Twórcy systemów relacyjnych wzmacniają ich interfejsy o pojęcia obiektowe oraz umożliwiają obiektowe perspektywy nad relacyjnymi strukturami danych.
- Nie istnieje użytkowa własność systemów relacyjnych, która nie mogłaby być zrealizowana w systemach obiektowych.

Garby modelu relacyjnego (1)

- Z góry **ustalony konstruktor typu danych** (relacja), rozszerzany ***ad hoc*** przez wytwórców systemów relacyjnych.

- **Brak możliwości rozszerzania typów**, ignorowanie zasad bezpieczeństwa typologicznego.

- **Brak złożonych obiektów**. Informacje o pojęciach wyróżnialnych i manipulowalnych w rzeczywistości są rozproszone w krotkach wielu tabel. Skojarzenie tych informacji następuje w zapytaniach SQL, przez co wzrasta złożoność zapytań oraz czas ich wykonania. Optymalizacja zapytań nie zawsze jest skuteczna.

- **Brak** wyspecjalizowanych **środków do realizacji powiązań pomiędzy danymi**.

- **Brak środków do przechowywania danych proceduralnych**.

- Wszelkie **informacje wykraczające poza strukturę relacyjną** (perspektywy, procedury bazy danych, BLOBy, aktywne reguły,...) są **implementowane *ad hoc***.

- **Brak środków do hermetyzacji i modularyzacji**: wykroczenie przeciwko zasadzie abstrakcji, tzn. oddzielenia implementacji od specyfikacji).

Garby modelu relacyjnego (2)

- **Brak uniwersalnych środków do manipulowania danymi**, co powoduje konieczność zanurzenia w języki programowania niższego poziomu (niezgodność impedancji).

Ubogie możliwości modelu relacyjnego powodują znaczne zwiększenie długości kodu aplikacji. Połączenie SQL z językiem programowania wymaga również dodatkowego kodu (szacuje się na 30%). Łącznie aplikacja (w porównaniu do systemów obiektowych) może zawierać nawet 70% nadmiarowego kodu.

Zdania SQL "wkodowane" do aplikacji obiektowej i operujące bezpośrednio na nazwach relacji i atrybutów są w wielu przypadkach niekorzystne, gdyż zmniejszają możliwości ponownego użycia oraz zmiany schematu. Lepszym rozwiązaniem jest dynamiczny SQL, który odwołuje się do informacji znajdującej się w katalogach. (Jest on jednak nieco wolniejszy.)

- **Niespójne mechanizmy wartości zerowych, brak wariantów**.

- **Złączenia (*joins*) są wolne**. Mimo sprawnych metod, takich jak ***hash join***, ***sort&merge join***, ***optymalizacji zapytań***, itd, złączenia powodują poważny narzut na wydajność. Należy ich unikać, np. poprzez denormalizację.

Czy model relacyjny był pomyłką?

Poglądy są podzielone. Na korzyść tej tezy przemawia fakt, że podstawowym założeniem było wykorzystanie matematycznych własności relacji. Od strony systemów komercyjnych, korzyści z matematyki są iluzją. Po co więc ograniczenia struktur danych i interfejsów, rzekomo "wymuszone" przez matematykę?

"*Relational databases set the commercial data processing industry back at least ten years*." (Dr. Henry G. Baker, Comm. ACM 35/4, 1992)

Jest to oczywiście twierdzenie niesprawdzalne. Nie wiadomo jak potoczyłaby się dziedzina baz danych, gdyby nie model relacyjny.

Podstawowym wkładem modelu relacyjnego była nie matematyka, a **założenie o logicznej niezależności danych**: uwolnienie programisty od myślenia na niskim poziomie, w kategoriach fizycznej organizacji danych. Jakkolwiek to założenie pojawiło się w czasach przed modelem relacyjnym, dopiero systemy relacyjne uczyniły go powszechnie obowiązującym faktem.

Tak czy inaczej, pozostaje rzeczywistość, której szybko zmienić się nie da ...

Rzeczywistość (1)

- Wiele aplikacji potrzebuje tylko warstwy trwałych danych, która w istocie jest ukryta przed użytkownikiem. Użytkownik dokonuje operacji na danych poprzez pewien z góry ustalony interfejs, który całkowicie izoluje go od struktury BD.
- Dla 90% rzeczywistych projektów systemy relacyjne są wystarczające. To powoduje zredukowanie zainteresowania systemami czysto obiektowymi.
- Łączne światowe inwestycje (komercyjne, akademickie, organizacyjne) w systemy relacyjnych baz danych są szacowane na ponad 100 miliardów dolarów. Jest mało prawdopodobne, że te inwestycje będą w krótkim czasie powtórzone dla modeli i systemów obiektowych baz danych. Nie oznacza to, że nie mają one szans; raczej, że ich rozwój, osiągnięcie dojrzałości i popularności będzie trwać dłużej niż przypuszcza wielu fanów obiektowości. Chyba, że nastąpi skok jakościowy...
- Nadzieje są związane z systemami obiektowo-relacyjnymi, które wzbogacają systemy relacyjne o pewne cechy obiektowości. Jest to podejście ewolucyjne. Pytanie, czy kiedyś zredukują złożoność odwzorowania modelu pojęciowego na model implementacyjny, pozostaje jednak otwarte.
- Niskie nakłady na pielęgnację (***maintenance***) oprogramowania jest podstawowym wymaganiem biznesu. Model obiektowy umożliwia zmniejszenie tych nakładów. Przejście na model relacyjny powoduje zwiększenie kosztów pielęgnacji kodu.

Rzeczywistość (2)

- Sprzymierzeńcem wszystkich wdrożonych technologii baz danych, w tym relacyjnych, jest ogromna bezwładność rynku zastosowań, który niechętnie zmienia swoje preferencje ze względu na zainwestowane duże pieniądze i czas.

- Klient baz danych nie tylko nie lubi kosztownych zmian; musi mieć także pewność, że nie pozostanie sam w swojej dziedzinie działalności lub rejonie geograficznym i może liczyć na zarówno środowisko specjalistów, jak i ogólną kulturę techniczną wytworzoną w związku z daną technologią.

- Systemy relacyjne opanowały dużą grupę „nisz ekologicznych" i można przyjąć jako pewnik, że pozostaną w nich przez kilka, kilkanaście, lub nawet kilkadziesiąt lat. Systemy obiektowe muszą poszukiwać innych nisz, które nie są zagospodarowane przez wcześniejsze technologie.

- Natomiast w dziedzinie projektowania baz danych odwrót od modelu relacyjnego nastąpił bardzo szybko (Chen, 1976, model encja-związek). **Obecnie nie istnieje metodyka projektowania nie oparta w jakiś sposób o pojęcia obiektowe.**

Schemat pojęciowy a systemy relacyjne

System relacyjny jako *back-end*, tj. baza implementacyjna. Na czubku systemu relacyjnego budowany jest ***front-end***, tj. zestaw interfejsów do zarządzania złożonymi obiektami, klasami, dziedziczeniem, itd. Podejście mające sporo opracowań oraz zaimplementowany co najmniej jeden prototyp (Starburst).

Wady: Podejście wymaga budowy nowego systemu; narzuty relacyjnego *back-end* na czasy wykonania mogą być istotne i trudne do wyeliminowania.

Obiektowe perspektywy nad strukturą relacyjną − możliwość istniejąca jak dotąd raczej w strefie akademickiej z kilku powodów: aktualizacja perspektyw, wydajność,...

Odwzorowanie schematu obiektowego na struktury relacyjne. Podejście tradycyjne (znane z modelu encja-związek).

Wady: niemożność odwzorowania wszystkich detali schematu obiektowego, zniekształcenie semantyki danych, konieczność wprowadzania sztucznych cech do schematu (niektórych atrybutów, itd.).

