

BAZY DANYCH

LITERATURA

- ★ **Beynon-Davies P. - Systemy baz danych – WNT Warszawa 2003**
- ★ **Ullman J.D., Widom J. – Podstawowy wykład z systemów baz danych – WNT Warszawa 2001**
- ★ **Date C.J. - Wprowadzenie do systemów baz danych - WNT Warszawa, 2000**
- ★ **L. Banachowski, A. Chędzyńska, K. Matejewski, E. Mrówka-Matejewska, K. Stencel - Bazy danych. Wykłady i ćwiczenia – Wydawnictwo PWSTK Warszawa 2003**
- ★ **Hernandez M.J. – Bazy danych dla zwykłych śmiertelników – Mikom 2004**
- ★ **Kowalski P – Podstawowe zagadnienia baz danych i procesów przetwarzania – Mikom 2005**

Wykład 1

Wprowadzenie do Baz Danych

Opis podstawowych cech systemu bazy danych, definicje podstawowych pojęć, znaczenie baz danych we współczesnych systemach informacyjnych

Jacek Bartman

jbartman@univ.rzeszow.pl

1. Bazy danych jako maszyny abstrakcyjne

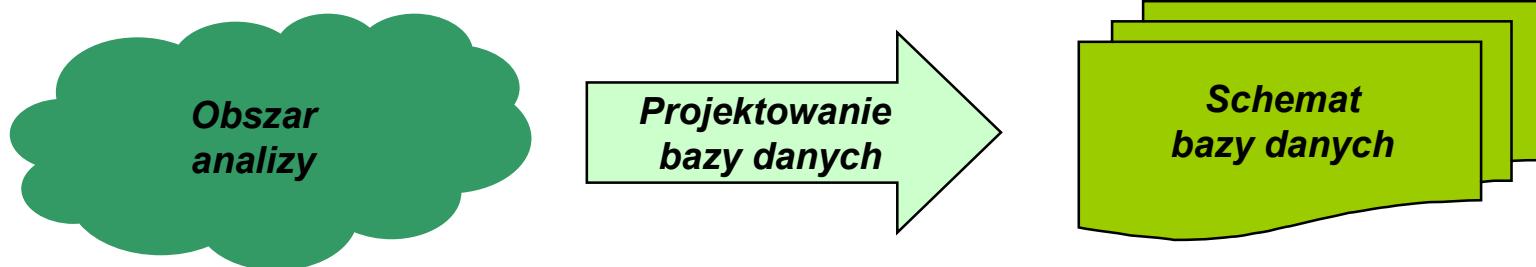
1.1. System bazy danych

★ Większość nowoczesnych przedsiębiorstw musi rejestrować dane dotyczące swojej codziennej działalności.

- Np.: Uniwersytet musi m.in. rejestrować:
 - jakich ma studentów i wykładowców,
 - jakie wykłady i moduły są przez nich prowadzone,
 - którzy wykładowcy uczą których modułów,
 - którzy studenci są zapisani na jakie moduły,
 - którzy studenci zaliczają które moduły.
- Dane mogą być wprowadzane przez różne osoby
- Dane można wykorzystać do różnych celów
 - np.: lista aktualnych studentów, wykorzystanie sal itp.

1.2 Obszar analizy

- ★ baza jest modelem pewnego aspektu rzeczywistości, rzeczywistość ta to obszar analizy
- ★ w przypadku uniwersytetu obszar analizy może obejmować m.in. moduły oferowane studentom i studentów zaliczających te moduły (rzeczy istotne dla uniwersytetu)
- ★ rzeczy istotne nazywa się **klasami** lub **encjami**,
 - np.: studenci, moduły,
- ★ klasy mogą być powiązane,
 - np.: którzy studenci jakie moduły zaliczają,
- ★ klasy mają **właściwości** lub **atrybuty**,
 - np.: studenci mają nazwiska, imiona, adresy itp..
- ★ klasy, atrybuty i związki muszą mieć jakąś reprezentację w bazie danych (baza musi być zaprojektowana)



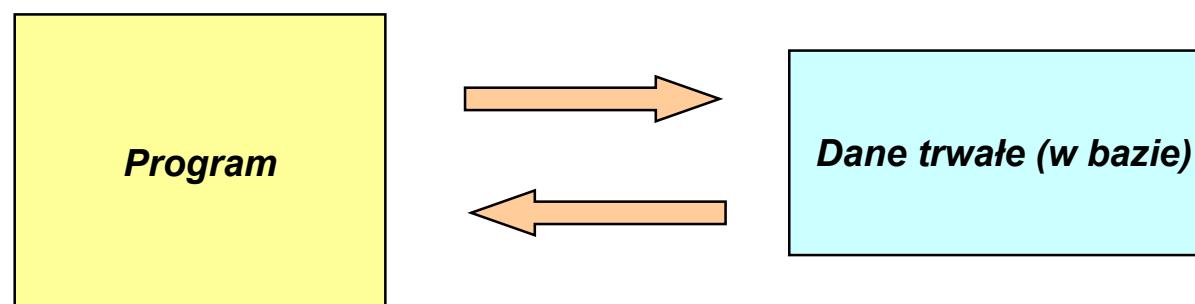
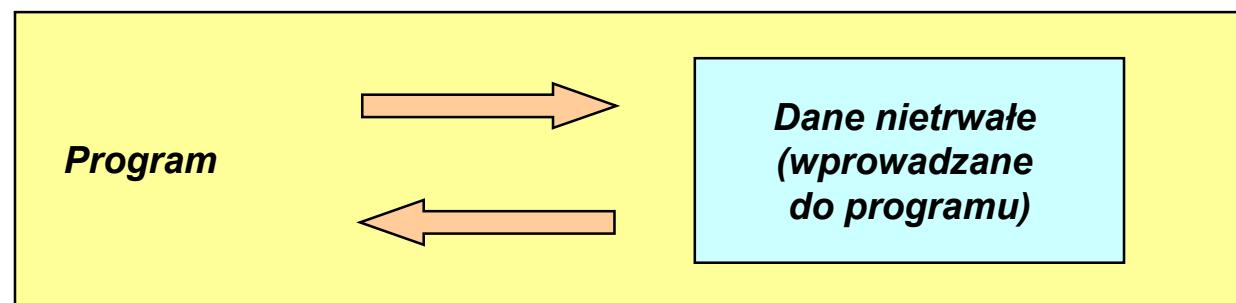
1.2.1. Bazy faktów



- ★ **Dane to fakty**, dana jest symbolem lub zbiorem symboli
- ★ Fakty same w sobie nie mają znaczenia, aby były użyteczne muszą być zinterpretowane.
- ★ Zinterpretowane dane to **informacje** (maja przypisaną semantykę).
 - np. : napis 43 sam nic nie znaczy, musi być podany kontekst
- ★ Bazę danych możemy uważać za zbiór faktów lub pozytywnych asercji na temat obszaru analizy za zbiór faktów.
- ★ W określonej chwili baza danych znajduje się w jakimś stanie (zawiera fakty prawdziwe w danej chwili).

1.2.2 Trwałość

- ★ Dane w bazie danych traktowane są jako trwałe (mogą być przechowywane przez pewien czas).
- ★ Dane wprowadzane przy terminalu przeznaczone do przetworzenia nie są trwałe.



1.2.3 Część intensjonalna i ekstensjonalna

- ★ Baza danych składa się z dwóch części: intensjonalnej i ekstensjonalnej

■ **Część intensjonalna** – zbiór definicji które opisują strukturę danych bazy, inaczej jest to schemat bazy danych (uzyskany w wyniku projektowania)

■ **Część ekstensjonalna** – łączny zbiór danych w bazie (zawartość bazy)

Schemat: Uniwersytet	Ekstensja (zawartość) —
Moduły: Systemy relacyjnych baz danych Projektowanie relacyjnych baz danych	
Studenci: Jan Kowalski urodzony 1 lipca 1980 Piotr Nowak urodzony 10 maja 1980 Maria Maj urodzona 18 stycznia 1981	
Zaliczają: Jan Kowalski zalicza Systemy relacyjnych baz danych	

Schemat: Uniwersytet	Intensja (schemat) —
	Klasy: Moduły – kursy prowadzone przez instytucję w semestrze Studenci – osoby zaliczające moduły w tej instytucji
	Związki: Studenci zaliczają moduły
	Atrybuty: Moduły mają nazwy Studenci mają nazwiska

1.3. Integralność

- ★ **Baza ma właściwości integralności gdy dokładnie odzwierciedla swój obszar analizy**
 - Np.: w uniwersyteckiej bazie danych integralność oznacza zapewnienie, że baza daje poprawne odpowiedzi na pytania typu „*ilu studentów jest zapisanych na moduł Systemy relacyjnych baz danych*”
- ★ **Integralność jest ważna szczególnie w bazach, których dane zmieniają się.**
- ★ **W zbiorze przyszłych stanów bazy są poprawne i niepoprawne integralność gwarantuje, iż baza nie przyjmie stanu niepoprawnego.**

Schemat: Uniwersytet	Ok. Stan poprawny
Moduły: <i>Systemy relacyjnych baz danych</i> <i>Projektowanie relacyjnych baz danych</i>	
Studenci: <i>Jan Kowalski urodzony 1 lipca 1980</i> <i>Piotr Nowak urodzony 10 maja 1980</i> <i>Maria Maj urodzona 18 stycznia 1981</i>	
Zaliczają: <i>Jan Kowalski zalicza Systemy relacyjnych baz danych</i> <i>Jan Kowalski zalicza Projektowanie relacyjnych baz danych</i>	

Schemat: Uniwersytet	Stan niepoprawny
Moduły: <i>Systemy relacyjnych baz danych</i> <i>Projektowanie relacyjnych baz danych</i>	
Studenci: <i>Jan Kowalski urodzony 1 lipca 1980</i> <i>Piotr Nowak urodzony 10 maja 1980</i> <i>Maria Maj urodzona 18 stycznia 1981</i>	
Zaliczają: <i>Jan Kowalski zalicza Systemy relacyjnych baz danych</i> <i>Piotr Nowak zalicza Podstawy programowania</i>	

1.3.1. Redundancja (nadmiarowość)

- * Redundancja to powtarzanie się danych w bazie (należy jej unikać)
- * Fakty w bazie danych nie powinny się powtarzać

Schemat: Uniwersytet

Moduły:

*Systemy relacyjnych baz danych
Projektowanie relacyjnych baz danych*

Studenci:

*Jan Kowalski urodzony 1 lipca 1980
Piotr Nowak urodzony 10 maja 1980
Maria Maj urodzona 18 stycznia 1981*

Zaliczają:

*→ Jan Kowalski zalicza Systemy relacyjnych baz danych
Piotr Nowak zalicza Projektowanie relacyjnych baz danych
→ Jan Kowalski zalicza Systemy relacyjnych baz danych*

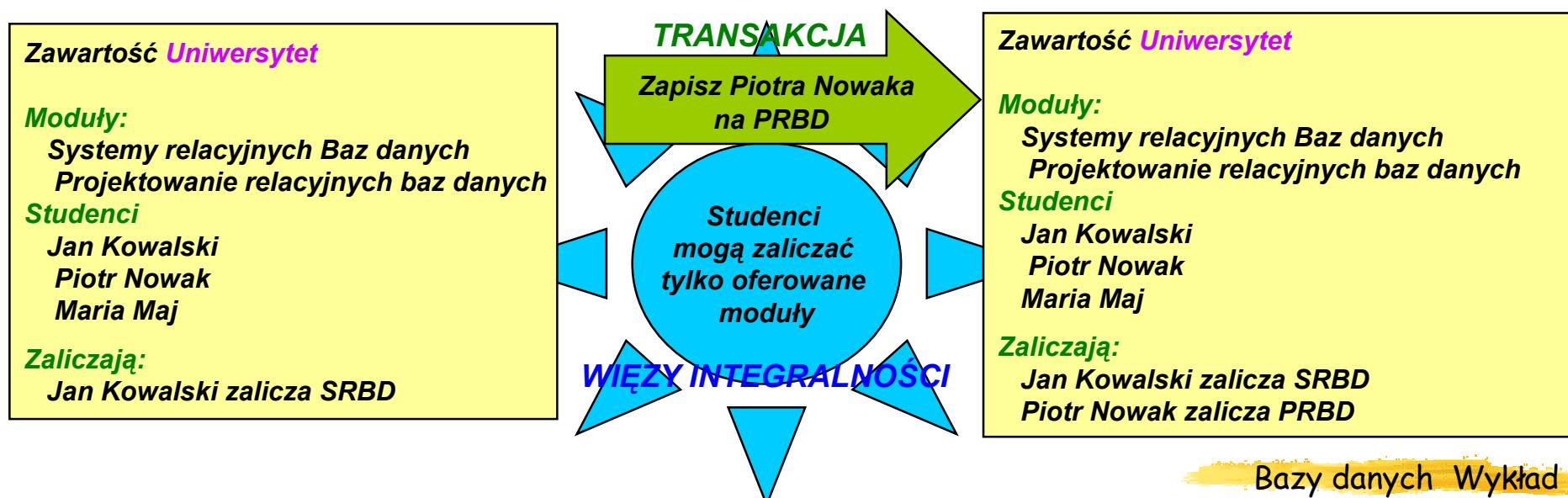
1.3.2 Transakcje

- * Transakcje to zdarzenia powodujące zmianę stanu bazy danych
- * Typem transakcji nazywamy transakcje o podobnej formie np.:

zapisz studenta na moduł

1.3.3. Więzy integralności

- ★ Integralność bazy danych jest realizowana przez więzy integralności
- ★ Więzy integralności to reguły określające w jaki sposób baza danych ma pozostać dokładnym odbiciem swego obszaru analizy
 - Wizy statyczne (niezmienne stanu) – sprawdzają czy wykonana transakcja nie zmienia stanu bazy w stan niepoprawny.
 - np.: czy studenci zaliczają tylko oferowane moduły
 - Więzy przejść – reguły, które wiążą ze sobą stany bazy danych. Stanowią ograniczenie nałożone na przejście:
 - np.: liczba modułów zaliczanych przez studenta nie może w semestrze zmniejszyć się do zera



1.4. Funkcje bazy danych

- ★ Do wykonywania operacji na bazie są potrzebne dwa rodzaje funkcji:
 - funkcje aktualizujące,
 - funkcje zapytań.

1.4.1. Funkcje aktualizujące

- ★ Funkcje aktualizujące dokonują zmian w danych
- ★ Transakcje mogą inicjować funkcje aktualizujące, które zmieniają bazę z jednego stanu w inny
 - Np. na uniwersytecie:
 - Zainicjuj semestr
 - Zaoferuj moduł
 - Anuluj moduł
 - Wpisz studenta na kurs
 - Przenieś studenta między modułami

1.4.2. Funkcje aktualizujące i więzy integralności

- ★ Z funkcją aktualizującą związaną jest ciąg warunków reprezentujących więzy integralności

■ np.:

Funkcja aktualizująca: Przenieś studenta X z modułu 1 do modułu 2

Warunki:

Student X zalicza moduł 1
Student X nie zalicza modułu 2
Moduł 2 jest oferowany

Akcje:

Zaprzecz, że student X zalicza moduł 1
Stwierdz, że student X zalicza moduł 2

- ★ Akcja danej funkcji aktualizującej może spowodować, że warunki innej funkcji aktualizującej staną się prawdziwe

■ np.:

Funkcja aktualizująca: Anuluj moduł 1

Warunki:

Moduł 1 istnieje
Liczba zapisanych studentów na moduł 1 jest mniejsza od 10

Akcje:

Zaprzecz, że moduł 1 istnieje

- Jeżeli funkcja „Anuluj moduł 1” jest aktualnie niewykonalna gdyż nie jest spełniony warunek 2. To wygenerowanie funkcji przeniesienia studenta z modułu 1 do innego, spowoduje zmniejszenie liczby studentów zapisanych na moduł 1 i może doprowadzić do sytuacji w której funkcja „Anuluj moduł 1” będzie wykonalna

1.4.3. Funkcje zapytań

* Funkcje zapytań wydobywają dane z bazy danych, nie modyfikując ich zawartości

■ np.:

- Czy moduł X jest oferowany
- Czy student X zalicza moduł Y
- Którzy studenci zaliczają moduł X ?
- Które moduły są obecnie oferowane?

Zwraca odpowiedź tak/nie

Zwraca zbiór wartości

Ten typ pytań jest częściej stosowany

1.5. Formalizmy

- ★ **Każdy system bazy danych musi używać jakiegoś formalizmu reprezentacji (konwencji zapisu).**
- ★ **Formalizm reprezentacji zbioru składnikowych i semantycznych konwencji, które umożliwiają opisywanie rzeczywistości.**
 - Składnia reprezentacji określa zbiór reguł łączenia symboli i układów symboli w celu tworzenia wyrażeń
 - Semantyka reprezentacji określa w jaki sposób należy interpretować wyrażenia
- ★ **W bazach danych idea formalizmu reprezentacji odpowiada pojęciu modelu danych**
 - Modele relacyjne dobrze reprezentują fakty, ale komplikują reprezentację funkcji aktualizujących
 - Modele obiektowe oferują dobrą reprezentację funkcji aktualizujących ale nie posiadają jednolitości dostępnych implementacji

1.6. Dostęp dla wielu użytkowników

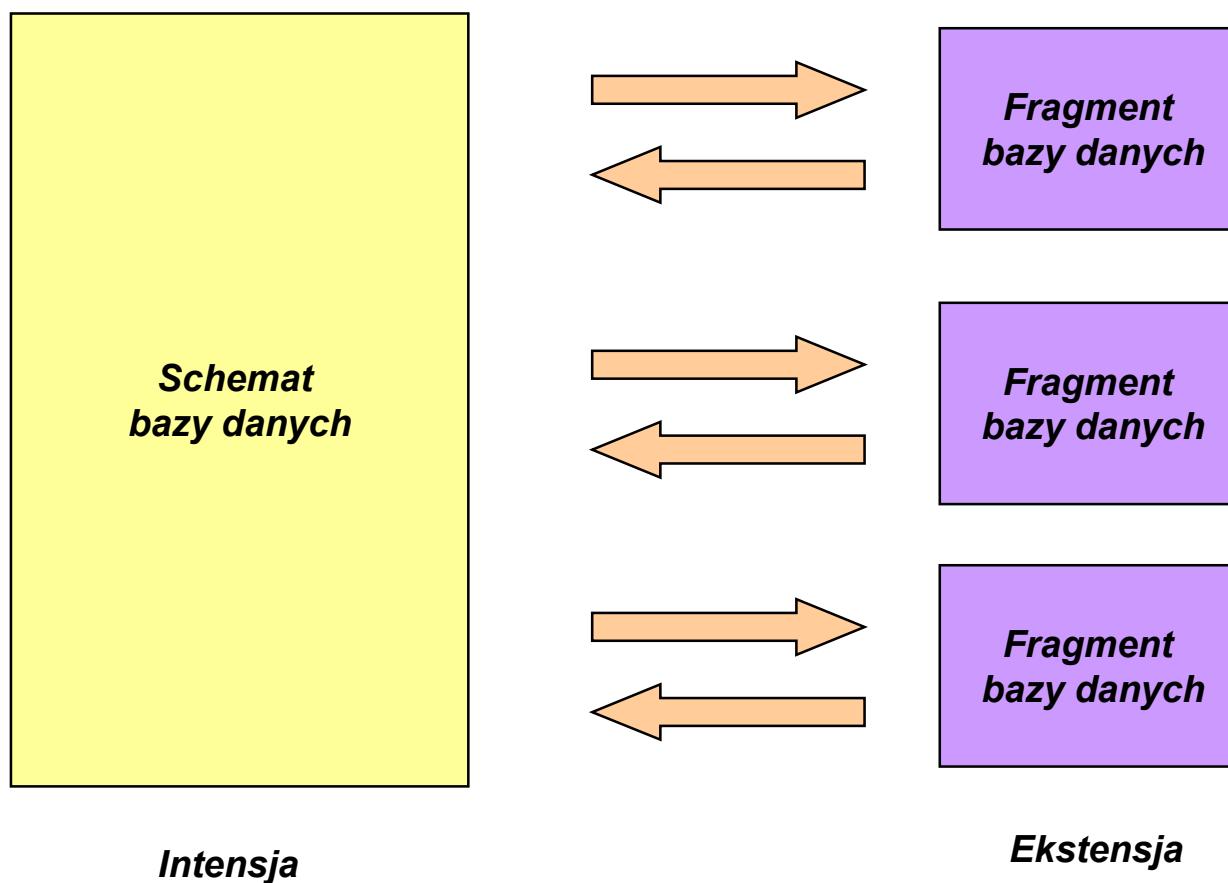
- ★ Większość baz danych jest używana przez wielu użytkowników – powstaje problem **współbieżności**
 - Np.: jeden użytkownik rejestruje studenta na jakiś moduł a inny w tym samym czasie usuwa ten moduł,
 - Jeden użytkownik sprzedaje miejsce w samolocie na określony rejs, a inny robi to samo
- Działania takie prowadziły by do niespójności bazy danych
- ★ Baza danych musi radzić sobie z konfliktami współbieżności

1.6.1. Perspektywy

- ★ Jednym z powodów, dla których bazy danych są używane przez wiele osób, jest to, że baza może być wykorzystywana do różnych celów.
 - Np.: w bazi uniwersyteckiej dziekanaty będą zainteresowana danymi o studentach, układający podziały godzin danymi o salach i wykładowcach.
- ★ Podzbiór danych będących w czymś zainteresowaniu nosi nazwę **perspektywy**

1.7. Rozproszenie

- ★ W systemie rozproszonych baz danych jeden rzeczywisty schemat jest przechowywany w kilku oddzielnych fragmentach i/lub kopach.
 - Np.: w bazie studenckiej dane o studentach mogą być przechowywane na wydziałach a dane kadrowo-płacowe w dziale administracyjnym



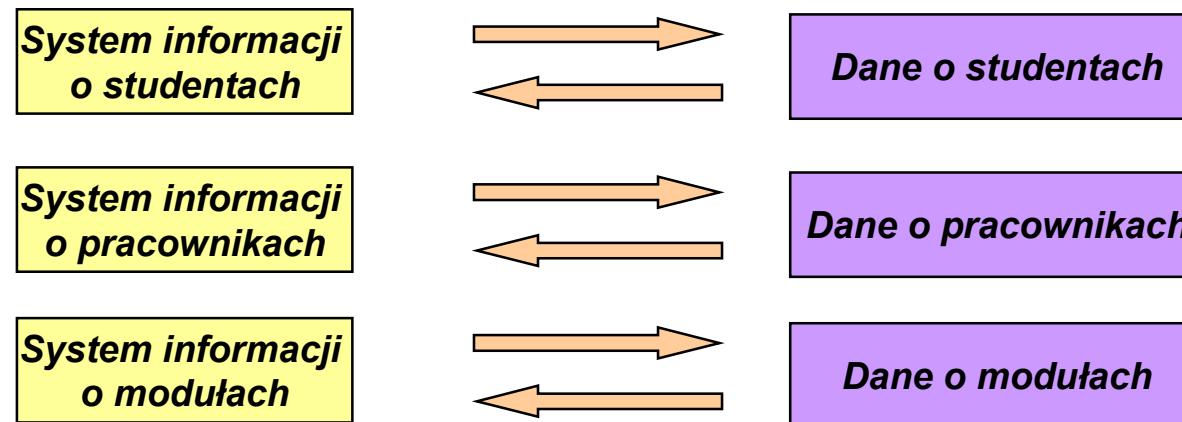
2. Podstawowe pojęcia

★ TEMatyka

- Określenie potrzeb systemów baz danych
- Definicje pojęć:
 - Baza danych
 - System zarządzania bazą danych
 - Model danych
- Rozróżnienie wielu znaczeń terminu model danych
- Historia zarządzania danymi
- Wprowadzenie do procesu tworzenia bazy danych

2.1. Rozwój systemu informacyjnego

- ★ Informatyzacja zwyczajowo odbywała się po kawałku (każdy dział oddzielnie)
- ★ Informatyzacja „po kawałku” najczęściej prowadzi do powstania wielu oddzielnych systemów z własnymi plikami, danymi i programami



- ★ Izolowane systemy nie reprezentują sposobu w jaki działa przedsiębiorstwo.
 - ★ Wymiana danych między systemami odbywa się poza nimi.
 - ★ Informacje wygenerowane przez kilka systemów są dla personelu mniej wartościowe, gdyż nie dają pełnego obrazu działalności przedsiębiorstwa.
 - ★ Dane mogą być powielane w wielu plikach używanych przez różne systemy.
-
- ★ **W nowoczesnych systemach informatycznych dane są zintegrowane – dzięki czemu mogą być wykorzystywane na wiele sposobów**

2.2. Bazy danych i systemy zarządzania bazą danych

- ★ W nowoczesnych systemach dane przechowuje się w sposób skonsolidowany w jednym miejscu
- ★ Dąży się do integracji Systemów Zarządzania Bazą Danych – SZBD (Database Management System – DBMS)

2.2.1 Co to jest baza danych ?

- ★ **Baza danych to zbiór powiązanych i trwałych danych opisujących rzeczywistość**
- ★ **Baza danych jest magazynem danych z nałożoną na niego strukturą wewnętrzną – zazwyczaj hierarchiczną**
 - Np.: bazą danych może być zbiór informacji o studentach – kartoteka a w niej teczki z informacjami o studentach – informacje na oddzielnych kartkach: dane personalne, zaliczenia, praktyki itp..
- ★ **Celem bazy danych jest najczęściej przechowywanie danych potrzebnych do codziennego funkcjonowania organizacji**

2.2.1 Właściwości bazy danych

- ★ **Współdzielenie danych** – dane mogą być używane przez wiele osób nawet w tym samym czasie
- ★ **Integracja danych** – w bazie nie może być redundancji danych (danych powtarzających się oraz zbędnych)
- ★ **Integralność danych** – baza danych musi dokładnie odzwierciedlać swój obszar analizy (w przypadku danych powiązanych zmiany winny być propagowane). Integralność często zapewnia się ograniczając do niej dostęp.
- ★ **Bezpieczeństwo danych** – polega głównie na wydzielaniu zakresu danych do których poszczególni użytkownicy mają określone prawa dostępu
- ★ **Abstrakcja danych** – stanowią model rzeczywistości
- ★ **Niezależność danych** – dane winny być oddzielone od procesów na nich operujących (organizacja danych ma być niewidoczna dla użytkowników)

2.3. Co to jest System Zarządzania Bazą Danych ?

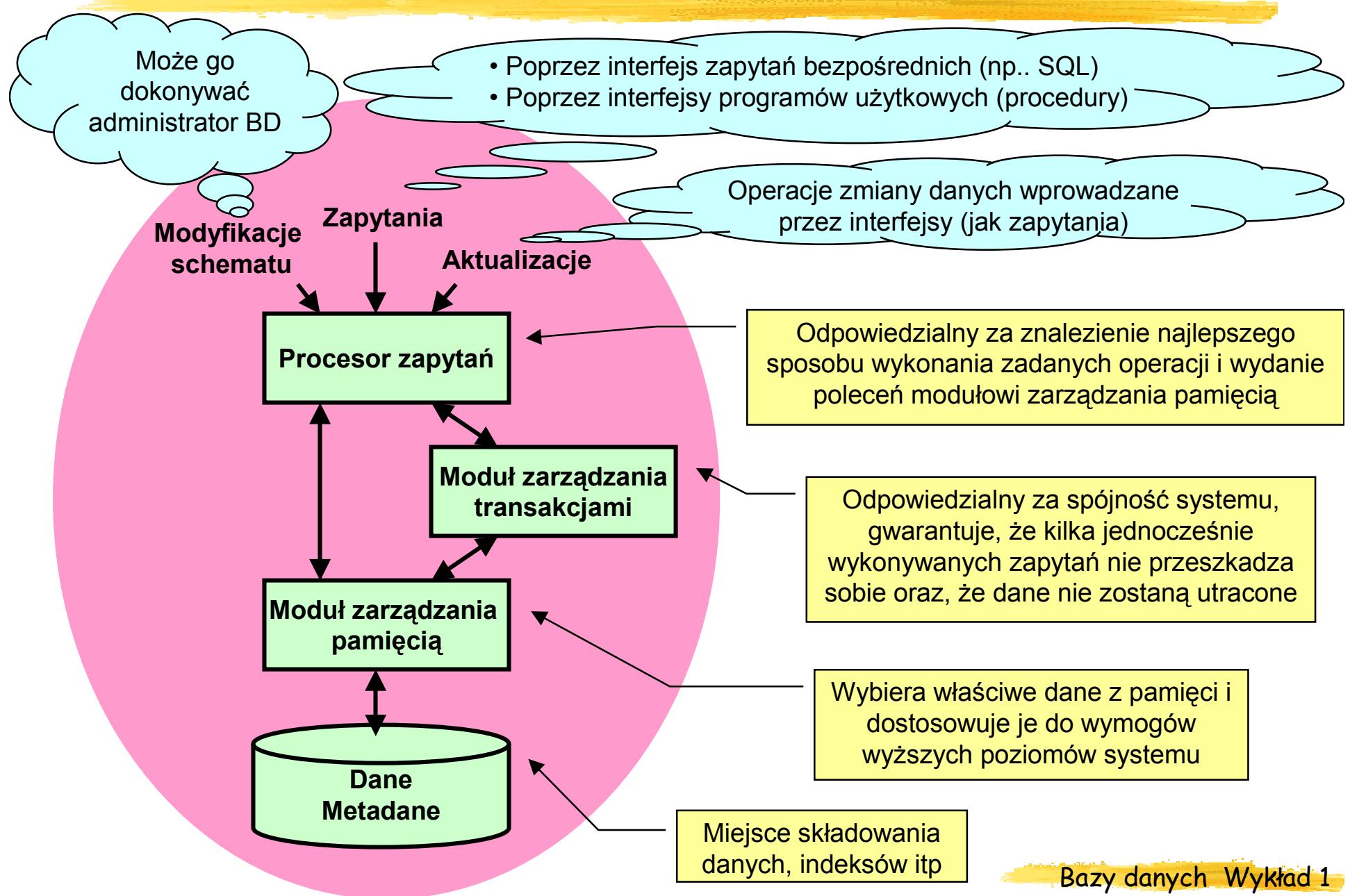
★ **System Zarządzania Bazą Danych SZBD (DBMS) jest to zbiór programów umożliwiających użytkownikom definiowanie, konstruowanie, manipulowanie bazą danych**

- definiowanie - specyfikacja typów danych, struktur, związków integralnościowych dla danych przechowywanych w bazie danych
- konstruowanie - proces zapisu danych na nośniku danych oraz ich kontrola przez SZBD
- manipulowanie - zapytania do bazy danych, modyfikacja, raportowanie

★ **Zadania Systemu Zarządzania Bazą Danych**

- **Pielęgnacja danych.** Umożliwienie użytkownikowi tworzenia nowych baz danych i określania ich struktury logicznej (przy pomocy specjalnego języka) oraz aktualizowania ich.
- **Wyszukiwanie danych.** Udostępnienie użytkownikowi możliwości tworzenia zapytań o dane.
- **Kontrola danych.** Sterowanie wielodostęmem do danych

2.3.1. Architektura SZBD (DBMS)



2.4. Modele danych

- ★ Każda baza danych, a także każdy SZBD muszą stosować się do zasad określonego modelu danych.
- ★ Pojęcie model danych jest niejednoznaczne; najczęściej używane jest w znaczeniu:
 - architektura danych
 - projekt danych (zbiór wymagań w odniesieniu do danych)

2.4.1. Model danych jako projekt

- ★ W tym sensie model danych to zintegrowany i niezależny od implementacji zbiór wymagań w odniesieniu do danych
 - Np.: model danych dla kartoteki studentów, model danych dla płac itp..

