

# Przetwarzanie i zarządzanie danymi

Uniwersytet Morski w Gdyni  
Narzędzia Informatyczne

# Przykładowe bazy danych

- Książka: telefoniczna, adresowa, kucharska
- Terminarze i organizery
- System zarządzania magazynem czy hurtownią
- Bazy danych w systemie bankowym, ubezpieczeniowym (informacje o kontach, informacje personalne...)
- Systemy rezerwacji miejsc (hotele, przychodzie, gabinety...)
- Systemy obsługi administracji (ZUS, NFZ)
- Powszechny Elektroniczny System Ewidencji Ludności
- Systemy wspomagające projektowanie CAD/CAM
- Magazyny (hurtownie) danych
- Systemy wspomagania inżynierii oprogramowania CASE
- Systemy z bazami wiedzy
- Systemy medyczne

# Wstęp

- Baza Danych (BD - *Database*) – jest to zbiór powiązanych ze sobą logicznie danych, zorganizowanych (uporządkowanych) zgodnie z określonym modelem danych (w ściśle określony sposób).
- Baza danych jest rodzajem systemu informatycznego, który został wyspecjalizowanym pod względem:
  - Wprowadzania,
  - Przechowywania,
  - Przetwarzania,
  - Udostępniania.

# System Baz Danych

- Sposób przechowywania informacji w bazie danych jest zdefiniowany i logiczny, między innymi dlatego bazy danych umożliwiają wygodny i szybki dostęp do zgromadzonych w nich danych. Forma jak i metody wykorzystywane przy przetwarzaniu i zarządzaniu danymi w bazie definiują system baz danych (SBD).
- SBD gwarantuje trwałość i spójność danych przechowywany w bazie.
- SBD zapewnia autoryzację dostępu do bazy danych.
- SBD zapewnia współbieżny dostęp do danych.
- SBD zapewnia efektywny dostęp do danych w środowisku scentralizowanym i rozproszonym.

# System Baz Danych

- Elementy składowe **SYSTEMU BAZ DANYCH (SBD)** to: trwała pamięć zewnętrzna, system zarządzania bazami danych i specjalizowane języki zapytań.
- W przypadku SBD trwała pamięć zewnętrzna to rodzaj pamięci masowej o dużej pojemności, przeznaczony do długotrwałego przechowywania danych.
- W SBD językami zapytań są języki wysokiego poziomu, za pomocą których użytkownicy komunikuje się z bazą danych.

# System Zarządzania Bazą Danych

- System Zarządzania Bazą Danych (SZBD) – jest to oprogramowanie zapewniające dostęp i przetwarzanie danych zawartych w bazie (*Database Management System*).
- System Zarządzania Bazą Danych (SZBD) – jest to oprogramowanie umożliwiające użytkownikom (też programom) definiowanie, tworzenie i zarządzanie bazą danych oraz kontrolowanie dostępu do niej.

# Cechy SBD

- **Niezawodność** - częstotliwość występowania błędów powinna być pomijalnie mała.

Niezawodność systemu baz danych zapewnia się przez:

- » dublowanie urządzeń pamięci zewnętrznych (systemy wielodyskowe)
- » mechanizmy księgowania
- » kontrolę poprawności zapisu
- » kody detekcji
- » korekcję błędów

- **Ochrona danych** – zgodność z rzeczywistością. Dane w bazie danych są odwzorowaniem rzeczywistości. Jeżeli modelowany fragment rzeczywistości ulegnie zmianie, baza danych również musi się zmienić.

# Cechy SBD

- **Trwałość danych** – oznacza, że dane powinny być przechowywane w pamięci tak długo, jak tego wymagają użytkownicy systemu baz danych.
- **Niezależność danych** – pozwala osiągnąć większą elastyczność, ponieważ programy wymieniające informacje z bazą danych są niezależne od przechowywania danych na dysku i szczegółów reprezentacji danych na dysku (np. format danych na dysku).