Projektowanie logiczne

Termin oznaczający **odwzorowanie modelu pojęciowego** (np. encja-związek lub obiektowego) **na model lub wyrażenia języka opisu danych konkretnego SZBD (np. relacyjnego)**.

Podstawowe problemy przy przechodzeniu na schemat logiczny:

- każda tabela musi być wyposażona w unikalny klucz,
- nie ma możliwości przechowywania wielu wartości jednego atrybutu,
- powiązania muszą być zaimplementowane jako tabele/relacje z kluczami obcymi,
- nie można zagnieżdżać danych,
- występują ograniczenia na rozmiar krotek, wartości elementarne i typy danych,
- brak dziedziczenia,
- brak wariantów (natomiast są wartości zerowe).

Dodatkowo należy uwzględnić:

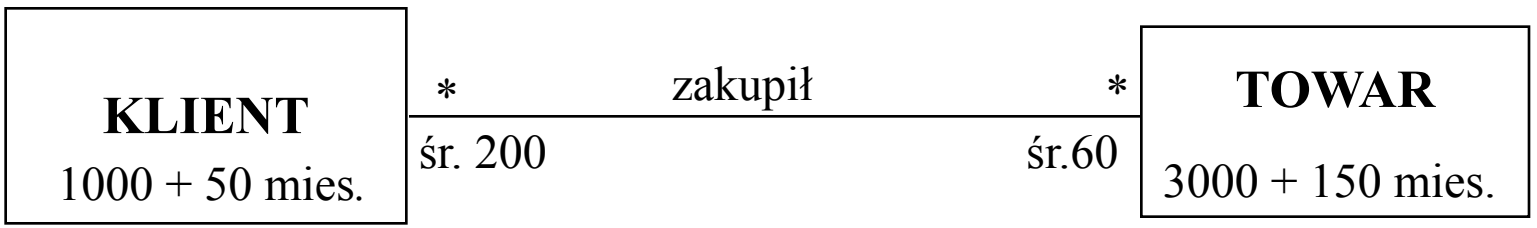
- cechy ilościowe (charakterystyka ilościowa danych i procesów),
- unikanie redundancji w danych (normalizacja 2NF, 3NF, BCNF),
- wymagania użytkowe: czas odpowiedzi, utylizacja pamięci (denormalizacja).

Przejście na schemat logiczny nie może być całkowicie automatyczne.

Charakterystyka ilościowa danych

INFORMACJE OPISUJĄCE DANE:

- **ZAJĘTOŚĆ PAMIĘCI** (liczba wystąpień danych),
- **ZMIENNOŚĆ** (spodziewany przyrost w czasie).



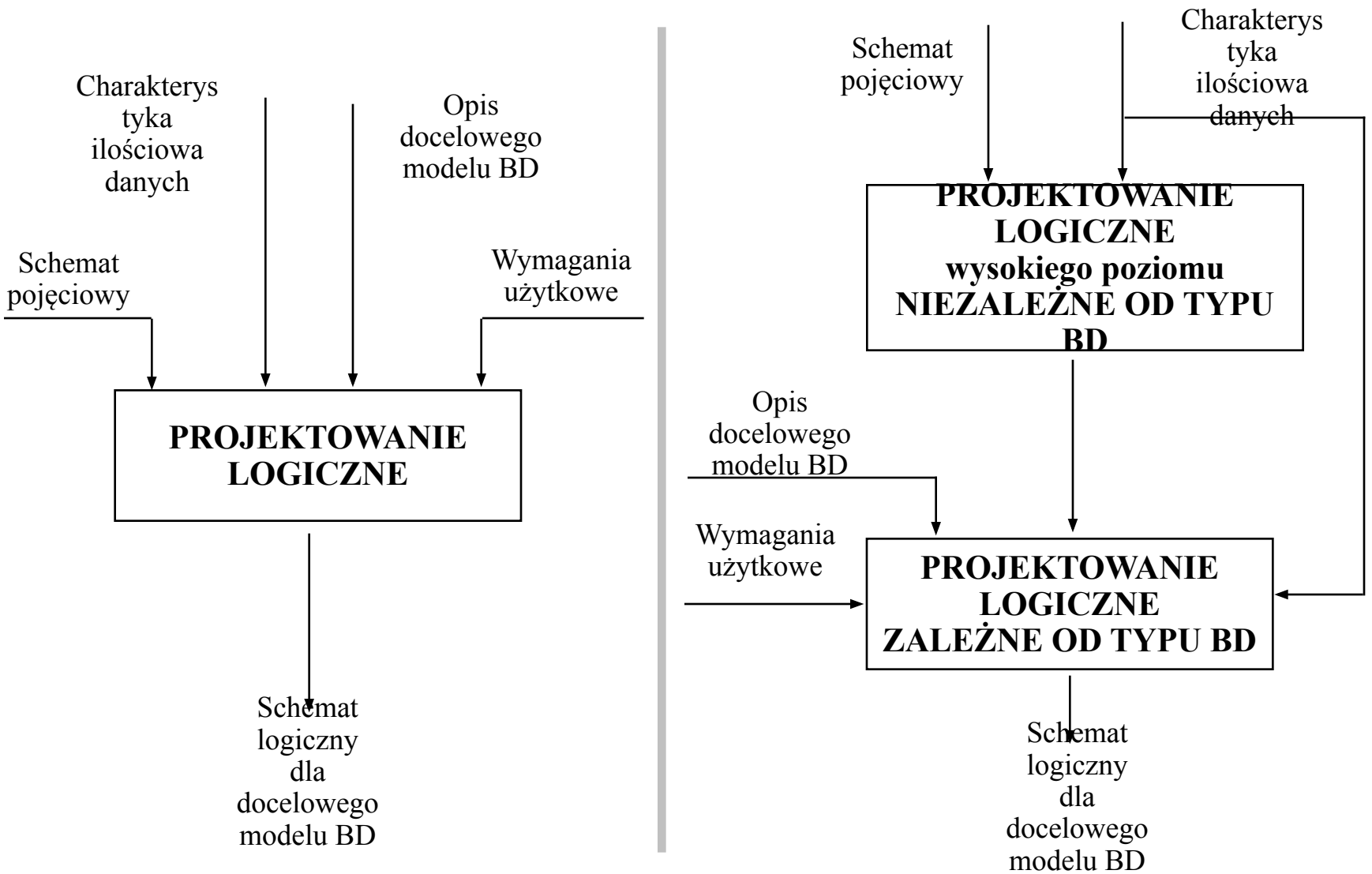
Charakterystyki ilościowe pozwalają określić fizyczne własności struktur danych. Istnieje sporo zaleceń i analiz pozwalających wykorzystać te własności.