2.4.2 Model danych jako architektura

- ★ W tym sensie model danych jest zbiorem ogólnych zasad posługiwania się danymi: model hierarchiczny, model sieciowy, model relacyjny, model obiektowy
- ★ Zbiór zasad które określają model danych można podzielić na trzy kategorie:
 - definicje danych (określają strukturę danych),
 - operowanie danymi (reguły operowania danymi),
 - integralność danych (określa które stany bazy są poprawne)

2.4.3. Typy architektonicznych modeli danych

- ★ Wyróżnia się trzy generacje architektonicznych modeli danych:
 - proste modele danych (pliki płaskie z rekordami)
 - klasyczne modele danych (hierarchiczne, sieciowe, relacyjne) - bazują na rekordach i trudno z samej bazy odczytać znaczenie informacji.
 - semantyczne modele danych (obiektowe)

2.5. System baz danych

★ **System Baz Danych to skomputeryzowany system przechowywania i przetwarzania danych**

★ **System Baz Danych składa się z następujących elementów:**

- modelu danych
- oprogramowania
 - System Zarządzania Bazą Danych
 - Inne oprogramowanie
- baz danych

★ **Często do systemu baz danych zalicza się również:**

- sprzęt
 - pamięci masowe
 - urządzenia systemowe
- użytkowników
 - programiści aplikacji – tworzą programy umożliwiające innym użytkownikom dostęp do bazy danych
 - użytkownicy końcowi – obsługujący bazę danych – wprowadzający dane, generujący raporty itp..
 - administratorzy BD – odpowiada za tworzenie i konserwacje rzeczywistej bazy danych, odpowiada za jej bezpieczeństwo

2.6 Historia zarządzania danymi

- ★ **4000 p.n.e – 1900 n.e – ręczne zarządzanie zapisami 4000**
 - pierwszy znany zapis obejmujący majątek królewski i podatki w Sumatrze.
- ★ **1900 – 1955 – zarządzanie zapisami na kartach perforowanych**
 - Maszyna Hollerith'a (bazująca na wynalazku Jacquarda z 1801 roku).
- ★ **1955 – 1970 – programowane zarządzanie zapisami**
 - 1950 rok wynalezienie taśmy magnetycznej (na jednej mieściły się dane z ok. 10 000 kart perforowanych)
 - praca wsadowa na plikach i rekordach (nie daje możliwości wychwycenia błędów aż do zakończenia wsadu)
- ★ **1965 – 1970 – sieciowe zarządzanie danymi on-line**
 - praca on-line, przetwarzanie bieżących transakcji
 - pliki indeksowane
 - zastosowanie bębnów i dysków magnetycznych
 - trzypoziomowa architektura bazy danych

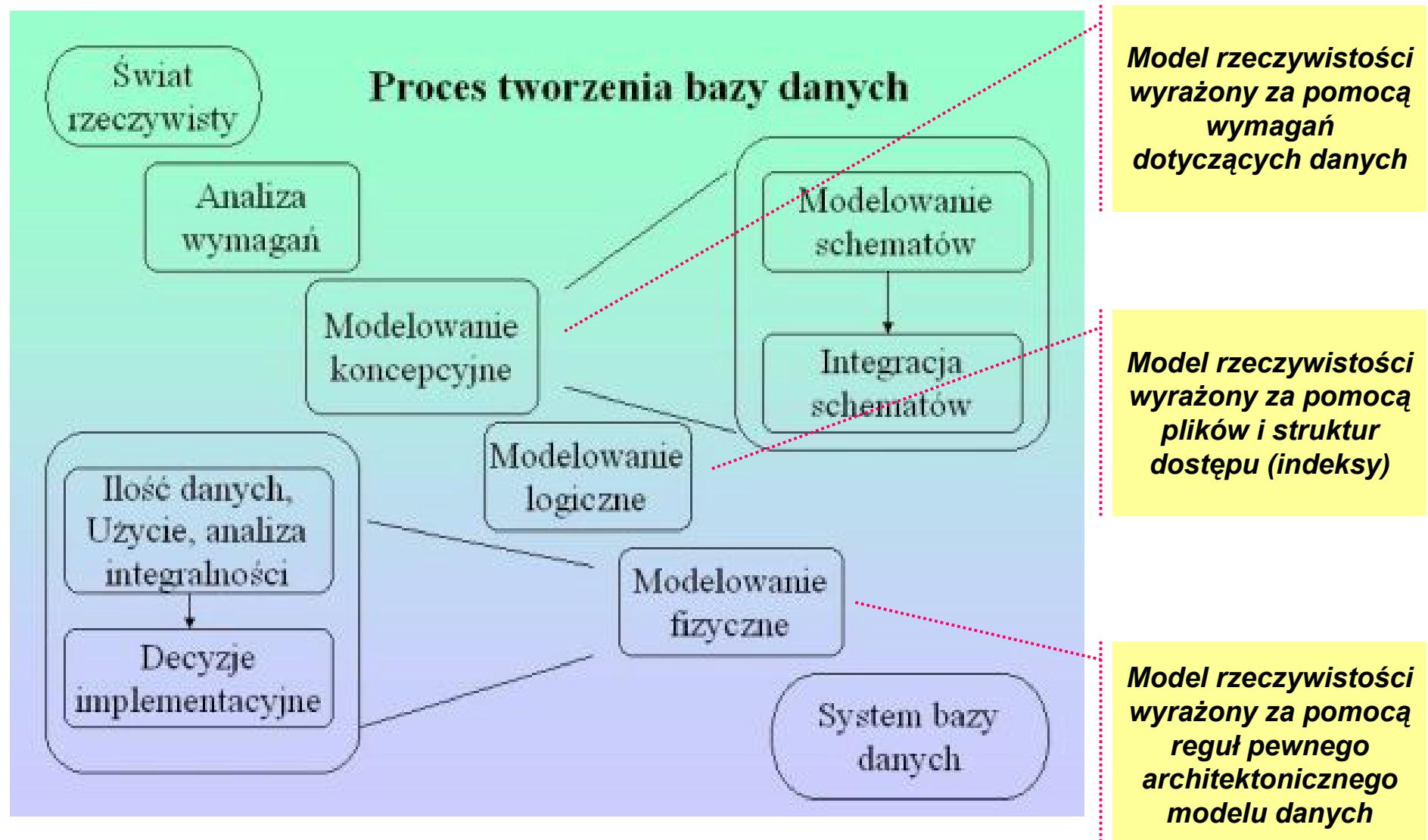
* 1980-1995 – relacyjne zarządzanie danymi

- 1970 – model Codd'a
- powszechnie SZBD na komputery PC
- najczęściej stosowany model
- architektura klient-server

* 1995- Obiektywne bazy danych

- dane aktywne
- oparte o mechanizmy dziedziczenia
- Często przechowują dane różnych typów np.: głos, obraz itp..

2.7. Tworzenie baz danych



Wykład 2

Wprowadzenie do Baz Danych

Warstwa zarządzania danymi,

Relacyjny model danych

Jacek Bartman

jbartman@univ.rzeszow.pl

3. Bazy danych i systemy informacyjne

* TEMatyka:

- określenie różnic pomiędzy danymi, informacją i wiedzą
- umiejscowienie systemów baz danych w systemach informacyjnych
- modele baz danych i ich związek z architekturą systemu informacyjnego
- współczesne obszary zastosowań systemów baz danych

3.1. Dane, informacja, wiedza

- ★ **Dane to fakty (zarejestrowane w bazie danych)**
 - Dane są zapisywane w postaci symboli, które coś reprezentują
- ★ **Informacja jest to przyrost wiedzy, który może być uzyskany na podstawie danych** (Tsitchizris i Lochovsky).
 - pojęcie informacji jest płynne i różnie interpretowane
 - Informacja ma charakter subiektywny
 - Informacje uzyskujemy poprzez zinterpretowanie danych
- ★ **Wiedza jest otrzymywana z informacji przez zintegrowanie z istniejącą wiedzą**

3.2. System informacyjny i systemy technologii informacyjnej

- ★ **System informacyjny to system, który dostarcza informacje do przedsiębiorstwa lub jego części (IS)**

- Przedstawia się je modelem wejście-proces-wyjście istniejącego w danym środowisku



- Środowisko systemu to wszystko co ma na niego wpływ i jest poza nim

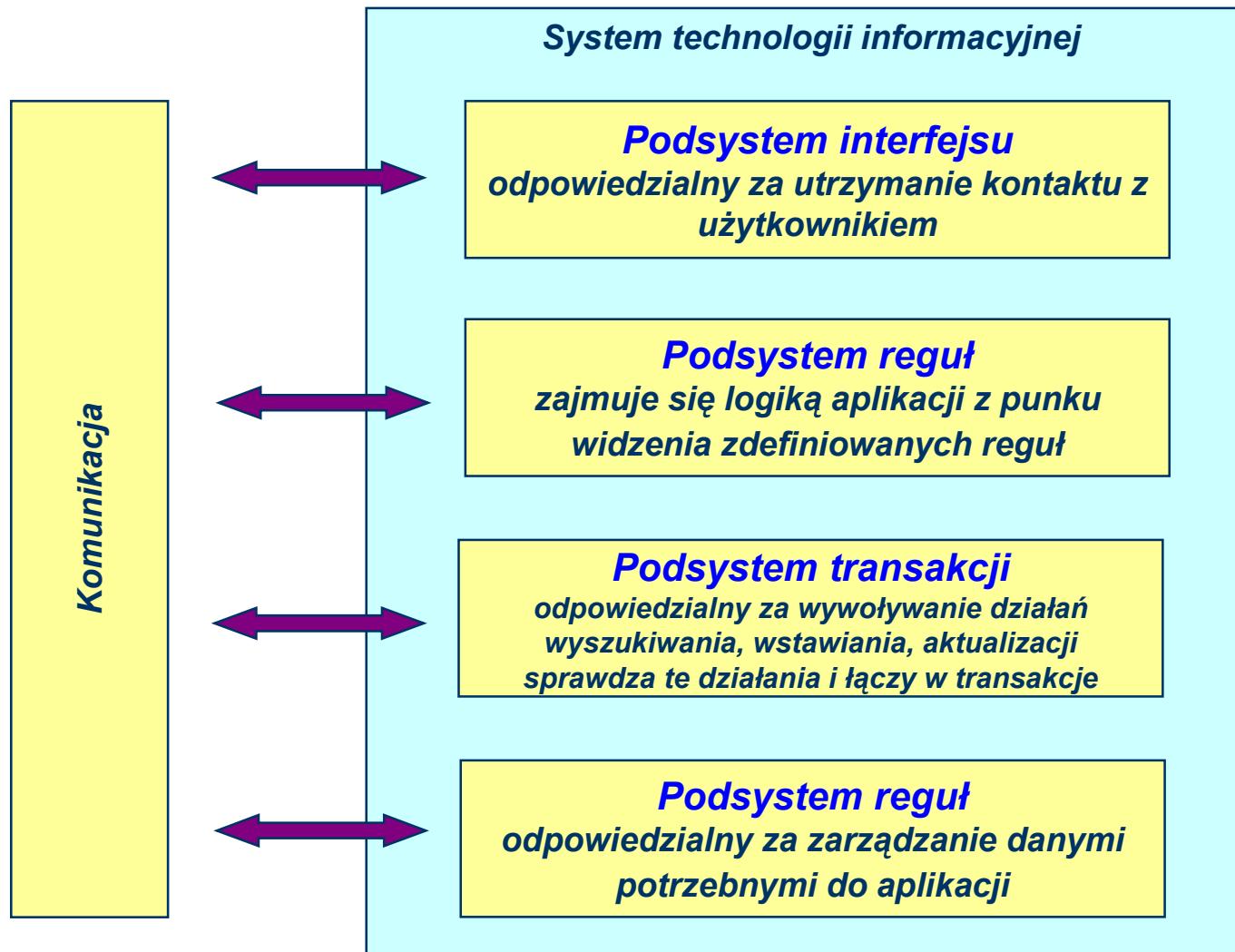
- ★ **System informacyjny obsługuje systemy związane z działalnością człowieka.**

- Przedsiębiorstwa najczęściej potrzebują kilku systemów informacyjnych bo posiadają kilka systemów związanych z działalnością człowieka.

- ★ **Technologia informacyjna (IT) obejmuje komputery wraz z oprogramowaniem oraz komunikację**

- Technologia informacyjna zapewnia środki do konstruowania nowoczesnych systemów informacyjnych
 - Systemy informacyjne mogą istnieć bez technologii informacyjnej (ale wtedy nie są to systemy technologii informacyjnej)

3.3. Warstwy systemu technologii informacyjnej

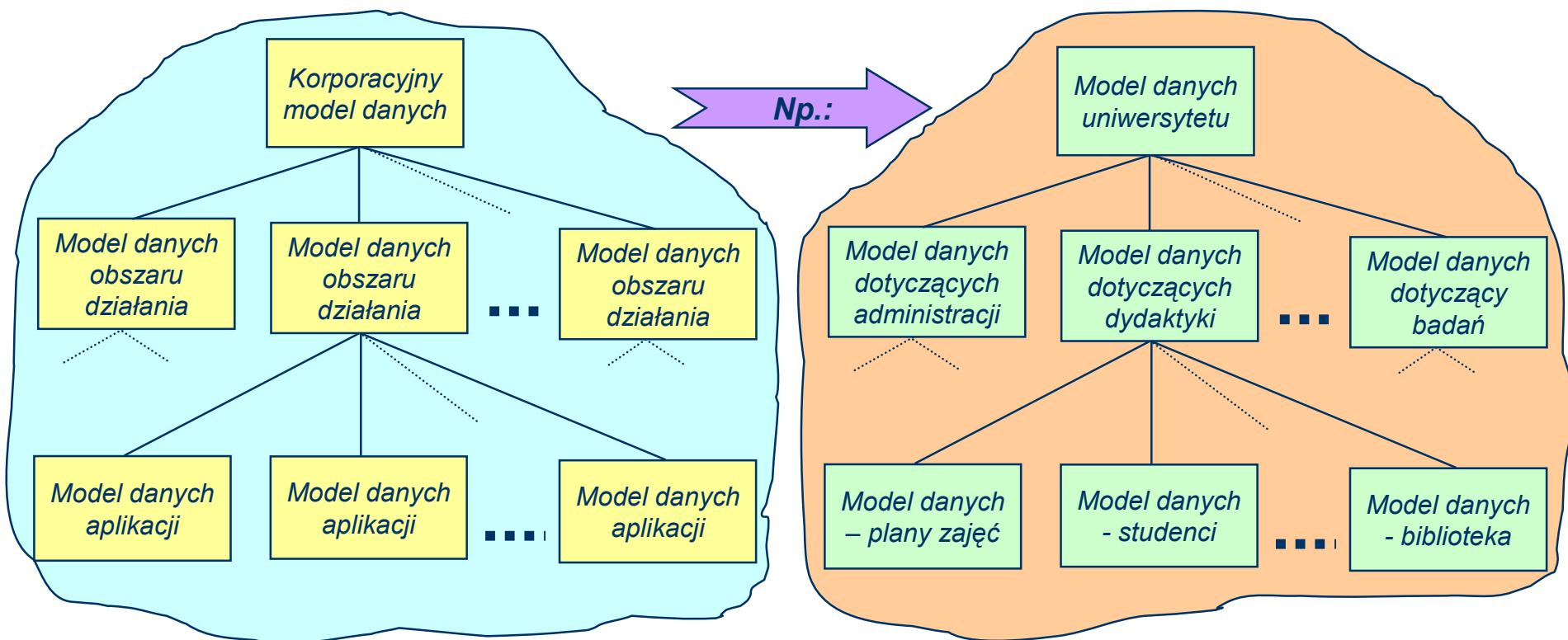


- * We współczesnych systemach podsystemy składowe mogą być rozproszone na kilku maszynach znajdujących się w różnych miejscach

- 
- ★ W starszych aplikacjach wszystkie warstwy były tworzone przy użyciu jednego narzędzia – języka programowania wysokiego poziomu najczęściej trzeciej generacji (np.. Cobol, Clipper).
 - ★ Od kilku (kilkunastu) lat poszczególne warstwy systemu tworzone są przez różne narzędzia:
 - narzędzia graficznego interfejsu użytkownika,
 - języki czwartej generacji (stosowane do kodowania reguł i logiki aplikacji),
 - systemy przetwarzania transakcji,
 - systemy zarządzania bazą danych.
 - oprogramowanie wspierające komunikację.

3.4. Modele danych aplikacji

- ★ Dane powinny być planowane i zarządzane – podstawowym narzędziem temu służącym są model danych
- ★ Model mogą być tworzone co najmniej na trzech poziomach:
 - korporacyjne modele danych (określają wymagania dla całego przedsiębiorstwa)
 - modele danych obszarów działania (biznesowych)
 - modele danych aplikacji.



3.4.1. Korporacyjne modelowanie danych i planowanie systemu informacyjnego

- * Odpowiednie zaplanowanie modelu danych dla każdego poziomu jest bardzo ważne**
- * Architektura systemu informacyjnego może być traktowana jako złożona z trzech warstw wynikających z rozróżnienia pomiędzy informacją, systemem informacyjnym i technologią informacyjną:**
 - architektura informacyjna składa się z działań związanych ze zbieraniem, przechowywaniem, rozdzielaniem i wykorzystywaniem informacji,
 - architektura systemów informacyjnych składa się z systemów niezbędnych do obsługi działań na elementach architektury informacyjnej,
 - architektura technologii informacyjnej składa się ze sprzętu, oprogramowania, urządzeń komunikacyjnych, wiedzy oraz umiejętności dostępnych w przedsiębiorstwie.
- * Istnieje związek pomiędzy poziomami modeli baz danych i poziomami architektury technologii informacyjnej.**
 - Strategia informacyjna – model korporacyjny danych
 - Strategia systemów informacyjnych – model danych obszarów działania
 - Strategia technologii informacyjnej – model danych aplikacji

3.5 Typy współczesnych baz danych

- ★ **Produkcyjne bazy danych** – obsługują standardowe funkcje przedsiębiorstwa.
Oferują tworzenie, odczytywanie, modyfikowanie i kasowanie danych.
 - Np.: baza o postępach studentów
- ★ **Bazy danych wspomagające decyzje** – służą do wyszukiwania informacji wspomagających decyzje w przedsiębiorstwie. Oferują tylko odczyt danych.
 - Np.: baza dotycząca rekrutacji na uniwersytecie
- ★ **Informacyjne bazy danych** – są to narzędzia dla użytkowników, mogą być aktualizowane, korzystają z baz produkcyjnych i wspomagających decyzje
 - Np.: wykładowca powinien mieć informacyjną bazę danych dotyczącą uczęszczania na zejścia studentów.

3.7 Rozwój zastosowań baz danych

- ★ **Hurtownie danych**

- Składnice danych (małe hurtownie)

- ★ **Internetowe bazy danych**

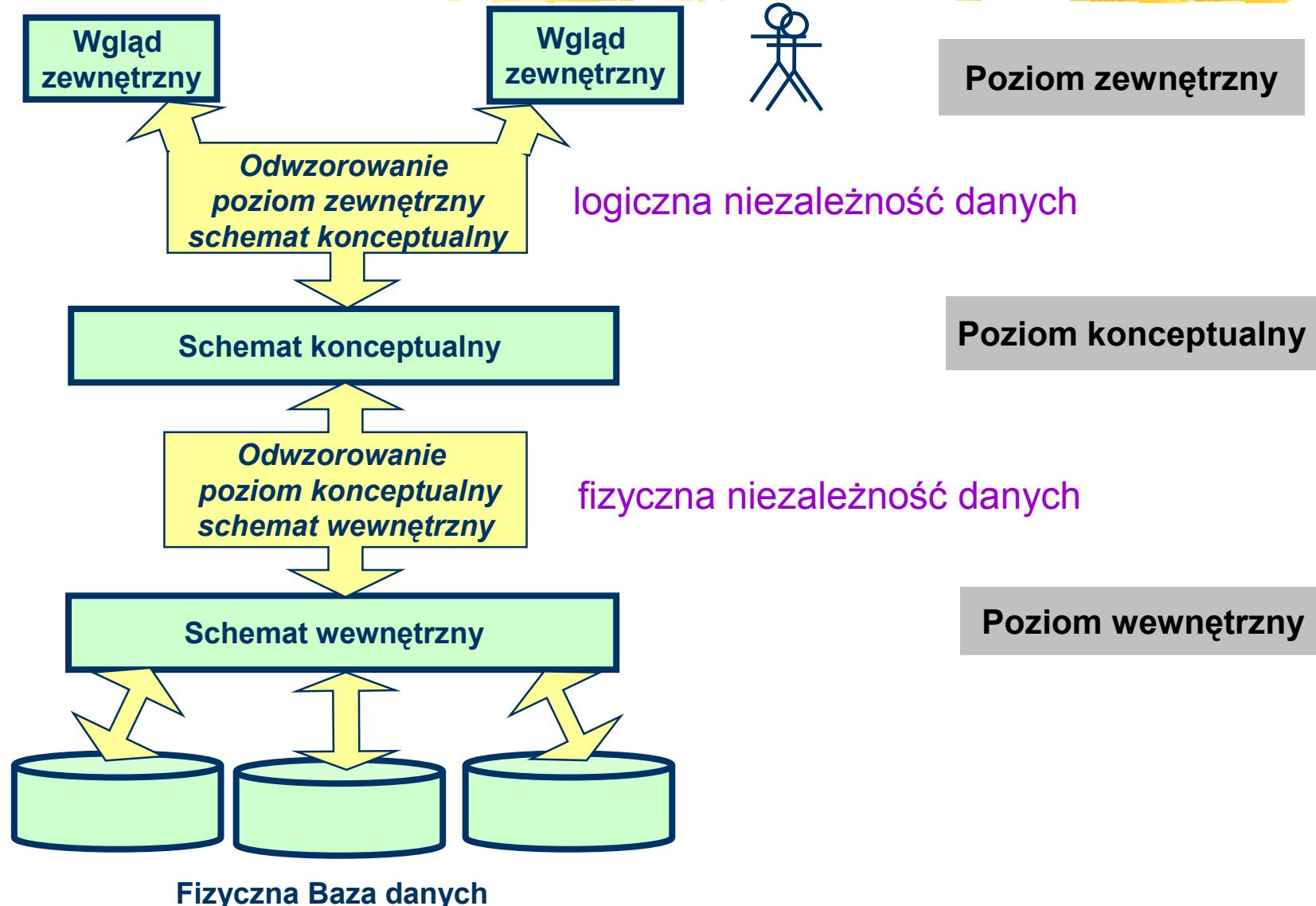
4. Warstwa zarządzania danymi

* TEMatyka

- Trzywarstwowa architektura SZBD (DBMS)
- Kluczowe funkcje SZBD
- Interfejsu do SZBD
- Jądro SZBD

4.1 Trzy warstwowa architektura SZBD

- ★ System zarządzania bazą danych jest pośrednikiem (buforem) pomiędzy programami użytkowymi, użytkownikiem końcowym i bazą danych.
- ★ W 1975 (ANSI-SPARC) zaproponował trzypoziomową architekturę SZBD:
 - **poziom zewnętrzny (użytkownika)** – opisuje jak użytkownicy widzą dane,
 - **poziom koncepcyjny (pojęciowy)** – opisuje widok wszystkich danych w bazie. Poziom ten opisuje logiczny widok baz danych, bez szczegółów dotyczących realizacji,
 - **poziom wewnętrzny (fizyczny)** – opisuje sposób przechowywania danych oraz metody dostępu do nich.
- ★ Pomiędzy warstwami istnieją dwa poziomy odwzorowania przekładające się na dwa poziomy niezależności danych:
 - **logiczna niezależność danych** – oznacza niewrażliwość schematów zewnętrznych na zmiany w schemacie koncepcyjnym,
 - **fizyczna niezależność danych** – oznacza niewrażliwość schematu koncepcyjnego na zmiany w schemacie fizycznym.



*W SZBD można wyróżnić: **jądro, interfejs i zestaw narzędzi**

4.2. Funkcje Systemu Zarządzania Bazą Danych

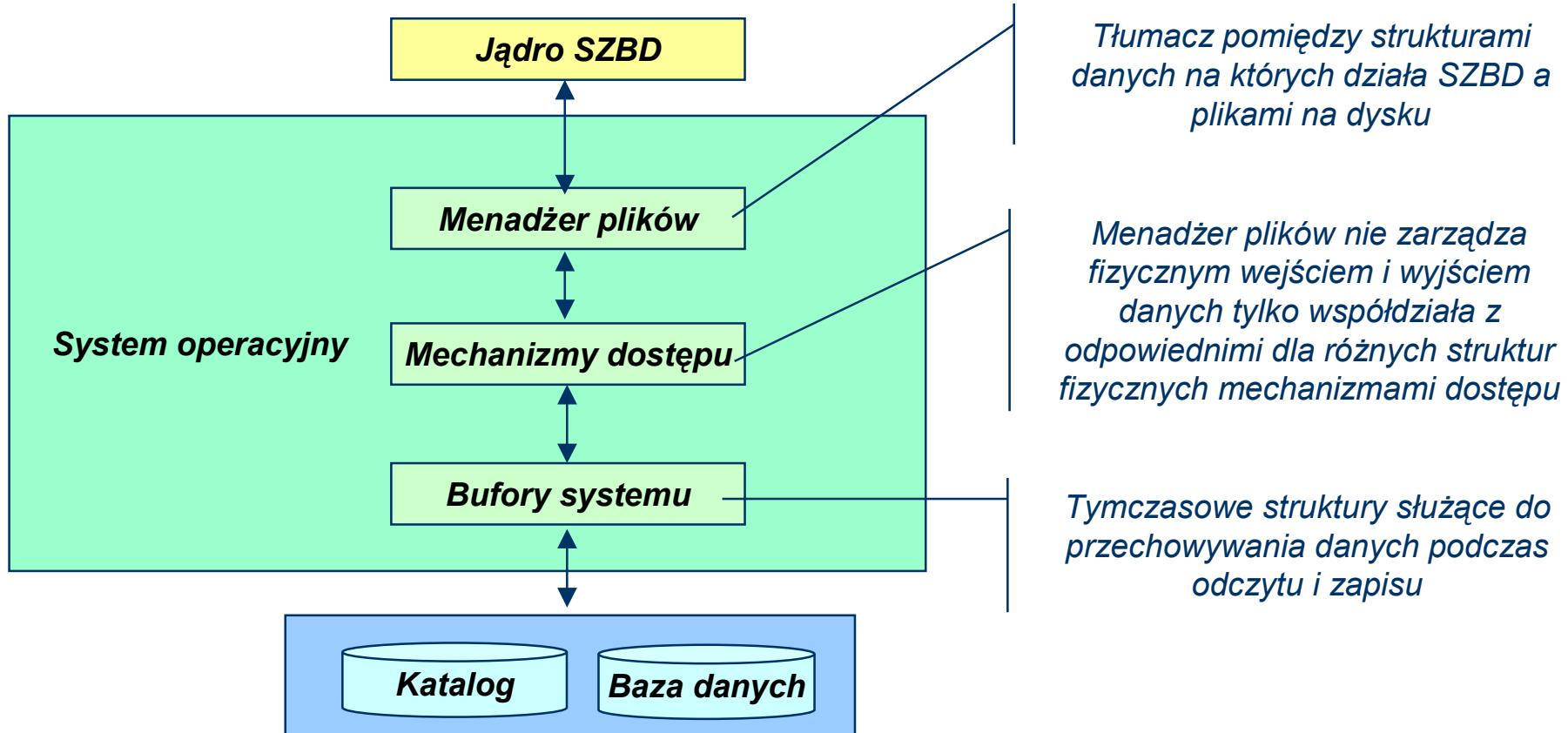
- ★ Funkcje CRUD (Create Read Update Delete),
- ★ Obsługa słownika danych – przechowywanie metadanych (danych o danych),
- ★ Zarządzanie transakcjami
- ★ Sterowanie współbieżnością
- ★ Odtwarzanie po awarii
- ★ Kontrola uprawnień użytkownika
- ★ Komunikacja danych (wymiana danych w systemie technologii informacyjnej)
- ★ Wymuszanie więzów integralności
- ★ Udostępnianie narzędzi do administrowania bazą danych (do importowania, eksportowania, monitorowania operacji, monitorowania wydajności)

4.3. Interfejs SZBD

- ★ Służy do powiązania jądra systemu z zestawem narzędzi
- ★ Składa się z subjęzyka bazy danych przeznaczonego do inicjowania funkcji SZBD. Składa się on z:
 - **języka definiowania danych (DDL)** służącego do tworzenia, usuwania i uzupełniania struktur danych oraz aktualizacji metadanych,
 - **języka operowania danymi (DML)** służącego do wykonywania operacji CRUD,
 - **języka integralności danych (DIL)** wykorzystywanego do określania więzów integralności,
 - **języka kontroli danych (DCL)** oferuje operacje przeznaczone dla administratora bazy
- ★ Przykładem takiego subjęzyka jest **SQL**

4.4. Jądro SZBD

- ★ Realizuje podstawowe funkcje zarządzania danymi (pkt 4.2.)

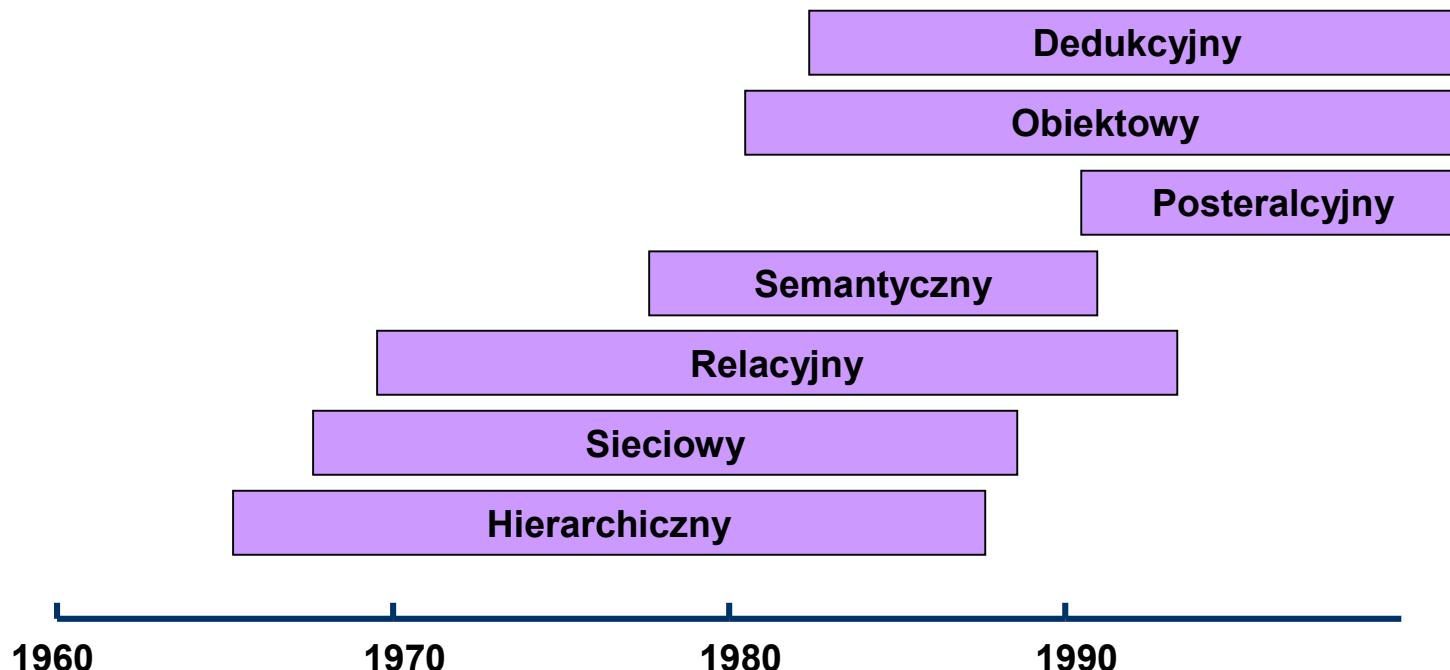


5. Model relacyjny

* TEMatyka:

- Modele danych – chronologiczny układ
- Definicja danych wg modelu relacyjnego
- Operowanie danymi
- Integralność danych w modelu relacyjnym

5.1. Chronologia opracowania modeli danych



- * Chronologia powstawania modeli danych nie odpowiada chronologii powstawania SZBD opartych na tych modelach

5.2. Ogólnie o modelu relacyjnym

- ★ Relacyjny model bazy danych został opublikowany w 1970 roku przez E.F. Codd'a. Model ten jest oparty jest na gałęziach matematyki zwanych teorią zbiorów i teorią predykatów.
- ★ Zasadą na której opiera się model relacyjny jest to, że typowa baza danych składa się z szeregu nieuporządkowanych tablic (relacji), którymi można manipulować używając nieproceduralnych operacji zwracających całe tablice.
- ★ Słowo „relacyjny” w modelu relacyjnym pochodzi od określenia relacja rozumianego jako specyficzna tablica.
- ★ Codd i inni teoretycy relacyjnych baz danych używają terminów:
„**relacja**”, „**atrybut**”, „**krotka**” tam gdzie większość stosuje pojęcia, odpowiednio:
„**tablice**”, „**kolumny**” i „**wiersze**” ale ma to swoje uzasadnienie.

5.3. Tworzenie definicji danych

- ★ Baza danych jest zbiorem struktur danych służącym do organizowania i przechowywania danych
- ★ W każdym modelu danych i w każdym SZBD musimy dysponować zbiorem reguł określających wykorzystanie struktur danych w aplikacjach bazy danych
- ★ Tworząc definicję danych używamy wewnętrznych struktur danych danego modelu danych w kontekście konkretnego zadania.
- ★ W modelu relacyjnym jest tylko jedna struktura danych – relacja (tablica)

5.3.1. Relacje (tablice)

- ★ **Relacja jest tabelą spełniającą następujące warunki:**
 - każda relacja w bazie danych ma jednoznaczna nazwę,
 - każda kolumna ma jednoznaczną nazwę w ramach relacji
 - wszystkie wartości w kolumnie są tego samego rodzaju (maja ta sama dziedzinę)
 - porządek kolumn w relacji nie jest istotny
 - wiersze mają unikalne wartości (są różne)
 - kolejność wierszy nie jest istotna
 - każde pole relacji winno zawierać elementarne wartości
- ★ **Tablice w modelu relacyjnym stosowane są do reprezentacji „różnych rzeczy” pochodzących ze świata rzeczywistego.**
- ★ **Każda tablica powinna reprezentować tylko jedną taką rzecz.**
- ★ **Rzeczy (jednostki) mogą być rzeczywistymi obiektami bądź zdarzeniami.**
Np. rzeczywistym obiektem może być klient, przedmiot z inventarza lub faktura.
Przykładami zdarzeń mogą być wizyty pacjentów, zlecenia, czy rozmowy telefoniczne.