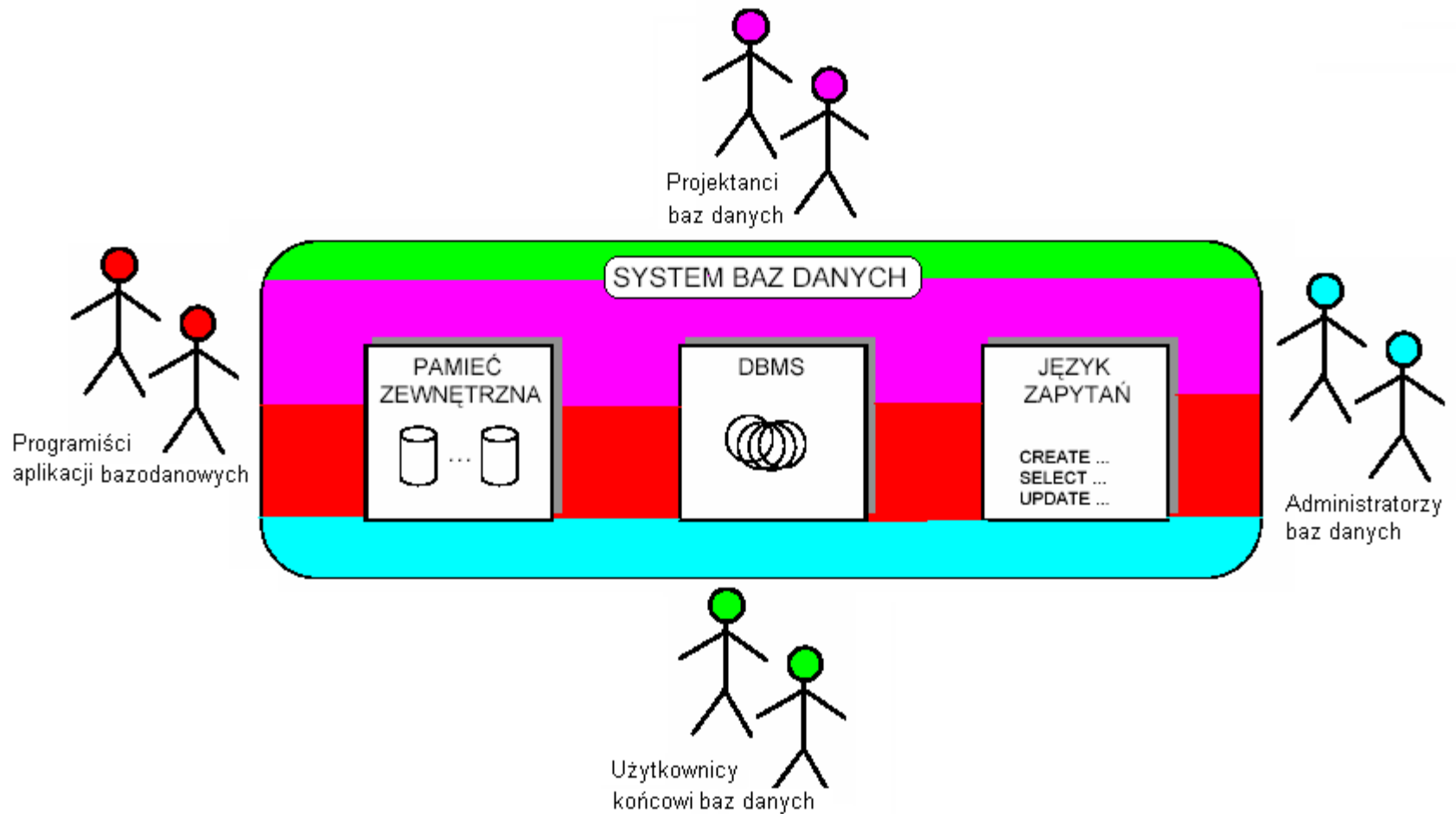


# Użytkownicy

Na rysunku przedstawiającym schemat Systemu Bazy Danych wyróżniono czterech podstawowych użytkowników. Pełnione przez nich role można zawęzić do:

- **rola twórców bazy danych**, których zadaniem jest zaprojektowanie i zaprogramowanie bazy danych – projektanci bazy danych i programiści aplikacji
- **rola użytkowników interaktywnych o różnych poziomach uprawnień**, których zadaniem jest przetwarzanie zapisanych w bazie informacji oraz zarządzanie istniejącą bazą danych - użytkownicy końcowi i administratorzy bazy danych

# System Baz Danych



# Rola i zadania administratora SBD

- **Administrator bazy danych (Data Base Administrator)**, to osoba sprawująca kontrolę nad dostępem do bazy danych i nad jej funkcjonowaniem.
- **Zadania administratora bazy danych:**
  - udzielanie użytkownikom i grupom użytkowników uprawnień dostępu do bazy danych (uprawnienia do tworzenia bazy danych, pobierania i modyfikowania danych i inne)
  - monitorowanie wydajności bazy danych
  - definiowanie strategii archiwizowania i odtwarzania danych po awariach
  - uruchamianie nowych strategii dostępu do danych

# Rola i zadania administratora SBD

- **Narzędzia, z których korzysta administrator bazy danych to:**
  - programy do ładowania bazy danych z plików zewnętrznych
  - programy do archiwizacji danych
  - programy do przywracania danych z archiwum
  - programy do zbierania statystyk i ich analizowania
  - programy do definiowania użytkowników i grup użytkowników
  - programy do operowania na *słowniku danych*
  - programy do analizowania dziennika bazy danych (*logu*)

# Języki zapytań

Najczęściej, w tworzonym SBD wykorzystywanych jest kilka języków programowania, które wzajemnie się uzupełniają. Z innego języka programowania korzysta się przy tworzeniu struktury bazy danych, a innego przy zarządzaniu danymi zapisanymi w bazie. Z różnych języków programowania korzystają użytkownicy bazy danych.

# Przykładowe wymagania na języki programowania:

- **Użytkownicy końcowi i administratorzy** używają zazwyczaj poleceń wybieranych z menu lub wprowadzają komendy za pośrednictwem wiersza poleceń.
- **Programiści aplikacji wymagają złożonych języków programowania, które udostępnią:**
  - instrukcje definiowania i operowania na danych z bazy danych,
  - instrukcje wejścia/wyjścia,
  - instrukcje i struktury danych typowe dla języków programowania (język *czwartej generacji* - 4GL);

# Języki wykorzystywane w SBD

- Typowe języki programowania wykorzystywane przy obsłudze SBD to: Java, C++, C#, Delphi, PHP, Cobol i inne.
- Wymienione języki programowania są zazwyczaj rozszerzane o instrukcje operujące na bazie danych. Są to głównie instrukcje języka zapytań bazujące na SQL.
- Podstały też modyfikacje języka zapytań SQL, które wykorzystywane są w predefiniowanych bazach danych, np.: PostgreSQL (T-SQL), Oracle (PL/SQL).

# Języki zapytań

=

**Język definiowania danych (DataDefinitionLanguage)**

+

**Język manipulowania danymi (DataManipulationLanguage)**

+

**Język kontrolowania danych (DataControl Language)**

- **DDL** - definiuje struktury danych, na których operują instrukcje DML
- **DML** - wykonuje operacje na danych zdefiniowanych przez DDL (wybieranie i aktualizowanie)
- **DCL** - kontroluje uprawnienia dostępu do danych



# Przykładowe technologie wykorzystywane przy realizacji systemów baz danych

- **Aplikacje Desktopowe:**

- Java 2 Standard Edition
- Microsoft .NET Framework
- MFC (Microsoft Foundation Classes)