Charakterystyka ilościowa procesów

INFORMACJE OPISUJĄCE PROCESY:

- **OPERACJE ELEMENTARNE (FUNKCJE UŻYTKOWE),**
- **TYP** (on-line, batch),
- **CZĘSTOTLIWOŚĆ ZACHODZENIA**
  (ew. dodatkowo rozkład w czasie),
- **FORMA** (ręczna, automatyczna),
- **SPOSÓB WYZWALANIA** (warunki - zdarzenia - wyzwalacze),
- **DOSTĘP DO ELEMENTÓW MODELU DANYCH .**

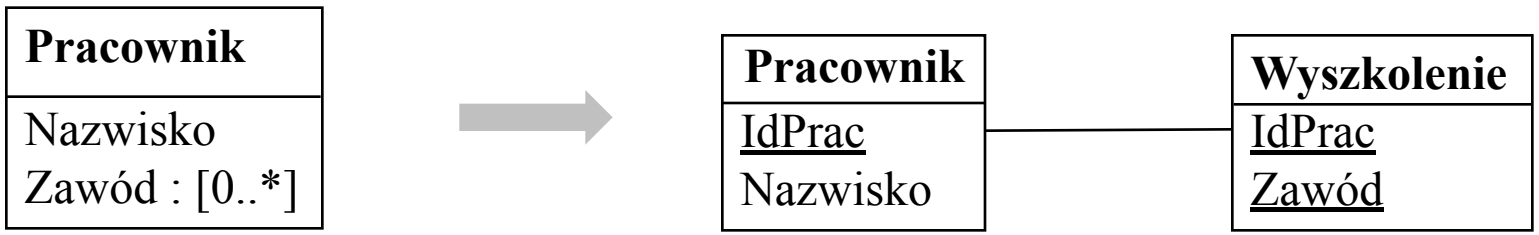
Proces projektowania logicznego



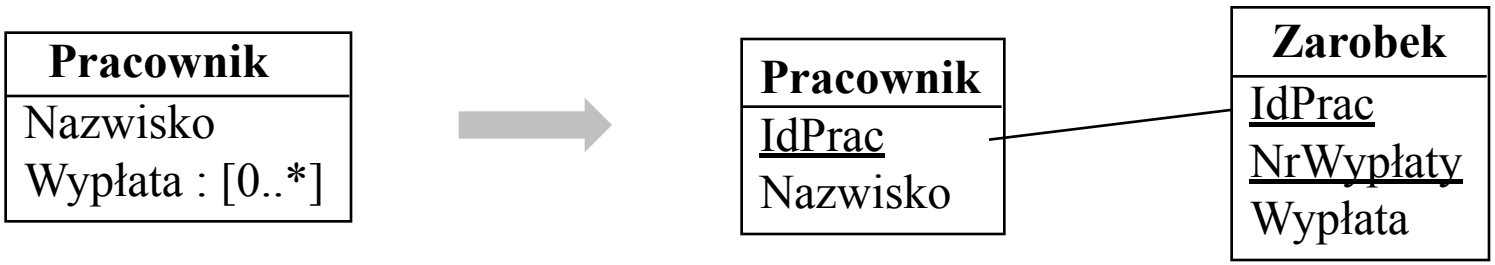
Odwzorowanie atrybutów powtarzalnych

Tabele relacyjne nie mogą przechowywać wielokrotnych wartości atrybutów. Model obiektowy (np. w UML) umożliwia zadeklarowanie takich atrybutów. Jest regułą, że takie atrybuty należy odwzorować jako odrębne tabele. Pojawią się także nowe atrybuty.

Pierwsza sytuacja: wartości powtarzalne nie mogą być identyczne**.**



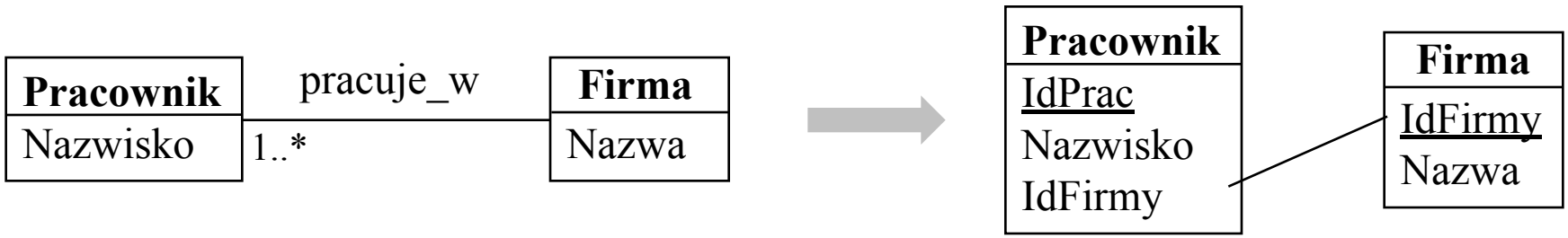
Druga sytuacja: wartości powtarzalne mogą być identyczne**.** Ten przypadek umożliwia również odwzorowanie sytuacji, gdy **porządek wielokrotnych wartości jest istotny**, poprzez wybór dodatkowego klucza (takiego jak NrWypłaty), który ustali ten porządek.



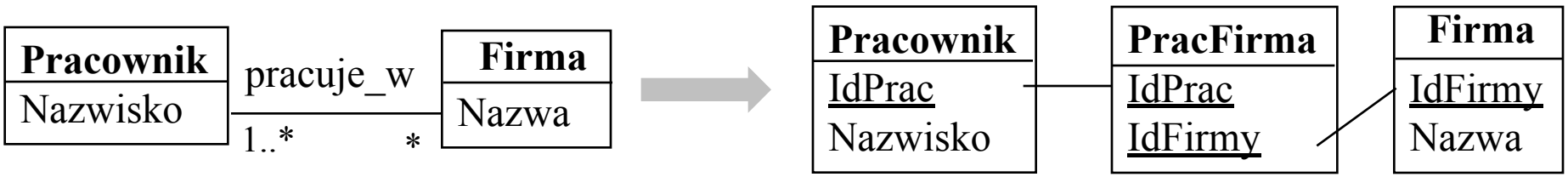
Odwzorowanie asocjacji

Dla liczności 1:1 można zaimplementować jako jedną tablelę.

Dla liczności 1:n można zaimplementować jako dwie tabele: atrybuty związku na ogół powodują konieczność zastosowania następnej metody.



Dla liczności m:n należy zaimplementować jako trzy tabele.



Odwzorowanie złożonych obiektów

- Podstawowa metoda odwzorowania obiektów złożonych polega **na spłaszczaniu ich struktury** (poprzez zamianę atrybutów złożonych na proste), usunięciu powtarzalnych atrybutów oraz różnych formach normalizacji (2NF, 3NF, 4NF, BCNF).

- Po tych zabiegach złożony obiekt jest reprezentowany jako zestaw krotek, często w wielu tabelach.

- Informacja o złożonym obiekcie jest utrzymywana w strukturze relacyjnej w postaci tzw. **integralności referencyjnej**. Polega ona na tym, że dla każdej wartości klucza obcego musi istnieć krotka posiadająca taką samą wartość klucza głównego. Nie wszystkie systemy relacyjne podtrzymują systemowo integralność referencyjną.

- Integralność referencyjna nie jest w stanie odwzorować całej semantyki złożonych obiektów. Np. zgubiona jest informacja, **co jest "korzeniem" obiektu, zgubione są reguły hermetyzacji obiektu, zgubiona jest semantyka niektórych operacji na obiektach (np. semantyka usuwania).** Istnieją propozycje wprowadzenia dodatkowej informacji do tabel, umożliwiającej przechowanie pełnej semantyki złożonych obiektów.

Odwzorowanie metod/operacji

Model relacyjny nie przewiduje specjalnych środków.

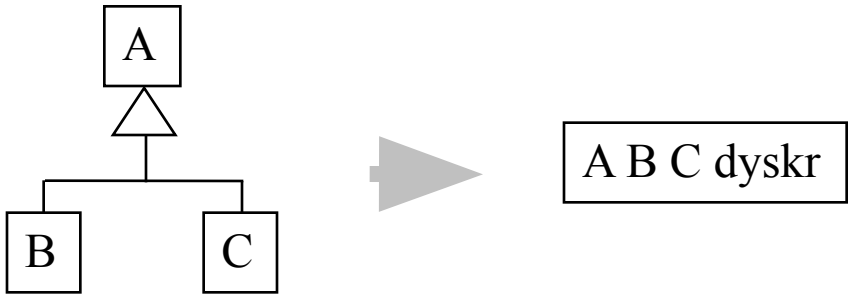
- Najczęściej są one odwzorowane na poziomie programów aplikacyjnych jako funkcje napisane w proceduralnych lub obiektowych językach programowania i dedykowane do obsługi pewnej tabeli/tabel w relacyjnej bazie danych.