Relacje – przykłady

Moduły			
Nazwa Modułu	Poziom	Kod Kursu	Nr Prac
Systemy relacyjnych baz danych	1	CSD	244
Projektowanie relacyjnych baz danych	1	CSD	244
Dedukcyjne bazy danych	4	CSD	445
Obiektowe bazy danych	4	CSD	445
Rozproszone bazy danych	2	CSD	247

Wykładowcy		
Nr Prac	Nazwisko Prac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A

Kursy	
Kod	Nazwa Modułu
CSD	Systemy relacyjnych baz danych Projektowanie relacyjnych baz danych Dedukcyjne bazy danych Obiektowe bazy danych Rozproszone bazy danych
BSD	Wprowadzenie do biznesu Podstawy księgowości

Relacje – przykłady

Studenci			
Nazwisko_Imie	MiejsceKszta	Rok	RokUr
Kozłowski Zbigniew	ED1084	1	1985
Szubart Joanna	ED1152	1	1986
Kluz Daniel	I21342	2	1982
Balawender Robert	MD8856	2	1982
Legień Krystyna	EZ9820	3	1983

krotka, wiersz, rekord

atrybut
kolumna

pole

kolumny = atrybuty

wiersze = krotki

liczba kolumn = stopień tabeli

liczba wierszy = liczebność tabeli

5.3.2. Klucz główny (pierwotny)

- ★ Model relacyjny nakazuje, żeby każdy wiersz w tablicy był unikalny. Jeśli pozwoli się na powtarzające się wiersze, wtedy dla programu bazy danych nie będzie istniał żaden sposób na jednoznaczne określenie pozycji danego wiersza.
- ★ Unikalność w tablicy zapewnia się wyznaczając klucz główny.
- ★ **Klucz główny to kolumna (lub grupa kolumn), która jednoznacznie identyfikuje każdy wiersz tabeli.**
- ★ **Każda tablica musi mieć dokładnie jeden klucz główny.**
- ★ **Wszystkie kolumny zawierające unikalne wartości nazywa się kluczami potencjalnymi (kandydującymi).**
- ★ **Spośród nich wybiera się jeden klucz główny, a pozostałe klucze nazywa się kluczami alternatywnymi.**
- ★ **Klucz prosty składa się tylko z jednej kolumny, natomiast klucz złożony zawiera dwie lub więcej kolumn.**

Wybór klucza głównego

* Klucz kandydujący musi mieć dwie cechy:

- jednoznaczność
- nie może być mieć wartości null

* Decyzja o wyborze klucza głównego powinna opierać się na:

- zasadzie minimalności (klucz powinien zawierać najmniejszą niezbędną ilość kolumn),
- stabilności (klucz powinien rzadko ulegać zmianom)
- prostoty / intuicyjności (klucz powinien być zarówno prosty jak i łatwy do zrozumienia przez użytkowników)

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A

Zawartość tabeli sugeruje unikalność każdego z atrybutów. Jednak z doświadczenia wiemy, iż Nazwiska pracowników mogą się powtórzyć. Podobnie status (typ pracownika) może być jednakowy dla kilku osób. Dlatego jedynie NrPrac nadaje się na klucz główny

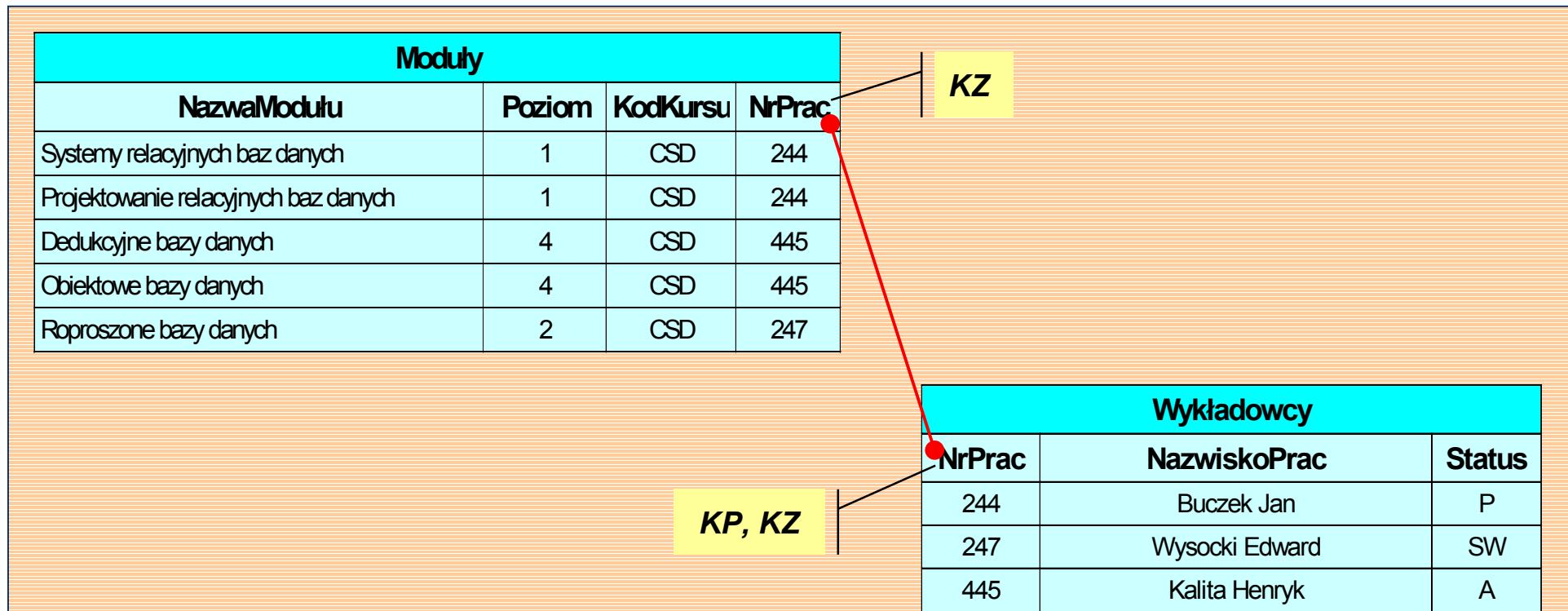
- * W wielu sytuacjach jako klucz główny najlepiej obrać jakiś arbitralny, statyczny numer (np.. numer pracownika, numer zlecenia, indeks, itp..), a unikać obierania kolumn zawierających opis tekstowy.
- * Do kluczy głównych, nie należy stosować liczb rzeczywistych, gdyż są one niedokładne

5.3.3. Dziedzina

- ★ Zbiór z którego kolumna może czerpać swoje wartości.
- ★ Dziedziny można traktować jako zdefiniowane przez użytkownika typy kolumn, nakładające na kolumnę reguły, do których muszą się stosować wartości zawarte w kolumnie, a także definiujące operacje, które można na kolumnach wykonywać.
- ★ Np.: dziedziną numerów pracownika jest zbiór możliwych numerów pracownika, dziedziną kolumny data urodzenia będą możliwe daty urodzenia.

5.3.4. Klucze obce (zewnętrzne)

- ✿ **Klucz obcy to taka kolumna (lub grupa kolumn), która zawiera odnośniki do klucza głównego z innej tablicy.**
- ✿ **Klucze główne, chociaż są specyfiką pojedynczych tablic, stanowią niezbędny element w definiowaniu powiązań między tablicami.**
- ✿ **Ważne jest, żeby klucz główny i klucz obcy miały to samo znaczenie i posiadały tą samą dziedzinę.**



5.3.5. Składnia definicji danych

- ★ Nie istnieje uzgodniona składnia wyrażania struktury relacji
- ★ Np.: dla bazy akademickiej

Dziedziny

NazwyModułów: CHARACTER(30)
Poziomy: INTEGER: {1, 2, 4}
KodyKursów: CHARACTER (3)
NryPrac: INTEGER
Statusy: CHARACTER: {P, SW, A, L, As, W}
NazwiskaPrac: CHARACTER(20)

Relacja MODUŁY

Atrybuty

NazwaModułu: NazwaModułu
Poziom: Poziomy
KodKursu: KodyKursów
NrPrac: NryPrac
Klucz główny NazwaModułu
Klucz obcy NrPrac do Wykładowcy

Relacja WYKŁADOWCY

Atrybuty

NrPrac: NryPrac
NazwiskoPrac: NazwiskaPrac
Status: Statusy
Klucz główny NrPrac

- ★ Jak opisuje się w MS ACCESS?

Przykład relacyjnej bazy danych

KLIENCI	<u>Id_klienta</u>	<u>Nazwa_firmy</u>	<u>Adres</u>	<u>Miasto</u>		
ZAMÓWIENIA	<u>Nr_zam</u>	<u>Data_zam</u>	<u>Id_klienta</u>	<u>Wartość_zam</u>		
POZYCJE_ZAMÓWIEŃ	<u>Nr_zam</u>	<u>Nr_katalog_towaru</u>	<u>Ilość_zam</u>			
TOWARY						
	<u>Nr_katalog_towaru</u>	<u>Rodzaj_towaru</u>	<u>Nazwa_producenta</u>	<u>Model</u>	<u>Cena_sprzed</u>	<u>Ilość_w_mag</u>
DOSTAWCY	<u>Id_dostawcy</u>	<u>Nazwa_firmy</u>	<u>Adres</u>	<u>Miasto</u>	<u>Kraj</u>	
OFERTY_DOSTAWCÓW	<u>Id_dostawcy</u>	<u>Nr_katalog_towaru</u>	<u>Cena</u>			
DOSTAWY	<u>Nr_katalog_towaru</u>	<u>Id_dostawcy</u>	<u>Data_zam</u>	<u>Data_przyjęcia</u>	<u>Ilość_zam</u>	

★ Dla każdego atrybutu (kolumny) powinna być określona dziedzina

5.4. Operowanie danymi

* Przetwarzanie danych wymaga:

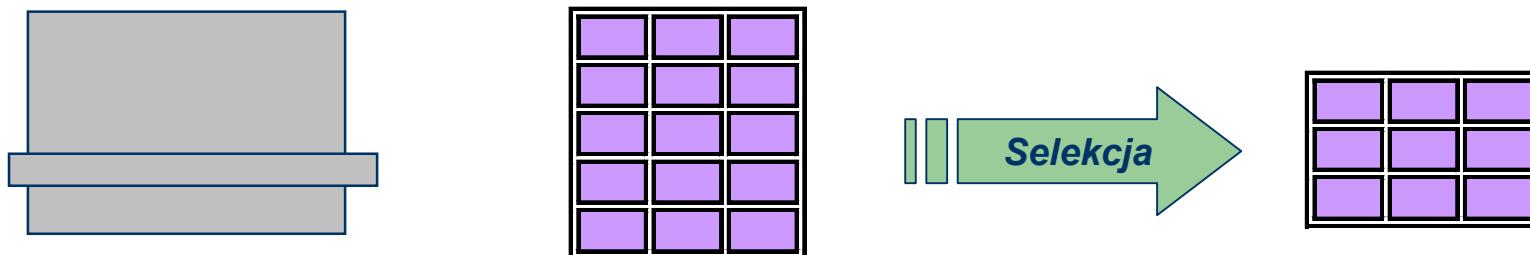
- wstawiania danych do relacji
- usuwania danych z relacji
- poprawiania danych w relacji
- wyszukiwania danych w relacji (służy do tego algebra relacyjna)

5.4.1. Algebra relacyjna

- ★ Algebra relacyjna jest zbiorem ośmiu operatorów.
- ★ Podstawowe operacje algebry relacji to:
 - wybór (selekcja, ograniczenie, restrykcja),
 - rzut (projekcja),
 - złączenie,
 - iloczyn,
 - suma,
 - przecięcie,
 - różnica,
 - iloraz.
- ★ Każdy operator bierze jedną lub więcej relacji jako argument i produkuje jedną relację jako wynik.
- ★ Algebra relacji jest proceduralnym językiem zapytań

Wybór (selekcja, restrykcja, filtrowanie)

- ★ Selekcja jest operatorem, który bierze jedną relację jako swój argument i produkuje w wyniku jedną relację w której umieszcza tylko wiersze spełniające zadany warunek (warunki).



- ★ Składnia operatora selekcji jest następująca:

RESTRICT <nazwa tabeli> [WHERE <warunek>] → <tabela wynikowa>

Moduły			
NazwaModułu	Poziom	KodKursu	NrPrac
Systemy relacyjnych baz danych	1	CSD	244
Projektowanie relacyjnych baz danych	1	CSD	244
Dedukcyjne bazy danych	4	CSD	445
Obiektowe bazy danych	4	CSD	445
Roproszone bazy danych	2	CSD	247

RESTRICT Moduły WHERE Poziom=4 → Poziom4

Poziom 4			
NazwaModułu	Poziom	KodKursu	NrPrac
Dedukcyjne bazy danych	4	CSD	445
Obiektowe bazy danych	4	CSD	445

Przykład:

drużyna	zawodnik	ile strzelił
Lechia	Kowalski	2
Lechia	Nowak	0
Arka	Aliński	0
Lechia	Kowalski	1
Lechia	Nowak	1
Pogoń	Wielgus	1

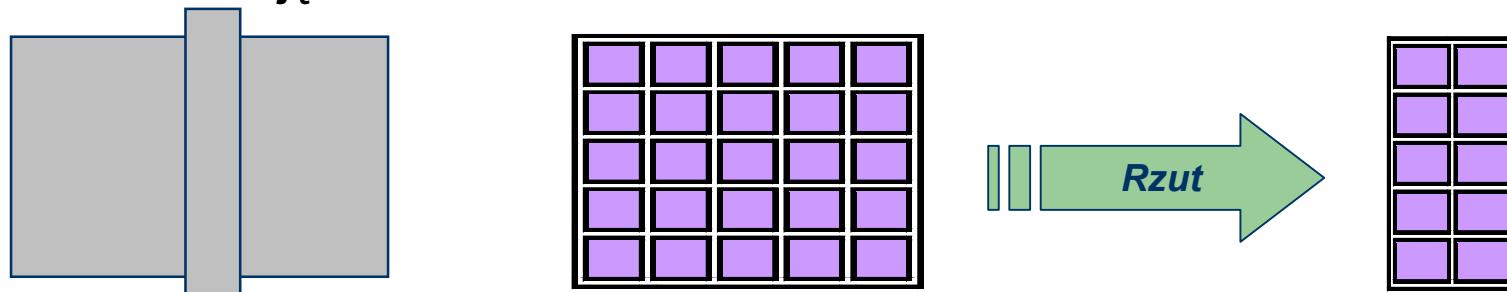
ile-strzelił > 0
→

drużyna	zawodnik	ile strzelił
Lechia	Kowalski	2
Lechia	Kowalski	1
Lechia	Nowak	1
Pogoń	Wielgus	1

Wybór zachowuje ilość kolumn a zmniejsza ilość wierszy.

Rzut

- ★ Operator rzutu wyciąga żądane atrybuty (kolumny) z jednej relacji i tworzy z nich nową relację



- ★ Składnia operatora rzutu jest następująca:

PROJECT <nazwa tabeli> (<lista kolumn>) → <tabela wynikowa>

PROJECT Moduły (NazwaModulu) → NazwMod

Moduły			
NazwaModułu	Poziom	KodKursu	NrPrac
Systemy relacyjnych baz danych	1	CSD	244
Projektowanie relacyjnych baz danych	1	CSD	244
Dedukcyjne bazy danych	4	CSD	445
Obiektowe bazy danych	4	CSD	445
Roproszone bazy danych	2	CSD	247



NazwMod
NazwaModułu
Systemy relacyjnych baz danych
Projektowanie relacyjnych baz danych
Dedukcyjne bazy danych
Obiektowe bazy danych
Roproszone bazy danych

Przykład:

drużyna	zawodnik	ile strzelił
Lechia	Kowalski	2
Lechia	Nowak	0
Arka	Aliński	0
Lechia	Kowalski	1
Lechia	Nowak	1
Pogoń	Wielgus	1

[zawodnik, ile-strzelił]

zawodnik	ile strzelił
Kowalski	2
Nowak	0
Aliński	0
Kowalski	1
Nowak	1
Wielgus	1

Wybór zachowuje ilość wierszy a zmniejsza ilość kolumn.

Iloczyn

- ★ Operator iloczynu tworzy z dwóch relacji jedną która składa się ze wszystkich możliwych kombinacji wierszy relacji wejściowych.
- ★ Iloczyn w praktyce jest używany bardzo rzadko.
- ★ Operator iloczynu wykorzystany jest do zdefiniowania operatorałączenia.
- ★ Składnia operatora iloczynu jest następująca:

`PRODUCT <tabela1> WITH <tabela2> → <tabela wynikowa>`

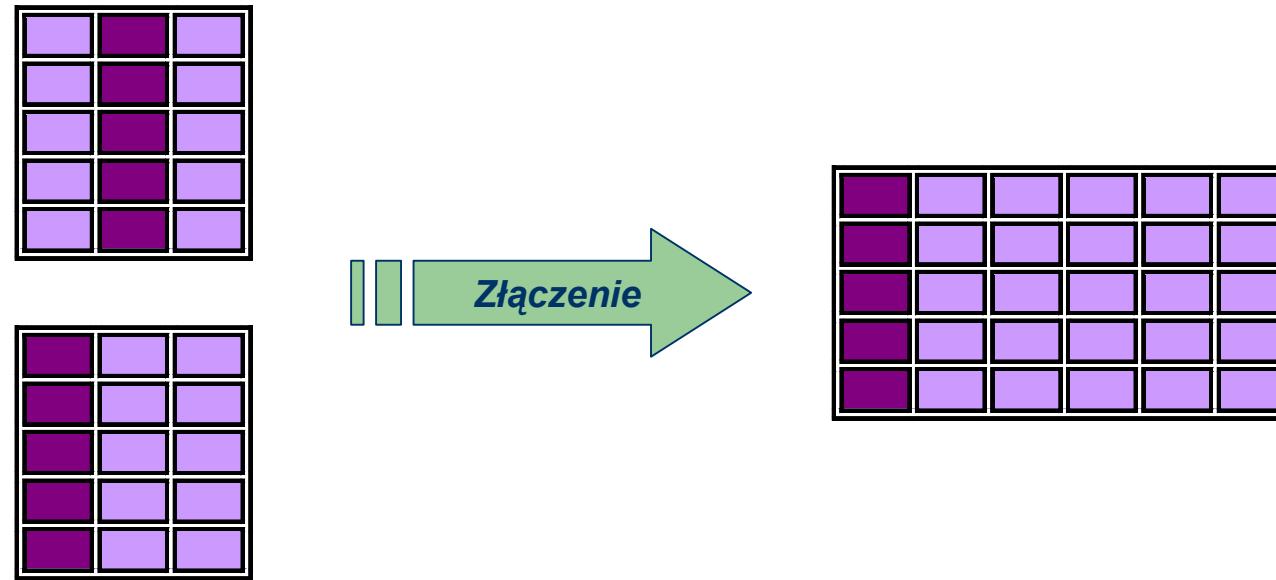
Moduły				Wykładowcy			
NazwaModułu	Poziom	KodKursu	NrPrac	NrPrac	NazwiskoPrac	Status	
Systemy relacyjnych baz danych	1	CSD	244	244	Buczek Jan	P	
Roproszone bazy danych	2	CSD	247	247	Wysocki Edward	SW	
				445	Kalita Henryk	A	

PRODUCT Moduł WITH Wykładowcy → ModWyk

ModWyk							
NazwaModułu	Poziom	KodKursu	NrPrac	NrPrac	NazwiskoPrac	Status	
Systemy relacyjnych baz danych	1	CSD	244	244	Buczek Jan	P	
Systemy relacyjnych baz danych	1	CSD	244	247	Wysocki Edward	SW	
Systemy relacyjnych baz danych	1	CSD	244	445	Kalita Henryk	A	
Roproszone bazy danych	2	CSD	247	244	Buczek Jan	P	
Roproszone bazy danych	2	CSD	247	247	Wysocki Edward	SW	
Roproszone bazy danych	2	CSD	247	445	Kalita Henryk	A	

Złączenie

- ★ Operator złączenia bierze dwie relacje i łączy je w jedną.



- ★ Wyróżnia się:
 - równozłączenie
 - złączenie naturalne
 - Złączenie zewnętrzne

Równozłoczenie

- ★ Operator równozłączzenia jest iloczynem kartezjańskim, po którym jest wykonywana selekcja (wybór) wybierająca tylko te wiersze, w których wartości w kolumnach złączenia są takie same.
- ★ Składnia operatora równozłączzenia jest następująca:

EQUIJOIN <tabela1> WITH <tabela2> → <tabela wynikowa>

Moduły			
NazwaModułu	Poziom	KodKursu	NrPrac
Systemy relacyjnych baz danych	1	CSD	244
Projektowanie relacyjnych baz danych	1	CSD	244
Rozproszone bazy danych	2	CSD	247

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A

EQUIJOIN Wykładowcy WITH Moduł → ModWykR

ModWykR						
NazwaModułu	Poziom	KodKursu	NrPrac	NrPrac	NazwiskoPrac	Status
Systemy relacyjnych baz danych	1	CSD	244	244	Buczek Jan	P
Projektowanie relacyjnych baz danych	1	CSD	244	244	Buczek Jan	P
Rozproszone bazy danych	2	CSD	247	247	Wysocki Edward	SW

Złączenie naturalne

- ★ Operator złączenia naturalnego jest iloczynem kartezjańskim, po którym jest wykonywana selekcja (wybór) wybierająca tylko te wiersze, w których wartości w kolumnach złączenia są takie same, oraz rzut usuwający powtarzające się kolumny
- ★ Składnia operatora złączenia naturalnego jest następująca:

`JOIN <tabela1> WITH <tabela2> → <tabela wynikowa>`

Moduły			
NazwaModułu	Poziom	KodKursu	NrPrac
Systemy relacyjnych baz danych	1	CSD	244
Projektowanie relacyjnych baz danych	1	CSD	244
Rozproszone bazy danych	2	CSD	247

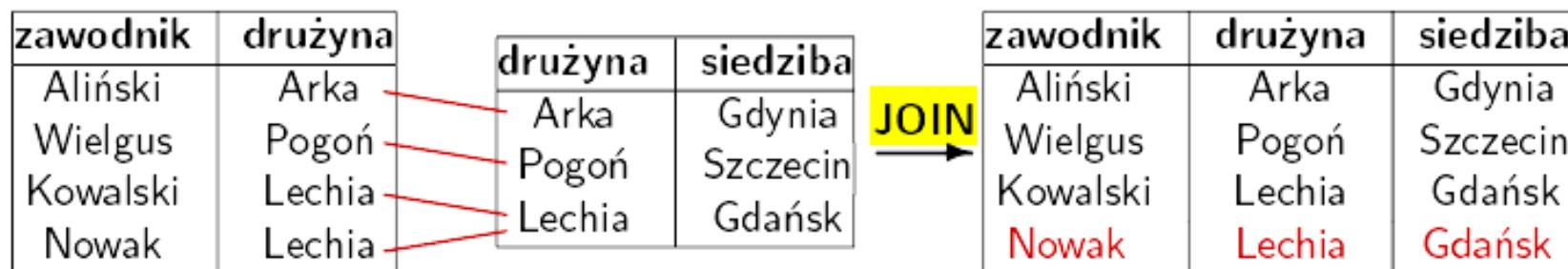
Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A

`JOIN Wykładowcy WITH Moduł → ModWykZN`

ModWykZN					
NazwaModułu	Poziom	KodKursu	NrPrac	NazwiskoPrac	Status
Systemy relacyjnych baz danych	1	CSD	244	Buczek Jan	P
Projektowanie relacyjnych baz danych	1	CSD	244	Buczek Jan	P
Rozproszone bazy danych	2	CSD	247	Wysocki Edward	SW

Złączenie tabelek

Przykład:



Atrybuty tabelki wynikowej = atrybuty obu tabel argumentowych
(bez powtórzeń)

Encje tabelki wynikowej = każda połączona z każdą
(bez sprzeczności na wspólnych atrybutach)

Złączenie zewnętrzne

- ★ Podobne do złączenia naturalnego, różnica polega na tym, iż pozostawiane są w relacji wynikowej wiersze nie posiadające odpowiedników w obu relacjach wyjściowych
- ★ Wyróżnia się złączenia zewnętrzne:
 - Lewostronne – zachowuje nie pasujące wiersze z tabeli będącej pierwszym argumentem,
 - Prawostronne – zachowuje nie pasujące wiersze z tabeli będącej drugim argumentem
 - Dwustronne – zachowuje nie pasujące wiersze z obu tabeli

Przykład złączenia zewnętrznego lewostronnego

The diagram illustrates a left outer join operation between two tables: 'Moduły' and 'Wykładowcy'. A large green arrow points from the 'Moduły' table to the resulting 'ModWykZZL' table.

Moduły Table:

Moduły			
NazwaModułu	Poziom	KodKursu	NrPrac
Systemy relacyjnych baz danych	1	CSD	244
Projektowanie relacyjnych baz da	1	CSD	244
Dedukcyjne bazy danych	4	CSD	445
Obiektowe bazy danych	4	CSD	445
Rozproszone bazy danych	2	CSD	247
Opracowanie baz danych	2	CSD	null
Administrowanie danymi	2	CSD	null

Wykładowcy Table:

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A
145	Zaborowski Jan	SW
447	Fusiarz Kamila	L

ModWykZZL Result Table:

ModWykZZL					
NazwaModułu	Poziom	KodKursu	NrPrac	NazwiskoPrac	Status
Systemy relacyjnych baz danych	1	CSD	244	Buczek Jan	P
Projektowanie relacyjnych baz danych	1	CSD	244	Buczek Jan	P
Dedukcyjne bazy danych	4	CSD	445	Kalita Henryk	A
Obiektowe bazy danych	4	CSD	445	Kalita Henryk	A
Rozproszone bazy danych	2	CSD	247	Wysocki Edward	SW
Opracowanie baz danych	2	CSD	null	null	null
Administrowanie danymi	2	CSD	null	null	null

Przykład złączenia zewnętrznego prawostronnego

The diagram illustrates an outer right join between two tables: 'Moduły' and 'Wykładowcy'. A green arrow points from the 'Moduły' table to the 'ModWykZZP' result table, indicating the direction of the join.

Moduły				Wykładowcy		
NazwaModułu	Poziom	KodKursu	NrPrac	NrPrac	NazwiskoPrac	Status
Systemy relacyjnych baz danych	1	CSD	244	244	Buczek Jan	P
Projektowanie relacyjnych baz da	1	CSD	244	247	Wysocki Edward	SW
Dedukcyjne bazy danych	4	CSD	445	445	Kalita Henryk	A
Obiektowe bazy danych	4	CSD	445	145	Zaborowski Jan	SW
Rozproszone bazy danych	2	CSD	247	447	Fusiarz Kamila	L
Opracowanie baz danych	2	CSD	null			
Administrowanie danymi	2	CSD	null			

ModWykZZP					
NazwaModułu	Poziom	KodKursu	NrPrac	NazwiskoPrac	Status
Systemy relacyjnych baz danych	1	CSD	244	Buczek Jan	P
Projektowanie relacyjnych baz danych	1	CSD	244	Buczek Jan	P
Dedukcyjne bazy danych	4	CSD	445	Kalita Henryk	A
Obiektowe bazy danych	4	CSD	445	Kalita Henryk	A
Rozproszone bazy danych	2	CSD	247	Wysocki Edward	SW
null	null	null	145	Zaborowski Jan	SW
null	null	null	447	Fusiarz Kamila	L

Przykład złączenia zewnętrznego dwustronnego

Moduły				Wykładowcy		
NazwaModułu	Poziom	KodKursu	NrPrac	NrPrac	NazwiskoPrac	Status
Systemy relacyjnych baz danych	1	CSD	244	244	Buczek Jan	P
Projektowanie relacyjnych baz da	1	CSD	244	247	Wysocki Edward	SW
Dedukcyjne bazy danych	4	CSD	445	445	Kalita Henryk	A
Obiektowe bazy danych	4	CSD	445	145	Zaborowski Jan	SW
Rozproszone bazy danych	2	CSD	247	447	Fusiarz Kamila	L
Opracowanie baz danych	2	CSD	null			
Administrowanie danymi	2	CSD	null			

ModWykZZD						
NazwaModułu	Poziom	KodKursu	NrPrac	NazwiskoPrac	Status	
Systemy relacyjnych baz danych	1	CSD	244	Buczek Jan	P	
Projektowanie relacyjnych baz danych	1	CSD	244	Buczek Jan	P	
Dedukcyjne bazy danych	4	CSD	445	Kalita Henryk	A	
Obiektowe bazy danych	4	CSD	445	Kalita Henryk	A	
Rozproszone bazy danych	2	CSD	247	Wysocki Edward	SW	
Opracowanie baz danych	2	CSD	null	null	null	
Administrowanie danymi	2	CSD	null	null	null	
null	null	null	145	Zaborowski Jan	SW	
null	null	null	447	Fusiarz Kamila	L	

Suma

- ★ Suma jest operatorem, który jako argumentów używa dwóch zgodnych relacji i produkuje jedną relację wyjściową, w której uwzględnia wszystkie różne wiersze z obu relacji.
- ★ Relacje zgodne to takie które mają identyczną strukturę i każda kolumna określona jest na tej samej dziedzinie
- ★ Składnia operatora sumy jest następująca:

`<tabela1> UNION <tabela2> → <tabela wynikowa>`

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A

Administracja		
NrPrac	NazwiskoPrac	Status
1010	Pawłowicz Maria	U
247	Wysocki Edward	SW

`Wykładowcy UNION Administracja → WykAdmSum`

WykAdmSum		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A
1010	Pawłowicz Maria	U

Przecięcie

- ★ Przecięcie jest operatorem, który jako argumentów używa dwóch zgodnych relacji i produkuje jedną relację wyjściową w której uwzględnia tylko identyczne wiersze z obu relacji.
- ★ Składnia operatora przecięcia jest następująca:

< tabela1 > **INTERSECTION** < tabela2 > → < tabela wynikowa >

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A

Administracja		
NrPrac	NazwiskoPrac	Status
1010	Pawłowicz Maria	U
247	Wysocki Edward	SW

Wykładowcy **INTERSECTION** Administracja → WykAdmPrz

WykAdmPrz		
NrPrac	NazwiskoPrac	Status
247	Wysocki Edward	SW

Różnica

- ★ Różnica jest operatorem, który jako argumentów używa dwóch zgodnych relacji i produkuje jedną relację wyjściową w której uwzględnia tylko te wiersze które występują jedynie w pierwszej relacji.
- ★ Składnia operatora przecięcia jest następująca:

<tabela1> DIFFERENCE <tabela2> → <tabela wynikowa>

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A

Administracja		
NrPrac	NazwiskoPrac	Status
1010	Pawłowicz Maria	U
247	Wysocki Edward	SW

Wykładowcy DIFFERENCE Administracja → WykAdm

WykAdm		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
445	Kalita Henryk	A

Iloraz

- ★ Iloraz jest operatorem, który jako argumentów używa dwóch relacji i produkuje jedną relację wyjściową. Jedna z tabel wejściowych musi być tabelą binarną (dwukolumnową) a druga unarną (jednokolumnową). Tabela unarna musi mieć dziedzinę zgodną z jedną z kolumn tabeli binarnej
- ★ Idea ilorazu polega na wybieraniu wartości z tabeli unarnej i porównywaniu z odpowiednią kolumną tabeli binarnej. Jeśli wszystkie wartości w tabeli unarnej są zgodne z taka samą wartością w tabeli binarnej to wartość jest wyprowadzana do tabeli wynikowej.

Chcemy znaleźć wspólny dzień w którym wykładowane są moduły: Systemy relacyjnych baz danych oraz Projektowanie relacyjnych baz danych

DniModułu	
NazwaModułu	Dzień
Systemy relacyjnych baz danych	04.02.2005
Projektowanie relacyjnych baz danych	02.02.2005
Projektowanie relacyjnych baz danych	04.02.2005
Obiektowe bazy danych	02.02.2005

ParyModułów	
NazwaModułu	
Systemy relacyjnych baz danych	
Projektowanie relacyjnych baz danych	

DniWspolne	
Dzień	
04.02.2005	

5.4.2. Algebra relacyjna - proceduralny język zapytań

- ★ Algebra relacyjna stanowi proceduralny język zapytań (ma właściwość domknięcia) – w celu wydobycia informacji z bazy danych określamy ciąg operatorów, w którym wynik z każdego kroku może być użyty jako argument wejściowy w następnych krokach.

Wypisz wszystkie moduły prowadzone przez Buczka Jana

1

```
RESTRICT Wykładowcy WHERE NazwiskoPrac = 'Buczek Jan' → Tab1  
JOIN Tab1 WITH Moduły ON NrPrac → Tab2  
PROJECT Tab2 (NazwaModułu) → Tab3
```

Wybiera z relacji Wykładowcy rekordy zawierające w polu *NazwiskoPrac* wyrażenie *Buczek Jan* i zapisuje w relacji *Tab1*

Wybiera z relacji *Tab2* kolumnę *NazwaModułu* i zapisuje w relacji *Tab3*

Łączy relacje *Tab1* oraz *Moduły* zapisując w tworzony relacji *Tab2* tylko te wiersze w których pola *NrPrac* w obu relacjach wejściowych są identyczne

2

```
JOIN Wykładowcy WITH Moduły ON NrPrac → Tab1  
RESTRICT Tab1 WHERE NazwiskoPrac = 'Buczek Jan' → Tab2  
PROJECT Tab2 (NazwaModułu) → Tab3
```

1a

```
PROJECT (JOIN Moduły WITH  
(RESTRICT Wykładowcy WHERE NazwiskoPrac = 'Buczek Jan') ON NrPrac) (NazwaModułu) → Tab1
```

Zagnieżdżona wersja zapisu 1. Wynik jednej operacji jest argumentem następnej

5.4.3. Rachunek relacyjny

- ★ Rachunek relacyjny stanowi alternatywę dla algebry relacyjnej
- ★ Rachunek relacyjny ma charakter nieproceduralny lub deklaracyjny (algebra proceduralny).
 - W rachunku formułujemy wyrażenie, które określa co ma być wyszukane, a nie jak to wyszukać
- ★ Rachunek relacyjny jest oparty na rachunku predykatów.
- ★ Istnieją dwa warianty rachunku relacyjnego:
 - rachunek relacyjny oparty na rekordach (podstawa SQL)
 - rachunek relacyjny oparty na dziedzinach (podstawa QBE – Query By Example)

Rachunek relacyjny na krotkach

- ★ Wyrażenie w tym rachunku ma postać:

```
RANGE OF <zmienna krotkowa> IS <nazwa tabeli>
GET <zmienna krotkowa>.<nazwa atrybutu>
INTO <tabela wynikowa>
WHERE <wyrażenie warunkowe>
```

Wybierz wykładowców p statusie SW

```
RANGE OF Kr1 IS Wykładowcy
GET Kr1.Status
INTO Tab1
WHERE Kr1.Status = SW
```

- ★ Instrukcja relacyjnego rachunku na krotkach może zawierać:
spójniki logiczne *and, or, not,*
kwantyfikatory egzystencjalne *Forsome* (istnieje)
kwantyfikatory uniwersalne *Forall* (dla każdego)

```
RANGE OF Kr1 IS Wykładowcy
GET Kr1.Status
INTO Tab1
WHERE Forsome Kr1 (Kr1.Status = SW)
```

Efekt jak wyżej

```
RANGE OF Kr2 IS Moduły
GET Kr2.Poziom
INTO Tab1
WHERE Forall Kr1 (Kr1.Status = SW)
```

Wydobędzie z bazy wszystkie poziomy
związane z modułami

5.4.4. Operacje dynamiczne na relacjach

* Istnieją trzy operacje dynamiczne (zarówno w algebrze relacyjnej jak i w rachunku):

- wstawianie

`INSERT <wartość>,<wartość>,...) INTO <nazwa tabeli>`

- usuwanie

`DELETE <nazwa tabeli> WITH <warunek>`

- modyfikowanie

`UPDATE <nazwa tabeli> WHERE <warunek> SET <nazwa kolumny>=<wartość>`

```
INSERT ('Systemy rozproszonych baz danych',2,CSD,247) INTO Moduły  
DELETE Moduły WITH Poziom = 1  
UPDATE Moduły WHERE Poziom = 2 SET Poziom = 1
```

5.5. Integralność danych

- ★ Integralność danych oznacza, iż baza danych stanowi odzwierciedlenie rzeczywistości.
- ★ Integralność zapewniają reguły integralności
- ★ W relacyjnym modelu danych istnieją dwa rodzaje wewnętrznych reguł integralności:
 - integralność encji (jednostek)
 - integralność referencyjna (odniesień)
- ★ Aby wyrazić wszystkie aspekty integralności stosuje się także tzw. integralności dodatkowe

5.5.1. Integralność encji (jednostek)

Zasada integralność encji mówi, że klucze główne nie mogą zawierać pól pustych

- ★ Uzasadnieni zasady jest oczywiste – nie da się jednoznacznie zidentyfikować wiersza w tablicy (krotki w relacji) jeśli klucze będą mogły być nieokreślone.
- ★ W przypadku kluczy złożonych żadna z indywidualnych kolumn nie może zawierać nieokreślonych (pustych) pól

5.5.2. Integralność referencyjna (odniesień)

Zasada integralności referencyjnej mówi, że klucz obcy nie może odnosić się do nieistniejących rekordów

* Z reguły wynika, że:

- do tablicy nie można dodać wiersza, którego wartość klucza obcego nie ma odpowiednika w tablicy odniesień,
- zmiana wartości klucza głównego w tablicy (lub usunięcie całego rekordu) nie może prowadzić do „osierocania” wierszy w innej tablicy (która poprzez klucz obcy wskazuje na modyfikowany klucz główny).