- **Aplikacje Serwerowe:**

- Java 2 Enterprise Edition
- ASP.NET
- PHP
- XHTML

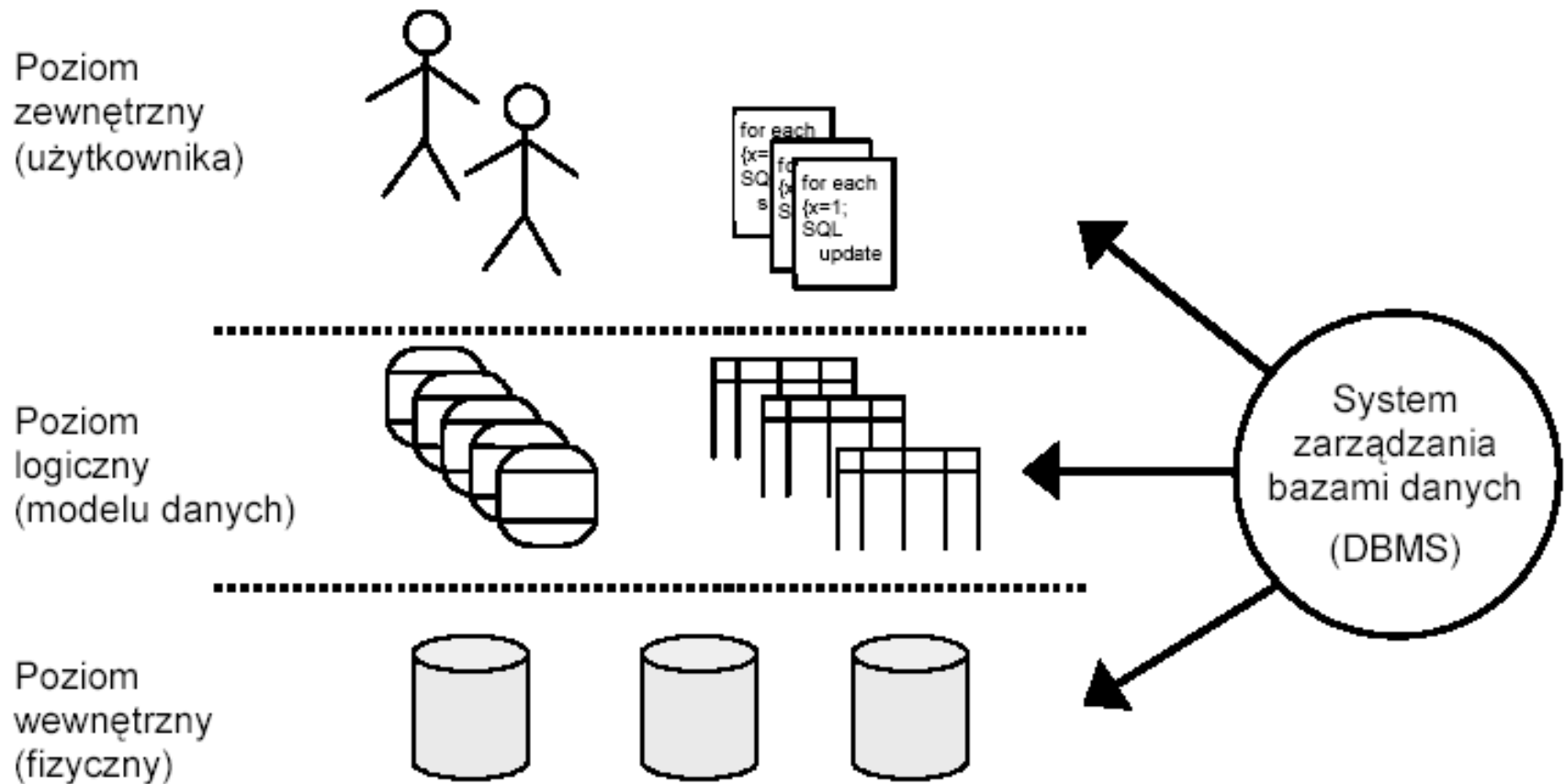
- **Bazy Danych:**

- Oracle
- MySQL
- PostgreSQL
- MS SQL Server
- ODBC
- Sybas
- IBM DB2
- InterBase
- FrontBase
- Solid