- Niekiedy w systemach relacyjnych mogą być odwzorowane w postaci procedur baz danych (w SQL) lub w postaci aktywnych reguł.

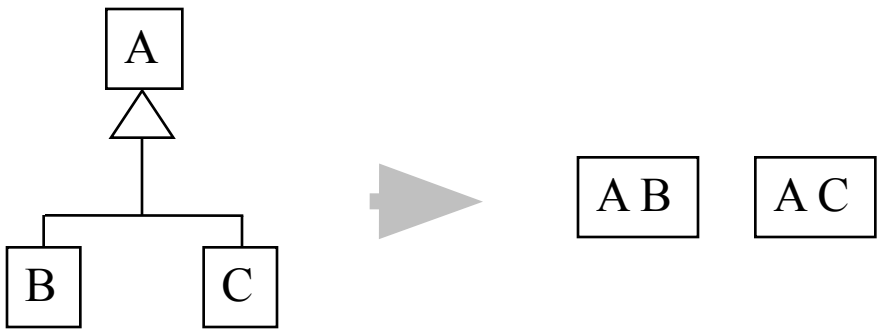
- Odwzorowanie polimorfizmu, przesłaniania, dynamicznego wiązania i hermetyzacji jest w zasadzie niemożliwe. Programista może napisać procedurę, która w środku ma przełączenie explicite na różne przypadki specjalizacji klas.

Trzy metody obejścia braku dziedziczenia

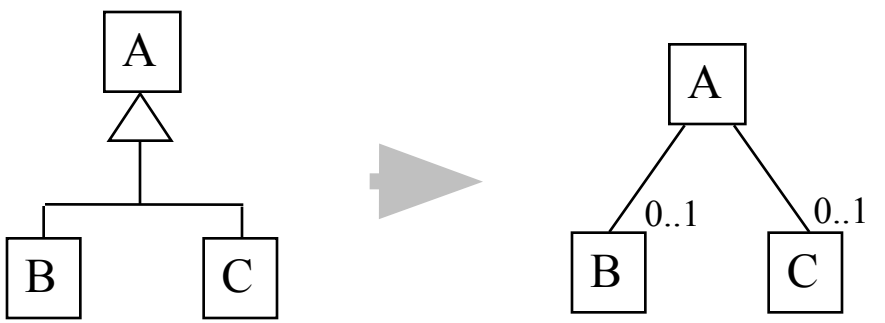
1 **Użycie jednej tabeli dla całego drzewa klas** poprzez zsumowanie wszystkich występujących atrybutów i powiązań w tym drzewie oraz dodanie dodatkowego atrybutu – dyskryminatora wariantu.



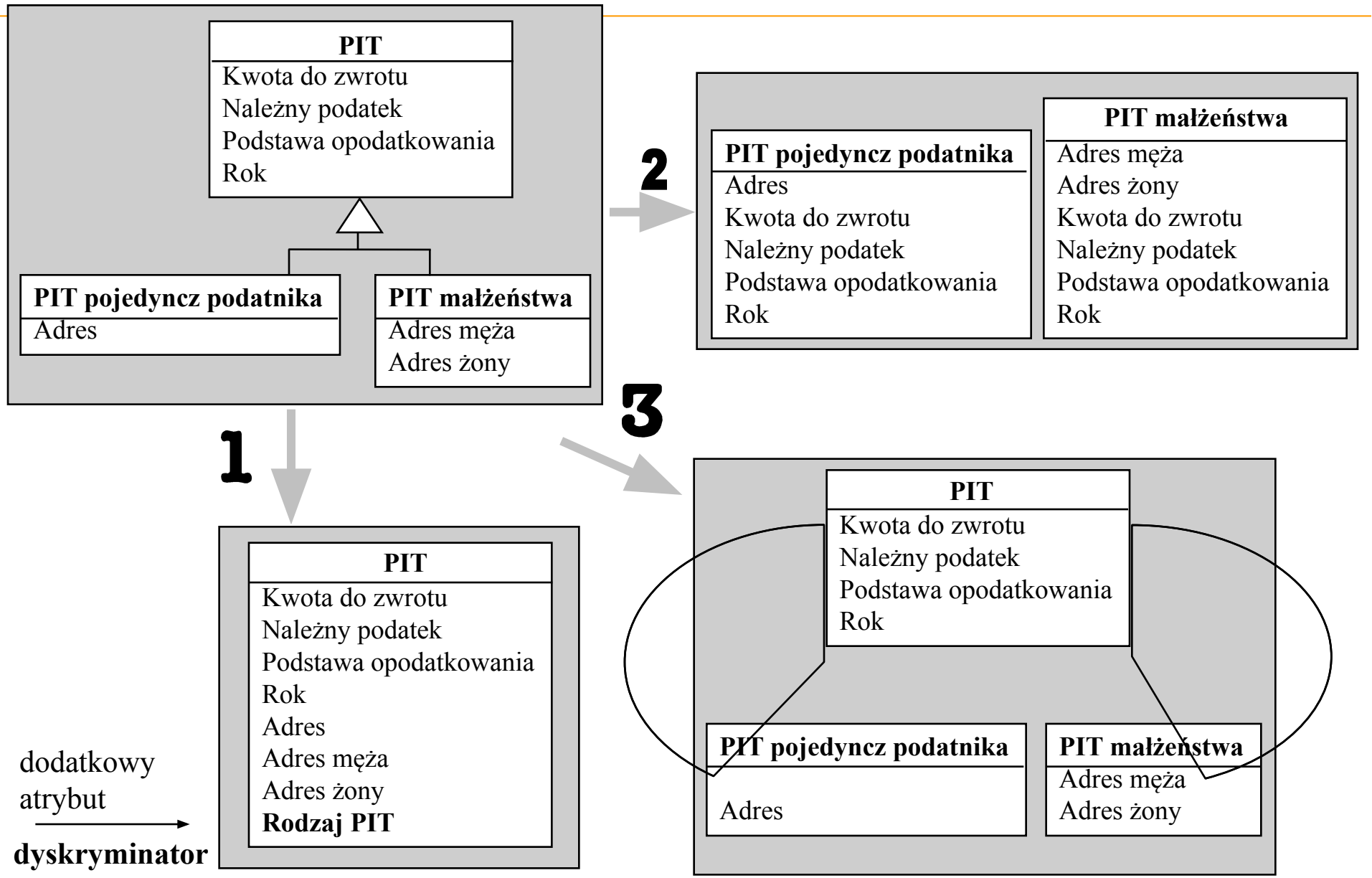
2 **Użycie oddzielnych tabel dla każdej podklasy**. Usunięcie nadklasy i przesunięcie jej atrybutów/powiązań do podklas.



3 **Użycie tabel dla każdej klasy.** Zamiana dziedziczenia na powiązania łączące nadklasę ze wszystkimi podklasami.



Przykład obejścia braku dziedziczenia



Zalety i wady każdej z trzech metod

Cecha	Jedna tabela dla hierarchii	Tabela dla każdej podklasy	Tabela dla każdej klasy
Łatwość implementacji	Prosta	Średnia	Trudna
Łatwość dostępu do danych	Prosta	Prosta	Średnia
Łatwość przypisania OID	Prosta	Średnia	Średnia
Związanie informacji	Bardzo duże	Duże	Małe
Dostęp	Bardzo szybki	Szybki	Wolny
Wspomaganie dla polimorfizmu	Słabe	Średnie	Duże
Konsumpcja pamięci	Duża	Mała	Mała

Prowadzenie słownika danych

Prowadzenie słownika jest konieczne dla przechowania informacji o sposobie odwzorowania modelu obiektowego na strukturę relacyjną.