* Utrzymanie integralności referencyjnej wymaga określenia więzów propagacji (zdefiniowania co zrobić w przypadku modyfikacji klucza głównego z wierszami powiązanymi z nim). Możliwe są trzy podejścia:

- zakazać dokonywania zmian klucza głównego w tablicy odniesień jeżeli istnieją do niego powiązania – metoda restrykcyjna
- propagować zmiany do wszystkich tablic zawierających referencje do zmienianych wierszy (w przypadku usuwania rekordu usuwać rekordy powiązane) – podejście ufne
- Wyzerować wszystkie wartości kluczy obcych odnoszących się do usuwanych rekordów – podejście wyważone.

5.5.3. Integralność dodatkowa

- ★ Dodatkowe więzy integralności pochodzą ze środowiska modelowanego przez bazę danych

Wykład 3

Normalizacja

6. Normalizacja

* TEMatyka:

- Cel normalizacji
- Etapy normalizacji
- Związki zależności funkcyjne i niefunkcyjne
- Postacie normalne
- Diagramy zależności
- Akomodacja – przekształcanie diagramu zależności w schemat relacyjny

Opis modelu relacyjnego w notacji nawiasowej

- ★ Metoda pozwala w postaci skróconej opisać definicję schematu bazy.
- ★ W opisie najpierw podajemy nazwę relacji (tabeli) a następnie w nawiasie kolejne atrybuty. Jako pierwsze podaje się atrybuty stanowiące klucz główny jednocześnie podkreślając je.

Dla przykładu opisanego wcześniej zapis będzie miał postać:

Moduły (NazwaModułu, NrPrac)

Wykładowcy (NrPrac, Pracownik)

Oceny (NazwaModułu, NrStud, TypOceny, Ocena)

Studenci (NrStudenta, Student)

6.1. Cel normalizacji

- ★ **Normalizacja to proces upraszczania struktury bazy danych w taki sposób, aby osiągnęła ona postać optymalną.**
- ★ Normalizacja wykonuje się na etapie projektowania modelu fizycznego danych
- ★ **Dzięki normalizacji można uniknąć anomalii – błędów lub niespójności w bazie danych (w tym również redundancji).**
- ★ **Można wyróżnić 3 rodzaje anomalii:**
 - anomalie przy wstawianiu rekordu
 - dopisanie rekordu powoduje dezaktualizację innego pala,
 - anomalie przy usuwaniu rekordu
 - usunięcie wiersza powoduje usunięcie większej ilości informacji niż żeśmy zamierzali,
 - anomalie przy modyfikacji rekordu
 - zmiana jednego rekordu powoduje konieczność zmiany zapisów w innych rekordach.

Przykład

- ★ Dysponujemy bazą danych z informacjami o studentach, modułach oraz wykładowcach na uniwersytecie

Moduły						
Nazwa Modułu	NrPrac	Pracownik	NrStud	Student	Ocena	TypOceny
Systemy relacyjnych baz danych	244	Buczek Jan	34698	Kowalski H.	4.0	zal.
Systemy relacyjnych baz danych	244	Buczek Jan	34698	Kowalski H.	3.5	egz.
Systemy relacyjnych baz danych	244	Buczek Jan	37653	Nowak R.	3.0	zal.
Systemy relacyjnych baz danych	244	Buczek Jan	34610	Lech M.	5.0	zal.
Projektowanie relacyjnych baz danych	244	Buczek Jan	34698	Kowalski H.	3.0	zal.
Projektowanie relacyjnych baz danych	244	Buczek Jan	34698	Kowalski H.	4.0	egz.
Obiektowe bazy danych	445	Kalita Henryk	35785	Woś S.	3.5	egz.
Roproszone bazy danych	247	Wysocki Edward	34789	Janda K.	5.0	zal.

- Co będzie gdy usuniemy studenta **Wosia**? – stracimy informację o Obiektowych bazach danych i wykładowcy **Kalicie Henryku** – anomalia przy usuwaniu
- Co będzie gdy zmienimy wykładowcę Rozproszonych baz danych? – musimy zmodyfikować dwa pola: **NrPrac** oraz **Pracownik** – anomalia przy modyfikacji
- Co będzie gdy wpiszemy nowego studenta na moduł? – możliwe to będzie dopiero po uzyskaniu przez niego pierwszego zaliczenia – anomalia przy wstawianiu

6.2. Etapy normalizacji

- 1. Zebranie danych**
- 2. Przekształcenie do pierwszej postaci normalnej (1PN)**
- 3. Przekształcenie do drugiej postaci normalnej (2PN)**
- 4. Przekształcenie do trzeciej postaci normalnej (3PN)**

Po znormalizowaniu do 3 PN najczęściej tablice są już pozbawione anomali, jeżeli nie to należy je:

- ★ Przekształcić do postaci normalnej Boyce'a-Codd'a (BCNF)
- ★ Przekształcić do czwartej postaci normalnej (4PN)
- ★ Przekształcić do piątej postaci normalnej (5PN)

Proces normalizacji jest włożony. To znaczy, że każda wyższa postać normalna jest podzbiorem postaci niższej.

6.2.1 Rozkład relacji (tablic) a normalizacja

Proces przekształcania nieznormalizowanego zbioru danych w pełni znormalizowaną bazę danych nosi nazwę dekompozycji odwracalnej (rozkładu odwracalnego)

* Cechy dekompozycji odwracalnej

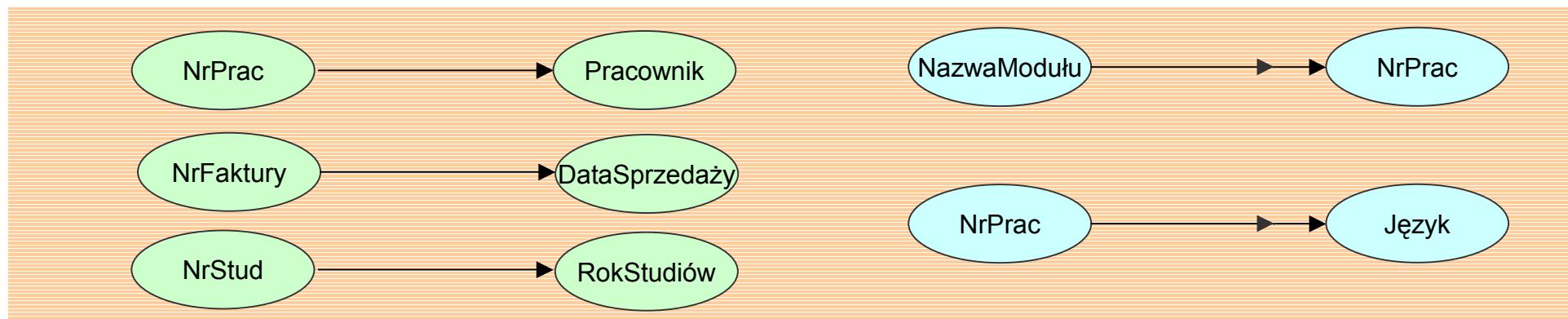
- usuwa redundancję z relacji
- można ją odwrócić przez naturalne złączenie
- powinna doprowadzić relacje do tzw. postaci normalnej
- nie powinna powodować utraty zależności istniejących w relacji pierwotnej

6.3. Zależności funkcyjne i niefunkcyjne (wielowartościowe)

- * Dwa elementy A i B są w związku zależności (związku determinowania), jeżeli pewne wartości elementu danych B występują zawsze z pewnymi wartościami elementu A.
- * **Zależność funkcyjna (determinowanie)** między elementami danych wskazuje kierunek związku. Jeżeli A determinuje B to związek jest od A do B nie odwrotnie

Element danych B jest **funkcyjnie zależny** od elementu danych A, jeżeli dla każdej wartości A istnieje jedna jednoznacznie określona wartość B

Element danych B jest **niefunkcyjnie zależny** od elementy danych A, jeżeli dla każdej wartości elementu danych A istnieje ograniczony zbiór wartości elementu B



- * 1 PN, 2 PN, 3 PN i BCNF dotyczą zależności funkcyjnych
- * 4 PN oraz 5 PN dotyczą zależności niefunkcyjnych

6.3.1. Zależności funkcyjne

- ★ Najważniejszy rodzaj więzów, z jakimi mamy do czynienia w modelu relacyjnym, dotyczy więzów jednoznaczności, które nazywa się również **zależnością funkcyjną**.
- ★ Nie istnieją metody pozwalające automatycznie określić zależności funkcyjne, aby to uczynić należy dokładnie przeanalizować znaczenie wszystkich atrybutów.

Filmy					
Tytuł	Rok	Długość	Typ	Producent	Gwiazda
Gwiezdne Wojny	1977	124	kolor	Fox	Carrie Fisher
Gwiezdne Wojny	1977	124	kolor	Fox	Mark Hamil
Gwiezdne Wojny	1977	124	kolor	Fox	Harrison Ford
Potężne Kaczory	1991	104	kolor	Disney	Emilio Estevez

W relacji można wyodrębnić zależności:

tytuł, rok → długość

tytuł, rok → typ

tytuł, rok → producent

Ponieważ wszystkie zależności mają po lewej stronie te same atrybuty można je zapisać:

tytuł, rok → długość, typ, producent

- ★ **Zależności funkcyjne (podobnie jak inne więzy) dotyczą schematu bazy danych, a nie określonej instancji**

Reguły dotyczące zależności funkcyjnych

- ★ Reguła przechodniości

Aksjomaty Armstronga

- ★ **Zbiór atrybutów określa funkcyjnie dowolny jego podzbiór** (Zwrotność)
 - np.: nr, nazwisko → nazwisko
- ★ **Jeżeli zbiór atrybutów X funkcyjnie określa zbiór atrybutów Y oraz Z jest innym zbiorem atrybutów wówczas suma zborów X i Z funkcyjnie określa sumę Y i Z**
 $X \rightarrow Y \Rightarrow X \cup Z \rightarrow Y \cup Z$ (Rozszerzenie)
 - np.: nr → nazwisko to zachodzi również nr, data → nazwisko, data
- ★ **Jeżeli $X \rightarrow Y$ i $Y \rightarrow Z \Rightarrow X \rightarrow Z$** (Przechodniość)
 - $X \rightarrow Y \Rightarrow X \cup Z \rightarrow Z$
 - $X \rightarrow Y$ i $X \rightarrow Z \Leftrightarrow X \rightarrow Y \cup Z$
 - **Jeżeli $A = \{A_1, A_2 \dots A_n\}$ to $X \rightarrow A \Leftrightarrow X \rightarrow A_1 \dots X \rightarrow A_n$**

Obliczanie domknięcia zbioru atrybutów

- ★ Bardzo często należy określić, które pojedyncze atrybuty są funkcjnie zależne od danego zbioru atrybutów.
- ★ **Algorytm X-domkniętości:**
 - Krok1: $X(0) = X$, $n=0$
 - Krok2: Jeżeli istnieje zależność $A \rightarrow B$ oraz $A \subset X(n)$ i $B \not\subset X(n)$ to $X(n+1) = X(n) \cup B$.
W przeciwnym wypadku zakończ algorytm
 - Krok 3: $n=n+1$ i wróć do kroku 2.

Rozważmy relację

Należności (nazwisko, ulica, miasto, województwo, data, wielkość)

Z następującymi zależnościami funkcyjnymi:

nazwisko \rightarrow ulica, miasto, województwo

nazwisko, data \rightarrow wielkość

miasto \rightarrow województwo

Wykonując algorytm X-domkniętości otrzymamy:

$X(0) = \{\text{nazwisko, data}\}$

$X(1) = \{\text{nazwisko, ulica, miasto, województwo, data}\}$

$X(2) = \{\text{nazwisko, ulica, miasto, województwo, data, wielkość}\}$

$X(3) = X(2)$

6.4. Pierwsza postać normalna 1PN

Relacja (tablica) jest w pierwszej postaci normalnej (1PN) wtedy i tylko wtedy, gdy każdy atrybut niekluczowy jest funkcjonalnie zależny od klucza głównego

- ★ Pierwsza postać normalna to warunek, że wszystkie wartości kolumn muszą być elementarne
- ★ Elementarne znaczy w tym przypadku niepodzielne. 1 PN wymaga, żeby dla każdej pozycji wiersz-kolumna w tablicy istniała tylko jedna wartość, a nie tablica lub lista wartości.
- ★ Jeśli w kolumnie przechowuje się całe listy wartości, wtedy trudno jest nimi operować.
- ★ 1 PN zabrania także istnienia powtarzających się grup, nawet jeśli miałyby być one złożone z kolumn elementarnych

Przejście do 1 PN – przykład

Moduły							
KP	NazwaModułu	NrPrac	Pracownik	NrStud	Student	Ocena	TypOceny
	Systemy relacyjnych baz danych	244	Buczek Jan	34698	Kowalski H.	4.0	zal.
						3.5	egz.
				34610	Lech M.	5.0	zal.
	Projektowanie relacyjnych baz danych	244	Buczek Jan	34698	Kowalski H.	3.0	zal.
						4.0	egz.
	Obiektowe bazy danych	445	Kalita Henryk	35785	Woś S.	3.5	egz.

* Atrybuty NrStud, Student, Ocena oraz TypOceny nie są funkcjonalnie zależne od klucza głównego, pozostałe są zależne. Należy więc utworzyć dwie tabele:

- jedna dla atrybutów zależnych
- drugą dla funkcjonalnie niezależnych atrybutów

Moduły			
NazwaModułu	NrPrac	Pracownik	

Oceny					
KP	NazwaModułu	NrStud	TypOceny	Student	Ocena
	Systemy relacyjnych baz danych	34698	zal.	Kowalski H.	4.0
	Systemy relacyjnych baz danych	34698	egz.	Kowalski H.	3.5
	Systemy relacyjnych baz danych	34610	zal.	Lech M.	5.0
	Projektowanie relacyjnych baz danych	34698	zal.	Kowalski H.	3.0
	Projektowanie relacyjnych baz danych	34698	egz.	Kowalski H.	4.0
	Obiektowe bazy danych	35785	egz.	Woś S.	3.5

Atrybuty NazwaModułu, NrStud i TypOceny utworzą klucz główny tabeli Oceny

6.5. Druga postać normalna 2PN

Relacja jest w drugiej postaci normalnej wtedy i tylko wtedy, gdy jest w 1PN i każdy atrybut niekluczowy jest w pełni funkcjonalnie zależny od klucza głównego

- ★ Tablica ma drugą postać normalną, jeśli jest w 1 PN i każda kolumna nie należąca do żadnego klucza potencjalnego jest całkowicie zależna od klucza głównego.
- ★ Innymi słowy, tablice powinny przechowywać dane dotyczące tylko jednej „rzeczy” (jednostki, obiektu, zdarzenia) oraz ta „rzecz” powinna być opisywalna przez jej klucz główny.
- ★ Tablica która jest w 1PN może nie być w 2PN tylko wtedy gdy posiada klucz główny złożony bo tylko wówczas któryś z atrybutów może być identyfikowany przez część klucza.

Przejście do 2 PN – przykład

Oceny					
KP	NazwaModułu	NrStud	TypOceny	Student	Ocena
	Systemy relacyjnych baz danych	34698	zal.	Kowalski H.	4.0
	Systemy relacyjnych baz danych	34698	egz.	Kowalski H.	3.5
	Systemy relacyjnych baz danych	34610	zal.	Lech M.	5.0
	Projektowanie relacyjnych baz danych	34698	zal.	Kowalski H.	3.0
	Projektowanie relacyjnych baz danych	34698	egz.	Kowalski H.	4.0
	Obiektowe bazy danych	35785	egz.	Woś S.	3.5

Tabela Oceny (niebieska) nie jest w 2PN gdyż pole Student nie zależy od całego klucza głównego a jedynie od atrybutu NrStud

Tabele w 2PN

Oceny					
KP	NazwaModułu	NrStud	TypOceny	Ocena	
	Systemy relacyjnych baz danych	34698	zal.	4.0	
	Systemy relacyjnych baz danych	34698	egz.	3.5	
	Systemy relacyjnych baz danych	34610	zal.	5.0	
	Projektowanie relacyjnych baz danych	34698	zal.	3.0	
	Projektowanie relacyjnych baz danych	34698	egz.	4.0	
	Obiektowe bazy danych	35785	egz.	3.5	

Studenci		
KP	NrStud	Student
	34698	Kowalski H.
	34610	Lech M.
	35785	Woś S.

Moduły			
KP	NazwaModułu	NrPrac	Pracownik
	Systemy relacyjnych baz danych	244	Buczek Jan
	Projektowanie relacyjnych baz danych	244	Buczek Jan
	Obiektowe bazy danych	445	Kalita Henryk

6.6. Trzecia postać normalna 3PN

Relacja jest w trzeciej postaci normalnej wtedy i tylko wtedy, gdy jest w 2PN i każdy atrybut niekluczowy jest bezpośrednio zależny od klucza głównego.

- ★ Tablica jest w trzeciej postaci normalnej jeśli jest w 2 PN i wszystkie kolumny nie należące do żadnego klucza potencjalnego są wzajemnie niezależne.

Przejście do 3 PN – przykład

Moduły		
NazwaModułu	NrPrac	Pracownik
Systemy relacyjnych baz danych	244	Buczek Jan
Projektowanie relacyjnych baz danych	244	Buczek Jan
Obiektowe bazy danych	445	Kalita Henryk

Jedynie tabela Moduły nie jest w 3PN gdyż pole Pracownik zależy od pola NrPrac a nie bezpośrednio od pola NazwaModułu.

Należy dokonać podziału tabeli

Tabele w 3 PN

Moduły	
NazwaModułu	NrPrac
Systemy relacyjnych baz danych	244
Projektowanie relacyjnych baz danych	244
Obiektowe bazy danych	445

Wykładowcy	
NrPrac	Pracownik
244	Buczek Jan
445	Kalita Henryk

Oceny				
KP	NazwaModułu	NrStud	TypOceny	Ocena
	Systemy relacyjnych baz danych	34698	zal.	4.0
	Systemy relacyjnych baz danych	34698	egz.	3.5
	Systemy relacyjnych baz danych	34610	zal.	5.0
	Projektowanie relacyjnych baz danych	34698	zal.	3.0
	Projektowanie relacyjnych baz danych	34698	egz.	4.0
	Obiektowe bazy danych	35785	egz.	3.5

Studenci	
NrStud	Student
34698	Kowalski H.
34610	Lech M.
35785	Woś S.

Metoda Bersteina

- ★ Dowolną relację można przekształcić do 3PN korzystając z metody Bernsteina:

Krok 1. Przekształć każdą zależność tak aby po prawej stronie był tylko jeden atrybut.

Krok 2. Wyeliminuj powtarzające się zależności

Krok 3. Przekształć zależność tak aby żaden podzbiór atrybutów stojący po lewej stronie nie określał prawej strony zależności.

Krok 4. Połącz zależności z takimi samymi lewymi stronami

Krok 5. Znajdź klucz dla relacji pierwotnej. Jeżeli żaden z kluczy nie jest zawarty w zbiorach z kroku poprzedniego to utwórz nowy zbiór z atrybutami klucza.

Krok 6. Jeżeli jakiś nowo utworzony zbiór jest rzutem (projekcją) innego, to wyeliminuj ten zbiór

- ★ Rozważmy relację

Magazyn (nr, mistrz, wydział, materiał, ilość, cena)

Z następującymi zależnościami funkcyjnymi:

nr → mistrz

mistrz → wydział

mistrz, materiał → ilość

materiał → cena

Krok 1. Zależności funkcyjne spełniają wymagania

Krok 2. Brak zależności redundancyjnych

Krok 3. Lewe strony są minimalne. Żaden podzbiór atrybutów z lewej strony nie określa prawej strony

Krok 4,5 . Otrzymujemy pięć relacji

R1 (nr, mistrz)

R2 (mistrz, wydział)

R3 (mistrz, materiał, ilość)

R4 (materiał, cena)

R5 (nr, materiał)

Krok 6. Żadna z otrzymanych relacji nie jest rzutem innej *nie zawiera się w niej)

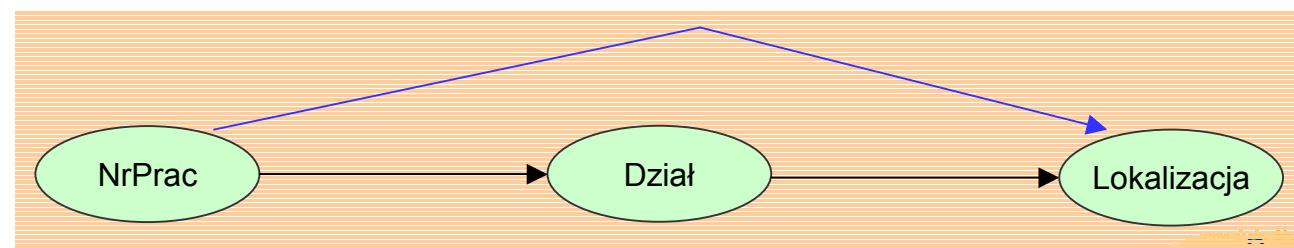
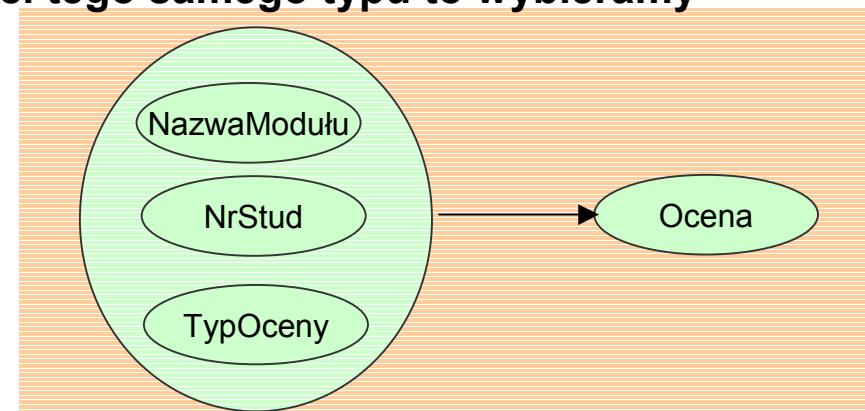
Magazyn					
Nr	Mistrz	Wydział	Materiał	Ilość	Cena
1	Malinowski	1	deski	3	80
2	Kowalski	1	parkiet	20	95
3	Malinowski	1	panele	50	35
4	Nowak	2	drzwi	1	850
5	Kowalski	1	drzwi	2	850

6.7. Diagramy zależności

- ★ **Klasyczna normalizacja** opisana jako proces rozkładu odwracalnego ma kilka wad:
 - wymaga aby zbiór danych był w pełni określony,
 - jest bardzo czasochłonna.
- ★ Alternatywą dla klasycznej normalizacji mogą być **diagramy zależności**.
- ★ **Zalety diagramów zależności:**
 - nie wymagają pełnego określenia danych
 - czytelny zapis graficzny

6.7.2. Pragmatyka rysowania diagramów zależności

- ★ Zależność między dwoma elementami danych można rysować tylko w jedną stronę (od A do B albo od B do A).
- ★ Jeżeli między elementami istnieje w jedną stronę zależność funkcyjna a w drugą niefunkcyjna to wybieramy kierunek zależności funkcyjnej.
 - Np.: w kierunku od NrPrac do NazwaWydziału jest zależność funkcyjna a w kierunku NazwaWydziału do NrPrac niefunkcyjna.
- ★ Jeżeli w obu kierunkach występują zależności tego samego typu to wybieramy kierunek, który dla nas jest wygodniejszy.
- ★ Zależności mogą być złożone – gdy złożenie kilku elementów determinuje jakiś inny element
- ★ Jeżeli A determinuje B a B determinuje C to mamy doczynienia z zależnością przechodnią. Wykrycie i usunięcie tego typu zależności jest ważnym elementem procesu normalizacji

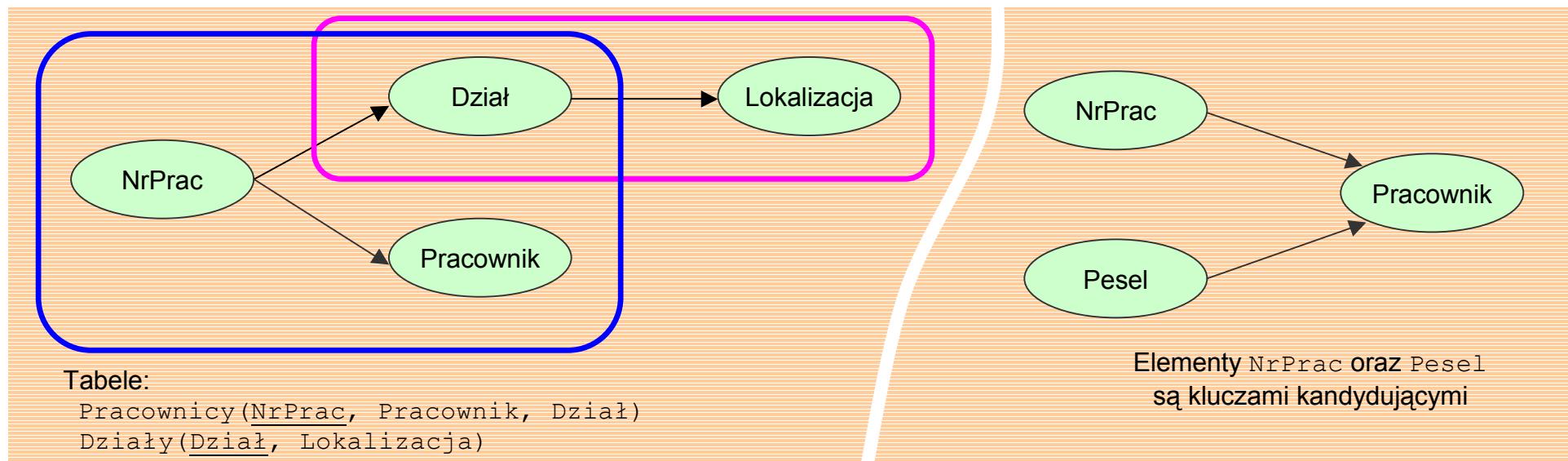


6.8 Akomodacja.

Akomodacja to proces przekształcania diagramu zależności w schemat relacyjny

Reguła Boyce'a-Codda

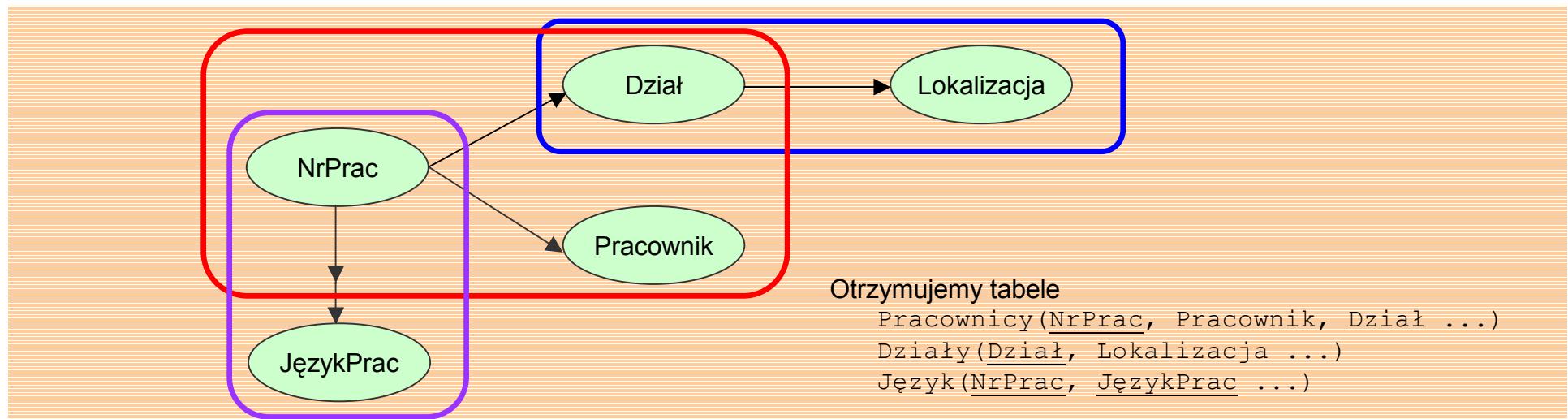
Każdy funkcjonalnie determinujący element staje się kluczem kandydującym tabeli. Wszystkie bezpośrednio zależne od niego elementy danych stają się niegłównymi atrybutami tabeli



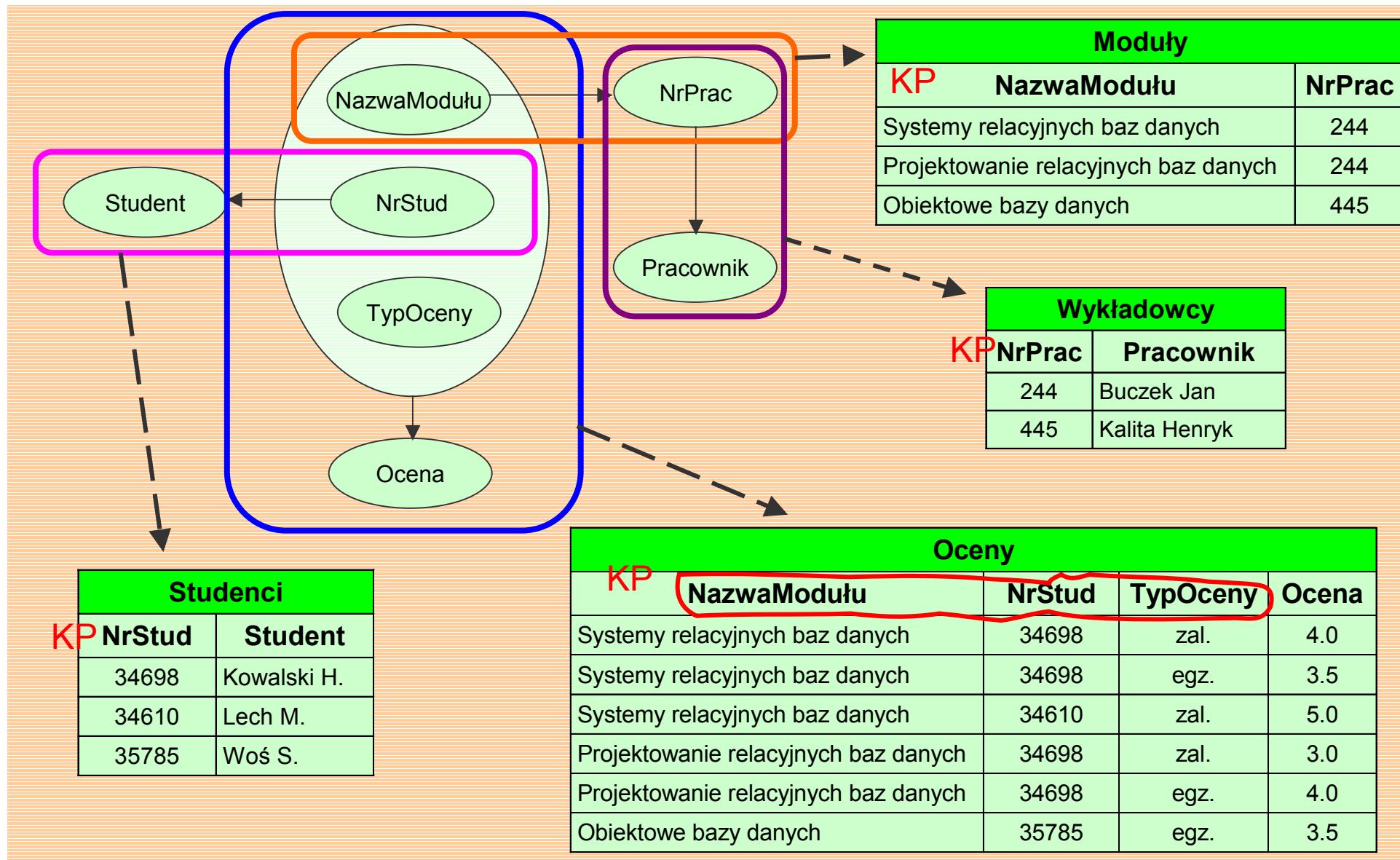
- * Liczba elementów determinujących (z których wychodzą strzałki) wskazuje liczbę wymaganych tabel
- * Element do którego wchodzi strzałka i z którego wychodzi strzałka stanowi klucz obcy

Każdy niefunkcyjnie determinujący element staje się częścią klucza głównego tabeli.

- ★ **Dokładnie tworzymy klucz główny z determinującego elementu danych i zależnych elementów danych wchodzących w skład związku niefunkcyjnego**



6.9. Rysowanie diagramów zależności i postacie normalne



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

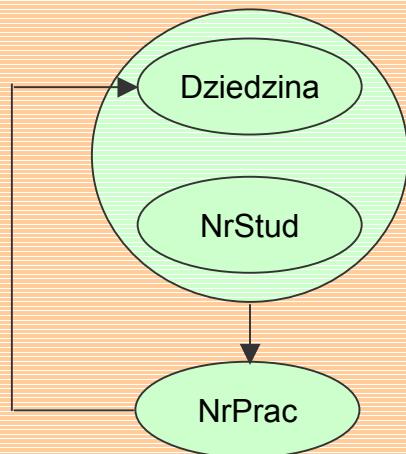
Relacja jest w postaci normalnej Boyce'a-Codda, jeśli tylko z nietrywialnych zależności wynika, że pewien nadklucz wyznacza funkcyjnie jakiś inny atrybut.

- ★ **Zależność funkcyjna nietrywialna** – zależność funkcyjna w której co najmniej jeden z atrybutów typu B znajduje się pośród atrybutów typu A
 - Np.: tytuł, rok → rok, długość
- ★ **Nadklucz** – zbiór atrybutów który zawiera klucz.

Przykład

* Założmy, iż mamy zamodelować sytuację:

- każdy student może specjalizować się w kilku dziedzinach,
- student ma jednego asystenta w każdej dziedzinie,
- każda dziedzina ma kilku asystentów, ale jeden asystent doradza tylko w jednej dziedzinie,
- każdy asystent doradza kilu studentom w jednej dziedzinie.