# Architektura Systemu Bazy Danych

- **Poziom zewnętrzny (użytkownika)**
  - zbiór zewnętrznych widoków, poprzez które użytkownicy widzą zawartość bazy danych (interfejs aplikacji).
  - widok jest zdefiniowany z wykorzystaniem języka zapytań.
- **Poziom wewnętrzny (fizyczny)**
  - fizyczna reprezentacja bazy danych w postaci plików dyskowych i algorytmów dostępu do nich.
- **Poziom logiczny (modelu danych)**
  - zawiera model konceptualny (myślowy) bazy danych. Model ten obejmuje całą zawartość bazy danych, tak jak ją widzi administrator lub właściciel bazy danych.

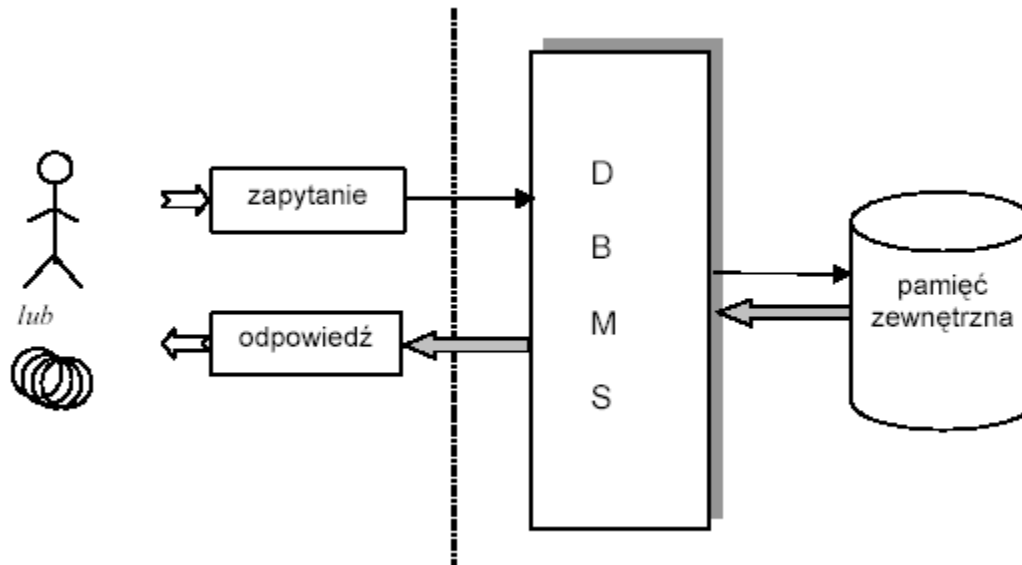
# Architektura Systemu Bazy Danych



# Dostęp do bazy danych

## UŻYTKOWNIK

- Człowiek przy terminalu, który interakcyjnie operuje na bazie danych, formułując żądania w języku :
- Program, który oprócz innych instrukcji zawiera instrukcje operujące na danych z bazy danych. Instrukcje te formułowane są w języku zapytań uzupełnionym przez język macierzysty.



# Modele Systemów Baz Danych

- **Modele danych to zintegrowany zbiór zasad opisujących dane, relacje, powiązania (stosunki) pomiędzy danymi, dozwolone operacje i ograniczenia nakładane na dane i operacje.**
- **Model danych jest próbą reprezentacji świata realnego i występujących w nim obiektów, zdarzeń oraz związków zachodzących między nimi.**

# Modele Systemów Baz Danych

- **modele proste**, gdzie dane są zorganizowane w postaci systemu plików.
- **modele klasyczne**, do których zalicza się model: hierarchiczny, sieciowy i relacyjny. Ten ostatni jest obecnie najbardziej popularny wśród obecnych systemów baz danych.
- **modele semantyczne** czyli znaczeniowe. 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.

# Model prosty (płaski)

- Rozważmy strukturę pliku wyposażenie.txt, w której wyróżnić można 5 pól:

KOD\_DZIAL, WYPOSAZENIE, SERIA, DATA, GWARANCJA.