Co powinien zawierać wiersz takiego słownika?

- nazwę odwzorowywanej klasy,
- nazwę odwzorowanego atrybutu tej klasy,
- nazwę kolumny, w którą taki atrybut jest odwzorowany,
- nazwę tabeli, która zawiera tę kolumnę.

Niekiedy potrzebna jest także następująca informacja:

- nazwę bazy danych, w którą odwzorowany jest dany fragment modelu,
- wskazanie, czy dany atrybut jest używany jako klucz główny,
- wskazanie, czy dany atrybut jest używany jako klucz obcy.

Normalizacja – 2NF, 3NF, BCNF,...

Polega na zdekomponowaniu tabeli na dwie lub więcej celem uniknięcia niekorzystnych własności: redundancji w danych oraz związanych z redundancją anomalii aktualizacyjnych.

Zależność funkcyjna pomiędzy atrybutami: wartość atrybutu **A2** zależy od wartości atrybutu **A1**, jeżeli dla każdego wyobrażalnego stanu bazy danych, dla każdej wartości atrybutu **A2** można przyporządkować dokładnie jedną wartość atrybutu **A1** taką, że **A2** = f(**A1**)

Druga forma normalna (2NF): nie ma atrybutów, które zależą funkcyjnie od części klucza.

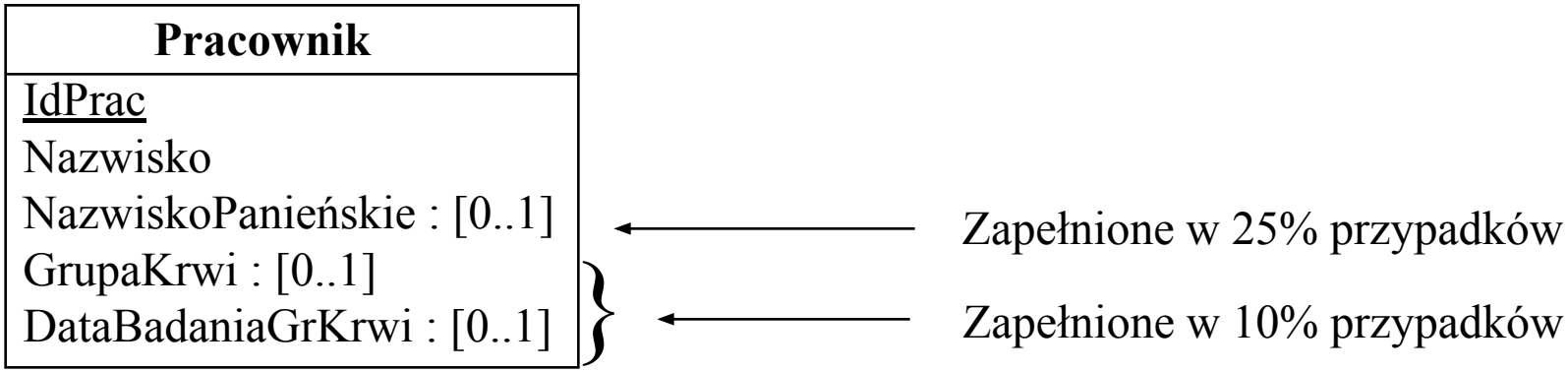
Trzecia forma normalna (3NF): nie ma zależności funkcyjnych tranzytywnych, tj. nie ma różnych atrybutów **A1**, **A2**, **A3** takich, że **A3** = f(**A2**) i **A2** = f(**A1**).

BCNF – każdy determinant (argument funkcji) jest kluczem kandydującym. Mocniejszy warunek od **3NF**, nie zawsze realizowalny.

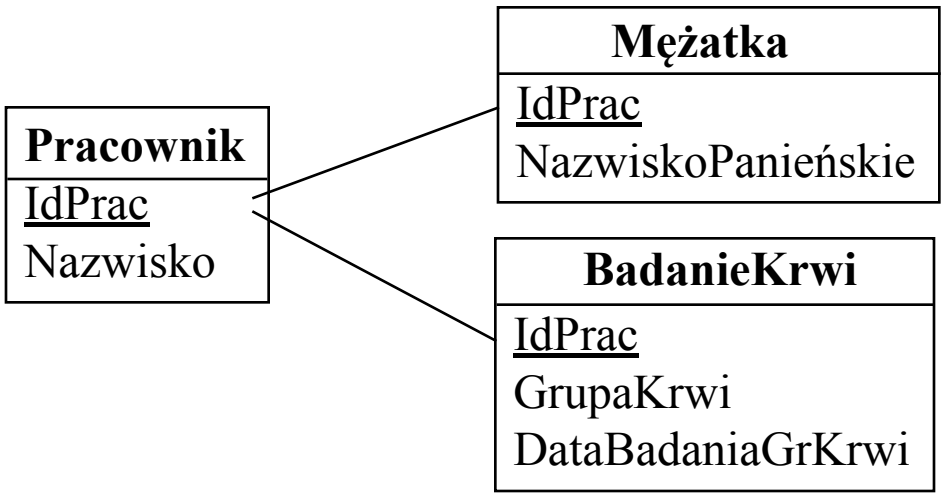
- Metodyki oparte na modelu encja-związek i metodyki obiektowe w naturalny sposób prowadzą do znormalizowanych schematów.
- Podejście top-down oraz tendencja do dekompozycji/separowania pojęć również w naturalny sposób prowadzą do znormalizowanych schematów.
- Czynniki inne niż zależności funkcyjne mogą okazać się bardziej istotne (wydajność --> denormalizacja).

Analiza wartości zerowych

Analiza ta, podobnie do zależności funkcyjnych, może nam przynieść informację o konieczności zdekomponowania danej tabeli na dwie lub więcej tabel.



To rozwiązanie implikuje, że ok. połowy BD będzie zapełnione wartościami zerowymi.



Brak wartości zerowych, objętość danych zmniejszyła się.

Wydajność może być gorsza ze względu na złączenia.

Analiza długich/złożonych wartości

Podobnie do wartości zerowych i zależności funkcyjnych, analiza długich wartości może być również podstawą do zdekomponowania tabeli na dwie lub więcej. Te same metody mogą dotyczyć złożonych atrybutów.

Pracownik
IdPrac : char[5]
Nazwisko : char[20]
ZwiązekZawodowy: char[100]

Załóżmy, że w zakładzie pracy działa 10 związków zawodowych, zaś pracowników jest 1000. Łatwo policzyć, że rozmiar tabeli będzie równy 125000 znaków. Dodatkowo, występuje możliwość popełniania błędów w pisowni nazwy związku.

Pracownik
IdPrac: char[5]
Nazwisko: char[20]
IdZZ: char[5]

ZwiązekZawodowy
IdZZ: char[5]
Nazwa: char[100]

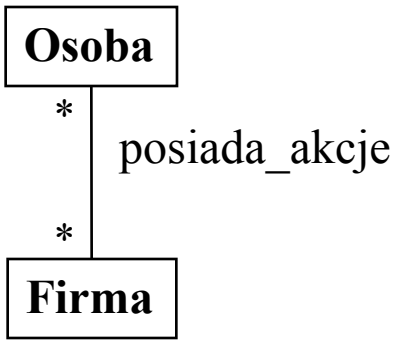
Po tym zabiegu rozmiar bazy danych zredukował się 4-krotnie.

Jak poprzednio, wydajność może być gorsza ze względu na złączenia.