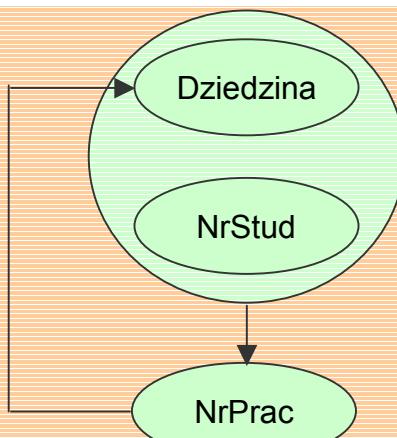


Specjalizacje (NrStud, Dziedzina, NrPrac)

Specjalizacje		
NrStud	Dziedzina	NrPrac
123456	Informatyka	234
234567	Systemy informacyjne	345
345678	Inżynieria oprogramowania	456

* Przedstawiony schemat spełnia wymogi 3PN

Przykład c.d.



Specjalizacje (NrStud, Dziedzina, NrPrac)

Specjalizacje		
NrStud	Dziedzina	NrPrac
123456	Informatyka	234
234567	Systemy informacyjne	345
345678	Inżynieria oprogramowania	456

★ Zauważmy że pomimo normalizacji do 3PN występują anomalie::

- zmiana specjalizacji przez studenta 123456 powoduje utratę informacji o NrPrac 234,
- nie można wstawić informacji o NrPrac 789, który jest asystentem z Informatyki tak długo jak długo jakiś student nie wybierze tej specjalności,
- usunięcie studenta 345678 powoduje jednoczesne usunięcie informacji o Nr 456.

★ Należy dokonać dekompozycji tabeli Specjalizacje:

■ Schemat 1

AsystenciStudenta (NrStudenta, NrPrac)

DziedzinyAsystenta (NrPrac, Dziedzina)

■ Schemat 2

AsystenciStudenta (NrStudenta, Dziedzina)

DziedzinyAsystenta (NrPrac, Dziedzina)

6.11. Czwarta postać normalna (4PN)

Relacja jest w czwartej postaci normalnej, wtedy i tylko wtedy gdy z nietrywialnych zależności wielowartościowych wynika, że pewien nadklucz wyznacza jakiś inny atrybut.

- ★ Relacja jest w 4PN, gdy zbiór atrybutów X określa wielowartościowo (niefunkcyjnie) Y to zachodzi jeden z następujących warunków:
 - Y jest puste lub zawiera się w X,
 - Suma zbiorów X i Y jest kompletnym zbiorem atrybutów dla danej relacji,
 - X zawiera klucz.
- ★ Aby przejść z 3 PN do 4 PN, szukamy tabel, które zawierają dwie lub więcej niezależnych zależności wielowartościowych.

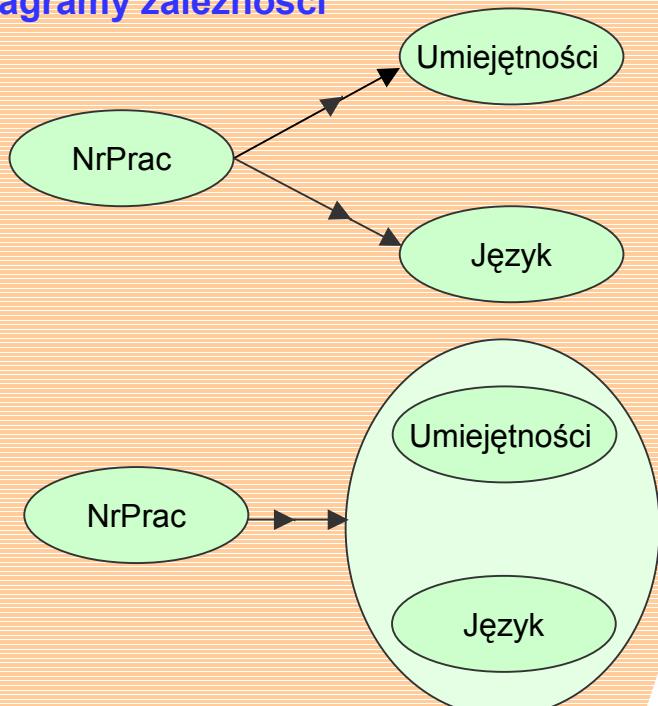
Przykład

★ Założmy, iż mamy przechowywać:

- informację o pracownikach,
- umiejętnościach pracowników,
- znajomości języków przez pracowników.

■ Pracownik może posiadać wiele umiejętności oraz znać kilka języków (umiejętności nie są związane ze znajomością języków).

Diagramy zależności



Tabele przed i po normalizacji do 4PN

Pracownicy		
NrPrac	Umiejętność	Język
123456	Obsługa komputera	angielski
123456	Obsługa komputera	francuski
123456	Prawo jazdy	angielski
234567	Prawo jazdy	niemiecki
234567	Obsługa komputera	angielski

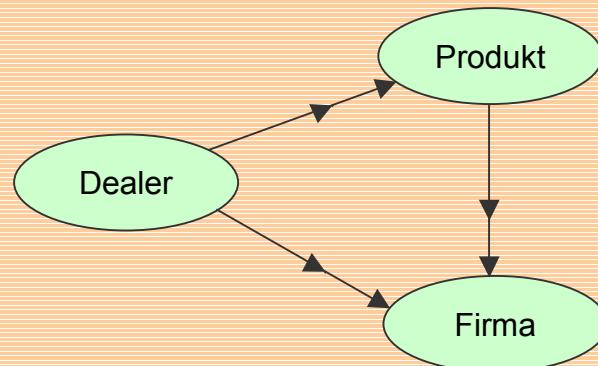
Umiejętności		Języki	
NrPrac	Umiejętność	NrPrac	Język
123456	Obsługa komputera	123456	angielski
123456	Prawo jazdy	123456	francuski
234567	Prawo jazdy	234567	niemiecki
234567	Obsługa komputera	234567	angielski

6.12. Piątą postać normalną (5PN)

Relacja jest w piątej postaci normalnej jeżeli jest w 4 PN i nie istnieje jej rozkład odwracalny na zbiór mniejszych tabel

* Założymy, iż mamy przechowywać:

- informację o dealerach samochodów,
- produkach sprzedawanych przez dealerów,



PunktySprzedaży		
Dealer	Producent	Typ
Auto Zbyt	Ford	osobowy
Auto Zbyt	Opel	dostawczy
Cztery Kółka	Ford	dostawczy
Cztery Kółka	Opel	osobowy

Dealerzy reprezentują firmy, firmy wytwarzają produkty i dealerzy sprzedają te produkty (diagram zależności). Nie można dokonać rozkładu struktury pokazanej na diagramie gdyż np.: Auto Zbyt sprzedaje samochody osobowe marki Ford i dostawcze marki Opel, a nie sprzedaje dostawczych fordów ani osobowych opel.

Wykład 5

SQL

Structured Query Language

7. SQL

* TEMATYKA:

- Podstawowe informacje o języku SQL
- Struktura języka SQL
- Wyszukiwanie informacji w tabelach
- Zakładanie tabel
- Wpisywania, aktualizacja, usuwanie danych z tabel

7.1. Podstawowe informacje o SQL

- ★ SQL stanowi najbardziej popularny mechanizm definiowania poleceń i modyfikacji w relacyjnych systemach baz danych (jest ratyfikowany jako standard języka relacyjnych baz danych).
- ★ Wszystkie DBMS powinny więc opierać się na tym standardzie.
- ★ SQL jest zaimplementowany w takich systemach baz danych (DBMS), jak: DB2, Oracle, InterBase, MySQL, dBase, Sybase, Informix, Paradox, MS SQL. Dzięki temu systemy informacyjne można przenosić na inne platformy.
- ★ Podstawowy rdzeń języka SQL jest implementacją algebry relacyjnej
- ★ Ujęty w standardzie ANSI/ISO w roku 1986
 - aktualizacje: SQL-89, SQL 2, SQL 3
- ★ Język SQL danej bazy, np. mySQL, zawiera:
 - polecenia SQL ujęte w standardzie
 - rozszerzenia standardu – polecenia specyficzne dla konkretnego systemu baz danych

Podstawowe informacje o SQL

- ★ SQL - Structured Query Language, strukturalny język zapytań
- ★ SQL jest językiem czwartej generacji, który został w ciągu wielu lat opracowany przez grupę badawczą IBM w późnych latach siedemdziesiątych. Język ten został stworzony dla relacyjnego systemu zarządzania bazą danych o nazwie DB2 (produkcie firmy IBM).
- ★ Stał się międzynarodowym standardem dla języków baz danych i występuje obecnie w produktach większości liczących się firm, zajmujących się sprzedażą oprogramowania dla baz danych.
- ★ SQL jest zaimplementowany w takich systemach baz danych (DBMS), jak: DB2, Oracle, InterBase, MySQL, dBse, Sybase, Informix, Paradox, MS SQL. Dzięki temu systemy informacyjne można przenosić na inne platformy.
- ★ Polecenia SQL mają postać zbliżoną do zdań w języku angielskim i są stosowane w celu uzyskania dostępu do danych i sterowania operacjami w bazie danych.
- ★ Podstawowy rdzeń języka SQL jest implementacją algebry relacyjnej
- ★ Ujęty w standardzie ANSI/ISO w roku 1986
 - aktualizacje: SQL-89, SQL 2, SQL 3
- ★ Język SQL danej bazy, np. MySQL, zawiera:
 - polecenia SQL ujęte w standardzie
 - rozszerzenia standardu – polecenia specyficzne dla konkretnego systemu baz danych

Typy składni języka SQL

* Składnię języka SQL dzieli się na trzy typy:

- **DML (Data Manipulation Language)** - stosowany przez wszystkich użytkowników bazy danych. Służy do wybierania i manipulowania danymi znajdującymi się w bazie. Za jego pomocą, można dodawać, usuwać, wybierać czy aktualniać dane.
 - Przykłady komend: SELECT- wydobywa dane z tabel, UPDATE- aktualnia dane w tabeli, DELETE - kasuje dane z tabeli, INSERT INTO - wprowadza dane do tabeli
- **DCL (Data Control Language)** - stosowany przez administratorów bazy danych. Służy on do zapewnienia bezpieczeństwa dostępu do danych znajdujących się w bazie. Za jego pomocą można przykładowo nadawać lub odbierać uprawnienia poszczególnym użytkownikom, czy całym grupom.
- **DDL (Data Definition Language)** - czyli język definiowania struktur danych jest wykorzystywany do utrzymywania struktury bazy danych. Dotyczy więc obiektów i poleceń jakie można na nich wykonywać.
 - Przykłady komend: CREATE TABLE - tworzy nową tabelę, ALTER TABLE - zmienia istniejącą tabelę, DROP TABLE - kasuje istniejącą tabelę, CREATE INDEX - tworzy indeks, DROP INDEX - usuwa indeks

* Integralność danych włączona jest do części definicji danych

* Język SQL nie zawiera zwykłych instrukcji sterowania IF ... THEN ..., DO ... WHILE i in.

Rodzaje poleceń SQL

* Polecenia SQL dotyczą:

- tworzenia i usuwania baz danych, tabel, kluczy
- wprowadzania, uaktualniania i usuwania danych
- wyszukiwania danych
- ustawiania praw dostępu do danych
- administracji bazą danych
- zarządzania transakcjami

* Sposób wprowadzania do bazy poleceń SQL:

- w programie działającym z linii poleceń
- w programie z graficznym interfejsem użytkownika
- w skryptach i programach komunikujących się z bazą danych
- pośrednio, przy użyciu graficznego interfejsu użytkownika

SQL - struktura języka, klauzule

★ Język definiowania struktur danych (DDL):

- **CREATE** - tworzenie tablic, perspektyw, indeksów;
- **DROP** - usuwanie obiektów;
- **ALTER** - modyfikacja struktury.

★ Język wyszukiwania i manipulowania danymi (DML):

- **SELECT** - wyszukiwanie danych w tabelach.
- **INSERT** - wstawianie danych do tabeli;
- **DELETE** - usuwanie danych;
- **UPDATE** - aktualizacja danych.

★ Język Nadzoru (DCL):

- **GRANT** - nadzorowanie uprawnień;
- **REVOKE** - odbieranie uprawnień;
- **BEGIN TRANSACTION ... END TRANSACTION** - programowanie transakcji;
- **ROLLBACK [WORK]** - przywrócenie tabeli do stanu przed transakcją.

★ Inne klauzule:

- **FROM** - nazwa tabeli;
- **WHERE** - warunek;
- **DISTINCT** - udostępnienie nie powtarzających się kolumn;
- **NOT, AND, OR** - spójniki wewnętrz warunku klauzuli.

SQL - struktura języka, typy danych

* Typy danych:

■ Typy napisowe

- **CHAR(N)** - napis znakowy o stałej długości, max długości 255 znaków
CHARACTER(N);
- **VARCHAR (N)** – napis znakowy o zmiennej długości, (do 255 znaków)
CHARACTER VARYING(N)
- **BIT(N)** – ciąg bitów o stałej długości N
- **BIT VARING(N)** – ciąg bitów o zmiennej długości, max N

■ Typy liczbowe

- **NUMERIC(M,N)**, - liczba stałoprzecinkowa o długości M z N miejscami po przecinku
DECIMAL(M,N)
- **INTEGER** - liczba całkowita (długość zależy od implementacji)
INT
- **SMALLINT** - liczba całkowita ze zmniejszoną liczbą cyfr (długość zależy od implementacji)
- **FLOAT(M,N)** - liczba rzeczywista zmiennoprzecinkowa
REAL

■ Typy daty i godziny

- **DATE**
- **TIME**
- **TIMESTAMP** – czas z uwzględnieniem ułamków sekund
- **INTERVAL** – przedział pomiędzy datami

■ Typy logiczne

- **LOGICAL** - wartości logiczne: .T. .F. (.T. .N.).

* Poszczególne implementacje języka różnią się w zakresie typów danych

SQL - struktura języka

★ Funkcje agregujące:

- **COUNT(nazwa pola)** – zlicza ilość rekordów;
- **AVG(nazwa pola)** – wyznacza średnią arytmetyczną wartości w kolumnie;
- **SUM(nazwa pola)** – liczy sumę wartości w kolumnie;
- **MIN(nazwa pola)** – zwraca minimalną wartość w kolumnie;
- **MAX(nazwa pola)** – zwraca wartość maksymalną kolumny.

★ Obiekty:

- **DATABASE** - baza danych;
- **TABLE** - jedna tabela w bazie danych;
- **INDEX** - tablica indeksów do tabeli;
- **VIEW** - widok, perspektywa z nazwą (część tabeli);
- **SYNONYM** - alternatywna nazwa tabeli lub widoku (synonim).

Zasady ogólne

- ★ Język SQL nie rozróżnia małych i wielkich liter w słowach kluczowych i nazwach (baz danych, tabel, indeksów i kolumn).
- ★ Język SQL rozróżnia litery w nazwach danych, trzeba pamiętać, że muszą być pisane dokładnie tak jak są umieszczone w tabeli.
- ★ Legalne są nazwy zbudowane ze znaków alfanumerycznych, nie zaczynające się od cyfry. Nie są dozwolone nazwy składające się wyłącznie z cyfr.
- ★ Nie należy w nazwach stosować znaków przestankowych i "@";
- ★ Każda komenda SQL kończy się średnikiem (;) i może składać się z wielu linii tekstu;
- ★ Wartości napisowe podaje się tak: "napis", lub tak: 'napis';
- ★ Wartości liczbowe zapisuje się w "zwykły" sposób, ewentualnie z kropką dziesiętną lub w notacji wykładniczej (np. -32032.6809e+10), gdy chodzi o wartości zmiennoprzecinkowe.

Zasady ogólne (2)

* Polecenie języka SQL składa się z dwóch rodzajów słów:

- Zarezerwowanych – które są integralną częścią języka i nie mogą być zmieniane ani dzielone pomiędzy wierszami.
- Zdefiniowanych przez użytkownika - reprezentują nazwy różnych obiektów bazy danych takich jak: indeksy, widoki, tabele, kolumny, relacje.

* Oznaczenia wykorzystane w składni poleceń

- Słowa pisane dużymi literami - słowa zarezerwowane np. SELECT, FROM, REVOKE, GROUP BY itd.
- Słowa pisane małymi literami - słowa nie zarezerwowane, zdefiniowane przez użytkownika. Są to więc np. nazwy przestrzeni tabel, nazwy kolumn, tabel itp.

Polecenie SELECT

- ★ Polecenie **SELECT** jest najpopularniejszym poleceniem języka SQL. Służy ono do wyszukiwania danych wg określonych w zapytaniu warunków.
- ★ Ogólna struktura polecenia **SELECT** jest następująca:

SELECT [nazwy kolumn, wyrażenia, funkcje]

FROM [nazwy tabel lub widoków]

WHERE [warunek wyboru wierszy]

GROUP BY [nazwy kolumn wg]

HAVING [warunek grupowania wybranych wierszy]

ORDER BY [nazwy lub pozycje kolumn]

Składnia polecenia SELECT

* Składnia polecenia (w notacji Bekusa):

```
SELECT [ALL | DISTINCT] { * | nazwa_kolumny [AS nowa_nazwa] [, ... ]}  
FROM nazwa_tabeli [alias] [, ... ]  
[WHERE warunek]  
[GROUP BY nazwa_kolumny]  
[HAVING warunek] [{UNION ALL] | INTERSECT | EXCEPT} SELECT ...]  
[ORDER BY nazwa_kolumny] [ASC | DESC]
```

- [xxx] – nie obowiązkowa obecność elementu w instrukcji
- { xxx } – obowiązkowa obecność elementu w instrukcji
- | – należy wybrać jeden z elementów rozdzielonych „|”
- ... – możliwość powtórzenia konstrukcji.

- ALL – zbiór ostateczny zawiera wszystkie rekordy spełniające warunki zapytania (również powtarzające się)
- DISTINCT – zbiór ostateczny zawiera wszystkie unikalne rekordy spełniające warunki zapytania (bez powtórzeń).
- *
- oznacza tu, że wszystkie kolumny wszystkich tabel będą włączone do zbioru ostatecznego.
- ASC – określa sortowanie w porządku rosnącym (takie jest domyślne)
- DESC – oznacza sortowanie w porządku malejącym

Realizacja instrukcji SELECT

- * Procedura wykonania instrukcji SELECT polega na realizacji klauzul FROM, WHERE, GROUP BY, HAVING i ORDER BY w następnej kolejności:
 1. **FROM** – określić nazwę (nazwy) tablicy (tablic), która jest potrzebna(i) dla formowania zbioru ostatecznego.
 2. **WHERE** – włączyć filtracje rekordów tabel. Uwzględniane będą tylko rekordy spełniające zapisany warunek.
 3. **GROUP BY** – sformować grupy rekordów, mających identyczne wartości w kolumnach tabeli, nazwa której jest podana w wyrażeniu tej instrukcji.
 4. **HAVING** – wykorzystać filtracje grupy rekordów. Warunek (warunki) tej filtracji jest wyznaczony w wyrażeniu instrukcji.
 5. **ORDER BY** – sformować rozkaz wyników ostatecznego zbioru. Rozkaz jest zadany w wyrażeniu tej instrukcji.

Wyszukiwanie – wybór kolumn (rzut)

- ★ Wyszukiwanie danych – wyświetlenie wybranych kolumn

```
SELECT rok, tytuł, wykonawca  
FROM albumy;
```

- ★ W ten sposób można uzyskać powtarzające się wyniki:

```
SELECT wykonawca  
FROM albumy;
```

- ★ Eliminacja powtórzeń wyników:

```
SELECT DISTINCT wykonawca  
FROM albumy;
```

albumy(tytuł, wykonawca, album, rok, gatunek)

tytuł	wykonawca	album	rok	gatunek

Wyszukiwanie – wybór wierszy. Operatory

* Wyszukiwanie rekordów spełniających zadany warunek – instrukcja WHERE

```
SELECT *  
FROM albumy  
WHERE wykonawca = 'Pink Floyd';
```

Restrykcja

Operatory używane w instrukcji SELECT ... WHERE:

- porównania: = <> < > <= >= <=>
- logiczne: NOT ! AND && OR || XOR
- IS NULL, IS NOT NULL
- *expr* BETWEEN *min* AND *max* (NOT BETWEEN)
- *expr* IN (*lista*) (NOT IN)

```
SELECT wykonawca, rok  
FROM albumy  
WHERE tytuł = 'The Best Of' AND rok < 1970;
```

Rzut + restrykcja

```
SELECT *  
FROM albumy  
WHERE wykonawca IN ('Pink Floyd', 'Dire Straits')  
AND (rok < 1975 OR rok BETWEEN 1979 AND 1983);
```

Sortowanie wyników

* Sortowanie wyników wg zadanej kolumny:

- `ORDER BY pole` – w porządku rosnącym
- `ORDER BY pole ASC` – jw.
- `ORDER BY pole DESC` – w porządku malejącym

```
SELECT *
FROM albumy
ORDER BY rok DESC;
```

```
SELECT *
FROM albumy
WHERE rok = 1989
ORDER BY wykonawca DESC;
```

Sortowanie wyników c.d.

- ★ Jeśli jest potrzeba sortowania wg dwóch (lub więcej) kolumn, jest to możliwe poprzez umieszczenie większej liczby kolumn w klauzuli ORDER BY, jak poniżej:

```
SELECT *
FROM albumy
ORDER BY rok DESC, wykonawca;
```

- ★ W klauzuli ORDER BY możemy się odwołać do kolumny poprzez wpisanie jej pozycji na liście SELECT.

```
SELECT wykonawca, album, rok
FROM albumy
ORDER BY 3, wykonawca;
```

- ★ W zapytaniu SELECT można użyć tylko jednej klauzuli ORDER BY, którą umieszczamy zawsze na końcu zapytania.

Grupowanie wyników

* Tworzenie zestawień przez grupowanie wyników:

- użycie funkcji, np. COUNT, SUM, MAX, MIN, AVG
- nazwanie kolumny z wynikami (opcjonalnie) – AS
- zgrupowanie wyników – GROUP BY

Przykład – obliczenie ilości albumów wszystkich wykonawców:

```
SELECT wykonawca, COUNT(*)  
FROM albumy  
GROUP BY wykonawca;
```

```
SELECT wykonawca, COUNT(*) AS ilosc  
FROM albumy  
GROUP BY wykonawca  
ORDER BY ilosc DESC;
```

Grupowanie wyników c.d.

- ★ Ograniczenie rekordów uzyskanych w wyniku grupowania - operator **HAVING**
- ★ Nie należy mylić instrukcji **WHERE** i **HAVING**!

Obliczenie ilości albumów wszystkich wykonawców.

Wyświetlenie tylko tych, którzy mają więcej niż 5 albumów:

```
SELECT wykonawca, COUNT(*) AS ilosc  
FROM albumy  
GROUP BY wykonawca  
HAVING ilosc > 5;
```

Ograniczenie liczby wyników

* Ograniczenie liczby zwracanych wyników – **LIMIT**

- **LIMIT *n*** – *n* pierwszych wyników
- **LIMIT *m, n*** – *n* wyników, pomijając *m* pierwszych

10 wykonawców o największej liczbie albumów:

```
SELECT wykonawca, COUNT(*) AS ilosc  
FROM albumy  
GROUP BY wykonawca    LIMIT 10;
```

20 kolejnych wyników (11-30):

```
SELECT wykonawca, COUNT(*) AS ilosc  
FROM albumy  
GROUP BY wykonawca    LIMIT 10,20;
```

Symbole wieloznaczne

- ★ Symbole wieloznaczne używane w instrukcji WHERE:
 - % - zastępuje dowolny ciąg znaków
 - _ - zastępuje jeden znak

- ★ Operator symboli wieloznacznych: LIKE, NOT LIKE

```
SELECT *
FROM albumy
WHERE album NOT LIKE 'The Best in 197_';
```

Wyszukiwanie przy użyciu polecenia SELECT

- ★ Aby wybrać wszystkie pola (kolumny) stosujemy „*” a brak warunku WHERE pozwoli wyszukać wszystkie kolumny z tabeli:

```
SELECT *
FROM Moduły;
```

- ★ Aby wybrać niektóre kolumny z tabeli, wpisujemy:

Rzut

```
SELECT NazwaModułu, Poziom
FROM Moduły;
```

- ★ Aby wybrać z tabeli tylko wiersze spełniające zadany warunek wpisujemy:

Restrykcja

```
SELECT *
FROM Moduły
WHERE Poziom = "1"
```

- ★ Aby wybrać niektóre kolumny z tabeli i tylko rekordy spełniające określony warunek wpiszemy

Rzut+Restrykcja

```
SELECT NazwaModułu, Poziom
FROM Moduły
WHERE Poziom = "1"
```

Wyszukiwanie w wielu tabelach

- ★ Pobieranie danych z więcej niż jednej tabeli

- ★ Przykład bazy danych – dwie tabele:

- albumy

IDA	Wykonawca	Album	Rok	Gatunek

- utwory

IDU	Utwór	Czas	IDA

- ★ Wybranie wszystkich możliwych kombinacji rekordów z obu tabel (iloczyn kartezjański):

```
SELECT *  
FROM albumy, utwory;
```

Wyszukiwanie w wielu tabelach c.d.

- ★ Uwzględnienie relacji między tabelami:

```
SELECT *
FROM albumy, utwory
WHERE albumy.IDA = utwory.IDA;
```

- ★ Łączy ze sobą rekordy obu tabel mające takie same dane w polach, które są połączone relacją:

- albumy

IDA	wykonawca	album	rok	gatunek

- utwory

IDU	utwór	czas	IDA

Wyszukiwanie w wielu tabelach

* Wybór kolumn:

```
SELECT albumy.wykonawca, albumy.album, utwory.utwor, utwory.czas  
FROM albumy, utwory  
WHERE albumy.IDA = utwory.IDA;
```

* Krótsza wersja – aliasy nazw tabel:

```
SELECT a.wykonawca, a.album, u.utwor, u.czas  
FROM albumy a, utwory u  
WHERE a.IDA = u.IDA;
```

IDA	wykonawca	album	rok	gatunek

IDU	utwór	czas	IDA

Tworzenie tabel

* Tworzenie tabeli wymaga podania:

- nazwy tabeli,
- nazw pól (kolumn) w tabeli,
- typów danych dla każdej kolumny,
- maksymalny rozmiar każdej kolumny.

```
CREATE TABLE nazwa_tabeli
    (nazwa_pola1 typ_pola1 [UNIQUE] [NOT NULL],
     nazwa_pola2 typ_pola2 [UNIQUE] [NOT NULL]
     ...
     nazwa_polaN typ_polaN [UNIQUE] [NOT NULL]);
```

```
CREATE TABLE albumy
    (id                      INT,
     wykonawca               VARCHAR(30),
     tytuł                   VARCHAR(30),
     rok                     YEAR,
     liczba-utworow          SMALLINT,
     opis                     TEXT);
```

Tworzenie tabel c.d.

- ★ Kolumny tabeli mogą być:

NOT NULL – nie mogą mieć wartości nul,

UNIQUE – jednoznaczne (nie powtarzające się),

- ★ Do definicji kolumny możemy dodać klauzulę określającą jej wartość domyślną:

nazwa_pola typ DEFAULT wartość

```
CREATE TABLE albumy
  (
    id                      INT NOT NULL UNIQUE,
    wykonawca               VARCHAR(30) NOT NULL,
    tytuł                   VARCHAR(30) NOT NULL,
    rok                     YEAR,
    liczba_utworow          SMALLINT DEFAULT 10,
    opis                    TEXT);

```

Tworzenie tabel – ustalanie kluczy

- ★ Podczas definiowania tabeli można określić klucze

- główny – dopisując odpowiednią klauzulę przy właściwej kolumnie

nazwa_pola typ PRIMARY KEY

lub dodając klauzulę na końcu definicji tabeli

PRIMARY KEY (nazwa_pola)

- zewnętrzny

FOREIGN KEY (nazwa_pola IDENTIFIES tabela_wskazywana)

```
CREATE TABLE Moduły
  (NazwaModułu      CHAR(15) PRIMARY KEY,
   Poziom            SMALLINT,
   KodKursy          CHAR(3),
   NrPrac            NUMBER(5)
   FOREIGN KEY (NrPrac IDENTIFIES Wykładowcy);
```

```
CREATE TABLE Wykładowcy
  (NrPrac           NUMBER(5),
   NazwiskoPrac     VARCHAR(20),
   Status            VARCHAR(10),
   PRIMARY KEY (NrPrac));
```

Tworzenie tabel – zasady integralności

- ★ Podczas definicji klucza zewnętrznego (obcego) można określić warunki propagacji w przypadku usunięcia rekordu zawierającego klucz (lub jego modyfikacji):
 - NO ACTION** – nie można usunąć klucza do którego odnosi się klucz obcy,
 - CASCADE** – powoduje automatyczne usunięcie powiązanych wierszy w tabeli klucza obcego po usunięciu wiersza w tabeli z kluczem głównym,
 - SET DEFAULT** – powoduje po usunięciu wiersza w tabeli z kluczem głównym, ustawienie wartości klucza obcego na wartość domyślną,
 - SET NULL** – powoduje po usunięciu wiersza w tabeli z kluczem głównym, ustawienie wartości klucza obcego na wartość NULL.

```
CREATE TABLE Moduły
  (NazwaModułu      CHAR(15) PRIMARY KEY,
   Poziom            SMALLINT,
   KodKursu          CHAR(3),
   NrPrac            NUMBER(5)
   FOREIGN KEY (NrPrac) REFERENCES Wykładowcy
   ON DELETE SET NULL
   ON UPDATE CASCADE);
```

```
CREATE TABLE Wykładowcy
  (NrPrac           NUMBER(5),
   NazwiskoPrac     VARCHAR(20),
   Status            VARCHAR(10),
   PRIMARY KEY (NrPrac));
```

Tworzenie tabel – dziedziny

- ★ Dziedziny możemy częściowo określić podając odpowiedni typ kolumn
- ★ Można ograniczyć zakres kolumn klauzulom **CHECK**

```
CREATE TABLE Moduły
  (NazwaModułu      CHAR(15) PRIMARY KEY,
   Poziom            SMALLINT,
   KodKursy          CHAR(3),
   NrPrac            NUMBER(5)
   FOREIGN KEY (NrPrac) REFERENCES Wykładowcy
   ON DELETE SET NULL
   ON UPDATE CASCADE
   CHECK (Poziom IN 1,2,3)
   CHECK (NrPrac BETWEEN 100 AND 10999);
```

- ★ Można zdefiniować własną dziedzinę

```
CREATE DOMAIN NazwaDziedziny AS Typ
  [DEFAULT wartość]
  [CHECK Zakres]
```

```
CREATE DOMAIN Poziomy AS INTEGER(1)
  DEFAULT 1
  CHECK (VALUE BETWEEN 1 AND 3);
```

Wykład 6

SQL

Structured Query Language

7. SQL

* TEMATYKA:

- Podstawowe informacje o języku SQL
- Struktura języka SQL
- Wyszukiwanie informacji w tabelach
- Zakładanie tabel
- Wpisywania, aktualizacja, usuwanie danych z tabel

Inne operacje definiowania struktur tabel

* Usuwanie tabeli:

```
DROP TABLE tabela;
```

```
DROP TABLE Wykładowcy;
```

* Zmiana struktury tabeli

■ Dodanie kolumny:

```
ALTER TABLE tabela ADD COLUMN kolumna typ;
```

```
ALTER TABLE Wykładowcy ADD COLUMN RokUr DATE
```

■ Usuwanie kolumny:

```
ALTER TABLE tabela DROP COLUMN kolumna;
```

```
ALTER TABLE Wykładowcy DROP COLUMN RokUr
```

Inne operacje na tabelach

- ★ Zmiana nazwy:

```
RENAME TABLE tabela TO nowa_tabela;
```

```
RENAME TABLE Moduły TO Przedmioty;
```

- ★ Wyświetlenie istniejących baz danych:

```
SHOW TABLES;
```

- ★ Wyświetlenie struktury tabeli:

```
DESCRIBE tabela;
```

```
DESCRIBE Przedmioty;
```

Inne operacje na tabelach c.d.

- ★ Modyfikacja typu kolumny:

```
ALTER TABLE tabela MODIFY kolumna typ;
```

```
ALTER TABLE Wykładowcy MODIFY NazwiskoPrac VARCHAR(35);
```

- ★ Zmiana nazwy kolumny:

```
ALTER TABLE tabela CHANGE kolumna nowa_kolumna typ
```

```
ALTER TABLE Wykładowcy CHANGE NazwiskoPrac Nazwisko VARCHAR(35);
```

- ★ Za pomocą ALTER TABLE możliwe jest również dodawanie i usuwanie atrybutów pól.

Atrybuty pól tabeli

- ★ Przy tworzeniu lub zmianie tabeli można podać opcjonalne atrybuty pól (kolumn) tabeli:

```
CREATE TABLE (pole typ atrybuty, ...);
```

- ★ Dostępne atrybuty:

- **NULL** – można nie podawać wartości (domyślnie)
- **NOT NULL** – wartość musi być podana
- **DEFAULT wartość** – gdy nie podamy wartości
- **AUTO_INCREMENT** – automatycznie zwiększany licznik
- **COMMENT 'opis'** – komentarz
- **PRIMARY KEY, INDEX** – indeksy główne

AUTO_INCREMENT i DEFAULT

- ★ AUTO_INCREMENT – nie wpisujemy danych, baza wpisuje aktualny stan licznika i zwiększa go o 1.
- ★ DEFAULT – jeżeli nie wprowadzimy danych, zostanie wpisana wartość domyślna

```
CREATE TABLE wykonawcy
  (
    id          INT NOT NULL AUTO_INCREMENT,
    wykonawca   VARCHAR(30),
    opis        TEXT DEFAULT ('brak opisu');

    INSERT INTO wykonawcy (wykonawca) VALUES ('XYZ');
```

Wynik: (1, 'XYZ', 'brak opisu')

Indeksowanie tabel

- ★ Na wybrane kolumny tabeli mogą być nakładane indeksy (klucze) w celu:
 - przyspieszenia wyszukiwania
 - zdefiniowania relacji pomiędzy tabelami (służą do tego klucze obce)
- ★ Tworzenie indeksów nie weszło do standardu SQL, jest jednak oferowane przez większość systemów komercyjnych
- ★ Tworzenie indeksu w już istniejącej tabeli:

```
CREATE INDEX indeks ON tabela(kolumna);
```

```
CREATE INDEX WgNazwisk ON Wykładowcy(NazwiskoPrac);
```

```
CREATE INDEX WgNazwStat ON Wykładowcy (NazwiskoPrac(10), Status);
```

Indeks wielokolumnowy,
indeksowanych 10 zanków
z pola NaziskoPrac

- ★ Usunięcie indeksu (nie usuwa danych!):

```
DROP INDEX indeks ON tabela;
```

```
DROP INDEX WgNazwisk ON Wykładowcy;
```

Wartości niepowtarzalne

- ★ **UNIQUE** – żadne dwa rekordy w tabeli nie mogą mieć jednakowych danych w indeksowanej kolumnie. Jest to jednocześnie INDEX.
- ★ Jeżeli indeksowana kolumna ma atrybut NOT NULL, dane w kolumnie muszą być unikatowe i muszą być wprowadzone.
- ★ Jeżeli indeksowana kolumna ma atrybut NULL, dane w kolumnie muszą być unikatowe, ale mogą nie być wprowadzane (pole może pozostać puste).