Variant 1 (separacja danych uzyskana za pomocą średnika)

```
KOD_DZIAL;WYPOSAZENIE;SERIA;DATA;GWARANCJA
SP;kalkulator;012432;17-mar-92;Nie
CH;kalkulator;452T54;08-wrz-88;Nie
ZA;komptuer PC;233RTQ;24-lip-89;Tak
EE;druk_las;984310WE;27-lut-92;Nie
AD;modem;RT671007;18-lip-92;Tak
RE;autom_sekretarka;126001;24-kwi-91;Tak
ZA;fax;67645IP;30-kwi-90;Tak
SP;druk_igłowa;8976500B;03-lis-91;Tak
RE;fax;4538IP;01-maj-92;Tak
```

# Model prosty (płaski)

- Rozważmy strukturę pliku wyposażenie.txt, w której wyróżnić można 5 pól:

KOD\_DZIAL, WYPOSAZENIE, SERIA, DATA, GWARANCJA.

Wariant 2 (separacja danych uzyskana za pomocą tabulacji)

KOD_DZIAL	WYPOSAZENIE	SERIA	DATA	GWARANCJA
SP	kalkulator	012432	17-mar-92	Nie
CH	kalkulator	452T54	08-wrz-88	Nie
ZA	komputer PC	233RTQ	24-lip-89	Tak
EE	druk_las	984310WE	27-lut-92	Nie
AD	modem	RT671007	18-lip-92	Tak
RE	autom_sekretarka	126001	24-kwi-91	Tak
ZA	fax	67645IP	30-kwi-90	Tak
SP	druk_igłowa	8976500B	03-lis-91	Tak
RE	fax	4538IP	01-maj-92	Tak



# Model prosty

- W przypadku danych zapisanych w pliku wyposażenie.txt zastosowano klucz podziału na rekordy i pola (variant 1 – średnik, variant 2 - tabulacja).
- W ogólnym przypadku, takim kluczem może być zarówno jakiś specjalny znak (na przykład kropka, czy przecinek -bazy CSV), jak też długość każdego pola z danymi.
- Do odpowiednich danych możliwy jest dostęp na podstawie znanego klucza, którego wyszukiwanie realizowane jest przez napisany w tym celu program, np. w C++ czy Delphi.
- Najczęściej kolejne rekordy oddzielone są znakami

# Model prosty – wady/zalety

- Dane w bazach płaskich mogą być wyłącznie ciągiem znaków. Co prawda dla niektórych języków programowania nie stwarza to żadnego problemu (patrz język Perl czy PHP), jednak dla języków niższego poziomu oznacza to częste wywoływanie funkcji konwertujących napisy do liczb, czy dat i odwrotnie.
- Ponieważ dane są zapisywane w postaci tekstowej, zatem zajmują dość sporo miejsca. Nie ma tutaj żadnej kompresji ani szybkiego dostępu do danych już skompresowanych.

# Model prosty – wady/zalety

- Dane w bazach są zorganizowane sekwencyjnie. Oznacza to konieczność dostępu sekwencyjnego, czyli bardzo powolnego i zasobożernego. Każde nowe rekordy są natomiast dopisywane na koniec danych.
- Nie ma praktycznie żadnego ograniczenia dostępu do danych w takiej bazy. Każda tablica jest po prostu jednym plikiem tekstowym. Ograniczać dostęp można więc tylko z poziomu systemu operacyjnego, co przy niektórych zabezpieczeniach systemów operacyjnych budzi poważne wątpliwości dotyczące bezpieczeństwa danych.