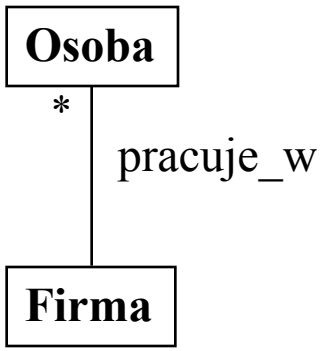
Klucze kandydujące

<u>Dla każdej klasy</u> można rozpatrzyć atrybut lub kombinację atrybutów, które mogą utworzyć klucz. Jeżeli takiego atrybutu(ów) nie ma, wówczas należy powołać klucz "sztuczny"; generowany automatycznie – OID.

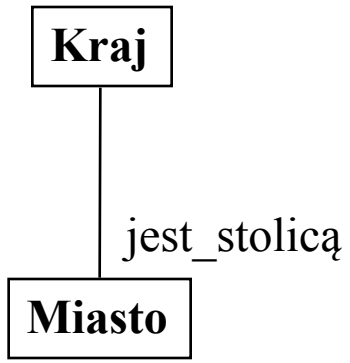
<u>Dla asocjacji:</u> kombinacja kluczy klas obiektów, połączonych daną asocjacją**.**



<u>Klucz kandydujący</u>:
{(Osoba, Firma)}



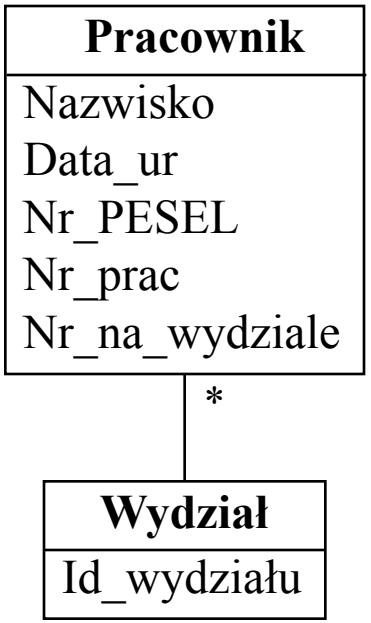
<u>Klucz kandydujący</u>:
{(Osoba)}



<u>Klucze kandydujące</u>:
{(Kraj), (Miasto)}

Wybór klucza

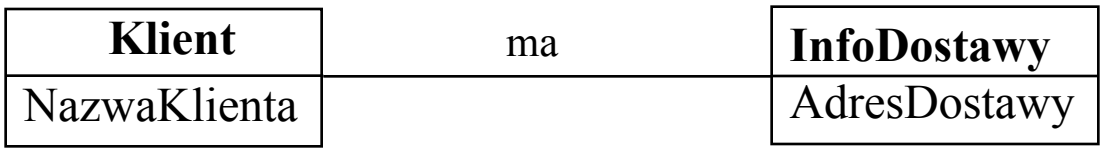
- **Może być wiele kluczy kandydujących, ale tylko jeden klucz główny.**
- **Klucze tabel nie powinny mieć znaczenia w dziedzinie przedmiotowej** (przeciwnie, niż postuluje główna doktryna modelu relacyjnego). Nawet trywialne zmiany w dziedzinie biznesu mogą podważyć dokonany wcześniej wybór klucza.
- **Klucze tabel nie powinny być złożone**; powinny być jednym atrybutem, co podważa sens dziesiątków prac teoretycznych. Praktyka pokazała, że złożone klucze (poza relacjami modelującymi związki) są powodem poważnych trudności wielu projektów.



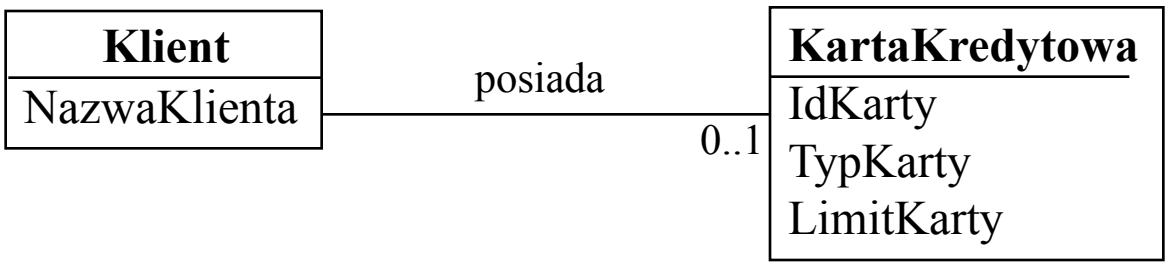
1. **Nazwisko + Data_ur**
2. **Nr_PESEL**
3. **Nr_prac**
4. **Nr_na_wydziale**

W większości przypadków, klucz powinien być generowany automatycznie.

Przejście na model relacyjny; przykłady (1)



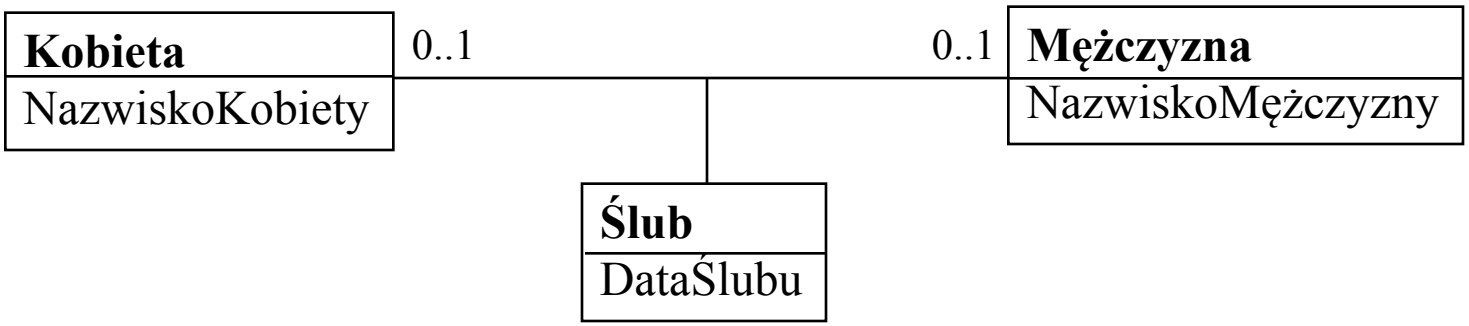
KlientDostawa (<u>IdKlienta</u>, NazwaKlienta, AdresDostawy)



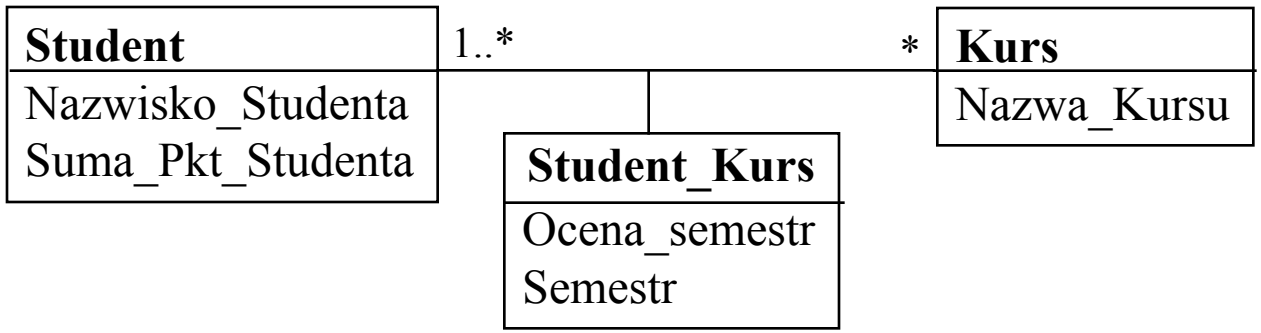
Klient (<u>IdKlienta</u>, NazwaKlienta)
KartaKredytowa (<u>IdKarty</u>, <u>TypKarty</u>, IdKlienta, LimitKarty)

Projektant nie zdecydował się na jedną tabelę, gdyż założył, że będzie zbyt dużo wartości zerowych.