Indeks główny

- ★ Indeks główny – PRIMARY KEY
- ★ identyfikuje jednoznacznie każdy rekord w tabeli
- ★ może istnieć tylko jeden w tabeli
- ★ jest typu UNIQUE
- ★ indeksowana kolumna otrzymuje automatycznie atrybut NOT NULL
- ★ ma nazwę PRIMARY (nie można podać własnej)
- ★ bierze domyślnie udział w relacjach z innymi tabelami

Tworzenie indeksu głównego

- ★ Tworzenie indeksu głównego podczas definiowania tabeli
– w definicji kolumny:

```
CREATE TABLE dane {  
    nazwisko CHAR(30) NOT NULL,  
    pesel      CHAR(11) PRIMARY KEY,  
};
```

- ★ To samo w definicji tabeli (może to być ind. wielokolumn.) CREATE TABLE dane {

```
    nazwisko CHAR(30) NOT NULL,  
    pesel      CHAR(11),  
    PRIMARY KEY (pesel)  
};
```

Tworzenie indeksu głównego

- ★ Tworzenie indeksu głównego w już istniejącej tabeli:
ALTER TABLEEEjDD PRIMARY KEY (pesel);

- ★ Usuwanie indeksu głównego w tabeli (nie usuwa danych!):
ALTER TABLEEEzP PRIMARY KEY;

Wstawianie danych do tabeli

- ★ Instrukcja wstawiania składa się z następujących elementów

- słowa kluczowego **INSERT INTO**
- nazwy tabeli
- listy atrybutów ujętej w nawiasy
- słowa kluczowego **VALUES**
- listy wartości przypisywanych poszczególnym atrybutom

```
INSERT INTO tabela(pole1 ... PoleN) VALUES (wartość1 ... wartośćN)
```

- ★ Wstawianie danych z podaniem wszystkich kolumn tabeli (należy zachować prawidłową kolejność)

```
INSERT INTO Wykładowcy  
VALUES (244, 'Buczek Jan', 'P');
```

- ★ Wstawianie danych do wybranych kolumn tabeli

```
INSERT INTO Wykładowcy (NrPrac, Status)  
VALUES (244, 'P');
```

- ★ Wstawianie do tabeli danych uzyskanych w wyniku zapytania:

```
INSERT INTO nowa (autor, dzieло)  
SELECT DISTINCT wykonawca, album  
FROM albumy;
```

Usuwanie rekordów

* Instrukcja usuwania rekordów z tabeli składa się z:

- słowa kluczowego **DELETE FROM**
- nazwy tabeli
- słowa kluczowego **WHERE**
- warunku

DELETE FROM tabela WHERE warunek

```
DELETE FROM Wykładowcy
WHERE NazwiskoPrac = 'Buczek Jan';
```

Uaktualnianie rekordów

* Instrukcja aktualizacji składa się z następujących elementów

- słowa kluczowego **UPDATE**
- nazwy tabeli
- słowa kluczowego **SET**
- listy wyrażeń, z których każde określa wartość pewnego atrybutu
- słowa kluczowego **WHERE**
- warunku

UPDATE tabela SET nowe_wartości WHERE warunek

```
UPDATE Moduły  
SET NrPrac = 260  
WHERE NrPrac = 244
```

Definiowanie perspektyw

- ★ Perspektywa w SQL jest relacją, która nie ma fizycznego odpowiednika (tabele istnieją faktycznie)
- ★ Perspektywy mogą być wykorzystywane w zapytaniach, a nawet niekiedy modyfikować
- ★ Deklaracja perspektywy składa się z:
 - słowa kluczowego CREATE VIEW
 - nazwy perspektywy
 - słowa kluczowego AS
 - zapytania, które określa zakres danych perspektywy
`CREATE VIEW perspektywa AS definicja`
- ★ Perspektywa z jednej tabeli

```
CREATE VIEW Profesorzy AS
    SELECT *
    FROM Wykładowcy
    WHERE Status = 'P' ;
```

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A

Definiowanie perspektyw c.d.

- ★ Perspektywa może wybierać dane z kilku tabel

```
CREATE VIEW Obsada AS  
    SELECT NazwaModułu, NazwiskoPrac  
    FROM Moduły, Wykładowcy  
    WHERE Moduły.NrPrac = Wykładowcy.NrPrac;
```

Moduły			
NazwaModułu	Poziom	KodKursu	NrPrac
Systemy relacyjnych baz danych	1	CSD	244
Projektowanie relacyjnych baz danych	1	CSD	244
Dedukcyjne bazy danych	4	CSD	445
Obiektowe bazy danych	4	CSD	445
Roproszone bazy danych	2	CSD	247

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A



Obsada	
NazwaModułu	NazwiskoPrac
Systemy relacyjnych baz danych	Buczek Jan
Projektowanie relacyjnych baz danych	Buczek Jan
Dedukcyjne bazy danych	Wysocki Edward
Obiektowe bazy danych	Wysocki Edward
Roproszone bazy danych	Kalita Henryk

Modyfikowanie perspektyw

- ★ Najczęściej perspektyw nie można modyfikować ! Gdzie zostanie umieszczony w prowadzony rekord, przecież mamy doczynienia ze strukturą logiczną ?
- ★ SQL 2 określa formalnie kiedy perspektywę można modyfikować:
 - perspektywa musi być zdefiniowana przez selekcję atrybutów tylko z jednej relacji (tabeli) R,
 - klauzula WHERE nie może zawierać zapytania dotyczącego relacji R
 - w klauzuli SELECT musi być wybranych wystarczająco dużo atrybutów. Pozostałe otrzymają wartość NULL lub wartości domyślne.
- ★ Do perspektyw modyfikowalnych można zarówno wstawać jak i usuwać rekordy.

Wartość NULL

- ★ Bardzo ważne zastosowanie NULL zachodzi przy wyliczaniu złączenia, gdy należy chronić dane, a krotka (rekord) z jednej relacji nie daje się połączyć z żadną krotką z innej relacji (złączenie zewnętrzne).
- ★ Wartości NULL są stosowane w krotkach wstawianych do relacji wówczas, gdy w poleceniu nie określono wszystkich składowych wstawianej krotki
- ★ Przy działaniach z wartością NULL należy pamiętać o dwóch regułach:
 - Jeśli jako argument działania arytmetycznego (*, +) występuje NULL i pewna inna wartość to wynikiem jest wartość NULL
 - $\text{NULL} + \text{???} = \text{NULL}$
 - $\text{NULL} * \text{???} = \text{NULL}$
 - Przy porównywaniu NULL z inną wartością przy pomocy operatorów algebraicznych (=, > , <) wynik otrzymuje wartość UNKNOWN (trzecia wartość logiczna obok TRUE i FALSE)
 - $\text{NULL} > \text{???} \Rightarrow \text{UNKNOWN}$
 - $\text{NULL} = \text{???} \Rightarrow \text{UNKNOWN}$
 - $\text{NULL} < \text{???} \Rightarrow \text{UNKNOWN}$

Pułapki: $0 * \text{NULL} = \text{NULL}$
 $\text{NULL} - \text{NULL} = \text{NULL}$

Wartość logiczna UNKNOWN

- Określając reguły wyznaczania wartości logicznych w których występuje wartość UNKNOWN przyjmuje się ją jako coś pośredniego pomiędzy wartościami TRUE = 1, FALSE = 0 i przypisuje wartość ½.

x	y	x AND y	x OR y	NOT x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Zadanie: Czy w wyniku wykonania selekcji zostaną wybrane wszystkie rekordy

`SELECT *`

`FROM Moduły`

`WHERE Poziom <= 3 OR Poziom > 3;`

NIE!

Jeżeli w polu Poziom występują wartości UNKNOWN

Złączenia w SQL2

złączenie naturalne (równozłączenie)

- ★ SQL wykonuje złączenia poprzez wskazanie wspólnych atrybutów w klauzuli WHERE instrukcji SELECT

```
SELECT NazwaModułu, KodKursu, NazwiskoPrac  
FROM Moduły M, Wykładowcy W  
WHERE M.NrPrac = W.NrPrac;
```

lub w SQL 2

```
SELECT NazwaModułu, KodKursu, NazwiskoPrac  
FROM Wykładowcy NATURAL JOIN Moduły;
```

Możliwe bo pola klucz zewnętrzny w tabeli Moduły ma identyczną nazwę jak klucz pierwotny w tabeli Wykładowcy

gdyby klucze były różne

```
SELECT NazwaModułu, KodKursu, NazwiskoPrac  
FROM Wykładowcy W NATURAL JOIN Moduły M  
ON M.NrPrac = W.NrPrac;
```

- ★ W przypadku wybrania wszystkich kolumn do relacji wyjściowej pola powtarzające się zostaną dołączone pojedynczo.
- ★ W relacji wyjściowej znajdują się tylko rekordy powiązane.

Złączenia w SQL2

złączenie naturalne (równozłączenie)

* Polecenie ma postać

Relacja1 NATURAL JOIN Relacja2

```
SELECT NazwaModułu, KodKursu, NazwiskoPrac  
FROM Wykładowcy NATURAL JOIN Moduły
```

lub

```
SELECT NazwaModułu, KodKursu, NazwiskoPrac  
FROM Wykładowcy W NATURAL JOIN Moduły M  
ON M.NrPrac = W.NrPrac;
```

Moduły			
NazwaModułu	Poziom	KodKursu	NrPrac
Systemy relacyjnych baz danych	1	CSD	244
Projektowanie relacyjnych baz danych	1	CSD	244
Rozproszone bazy danych	2	CSD	247

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A

ModWykR						
NazwaModułu	Poziom	KodKursu	NrPrac	NrPrac	NazwiskoPrac	Status
Systemy relacyjnych baz danych	1	CSD	244	244	Buczek Jan	P
Projektowanie relacyjnych baz danych	1	CSD	244	244	Buczek Jan	P
Rozproszone bazy danych	2	CSD	247	247	Wysocki Edward	SW

Złączenia w SQL2

Złączenie zewnętrzne

- ★ Podobne do złączenia naturalnego, różnica polega na tym, iż pozostawiane są w relacji wynikowej wiersze nie posiadające odpowiedników w obu relacjach wyjściowych
- ★ Wyróżnia się złączenia zewnętrzne:

- Lewostronne – zachowuje nie pasujące wiersze z tabeli będącej pierwszym argumentem,

`Relacja1 LEFT OUTER JOIN Relacja 2`

- Prawostronne – zachowuje nie pasujące wiersze z tabeli będącej drugim argumentem

`Relacja1 RIGHT OUTER JOIN Relacja 2`

- Dwustronne – zachowuje nie pasujące wiersze z obu tabeli

`Relacja1 FULL OUTER JOIN Relacja 2`

Przykład złączenia zewnętrznego lewostronnego w SQL2

```
SELECT *  
FROM Moduły LEFT OUTER JOIN Wykładowcy
```

Moduły				Wykładowcy		
NazwaModułu	Poziom	KodKursu	NrPrac	NrPrac	NazwiskoPrac	Status
Systemy relacyjnych baz danych	1	CSD	244	244	Buczek Jan	P
Projektowanie relacyjnych baz da	1	CSD	244	247	Wysocki Edward	SW
Dedukcyjne bazy danych	4	CSD	445	445	Kalita Henryk	A
Obiektowe bazy danych	4	CSD	445	145	Zaborowski Jan	SW
Rozproszone bazy danych	2	CSD	247	447	Fusiarz Kamila	L
Opracowanie baz danych	2	CSD	null			
Administrowanie danymi	2	CSD	null			

NazwaModułu	Poziom	KodKursu	NrPrac	NazwiskoPrac	Status
Systemy relacyjnych baz danych	1	CSD	244	Buczek Jan	P
Projektowanie relacyjnych baz danych	1	CSD	244	Buczek Jan	P
Dedukcyjne bazy danych	4	CSD	445	Kalita Henryk	A
Obiektowe bazy danych	4	CSD	445	Kalita Henryk	A
Rozproszone bazy danych	2	CSD	247	Wysocki Edward	SW
Opracowanie baz danych	2	CSD	null	null	null
Administrowanie danymi	2	CSD	null	null	null

Przykład złączenia zewnętrznego prawostronnego w SQL2

```
SELECT *  
FROM Moduły RIGHT OUTER JOIN Wykładowcy
```

Moduły			
NazwaModułu	Poziom	KodKursu	NrPrac
Systemy relacyjnych baz danych	1	CSD	244
Projektowanie relacyjnych baz da	1	CSD	244
Dedukcyjne bazy danych	4	CSD	445
Obiektowe bazy danych	4	CSD	445
Rozproszone bazy danych	2	CSD	247
Opracowanie baz danych	2	CSD	null
Administrowanie danymi	2	CSD	null

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A
145	Zaborowski Jan	SW
447	Fusiarz Kamila	L

NazwaModułu	Poziom	KodKursu	NrPrac	NazwiskoPrac	Status
Systemy relacyjnych baz danych	1	CSD	244	Buczek Jan	P
Projektowanie relacyjnych baz danych	1	CSD	244	Buczek Jan	P
Dedukcyjne bazy danych	4	CSD	445	Kalita Henryk	A
Obiektowe bazy danych	4	CSD	445	Kalita Henryk	A
Rozproszone bazy danych	2	CSD	247	Wysocki Edward	SW
null	null	null	145	Zaborowski Jan	SW
null	null	null	447	Fusiarz Kamila	L

Przykład złączenia zewnętrznego dwustronnego w SQL2

```
SELECT *
FROM Moduły FULL OUTER JOIN Wykładowcy
```

Moduły			
NazwaModułu	Poziom	KodKursu	NrPrac
Systemy relacyjnych baz danych	1	CSD	244
Projektowanie relacyjnych baz da	1	CSD	244
Dedukcyjne bazy danych	4	CSD	445
Obiektowe bazy danych	4	CSD	445
Rozproszone bazy danych	2	CSD	247
Opracowanie baz danych	2	CSD	null
Administrowanie danymi	2	CSD	null

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A
145	Zaborowski Jan	SW
447	Fusiarz Kamila	L

NazwaModułu	Poziom	KodKursu	NrPrac	NazwiskoPrac	Status
Systemy relacyjnych baz danych	1	CSD	244	Buczek Jan	P
Projektowanie relacyjnych baz danych	1	CSD	244	Buczek Jan	P
Dedukcyjne bazy danych	4	CSD	445	Kalita Henryk	A
Obiektowe bazy danych	4	CSD	445	Kalita Henryk	A
Rozproszone bazy danych	2	CSD	247	Wysocki Edward	SW
Opracowanie baz danych	2	CSD	null	null	null
Administrowanie danymi	2	CSD	null	null	null
null	null	null	145	Zaborowski Jan	SW
null	null	null	447	Fusiarz Kamila	L

Iloczyn kartezjański (łączenie krzyżowe) w SQL2

- ★ Operator iloczynu kartezjańskiego ma bardzo ograniczone zastosowanie.

Relacja1 **CROSS JOIN** Relacja2

```
SELECT *
FROM Moduły CROSS JOIN Wykładowcy
```

Moduły			
NazwaModułu	Poziom	KodKursu	NrPrac
Systemy relacyjnych baz danych	1	CSD	244
Roproszone bazy danych	2	CSD	247

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A



NazwaModułu	Poziom	KodKursu	NrPrac	NrPrac	NazwiskoPrac	Status
Systemy relacyjnych baz danych	1	CSD	244	244	Buczek Jan	P
Systemy relacyjnych baz danych	1	CSD	244	247	Wysocki Edward	SW
Systemy relacyjnych baz danych	1	CSD	244	445	Kalita Henryk	A
Roproszone bazy danych	2	CSD	247	244	Buczek Jan	P
Roproszone bazy danych	2	CSD	247	247	Wysocki Edward	SW
Roproszone bazy danych	2	CSD	247	445	Kalita Henryk	A

Suma w SQL2

- ★ Suma daje możliwość połączenia dwóch zgodnych relacji.

Relacja1 **UNION** Relacja2

- ★ W praktyce najczęściej dokonuje się sumowania wyników dwóch zapytań.

```
SELECT *
FROM Wykładowcy
UNION
SELECT *
FROM Administracja
```

Wykładowcy		
NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A

Administracja		
NrPrac	NazwiskoPrac	Status
1010	Pawłowicz Maria	U
247	Wysocki Edward	SW

NrPrac	NazwiskoPrac	Status
244	Buczek Jan	P
247	Wysocki Edward	SW
445	Kalita Henryk	A
1010	Pawłowicz Maria	U

Instrukcje sterowania dostępem do danych

- ★ Każda SZBD musi zawierać mechanizm gwarantujący dostęp do baz danych tylko dla upoważnianych (przez administratora) użytkowników.
- ★ Język SQL zawiera instrukcje **GRANT** oraz **REVOKE** dla zabezpieczenia danych.
- ★ Mechanizm zabezpieczeń wykorzystuje **identyfikatory** użytkowników oraz ich **prawa**.

- ★ Identyfikator rejestruje administrator bazy danych (DBA). Każdy identyfikator jest połączony z hasłem.
- ★ Identyfikator użytkownika pozwala wyznaczyć obiekty bazy danych, z którymi może on pracować.

Instrukcje sterowania dostępem

- ★ Prawa to są działania, które użytkownik może wykonywać z daną relacją bazy danych.
- ★ W SQL2 określono następujące prawa:
 - **SELECT** – prawo do wybierania danych z tabeli;
 - **INSERT** – prawo do wstawiania nowych rekordów do tabeli;
 - **UPDATE** – prawo do modyfikowania danych w tabeli;
 - **DELETE** – prawo do usuwania rekordów z tabeli;
 - **REFERENCES** – prawo do odwoływanego się do danej struktury w więzach integralności.
 - **USAGE** – prawo do wykorzystywania dziedzin i elementów schematu w deklaracjach
- ★ Przywileje **INSERT** oraz **UPDATE** mogą dotyczyć pojedynczych kolumn w tabeli.
- ★ Każdy obiekt, który jest stworzony w środowisku SQL, ma właściciela i właściciel ma wobec niego wszelkie prawa.

Nadawanie praw – instrukcja GRANT

* Do nadawania praw służy instrukcja GRANT

* Instrukcja **GRANT** składa się z:

- słowa kluczowego GRANT,
- listy złożonej z przydzielanych praw,
- słowa kluczowego ON
- elementu bazy danych (tabela, perspektywa, dziedzina itp...) do którego nadajemy prawa
- słowa kluczowego TO
- listy użytkowników którym przydzielamy prawa
- opcjonalnie z klauzuli WITH GRANT OPTION

```
GRANT prawa | ALL PRIVILEGES  
    ON element_bazy_danych  
    TO użytkownicy | PUBLIC  
    [WITH GRANT OPTION]
```

ALL PRIVILEGES – wszystkie prawa;
PUBLIC – wszyscy użytkownicy;
WITH GRANT OPTION – prawo do przekazywania praw

Nadajmy użytkownikowi o identyfikatorze Maria prawo do wstawiania i wybierania elementów w tabeli Moduły oraz wszystkie prawa z możliwością ich przekazywania w tabeli Wykładowcy

```
GRANT SELECT, INSERT  
    ON Moduły  
    TO Maria  
  
GRANT ALL PRIVILEGES  
    ON Wykładowcy  
    TO Maria  
    WITH GRANT OPTION
```

Odbieranie praw – instrukcja REVOKE

★ Nadane prawa można w każdej chwili odebrać

★ Instrukcja **REVOKE** składa się z:

- słowa kluczowego REVOKE,
- listy złożonej z odbieranych praw praw,
- słowa kluczowego ON
- elementu bazy danych (tabela, perspektywa, dziedzina itp...) do którego obieramy prawa
- słowa kluczowego FROM
- listy użytkowników którym odbieramy prawa
- opcjonalnie z klauzul CASCADE, RESTRICT

Odbierzmy użytkownikowi o identyfikatorze Maria prawo do wstawiania elementów w tabeli Moduły oraz prawo do przekazywania praw z tabeli Wykładowcy

```
REVOKE SELECT  
    ON Moduły  
    FROM Maria  
  
REVOKE GRANT OPTION FOR  
    ALL PRIVILEGES  
    ON Wykładowcy  
    FROM Maria
```

```
REVOKE [GRANT OPTION FOR] prawa | ALL PRIVILEGES  
    ON element_bazy_danych  
    FROM użytkownicy | PUBLIC  
    [CASCADE | RESTRICT]
```

GRANT OPTION FOR	– odebranie prawa do przekazywania praw
ALL PRIVILEGES	– wszystkie prawa;
PUBLIC	– wszyscy użytkownicy;
CASCADE	– odbiera prawa nadane przez osobę której prawa odbieramy
RESTRICT	– nie odbierać praw jeżeli powoduje to usunięcie praw innych niż jawnie wyspecyfikowane

★ Po odebraniu praw ogólnych prawa szczegółowe obowiązują dalej

Operacje na bazach danych

- ★ **Tworzenie bazy danych:**

- ★ CREATE DATABASE baza;

- ★ **Usuwanie całej bazy:**

- ★ DROP DATABASE baza;

- ★ **Wyświetlenie istniejących baz danych:**

- ★ SHOW DATABASES;

- ★ **Przełączenie się na inną bazę danych:**

- ★ USE baza;

Funkcje SQL

- ★ Język SQL udostępnia szereg funkcji umożliwiających wykonywanie operacji na danych w zapytaniach.
- ★ Funkcje:
 - matematyczne
 - tekstowe
 - daty i czasu
- ★ Funkcje te mogą być wykorzystywane w instrukcji SELECT, w warunku wyboru kolumn lub w warunku wyboru wierszy.

Funkcje SQL c.d.

- ★ Przykład stosowania funkcji w instrukcji SELECT
- ★ Funkcja UPPER() zamienia litery na wielkie.

- ★ Użycie w warunku wyboru kolumn – zamienia litery na wielkie w zwracanych danych:
 - ★ `SELECT UPPER(wykonawca) FROM albumy;`
- ★ Użycie w warunku wyboru rekordu – zawartość pola po konwersji musi odpowiadać warunkowi:
 - ★ `SELECT * FROM albumy
WHERE UPPER(wykonawca)='U2';`

Funkcje matematyczne

- * ABS(x) – wartość bezwzględna
- * SIGN(x) – znak liczby (-1, 0, 1)
- * MOD(m,n) – reszta z dzielenia M/N
- * FLOOR(x) – zaokrąglenie w dół
- * CEIL(x) – zaokrąglenie w górę
- * ROUND(x) – zaokrąglenie do najbliższej l. całkowitej
- * m DIV n – część całkowita z dzielenia m/n
- * EXP(x) – e^x
- * LN(x), LOG2(x), LOG10(x), LOG(b,x) – logarytmy
- * POWER(x,y) = $x^y \text{SQRT}(x)$ – pierwiastek kwadratowy
- * PI() – wartość π
- * SIN(x), COS(x), TAN(x), COT(x) – funkcje trygonometr.
- * ASIN(x), ACOS(x), ATAN(x) – odwrotne funkcje tryg.
- * CRC32('wyr') – kod CRC wyrażenia *wyr*
- * RAND() – liczba losowa od 0 do 1
- * LEAST(x,y,...) – najmniejsza wartość z listy
- * GREATEST(x,y,...) – największa wartość z listy
- * DEGREES(x), RADIANS(x) – konwersja stopnie/radiany
- * TRUNCATE(x,d) – skrócenie x do *d* miejsc po przecinku

Funkcje tekstowe (1)

- ★ **ASCII(x)** – kod ASCII znaku
- ★ **ORD(x)** – suma na podstawie kodów ASCII
- ★ **CONV(x,m,n)** – konwersja między systemami liczbowymi
- ★ **BIN(x), OCT(x), HEX(x)** – konwersja między systemami
- ★ **CHAR(x)** – ciąg złożony ze znaków o podanych kodach
- ★ **CONCAT(s1,s2,...)** – łączy podane napisy w jeden
- ★ **CONCAT_WS(sep,s1,s2,...)** – łączy napisy separatorem
- ★ **LENGTH(s)** – długość napisu
- ★ **LOCATE(s1,s2,p)** – pozycja napisu *s1* w *s2* (szuk. od *p*)
- ★ **INSTR(s1,s2)** – pozycja napisu *s2* w *s1*
- ★ **LPAD(s1,n,s2)** – poprzedza *s1* ciągiem *s2* do długości *n*
- ★ **RPAD(s1,n,s2)** – dopisuje do *s1* ciąg *s2* do długości *n*
- ★ **LEFT(s,n)** – *n* pierwszych znaków z napisu *s*
- ★ **RIGHT(s,n)** – *n* ostatnich znaków z napisu *s*
- ★ **SUBSTRING(s,m,n)** – *n* znaków z napisu *s* od poz. *m*
- ★ **SUBSTRING_INDEX(s,sep,n)** – część napisu *s* przed *n*-tym wystąpieniem separatora *sep*

Funkcje tekstowe cd

- ★ **LTRIM(s)** – usuwa początkowe spacje
- ★ **RTRIM(s)** – usuwa końcowe spacje
- ★ **TRIM(s)** – usuwa początkowe i końcowe spacje
- ★ **SPACE(n)** – napis złożony z *n* spacji
- ★ **REPLACE(s1,s2,s3)** – zamień s2 na s3 w napisie s1
- ★ **REPEAT(s,n)** – napis z *n* powtórzeń s
- ★ **REVERSE(s)** – odwraca napis s
- ★ **INSERT(s1,m,n,s2)** – wstawia *n* znaków s2 do s1 na poz. *m*
- ★ **ELT(n,s1,s2,...)** – zwraca *n*-ty napis ze zbioru
- ★ **FIELD(s,s1,s2,...)** – zwraca indeks napisu s w zbiorze
- ★ **LOWER(s)** – zmienia litery na małe
- ★ **UPPER(s)** – zmienia litery na wielkie
- ★ **LOAD_FILE(plik)** – odczytuje zawartość pliku
- ★ **QUOTE(s)** – poprzedza znaki specjalne znakiem '\'
- ★ **STRCMP(s1,s2)** – porównanie dwóch napisów

Funkcje daty i czasu (1)

- * DATE(s) – pobiera datę z wyrażenia s
- * TIME(s) - pobiera czas z wyrażenia s
- * TIMESTAMP(s) – pobiera datę i czas z wyrażenia s
- * DAYOFWEEK(data) – podaje dzień tygodnia
- * DAYOFMONTH(data) – podaje dzień miesiąca
- * DAYOFYEAR(data) – podaje dzień w roku
- * MONTH(data) – podaje numer miesiąca
- * DAYNAME(data) – podaje nazwę dnia
- * MONTHNAME(data) – podaje nazwę miesiąca
- * WEEK(data) – podaje numer tygodnia (od 0)
- * WEEKOFYEAR(data) – podaje numer tygodnia (od 1)
- * YEAR(data) – podaje rok
- * YEARWEEK(data) – podaje rok i numer tygodnia
- * HOUR(data) – podaje godzinę
- * MINUTE(data) – podaje minutę
- * SECOND(data) – podaje sekundę
- * MICROSECOND(data) – podaje ułamki sekundy
- * PERIOD_ADD(p,m) – dodaje *m* miesięcy do daty *p*
- * PERIOD_DIFF(p1,p2) – różnica dwóch dat

Funkcje daty i czasu (2)

- * **DATE_ADD(data, INTERVAL wyr typ)**
– dodaje do daty podany czas
- * **DATE_SUB(data, INTERVAL wyr typ)** – odejmowanie daty
- * **ADDDATE(data,n)** – dodaje *n* dni do daty
- * **SUBDATE(data,n)** – odejmuje *n* dni od daty
- * **ADDTIME(s1,s2)** – dodawanie czasu
- * **SUBTIME(s1,s2)** – odejmowanie czasu
- * **EXTRACT(typ FROM data)** – pobranie części daty
- * **TO_DAYS(data)** – zamienia datę na numer dnia
- * **FROM_DAYS(n)** – zamienia numer dnia na datę
- * **DATE_FORMAT(data,format)** – formatowanie daty
- * **TIME_FORMAT(czas,format)** – formatowanie czasu
- * **MAKEDATE(rok,dzień)** – podaje datę
- * **MAKETIME(godz,min,sek)** – podaje czas
- * **CURDATE()** – bieżąca data
- * **CURTIME()** – bieżący czas
- * **NOW()** – bieżąca data i czas
- * **UNIX_TIMESTAMP()** – bieżąca data i czas w formacie UNIX
- * **SEC_TO_TIME(sek)** – konwersja sekund na czas
- * **TIME_TO_SEC(czas)** – konwersja czasu na sekundy

Przykład operacji na datach

- ★ **Dodawanie i odejmowanie**

- ★ DATE_ADD('1997-12-31 23:59:59', INTERVAL 1 DAY);
- ★ DATE_ADD('1997-12-31 23:59:59',
INTERVAL '1:1' MINUTE_SECOND);
- ★ DATE_SUB('1998-01-02', INTERVAL 31 DAY);

- ★ **Pobranie części daty**

- ★ EXTRACT(YEAR FROM "1999-07-02");

- ★ **Formatowanie daty**

- ★ DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');

Funkcje konwersji

- ★ Konwersje typów danych
- ★ CAST(wyr AS typ) – zmiana *wyr* na *typ*
- ★ CONVERT(wyr,typ) – jw.
- ★ CONVERT(wyr USING kod) – zmiana strony kodowej

- ★ Typy: BINARY, CHAR, DATE, DATETIME,
 SIGNED, TIME, UNSIGNED

Transakcje

- ★ Domyślnie wszystkie instrukcje są wykonywane od razu po ich wprowadzeniu – zmiana danych w bazie.
- ★ W pewnych sytuacjach nie chcemy aby wykonywane operacje modyfikowały fizyczny zbiór danych.
- ★ Tryb transakcji – wprowadzane operacje zostaną wykonane dopiero po podaniu odpowiedniej komendy.

Transakcje c.d.

- ★ **START TRANSACTION** – rozpoczęcie transakcji
- ★ Kolejne operacje są zapamiętywane, ale nie są wykonywane.
- ★ **COMMIT** – wykonanie operacji z całej transakcji
- ★ **ROLLBACK** – cofnięcie do początku transakcji

- ★ Niektóre komendy automatycznie wykonują COMMIT, np. **CREATE INDEX**, **DROP INDEX**, **DROP TABLE**, **DROP DATABASE**, **ALTER TABLE**, **RENAME TABLE**, **TRUNCATE**
- ★ Możliwe jest ustawienie w trakcie transakcji punktów zapisu za pomocą komendy **SAVEPOINT *nazwa***

- ★ Wykonanie komendy **ROLLBACK TO SAVEPOINT *nazwa***
- ★ powoduje cofnięcie do punktu zapisu o podanej nazwie

- ★ COMMIT nadal wykonuje całą transakcję.

Blokowanie tabel

- ★ W pewnych sytuacjach potrzebne jest czasowe zablokowanie tabeli, aby inny użytkownik nie zmodyfikował danych.
- ★ Blokowanie:
 - ★ `LOCK TABLES tabela1 typ, tabela2 typ, ... ;`
- ★ Typ blokady:
 - READ – blokada do odczytu
 - WRITE – blokada do zapisu
- ★ Odblokowanie tabel:
 - ★ `UNLOCK TABLES;`

Wykład 7

Diagramy związków encji

Diagramy związków encji

★ TEMatyka:

- Składowe diagramów: zbiory encji, atrybuty, związki

Składowe diagramów związków encji

- Zbiory encji
- Atrybuty – opisują encje
- Związki – opisują połączenia pomiędzy zbiorami encji.

Encje

★ Encja

- jednoznaczny składnik badanej rzeczywistości - dziedziny konceptualnej
- składnik w którym są przechowywane dane, wystąpienie encji
 - Przykład: **Szkoła, Uczeń, Sprzedaż, Naprawa, Samochód itp.**
 - ◆ **encja: samochód** (właściwie typ encji)
 - ◆ **instancja : Skoda ZNA6362** (właściwie encja)

★ **Encja** (obiekt) coś co istnieje, co jest odróżnialne od innych, o czym informację trzeba znać lub przechowywać. Encje o tych samych własnościach tworzą typy (zbiory) encji.

- AUTOR – dane osobowe autorów,
 - KSIĄŻKA – dane o książce,
 - WYDAWNICTWO – dane o wydawnictwie
- Reprezentacją graficzną encji jest ramka (prostokąt).
- Encje są opisywane przez atrybuty.
- | |
|---------------|
| AUTOR |
| Nazwisko |
| Imię |
| Rok urodzenia |
| ... |

★ Należy odróżniać typ encji oraz jej instancje (egzemplarze) np. *Osoba* jako *typ* i jako konkretny *obiekt* (instancja, egzemplarz).

★ Encje są opisywane za pomocą rzeczowników lub wyrażeń rzeczownikowych w liczbie pojedynczej

Atrybuty

* Atrybuty

- opisują właściwości encji są elementami charakteryzującymi encje i związki w dziedzinie konceptualnej
 - zbiór atrybutów jest zestawem jednoznacznie opisującym encję
-
- * Identyfikowany atrybut powinien opisywać encję, przy której się go umieszcza (a nie związki z innymi encjami)!
 - Np...: *Numer miejsca w kinie* jest atrybutem encji *Miejsce na widowni*, a nie atrybutem encji *Bilet*, na którym się pojawia.

Związki

★ **Związki** – opisują połączenia pomiędzy zbiorami encji.