# Model prosty (płaski) – wady/zalety

- Nie ma żadnego ograniczenia w kolejności dostępu do takiej bazy danych. Można sobie wyobrazić, że jeśli do takiej bazy danych dostęp miałyby więcej niż jedna osoba, wtedy istnieje duże prawdopodobieństwo, że podczas gdy jedna osoba chciałaby czytać pewne dane, inna może chcieć je w tym momencie usunąć. Jest to dość niebezpieczna sytuacja - nie ma określonych priorytetów i tzw. zamykania na jakiś czas dostępu do danych.
- Nie ma zdefiniowanych relacji między danymi, zatem dane mogą się niepotrzebnie wielokrotnie powtarzać.

# Model prosty (płaski) – wady/zalety

- Jeśli separatorem pól danych są znaki specjalne, to oznacza to, że w zawartości pól nie może wystąpić taka sekwencja znaków, która normalnie rozdziela pola wierszy.
- Wybrnąć z tego można tylko w jeden sposób: należy zdefiniować naprawdę unikalny ciąg znaków, który rozdziela pola wierszy. Jednak taka definicja w dalszej perspektywie jest mało prawdopodobna, gdyż i tak może się zdarzyć, że w przyszłości ktoś wprowadzi do takiej bazy dane, które normalnie dzielą dane na pola.

# Model prosty (płaski) – wady/zalety

- Jeśli kluczem do podziału pól danych jest ściśle zdefiniowana długość pól danych, to istnieje dość ograniczony zasięg do definicji danych, gdyż pole o ograniczonej długości może pomieścić tylko dane o ograniczonej długości. Takie ograniczenie może w przyszłości spowodować obcinanie danych do określonej w programie długości danych, w przypadku, kiedy do bazy zostaną wprowadzone dane o długości większej niż przewidziane wcześniej przy projektowaniu bazy danych.
- Jedynym rozwiązaniem jest tutaj zdefiniowanie wcześniej odpowiednio długiego pola, tak aby w przyszłości nie było możliwe nigdy przekroczenie wymaganej długości. Jednak takie postępowanie powoduje, że dane, które normalnie zajmują bardzo niewiele miejsca w takiej bazie danych, w której na zapas zdefiniowano dla danych więcej miejsca, mogą zajmować kilkakrotnie więcej miejsca niż by musiały zajmować ze względu na swoją długość.

# Model relacyjny

- Relacyjny model danych został opracowany przez E. F. Codd w latach 70-80.
- Ze względu na proste zasady nim rządzące model relacyjny stanowi podstawową architekturę dla większości popularnych dziś systemów baz danych.

# Model relacyjny

- Podstawową strukturą danych jest relacja. W bazach danych relacja reprezentowana jest w postaci tabeli.
- Relacja jest zbiorem wierszy, z których każdy ma taką samą strukturę, lecz różne wartości.
- Każdy wiersz musi posiadać co najmniej jedną kolumnę w tablicy.



# Model relacyjny

- Przykładowa tabela z wyróżnionymi w niej wierszami, kolumnami i polami.

The diagram illustrates a relational database table with the following structure and data:

PESEL	Imię	Nazwisko	Wykształcenie
72030400264	Adam	Kowalski	Podstawowe
65083104569	Marcin	Kowalski	Średnie
71123108664	Marcin	Dąbrowski	Wyższe
85041406333	Anna	Kowalska	Podstawowe
46111006545	Anna	Dąbrowska	Wyższe

Annotations and their meanings:

- wiersz = rekord = encja**: Points to the first row of the table.
- kolumna = atrybut**: Points to the 'Imię' column.
- pole**: Points to the 'Wyższe' cell in the 'Wykształcenie' column.
- wartość**: Points to the 'Wyższe' cell in the 'Wykształcenie' column.

# Model relacyjny

- Najważniejsze własności tablicy:
  - wiersze są unikalne (**wiersz = rekord = encja = krotka**)
  - kolumny są unikalne (**kolumna = atrybut**)
  - kolejność wierszy w tablicy nie ma znaczenia
  - kolejność kolumn nie ma znaczenia
  - wartości pól są atomowe, tzn. swoiste, niezależne od innych, ale o określonym typie.