Przejście na model relacyjny; przykłady (2)

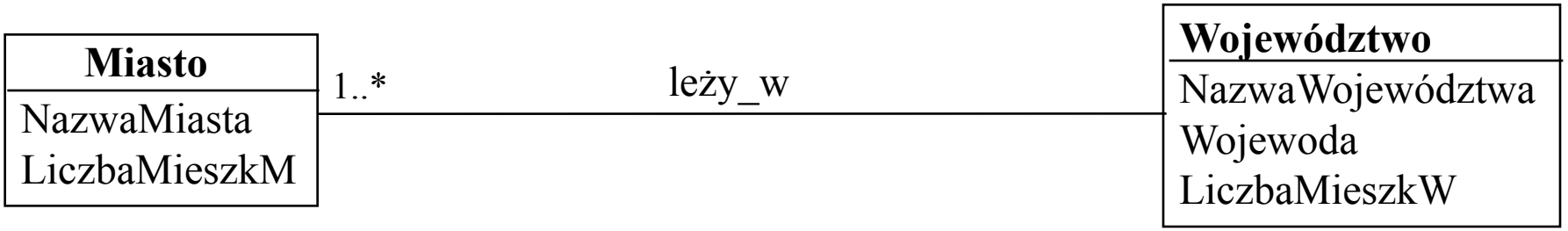


Kobieta(IdKobiety, NazwiskoKobiety)
Mężczyzna(IdMężczyzny, NazwiskoMężczyzny)
Ślub(IdKobiety, IdMężczyzny, DataŚlubu)

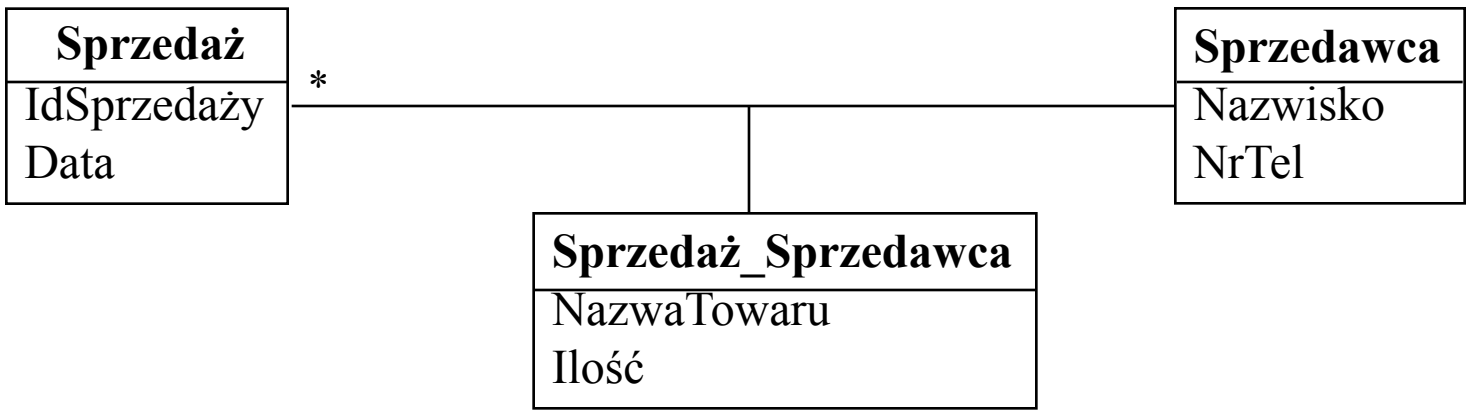


Student (Id_Studenta, Nazwisko_Studenta, Suma_Pkt_Studenta)
Kurs (Id_Kursu, Nazwa_Kursu)
Student_Kurs (Id_Studenta, Id_Kursu, Semestr, Ocena_semestr)

Przejście na model relacyjny; przykłady (3)

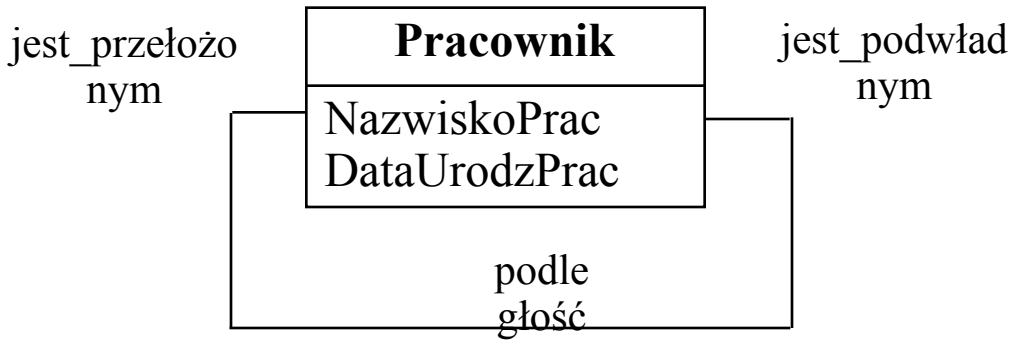


Miasto(NazwaMiasta, NazwaWojewództwa, LiczbaMieszkM)
Województwo(NazwaWojewództwa, Wojewoda, LiczbaMieszkW)



Sprzedawca(IdSprzedawcy, Nazwisko, NrTel)
Sprzedaż(IdSprzedaży, Data)
Sprzedaż_Sprzedawca (IdSprzedaży, IdSprzedawcy, NazwaTowaru, Ilość)

Przejście na model relacyjny; przykłady (4)



M:N

Pracownik (IdPracownika, NazwiskoPrac, DataUrodzPrac)
Podległość (IdPracPodwład, IdPracPrzełoż)

1:N

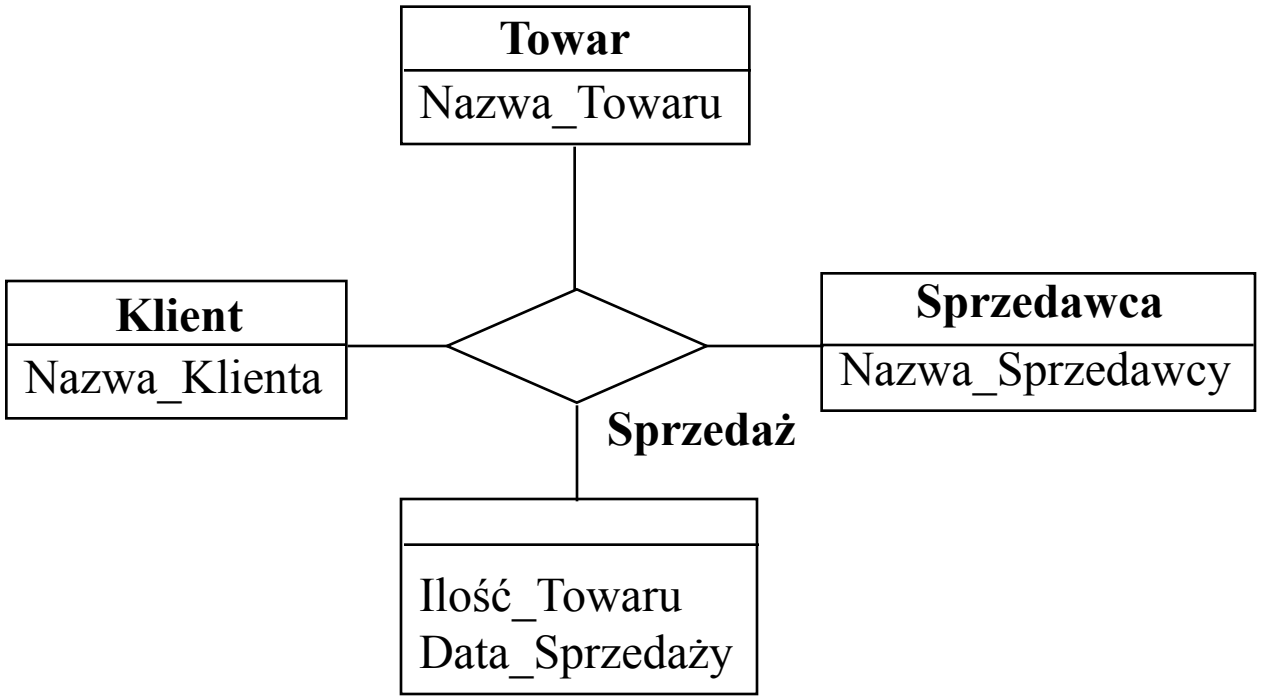
Pracownik (IdPracownika, NazwiskoPrac, DataUrodzPrac)
Podległość(IdPracPodwład, IdPracPrzełoż)