- związek stanowi naturalne powiązanie pomiędzy dwoma lub więcej encjami w danej dziedzinie konceptualnej
- ogólny podział związków:
 - binarne (dwuargumentowe) - obejmuje dwie encje,
 - wielorakie (wieloargumentowe) - obejmuje więcej niż dwie encje
- w modelowaniu związku istotne są
 - liczebność związku - stosunek liczebnościowy między wystąpieniami encji uczestniczącymi w danym związku
 - reguły biznesowe

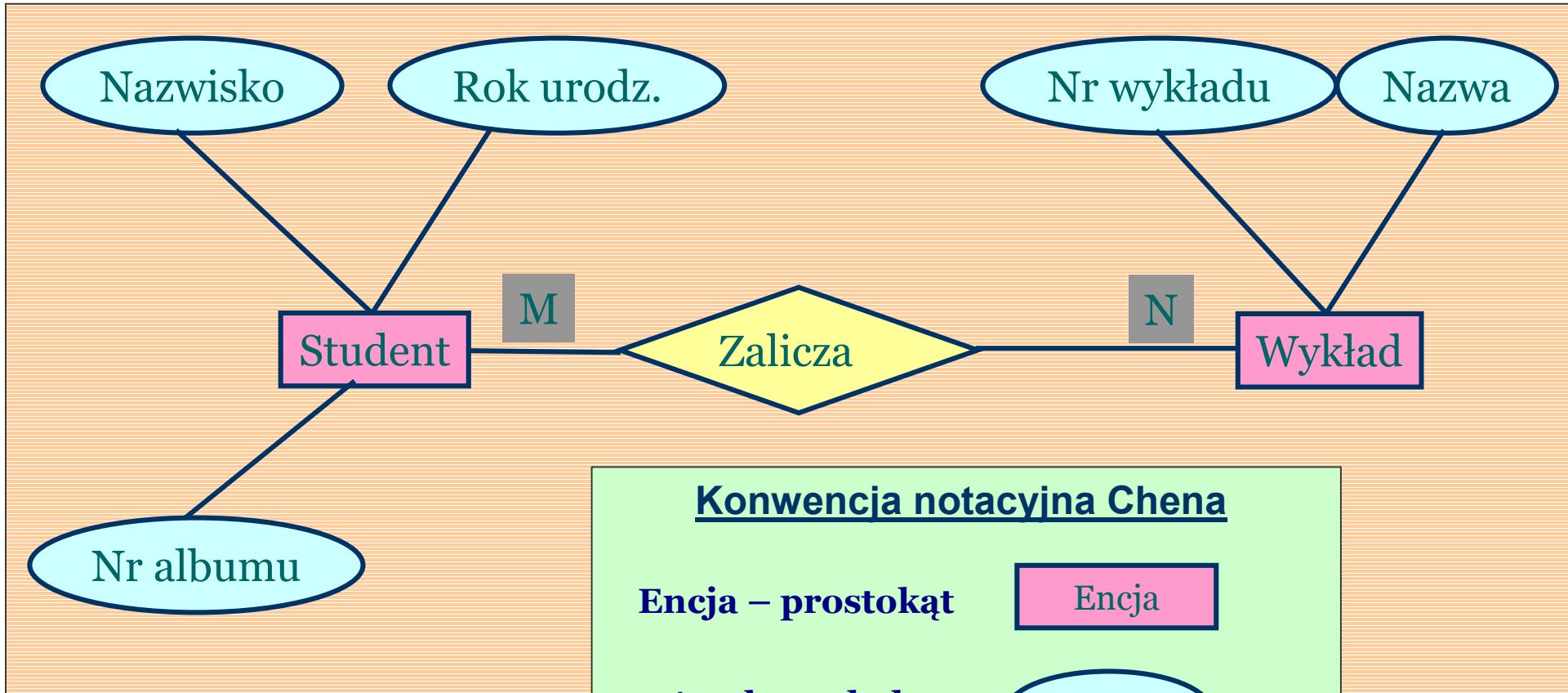
★ Każdy związek określa pewną relację między zbiorami egzemplarzy encji wchodzącyimi w skład związku - *instancję związku*.

★ Przykładowe związki

- dwuargumentowe (binarne):
 - autor pisze książkę,
 - wydawnictwo wydaje książki,
 - książka ma swoje egzemplarze,
 - pracownik pracuje w dziale,
 - kraj eksportuje towary,
- trójargumentowe:
 - czytelnik wypożycza książki z danej dziedziny,

Przykład diagramu związków encji (ERD)

Struktura konceptualna może być przedstawiona za pomocą diagramu ER



Konwencja notacyjna Chena

Encja – prostokąt

Encja

Atrybut – koło

Atrybut

Związek - romb

Związek

Liczebność (typy związków encji)

* Liczebność dotyczy liczby instancji biorących udział w związku.

* Typy związków dwuargumentowych:

■ jeden do jeden - 1:1

- osoba kieruje jedną szkołą
szkoła jest kierowana przez jedną osobę

■ jeden do wiele - 1:N

- wydawnictwo wydaje wiele książek
książka jest wydana przez jedno wydawnictwo

■ wiele do wiele - M:N

- autor pisze wiele książek
książka jest napisana przez wielu autorów

* Często na każdej końcówce linii związku podaje się minimalna oraz maksymalną dopuszczalną liczebność

* Nie ma standardowej notacji do opisu liczności (krotność)

Krotności....

Information Engineering style

one to one



one to many (mandatory)



many



one or more (mandatory)



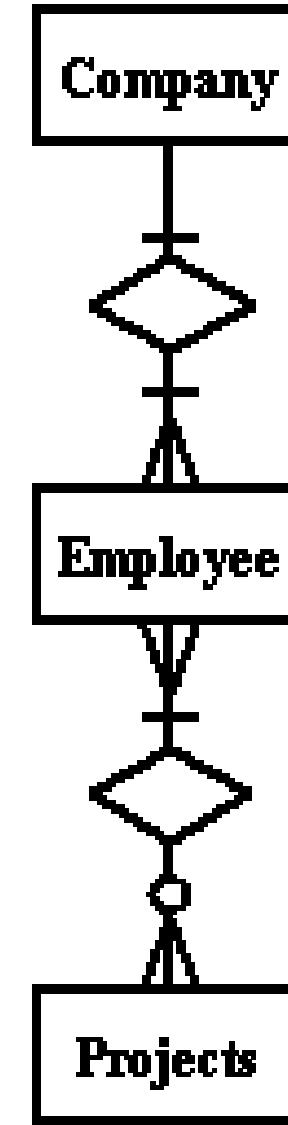
one and only one (mandatory)



zero or one (optional)



zero or many (optional)



Krotności....

Bachman style



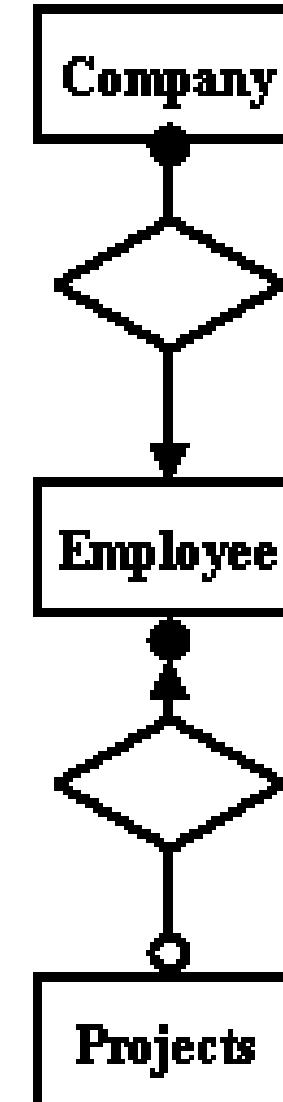
one to one



zero or more to one or more

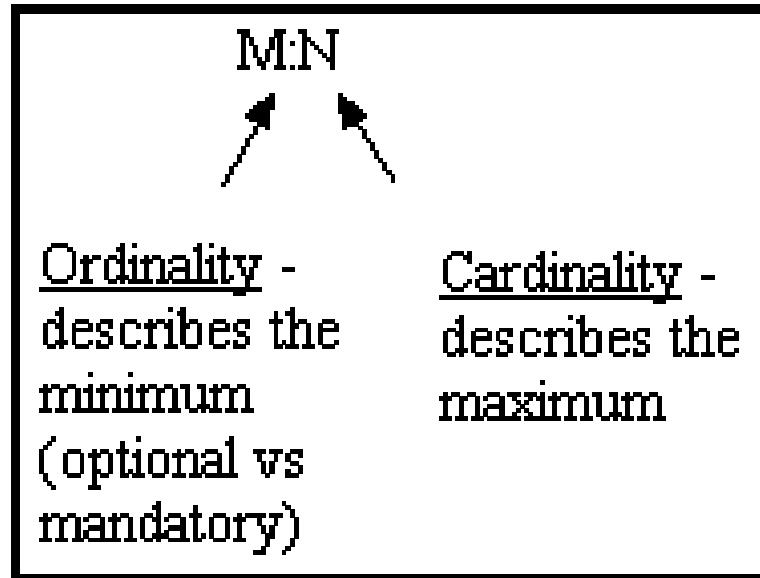


one to one or more



Krotności....

Chen style



1:N ($n=0,1,2,3\dots$)

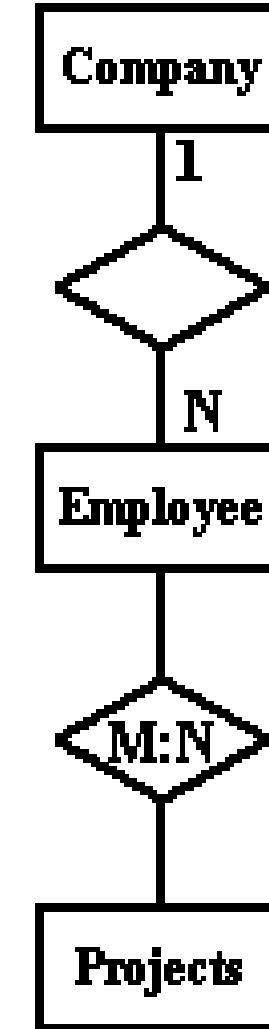
one to zero or more

M:N (m and $n=0,1,2,3\dots$)

zero or more to zero or more
(many to many)

1:1

one to one



Krotności....

Martin style

1 - one, and only one (mandatory)

* - many (zero or more - optional)

1...* - one or more (mandatory)

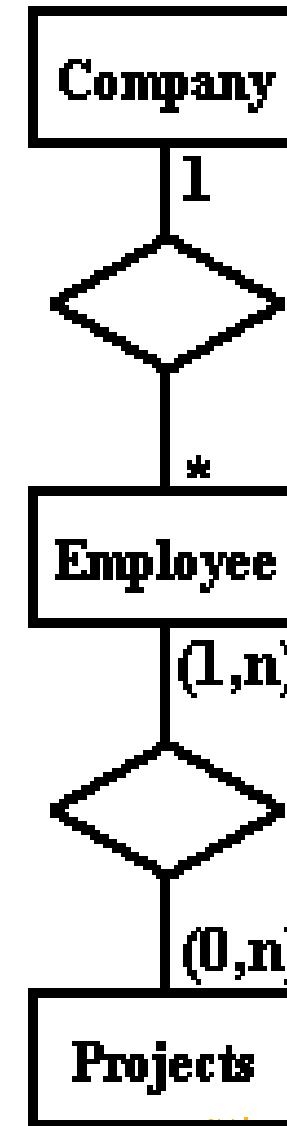
0...1 - zero or one (optional)

(0,1) - zero or one (optional)

(1,n) - one or more (mandatory)

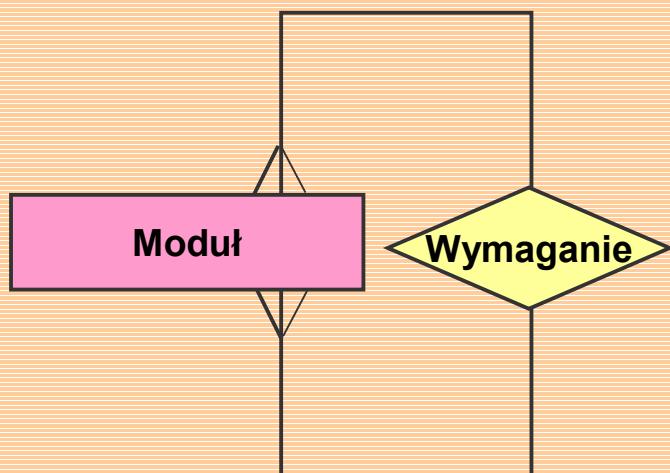
(0,n) - zero or more (optional)

(1,1) - one and only one (mandatory)



Związki rekurencyjne

- * Najczęściej związki są binarne.
- * Jeżeli związek jest jednoargumentowy (dotyczy tylko jednej encji) to mówimy o nim, że jest rekurencyjny

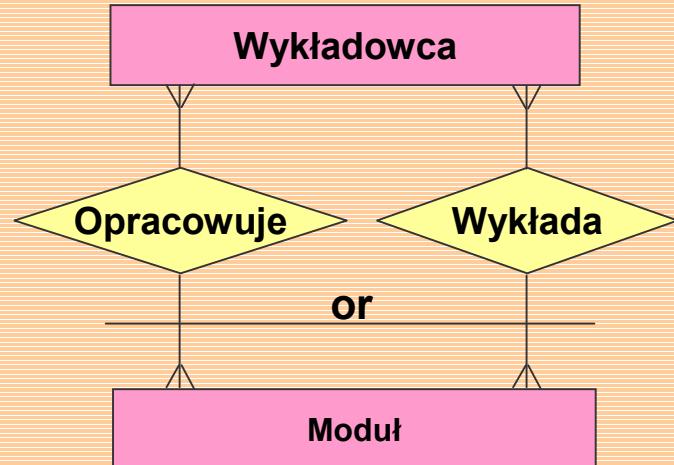


Moduł może mieć określone wymagania co do modułów go poprzedzających a sam może być wymagany przed innymi.
Np. Elektronika wymaga aby poprzedzała ją Elektrotechnika, sama zaś winna poprzedzać Architekturę komputerów

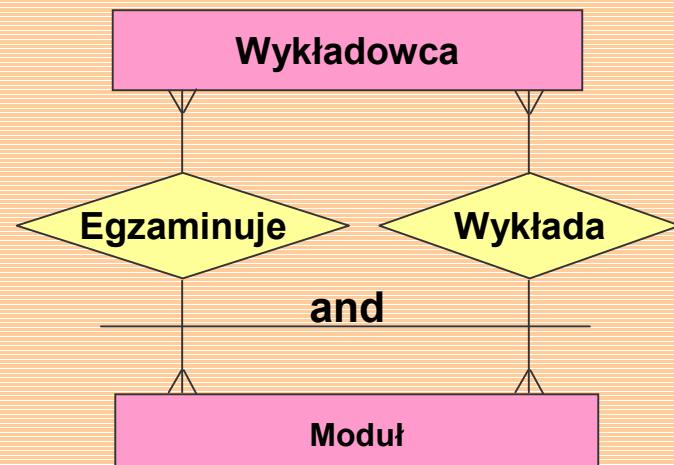
Modelowanie semantyczne

- ★ Niektóre dane można modelować na wiele sposobów, często wymiennie używając encji, związków i atrybutów;
- ★ Przykład : umowa kupna jako encja (sprzedajacy, kupujacy, cena) albo jako dwie encje Sprzedajacy, Kupujacy połączone związkiem Kupno;
- ★ Wybieramy model najodpowiedniejszy z punktu widzenia projektowanego systemu;
- ★ Dane reprezentujemy tak jak one są widziane w organizacji.

Zawieranie i wykluczanie



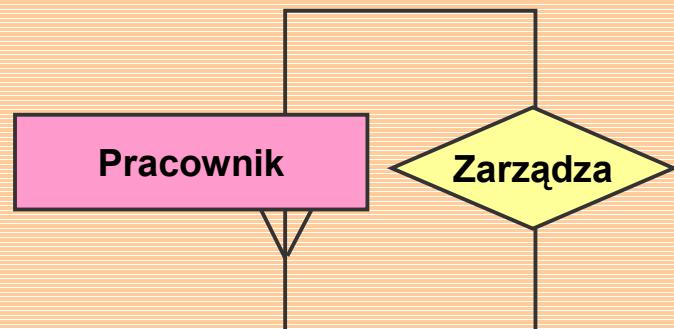
Moduł nie może być jednocześnie opracowywany i wykładany. Związki **wykluczają** się wzajemnie



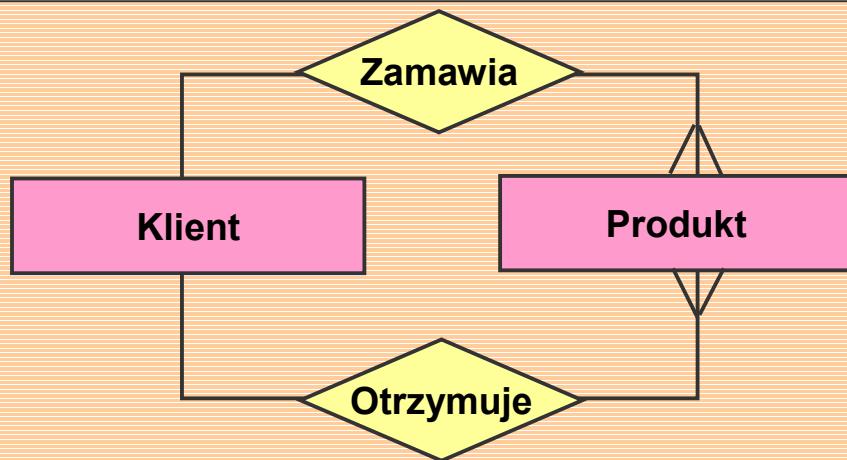
Wykładowca nie może wykładać modułu i nie egzaminować z niego. Związki **zawierają** się wzajemnie

Role w związkach

- ★ Zdarza się, że jeden zbiór encji występuje w danym związku więcej niż jeden raz. Wówczas trzeba narysować tyle krawędzi pomiędzy związkiem a zbiorem ile razy pojawia się on w związku.
- ★ Każda krawędź symbolizuje inną rolę



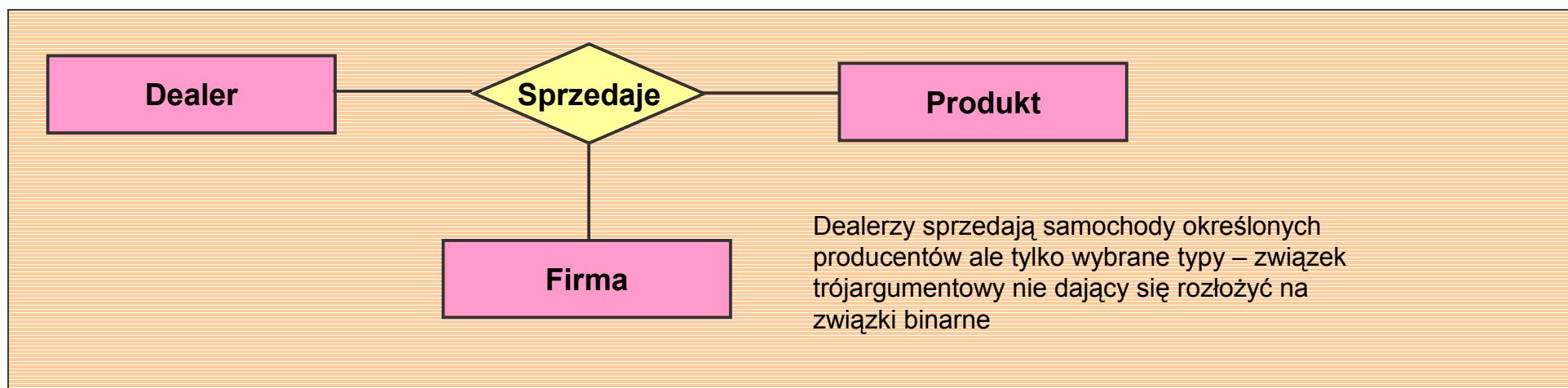
Pracownik występuje w dwóch rolach: jako kierownik i jako podwładny. Pracownik może być podwładnym w jednym związku i kierownikiem w innym



Klient zamawia towar oraz klient otrzymuje towar

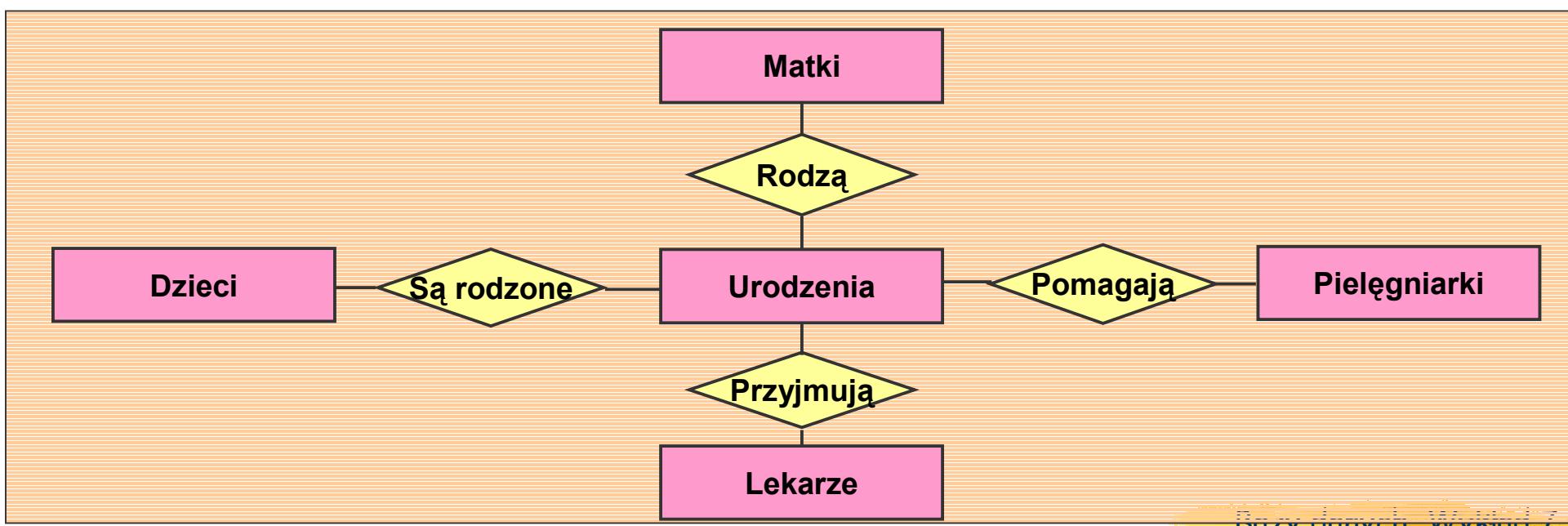
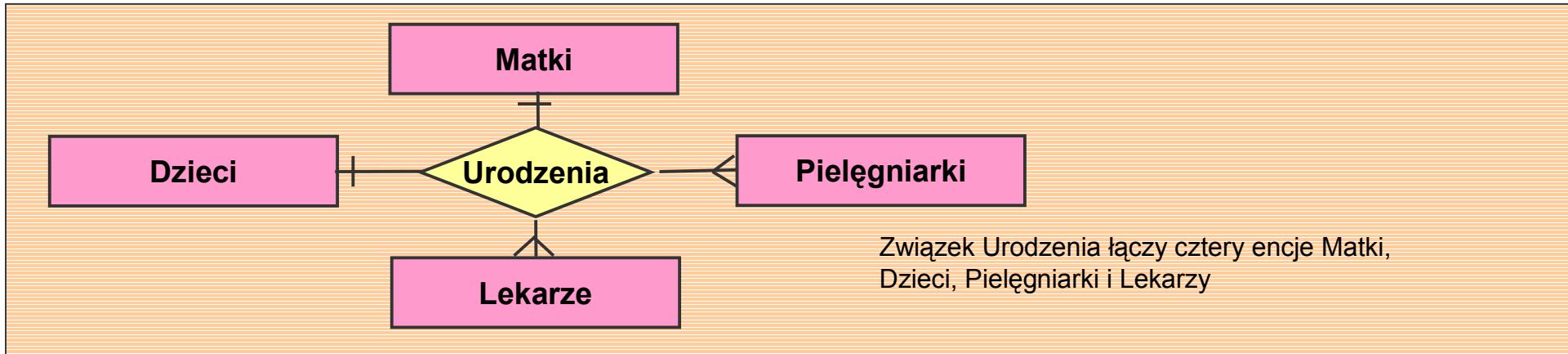
Związki wieloargumentowe

- ★ Związków wieloargumentowych łączą więcej niż 2 encje.
- ★ Związki wieloargumentowe należy rozbić na związki binarne lub trójargumentowe (jeżeli nie da się ich przekształcić na binarne).
- ★ Związki trójargumentowe reprezentują związki w piątej postaci normalnej



Przekształcanie związków wieloargumentowych w binarne

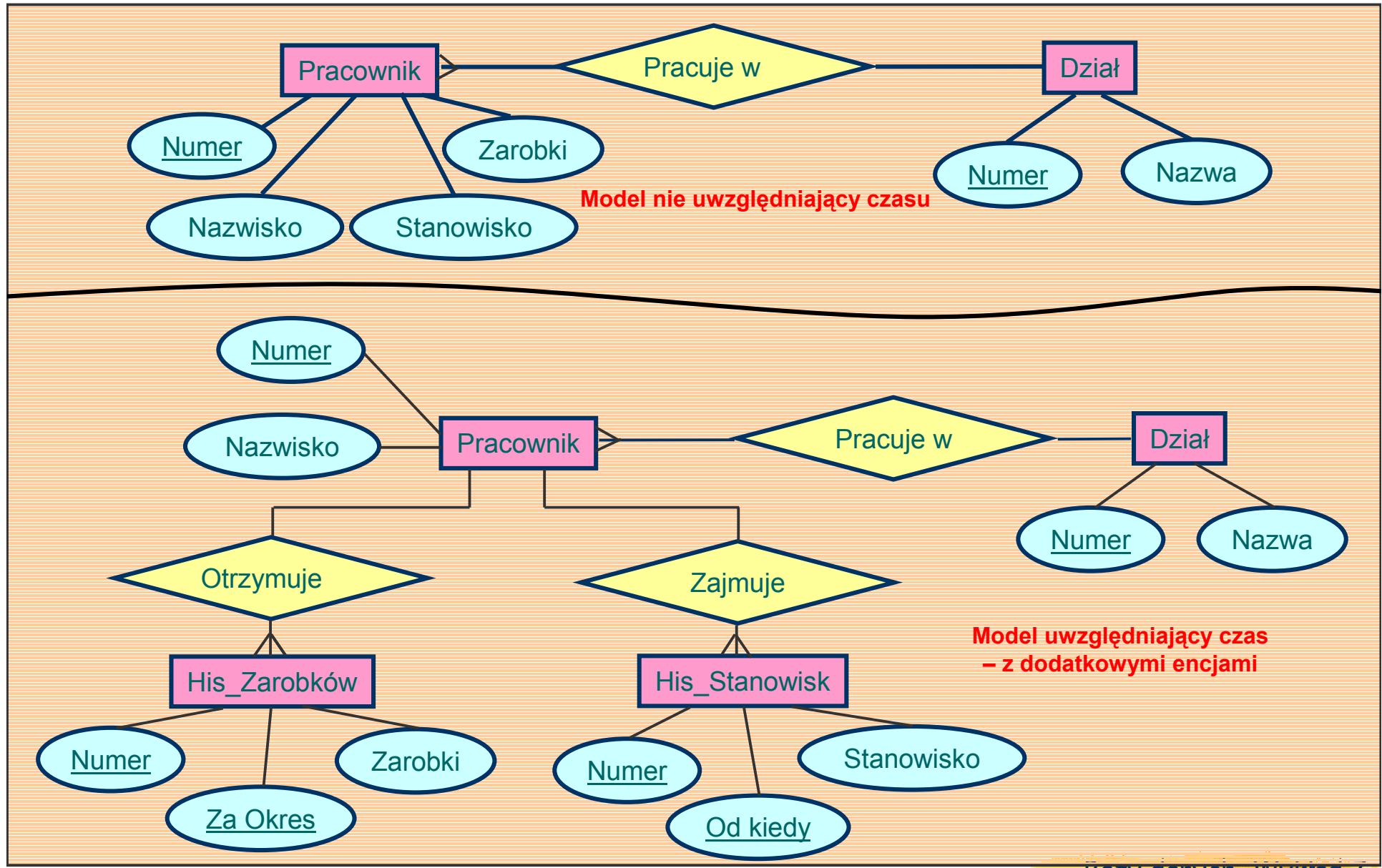
- ★ W celu przekształcenia związku wieloargumentowego na binarny tworzy się dodatkowy zbiór encji i odpowiednie związki łączące go z encjami wchodząymi w skład związku pierwotnego (wieloargumentowego)



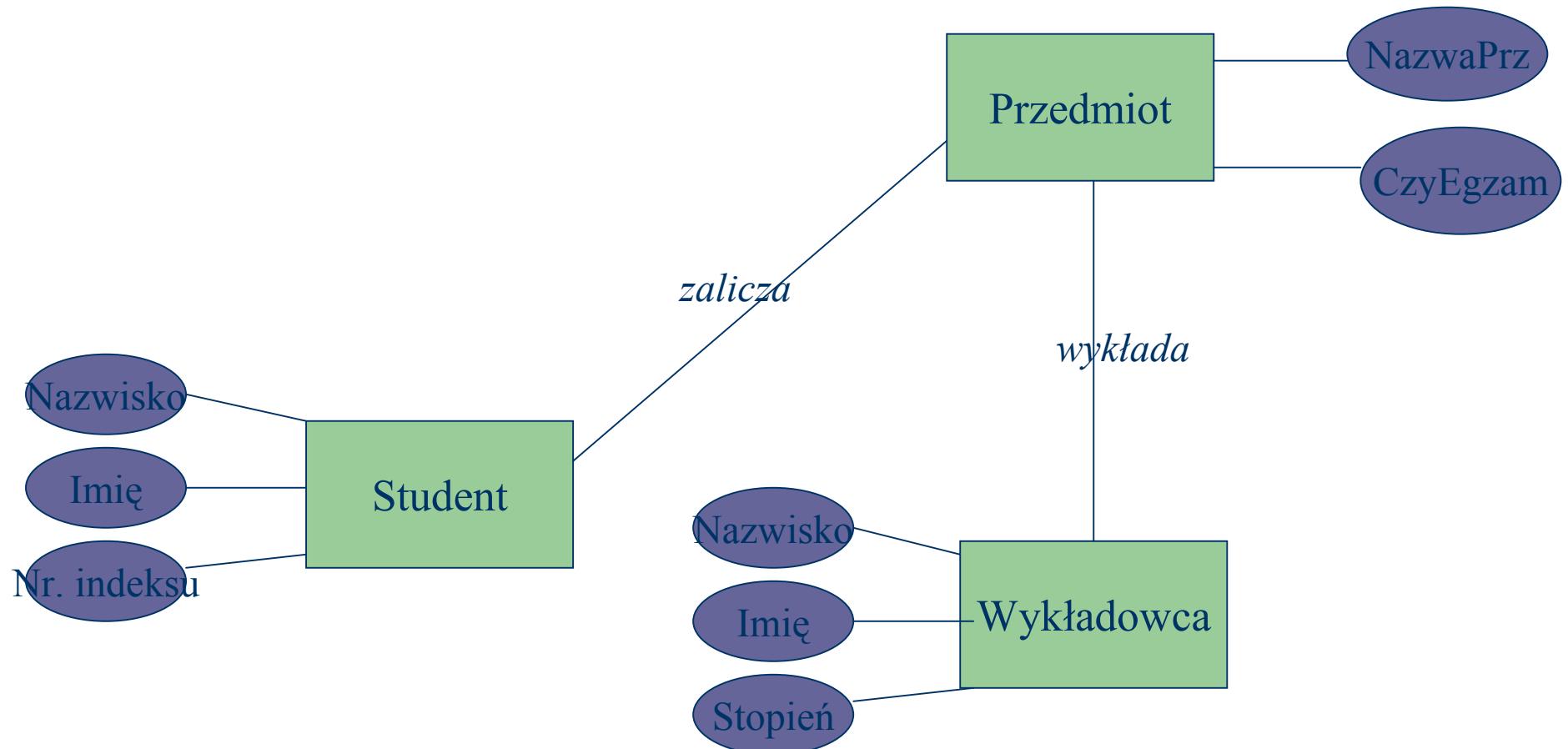
Modelowanie czasu

- ★ Problemem, przed którym często staje projektant schematu bazy danych jest uwzględnienie w modelu danych zmian w czasie.
 - Interesuje nas:
 - ile zarabia aktualnie pracownik, na jakim jest zatrudniony stanowisku, w którym aktualnie pracuje dziale,
 - ile zarabiał w zeszłym roku, jakie piastował stanowiska, w jakich działach pracował od początku zatrudnienia.

Modelowanie czasu - przykład



Nasz diagram ER



Wykład 8

KLASYCZNE I OBIEKTOWE MODELE DANYCH

SPIS TREŚCI

1. [Wstęp.](#)
2. [Podstawowe wady dotychczasowych baz danych i systemów zarządzania.](#)
3. [Obiektowy, podstawowy model danych.](#)
4. [Obiektowy model danych.](#)
5. [Obiektowa baza danych.](#)
6. [Pojęcia związane z obiektowym modelem bazy danych.](#)
7. [Obiekt.](#)
8. [Klasa.](#)
9. [Atrybut.](#)
11. [Metoda.](#)
12. [Hermetryzacja.](#)
13. [Relacyjna a obiektowa struktura aplikacji.](#)
14. [System zarządzania bazą danych.](#)
15. [Interfejs obiektowych baz danych.](#)
16. [Obiektowy model danych.](#)
17. [Demony w obiektowej bazie danych.](#)
18. [Zasady zachowania spójności bazy.](#)
19. [Projekt języka dla obiektowej bazy danych.](#)
20. [Fazy występujące w czasie wykonywania zapytań w systemach baz danych.](#)
21. [Jak przechowywane są obiekty złożone?](#)
22. [Co stanowi zasoby, i jak ma być zapewniona spójność danych.](#)
23. [Zamknięcia stosowane dla obiektowych baz danych.](#)
24. [Strategie dostępu jednoczesnego w obiektowych bazach danych.](#)
25. [Najpopularniejsze systemy obiektowych baz danych.](#)
26. [Obiektowo - relacyjna baza danych.](#)
27. [Korzyści płynące z zastosowania obiektowości.](#)

Generacje baz danych

- ★ Generacje baz danych
 - systemy plików (takie jak: ISAM i VSAM),
 - hierarchiczne baz danych (ISM, System 2000),
 - sieciowe baz danych CODASYL (m.in. IDS, IDMS),
 - relacyjnymi bazami danych.
 - obiektowe bazy danych
 - postrelacyjne (obiektowo-relacyjne) bazy danych
- ★ Przejście z jednej generacji do kolejnej było zawsze wymuszane wzrostem złożoności programów użytkowych baz danych, kosztów implementowania, utrzymywania i powiększania tych programów.
- ★ Niewygodny dostęp do danych (przy pomocy wskaźników) oraz brak niezależności danych doprowadziły do rozwoju systemów czwartej generacji - relacyjnych baz danych.
- ★ Mimo, że obecnie relacyjne bazy danych znajdują powszechnie zastosowanie na rynku, posiadają pewne ograniczenia, uniemożliwiające ich wykorzystanie do niektórych typów programów użytkowych. Spowodowane jest to faktem, iż relacyjne bazy danych (podobnie zresztą jak bazy poprzednich generacji) rozwijane były z myślą o konwencjonalnych programach użytkowych służących do obsługi magazynu, listy płac, księgowości, itp.

Modele danych

Modele danych (a w odniesieniu do konkretnej realizacji mówi się często o architekturze systemu baz danych) można rozumieć jako zbiór ogólnych zasad posługiwania się danymi. Zbiór ten obejmuje trzy główne części:

Definicje danych: zbiór reguł określających strukturę danych, tj. to co wcześniej określałem jako logiczną strukturę bazy danych (w odróżnieniu od niższego poziomu organizacji zapisu stosowanego wewnętrznie przez jądro bazy danych);

Operowanie danymi: zbiór reguł dotyczących procesu dostępu do danych i ich modyfikacji;

Integralność danych: zbiór reguł określających, które stany bazy danych są poprawne (a więc zarazem jakie operacje prowadzące do modyfikacji danych są dozwolone).

Brak jednoznacznych definicji pojęcia model danych

Model danych

Model danych (data model) jest pojęciem, którego ścisłe zdefiniowanie stwarza problem. Jego znaczenie jest wypadkową następujących przymiotów:

- metajęzyk (pojęcia, terminologia) do mówienia o danych, o systemach baz danych i o przetwarzaniu danych;
- sposób rozumienia organizacji danych i ideologiczne lub techniczne ograniczenia w zakresie konstrukcji, organizacji i dostępu do danych;
- języki opisu i przetwarzania danych, w szczególności: diagramy struktur danych, języki opisu danych i języki zapytań;
- ogólne założenia dotyczące architektury i języków systemu bazy danych;
- ideologie lub teorie (matematyczne) dotyczące struktur danych i dostępu do danych

Główne typy (lub generacje) modeli danych

Proste modele danych.

Dane zorganizowane są w strukturę rekordów zgrupowanych w plikach. Głównymi dostępnymi operacjami są operacje na rekordach (ewentualnie na ich poszczególnych polach).

Klasyczne modele danych.

Należą do nich modele **hierarchiczne**, **sieciowe** i **relacyjne**. Modele relacyjne stanowią najbardziej popularną obecnie podstawę architektur systemów baz danych.

Semantyczne modele danych.

Semantyka to inaczej **znaczenie**. Klasyczne modele danych nie dostarczają łatwego sposobu odczytania informacji o semantyce danych, stąd podejmuje się próby stworzenia innych modeli, uzupełniających ten brak. Przykładem częściowej realizacji tego programu są **obiektowe** modele danych.

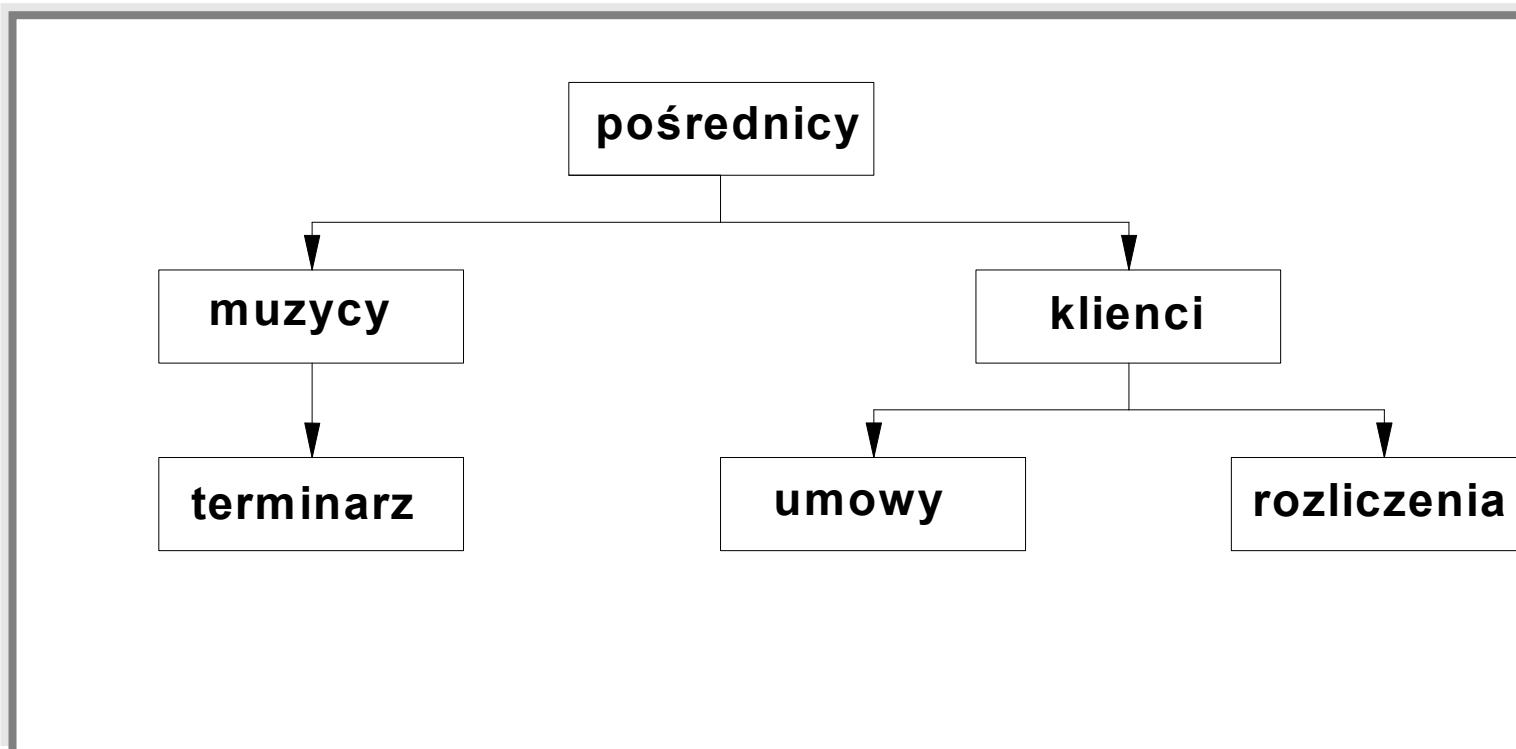
Cechy modeli klasycznych

- ★ struktura bazy danych opiera się na rekordach różnych typów o stałym formacie
- ★ każdy rekord definiowany jest poprzez stałą liczbę atrybutów
- ★ każdy atrybut jest zazwyczaj określonego rozmiaru co ułatwia implementację
- ★ nie zawierają mechanizmów bezpośredniej reprezentacji kodu w bazie danych
- ★ z modelami związane są języki zapytań umożliwiające realizację zapytań oraz modyfikacji danych
- ★ opis danych na poziomie konceptualnym oraz na poziomie wglądu
- ★ specyfikacja całościowa konceptualnej struktury bazy danych
- ★ opis implementacji bazy danych na wysokim poziomie

Hierarchiczne bazy danych

- ★ W hierarchicznym modelu danych (HMBD) dane mają strukturę, którą można najprościej opisać jako odwrócone drzewo.
- ★ Jedna z tabel pełni rolę „korzenia” drzewa, a pozostałe mają postać „gałęzi” biorących swój początek w korzeniu.
- ★ Relacje w HMBD są reprezentowane w kategoriach *ojciec/syn (nadrzędny/podrzędny)*. Oznacza to że *tabela-ojciec* może być powiązana w wieloma *tabelami/synami*, lecz pojedynczy *syn* może mieć tylko jednego ojca. Tabele te mogą być powiązane jawnie, przez *wskaźniki*, lub przez fizyczną organizację rekordów wewnątrz tabel. Aby uzyskać dostęp do danych
- ★ W modelu hierarchicznym, użytkownik zaczyna przeszukiwanie od korzenia i przedziera się przez całe drzewo danych, aż do interesującego go miejsca. Oznacza to jednak, że użytkownik musi dobrze znać strukturę bazy danych.

Hierarchiczne bazy danych



* Baza danych pośredników:

- każdy pośrednik pracuje dla kilku muzyków i ma pewną liczbę klientów, którzy zamawiają u niego obsługę muzyczną różnych imprez.
- klient zawiera umowę z danym muzykiem przez pośrednika i u tego właśnie pośrednika uiszcza należność za usługę

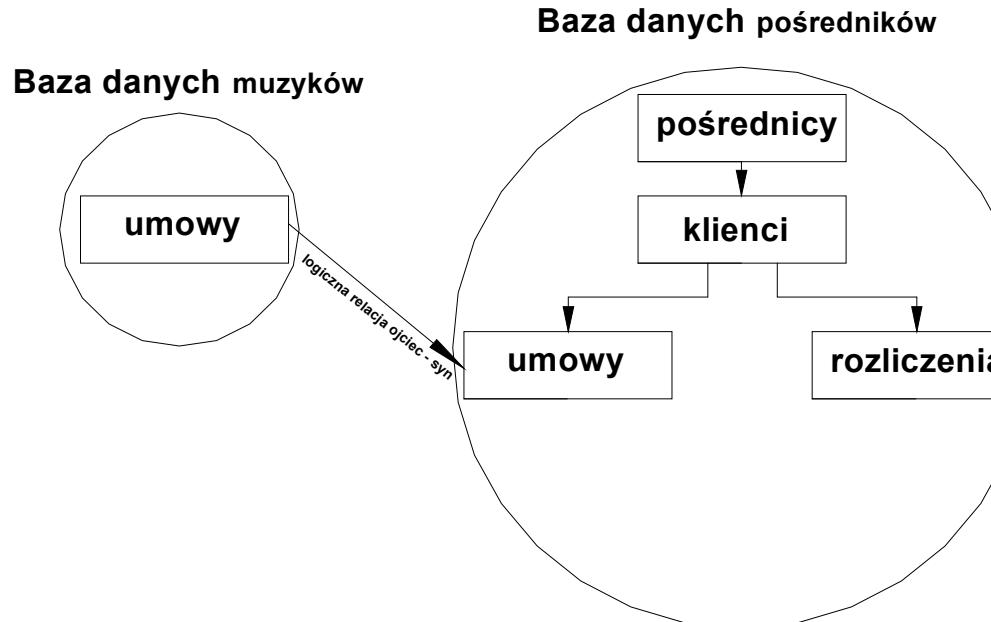
Hierarchiczne bazy danych - zalety

- potrzebne dane można szybko przywołać, ponieważ poszczególne tabele są ze sobą bezpośrednio powiązane.
- automatycznie wbudowana *integralność odwołań*. Oznacza to, że rekord w *tabeli-synu* musi być powiązany z *istniejącym* rekordem w *tabeli-ojcu*. Jeśli jeden z rekordów w *tabeli-ojcu* zostanie skasowany ulegną również wszystkie powiązane z nim rekordy w *tabelach-synów*.

Hierarchiczne bazy danych - wady

- ★ **problemy z dopisywaniem danych do tabeli-syna** - jeżeli musimy zapisać w *tabeli-synu* rekord, który nie byłby powiązany z żadnym z rekordów w *tabeli-ojcu*.
 - W przykładzie nie możemy dodać do tabeli muzyków nowej osoby, dopóki nie przypiszemy jej pośrednika w tabeli pośredników.
- ★ **nadmiarowe dane**. Wynikają z niezdolności modelu do obsługi złożonych relacji.
 - W przykładzie między klientami i muzykami istnieje relacja *wiele-do-wielu*: jeden muzyk gra dla wielu klientów, a jeden klient może zatrudniać wielu muzyków.
 - Ponieważ taki rodzaj relacji nie może zostać bezpośrednio wpisany w model HMBD, zachodzi konieczność wprowadzenia nadmiarowych danych do tabel terminarzy i umów. Tabela terminarzy będzie zawierać dane o klientach (takich np. jak: imię, nazwisko, adres oraz numer telefonu), informujące o miejscu występu danego muzyka, lecz dane te pojawiają się również w tabeli klientów. Ponadto tabela terminarzy będzie zawierać dane o muzykach (na przykład: imię, nazwisko, telefon i gatunek muzyki), mówiące nam, którzy muzycy grają dla danego klienta. Te same dane znajdują się w tabeli muzyków. Ta nadmiarowość tworzy możliwość sytuacji, w których pewne dane zostają wprowadzone w sposób niekonsekwentny, co z kolei zaburza integralność bazy.
 - Problem nadmiarowych danych można obejść przez utworzenie osobnej hierarchicznej bazy danych dla muzyków i drugiej - dla pośredników. Nowa baza muzyków będzie składać się jedynie z tabeli muzyków, natomiast baza pośredników będzie zawierać tabele pośredników, klientów, rozliczeń i umów. Tabela terminarzy nie jest już potrzebna, ponieważ można zdefiniować *logiczną relację ojciec-syn* między tabelą umów w bazie pośredników oraz tabelą muzyków w bazie muzyków. Po wprowadzeniu takiej relacji z bazy będzie można wyczytać informacje takie jak: lista muzyków, z którymi zawarł umowy dany klient, czy terminarz występów danego muzyka. Metoda ta spełnia swoje zadanie, jeśli ją zastosujemy. Twórca bazy danych musi jednak być pewien konieczności uwzględnienia takiej relacji. W tym wypadku była ona dość oczywista, lecz wiele relacji jest znacznie bardziej skomplikowanych i potrzeba ich wprowadzenia może zostać zauważona dopiero w bardzo późnych stadiach procesu projektowania lub też - o zgrozo - po oddaniu bazy do użytku.

Hierarchiczne bazy danych

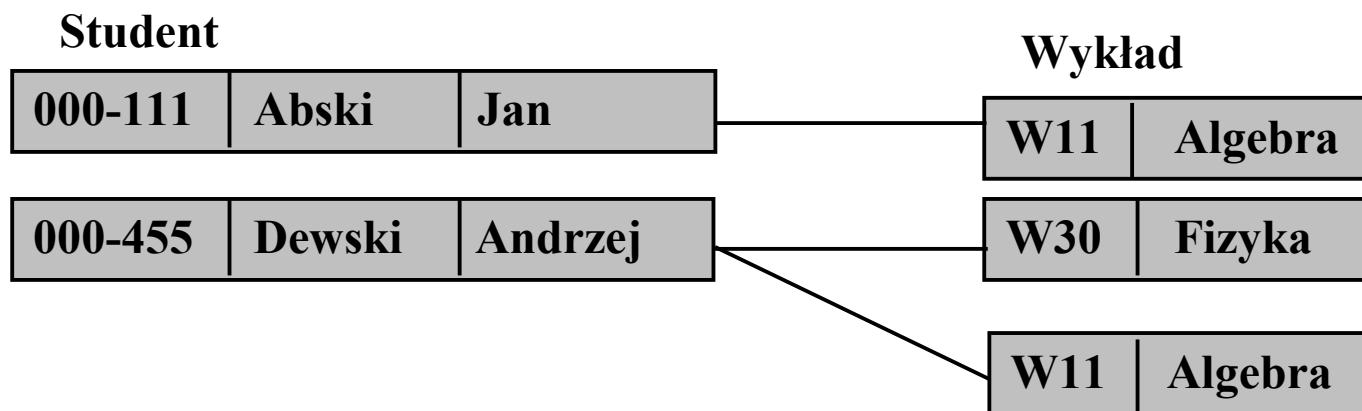


Rysunek 1.2. Wykorzystanie dwóch hierarchicznych baz danych do rozwiązania problemu relacji wiele-do-wielu

Hierarchiczny model bazy danych był z powodzeniem stosowany w systemach zapisu taśmowego, wykorzystywanych w komputerach typu *mainframe* do późnych lat siedemdziesiątych, i zdobył dużą popularność w firmach polegających na tych systemach. A jednak, pomimo tego, iż HMBD zapewniał szybki, bezpośredni dostęp do danych i znajdował zastosowanie w rozwiązywaniu wielu problemów, powoli narastała potrzeba wprowadzenia nowego modelu, który nie wymagałby wprowadzania takiej ilości nadmiarowych danych i złożonych relacji.

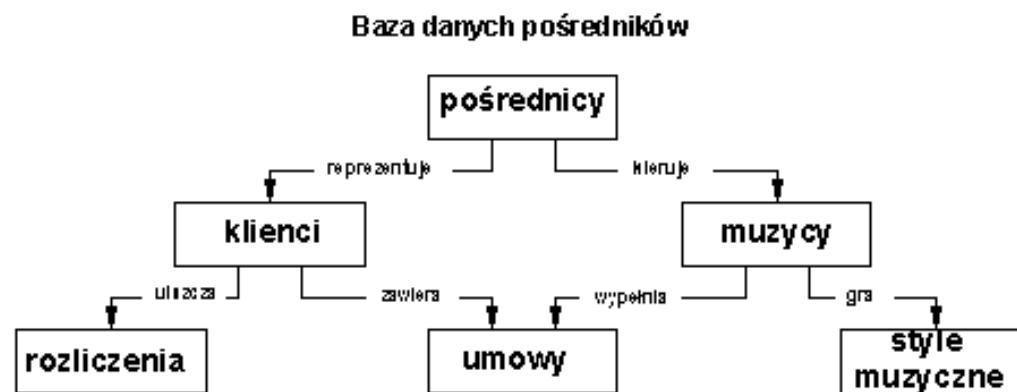
Sieciowy model danych

- ★ Sieciowy model bazy danych (SMBD) został stworzony głównie w celu rozwiązania problemów związanych z modelem hierarchicznym.
- ★ Tak jak w HMBD, strukturę SMBD można sobie wyobrazić jako odwrócone drzewo. Różnica polega jednak na tym, że w przypadku SMBD kilka drzew może dzielić ze sobą gałęzie, a każde drzewo stanowi część ogólnej struktury bazy danych.
- ★ Relacje w SMBD są definiowane przez *kolekcje* (ang. *set structureś*).
- ★ *Kolekcja* jest niejawną konstrukcją łączącą dwie tabele przez przypisanie jednej z nich roli *właściciela*, a drugiej - roli *członka*. (Stanowiło to znaczny postęp w porównaniu z relacjami *ojciec--syni*).
- ★ Kolekcje umożliwiają wprowadzanie relacji *jeden-do-wielu*, co oznacza, że w obrębie danej struktury każdy rekord z *tabeli-właściciela* może być powiązany z dowolną ilością rekordów *tabeli-członka*, lecz pojedynczemu rekordowi w *tabeli-członku* może odpowiadać tylko jeden rekord w *tabeli-właścicielu*. Ponadto każdy rekord w *tabeli-członku* musi być powiązany z istniejącym rekordem w *tabeli-właścicielu*.
 - Przykładowo, każdy student ma przyporządkowane przedmioty, które studiuje



Sieciowy model danych

- * **Baza danych pośredników.** W przykładzie pośrednik pracuje dla kilku muzyków i ma pewną liczbę klientów, którzy zamawiają u niego obsługę muzyczną różnych imprez i uiszczają opłaty. Każdy muzyk może zawrzeć wiele oddzielnych umów i specjalizować się w wielu różnych gatunkach muzyki.



Sieciowy model danych

- ★ Między każdymi dwoma tabelami można zdefiniować dowolną liczbę powiązań (kolekcji), a każda tabela może uczestniczyć w wielu różnych kolekcjach.
 - Przykładowo: tabela klientów jest powiązana z tabelą opłat przez kolekcję *uiszcza* oraz z tabelą umów przez kolekcję *zawiera*. Tabela umów, oprócz tabeli klientów, jest powiązana również z tabelą muzyków przez kolekcję *wypełnia*.
- ★ W SMBD dostęp do odpowiednich danych można uzyskać, poruszając się po kolekcjach. W przeciwieństwie do modelu hierarchicznego, gdzie poszukiwanie danych należało zacząć od podstaw, w SMBD użytkownik może rozpocząć od dowolnej tabeli, a następnie poruszać się w górę lub w dół po tabelach z nią powiązanych.

Sieciowy model danych – zalety i wady

* Zalety:

- szybkość, z jaką można odczytywać z niego dane. Ponadto użytkownik ma możliwość tworzenia znacznie bardziej złożonych zapytań niż miało to miejsce w modelu hierarchicznym.

* Wady

- ★ można uznać jednak to, że, podobnie jak w HMBD, użytkownik musi mieć dobre wyobrażenie o strukturze używanej bazy danych.
 - W przykładzie z pośrednikami na użytkownika spada ciężar pamiętania, przez które kolekcje należy przejść, aby dowiedzieć się, czy - przykładowo - należność za daną umowę została już uiszczona.
- niemożność zmiany struktury bazy danych bez ponownego tworzenia obsługujących ją programów. Jak już wiemy, relacje w SMBD są zdefiniowane za pomocą kolekcji. Danej kolekcji nie można zmienić bez modyfikowania aplikacji bazodanowej, ponieważ aplikacja ta polega na kolekcjach do wyszukiwania odpowiednich danych. Jeśli któraś kolekcja ulegnie zmianie, *wszystkie* odwołania do tej kolekcji zawarte w aplikacji obsługującej bazę danych muszą również zostać zmienione.
- ★ Pomimo iż sieciowy model logiczny był zdecydowanym krokiem naprzód w porównaniu z modelem hierarchicznym, kilku ludzi w środowisku bazodanowym nadal uważało, że musi istnieć lepszy sposób na przechowywanie i obsługę dużych ilości danych. W miarę jak rozwijał się przemysł informatyczny, użytkownicy chcieli uzyskiwać odpowiedzi na coraz bardziej złożone zapytania. To z kolei kładło coraz większy ciężar na istniejące struktury baz danych. I tak dochodzimy do modelu relacyjnego.

Wady klasycznych baz danych i SZBD

- ★ model danych (szczególnie relacyjny) jest zbyt prosty do zamodelowania złożonych, zagnieżdżonych encji,
- ★ systemy baz danych oferują tylko kilka prostych typów danych (integer, character), nie obsługują ogólnych typów danych występujących w językach programowania,
- ★ model danych nie zawiera często używanych pojęć semantycznych (np.: generalizacja, agregacja), a co za tym idzie SZBD nie oferuje mechanizmów do reprezentacji takich związków i zarządzania nimi (programiści muszą sami zaimplementować takie mechanizmy w aplikacjach),
- ★ zbyt wolne działanie systemów baz danych z programami użytkowymi wymagającymi szybkich i skomplikowanych obliczeń (szczególnie programami symulacyjnymi),
- ★ różnice między językami baz danych (SQL, DL/1, CODASYL DML), a językami programowania (COBOL, FORTRAN, PL/1, C++) zarówno pod względem wykorzystywanego modelu danych, jak i struktur danych,
- ★ systemy baz danych nie dostarczają narzędzi do reprezentowania i zarządzania temporalnymi aspektami baz danych (m.in.: pojęciem czasu, wersjami obiektów i schematu).

Obiektowy model danych

Obiektowy model danych (*object model, object-oriented model*) jest modelem danych, którego podstawą są pojęcia obiektywości, m.in.: obiekt, klasa, dziedziczenie, hermetyzacja.

- ★ Brak powszechnie akceptowalnych definicji modelu obiektowego
- ★ Trwają prace nad ustandaryzowaniem pojęć obiektowych w dziedzinie baz danych, prowadzone m.in.: przez ODMG (*Object Database Management Group*).
- ★ Standard zaproponowany przez ODMG stworzony został w oparciu o trzy istniejące standardy dotyczące baz danych (SQL-92), obiektów (OMG) i obiektowych języków programowania (ANSI).
- ★ Brak standardu wynika z faktu, iż rozwój podejścia obiektowego następował w trzech różnych obszarach:
 - językach programowania,
 - sztucznej inteligencji,
 - bazach danych.

Obiektowa baza danych

Obiektowa baza danych (*Object Database, Object-Oriented Database*) jest zbiorem obiektów, których zachowanie, stan i związki zostały określone zgodnie z obiektywnym modelem danych.

- ★ Przyjęcie obiektowego modelu, jako podstawy bazy danych, implikuje naturalną jej rozszerzalność bez konieczności wprowadzania zmian do istniejącego systemu.
- ★ Możliwe są dwa sposoby rozszerzenia obiektowych bazy danych:
 - rozszerzanie zachowania się
 - dziedziczenie.
- ★ Zachowanie się obiektu może zostać rozszerzone przez dołączenie dodatkowych programów (metod) do już istniejących. W dowolnym momencie życia systemu można zdefiniować nowe klasy korzystając już z istniejących lub przedefiniować istniejące.



Pojęcia związane z obiektowym modelem bazy danych:

Obiekty

Obiekt jest zbiorem danych i procedur. Dane są gromadzone w atrybutach obiektu. Procedury są zdefiniowane za pomocą metod obiektu. Metody są uaktywniane przez komunikaty przekazywane między obiektami.

- ★ Obiekty muszą mieć właściwość hermetyzacji. Dane i proces są zapakowane razem w ramach jednego interfejsu i udostępniane z zewnątrz w sposób kontrolowany.
- ★ Obiekt jest instancją klasy.
- ★ Obiekty mogą być rekurencyjnie powiązane między sobą związkami semantycznymi, - związki między obiektami reprezentowane są poprzez referencje (odwołania), które są wartościami atrybutów obiektu. Odwołanie do konkretnego obiektu w systemie jest realizowane przez wysłanie do niego komunikatu;
- ★ **Identyfikator** obiektu umożliwia jednoznaczne odwołanie do obiektu (jest niepowtarzalny w systemie)

Klasy

Klasa (class) - byt semantyczny rozumiany jako miejsce przechowywania (specyfikacji i definicji) takich cech grupy podobnych obiektów, które są dla nich niezmienne: atrybutów, metod, ograniczeń dostępu, dozwolonych operacji na obiektach, wyjątków, itp.

- ★ Klasa stanowi wzorzec dla tworzonego obiektu. W przypadku baz danych definiują schemat bazy danych
- ★ W systemach obiektowych, klasa jest traktowana jako obiekt (klasowy), w celu zagwarantowania jednolitego posługiwania się komunikatami. Dlatego z klasą mogą być związane atrybuty i metody klasowe.
- ★ W obiektowej bazie danych, dziedzina dowolnego atrybutu może być klasa, co zwiększa semantyczną spójność bazy (zwalnia od określania więzów spójności bezpośrednio w zbiorach wartości).
- ★ Również klasa (lub kilka klas), jako agregacja powiązanych ze sobą obiektów w bazie danych, stanowi cel do sformułowania zapytania. Zwykle klasy wiążą się ze sobą poprzez hierarchię (lub inną strukturę) dziedziczenia.

Uogólnienie – mechanizm abstrakcji lub proces, zgodnie z którym jest tworzony obiekt wyższego rzędu w celu podkreślenia podobieństw pomiędzy obiektami niższego rzędu

Atrybuty

Atrybut (attribute) jest częścią definicji klasy, specyfikacja atrybutu polega na podaniu jego nazwy i więzów semantycznej spójności, obejmujących: dziedzinę atrybutu, jednoznaczność wartości, dopuszczalność wartości NULL, itp.

- * Wartości atrybutów obiektu opisują jego stan.
- * Dziedziną atrybutu może być jakakolwiek klasa ze swoim własnym zbiorem atrybutów lub klasa wartości pierwotnych (np.: integer, string).
- * Wartością atrybutu może być instancja klasy będącej jego dziedziną lub instancja dowolnej podklasy z hierarchii klas zakorzenionej w klasie stanowiącej dziedzinę atrybutu;

Metody

Metoda (method) jest procedurą, funkcją lub operacją przypisaną do klasy i dziedziczoną przez jej podklasy.

- ★ Metoda działa na stanie obiektu tej klasy (czyli operuje wartościami atrybutów).
- ★ Kod w języku programowania implementujący metodę nazywamy ciałem metody.
- ★ Metoda abstrakcyjna specyfikowana jest w klasie, ale jej działanie może być przedefiniowane w dowolnej z jej podklas
- ★ W bazach danych metody występują w czterech postaciach
 - metody tworzenia – operacje służące do tworzenia nowych instancji klasy,
 - metody usuwania – operacje służące do usuwania zbędnych obiektów,
 - metody dostępu – operacje służące do przekazywania właściwości obiektów,
 - metody przekształcania – operacje służące do przekazywania nowych obiektów uzyskanych z obiektów istniejących



Dziedziczenie

* Typy dziedziczenia:

- dziedziczenie struktury – podkласa dziedziczy atrybuty swojej nadklasy
- dziedziczenie zachowania – podklasa dziedziczy metody swojej nadklasy

* Rodzaje dziedziczenia:

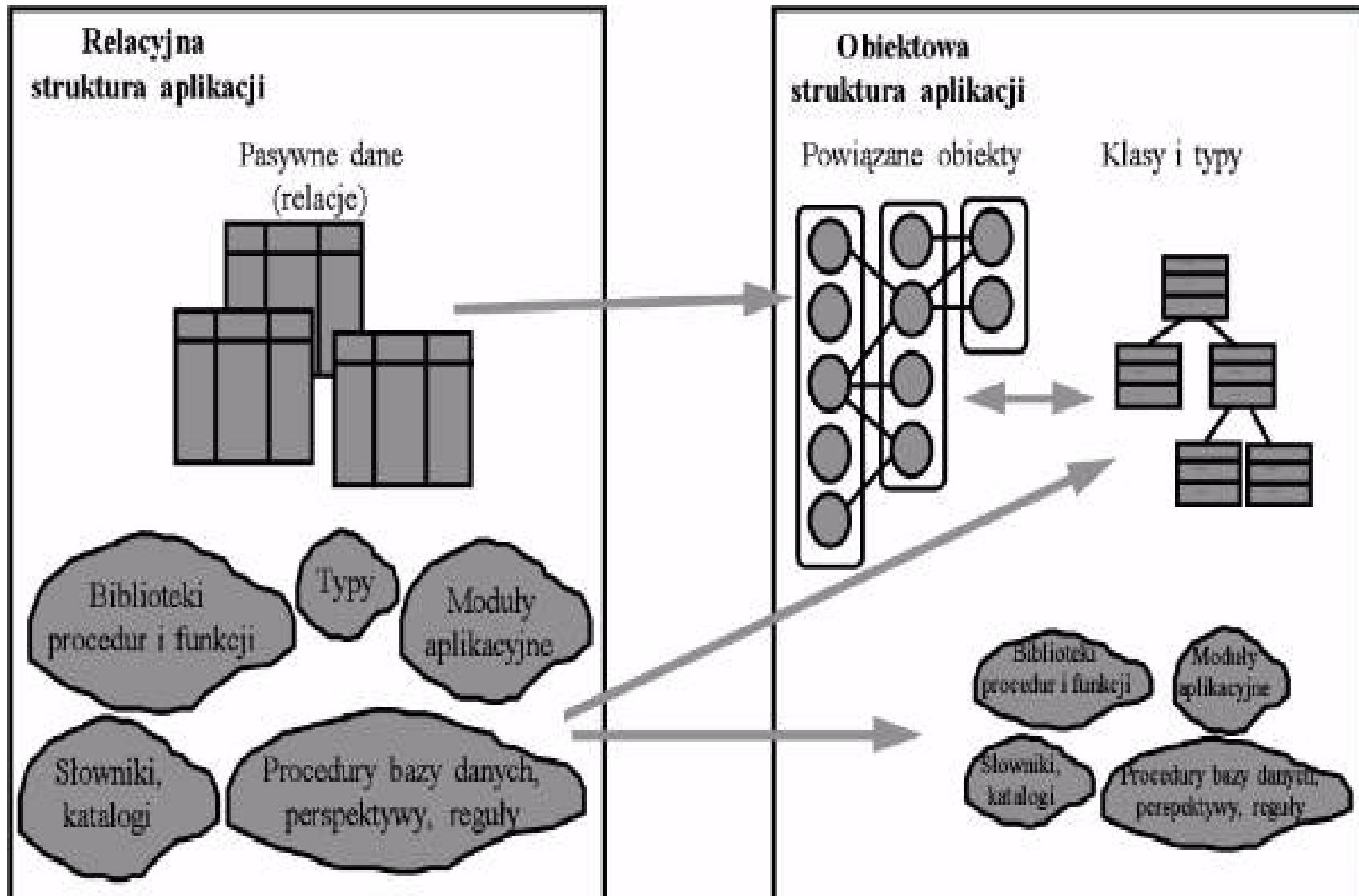
- dziedziczenie pojedyncze – klasa może być podklassą tylko jednej nadklasy
- dziedziczenie wielokrotne – klasa może dziedziczyć od więcej niż jednej nadklasy

Hermetyzacja

Hermetyzacja (encapsulation) – upakowanie razem danych i procedur w zdefiniowanym interfejsie.

- ★ Hermetyzacja jest podstawową techniką abstrakcji, tj. ukrycia wszelkich szczegółów danego przedmiotu lub bytu programistycznego, które na danym etapie rozpatrywania (analizy, projektowania, programowania) nie stanowią jego istotnej charakterystyki.
- ★ W „obiektywości” wyróżnia się:
 - koncepcję **ortodoksyjnej hermetyzacji** (Smalltalk), w której wszelkie operacje, które można wykonać na obiekcie są określone przez metody przypisane do obiektu (znajdujące się w jego klasie i nadklasach). Bezpośredni dostęp do atrybutów obiektu jest niemożliwy,
 - **hermetyzacja ortogonalna** (C++), gdzie dowolny atrybut obiektu (i dowolna metoda) może być prywatny (niedostępny z zewnątrz) lub publiczny;

Relacyjna i obiektowa struktura aplikacji



Zalety obiektowych baz danych

- ★ złożone obiekty
- ★ typy danych definiowane przez użytkownika
- ★ tożsamość obiektów (identyfikator), trwałość
- ★ hermetyzacja, hierarchia, dziedziczenie
- ★ rozszerzalność
- ★ zgodność we wszystkich fazach życia bazy i danych
- ★ metody i funkcje przechowywane wraz z danymi
- ★ nowe możliwości (wersjonowanie, rejestracja zmian, powiadamianie ...)
- ★ możliwość nowych zastosowań mniejszym kosztem (bazy multimedialne, przestrzenne, bazy aktywne...)
- ★ porównywalna wydajność (i wciąż rośnie)

Wady obiektowych baz danych

- ★ brak optymalizacji zapytań (rozumianej jak w poprzednich modelach)
- ★ niedopracowane mechanizmy zarządzania dużą bazą obiektów, sterowania wersjami, ...
- ★ mała liczba ekspertów od technik obiektowych
- ★ nie wiadomo z jakimi kosztami wiąże się migracja dużych systemów
- ★ brak dopracowanych standardów