Różnica w kluczach

1:1 i 1:N

Pracownik(IdPracownika, NazwiskoPrac, DataUrodzPrac, IdPracPrzełoż)

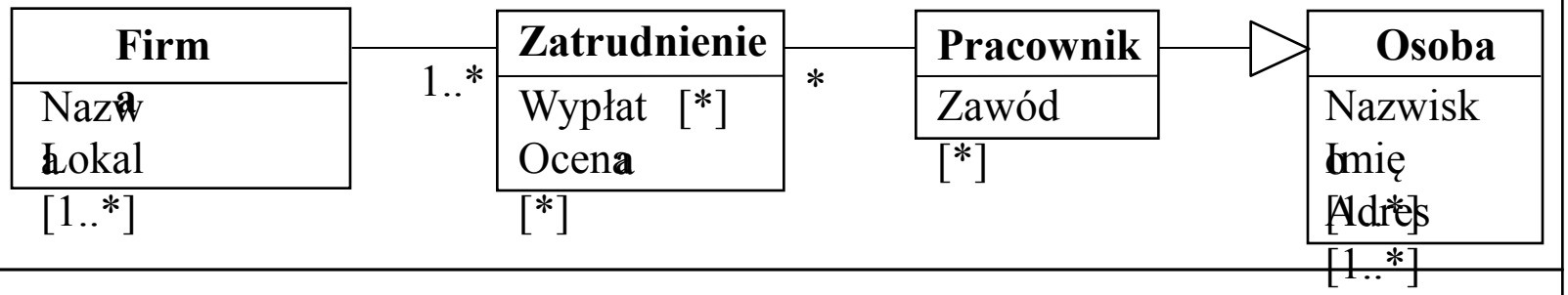
Przejście na model relacyjny; przykłady (5)



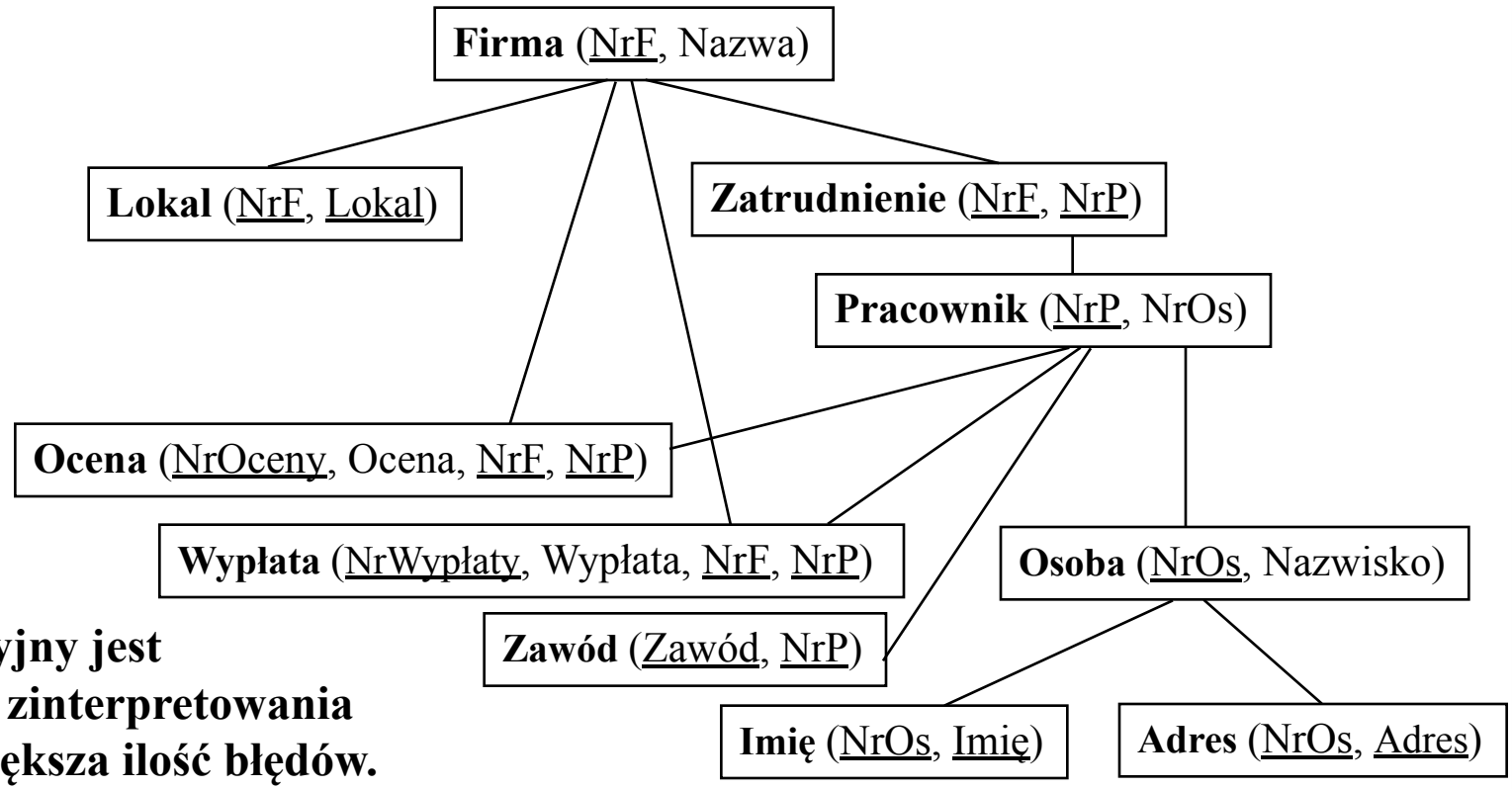
Klient (Id_Klienta, Nazwa_Klienta)
Towar (Id_Towaru, Nazwa_Towaru)
Sprzedawca (Id_Sprzedwcy, Nazwa_Sprzedawcy)
Sprzedaż (Id_Klienta, Id_Sprzedawcy, Id_Towaru, Data_Sprzedaży, Ilość_Towaru)

Przejście na model relacyjny; przykłady (6)

Pojęciowy schemat obiektowy:



Schemat realizacyjny (relacyjny):



Schemat relacyjny jest trudniejszy do zinterpretowania − w efekcie większa ilość błędów.