# Model relacyjny

- Najważniejsze własności tablicy (cd):
  - brakowi wartości odpowiada wartość specjalna NULL zgodna z każdym typem kolumny (chyba, że została jawnie wykluczona przez definicję typu kolumny).
  - każda tabela zawiera klucz główny -- kolumnę (lub kolumny), której wartości jednoznacznie identyfikują wiersz (a więc, wartości klucza głównego nie mogą się powtarzać w obrębie tabeli).
  - Wartością klucza głównego nie może być NULL.

# Model relacyjny

- Wiersze występujące w jednej tabeli nie powinny się powtarzać - zabezpieczeniem przed ich powtórzeniem się jest wprowadzenie klucza głównego (w przykładzie powyżej było to pole PESEL ).
- Wiersze w odróżnieniu od kolumn są dynamiczne (zmieniają się), ponieważ tego wymaga zasada działania relacyjnej bazy danych.
- Wiersze w tabeli są: dopisywane, modyfikowane lub kasowane.

# Model relacyjny

- Na etapie projektowania tabeli ustalana jest liczba i nazwa kolumn w niej zawartych.
- Rzadko dopisuje się lub kasuje kolumny - ponieważ każda z kolumn reprezentuje pewną własność modelowanej rzeczywistości.
- Bazy danych posiadają mechanizmy do zmiany definicji tabeli, w tym do zmiany ilości i nazw kolumn.

# Klucze

## Własności klucza:

- **Jednoznaczność:**

- Nie istnieją takie dwa wiersze, które mają takie same wartości kolumn kluczowych.
- Innymi słowy: wartość klucza jednoznacznie identyfikuje wiersz.

- **Minimalność** (dot. kluczy złożonych):

- Nie można z klucza usunąć żadnej kolumny bez naruszenia wymogu jednoznaczności.
- Innymi słowy każda składowa klucza złożonego jest potrzebna.

# Klucze

- **Klucz wiersza (klucz kandydujący)** – to kolumna (lub zbiór kolumn) tabeli spełniający warunki: jednoznaczności i minimalności.
- W praktyce oznacza to, że kombinacja wartości tych kolumn jest unikalna w ramach całej tabeli, oraz spośród kolumn klucza kandydującego nie można wybrać podzbioru kolumn, który również zapewnia unikatowość wierszy w tabeli.

# Klucze

- Ze względu na liczbę kolumn tworzących klucz wyróżniamy:
  - Klucz prosty – obejmuje tylko jedną kolumnę
  - Klucz złożony – obejmuje kilka kolumn



# Klucze

- Ze względu na budowanie relacji między tabelami wyróżniamy:
  - **Klucz główny** (inaczej **klucz podstawowy**) tabeli jest arbitralnie wybranym kluczem kandydującym relacji. Pozostałe klucze jeśli istnieją są kluczami alternatywnymi.  
Umożliwia jednoznaczne wskazanie na wiersz w tabeli, w której jest powołany.
  - **Klucz obcy** - jest wykorzystywany do tworzenia powiązań (relacji) między kilkoma oddzielnymi tabelami.

# Klucze

**Klienci**

KlientID	Imię	Nazwisko	Adres
101	Joanna	Kowalska	Wierzbowa 25
103	Anna	Kowalska	Narcyzowa 1
104	Anna	Dąbrowska	Fiołkowa 11

Klucz podstawowy

Klucz obcy

**Zamowienia**

ZamowienieID	KlientID	Wartość	Data
302	101	255,2	2006-01-12
303	103	1651,1	2006-01-12
304	101	12,25	2006-01-14
305	101	1321,1	2006-01-14

# Klucze

## **Uwaga:**

- Ponieważ tabela jest zbiorem, zawsze ma przynajmniej jeden klucz (w zbiorze nie ma powtarzających się elementów, zatem nie ma dwóch identycznych wierszy i na pewno istnieje pewna kombinacja kolumn, która spełnia warunek jednoznaczności i minimalności. W skrajnym przypadku jest to kombinacja wszystkich kolumn tabeli.

## **Wniosek:**

- Każda tabela ma klucz główny.

# Model relacyjny

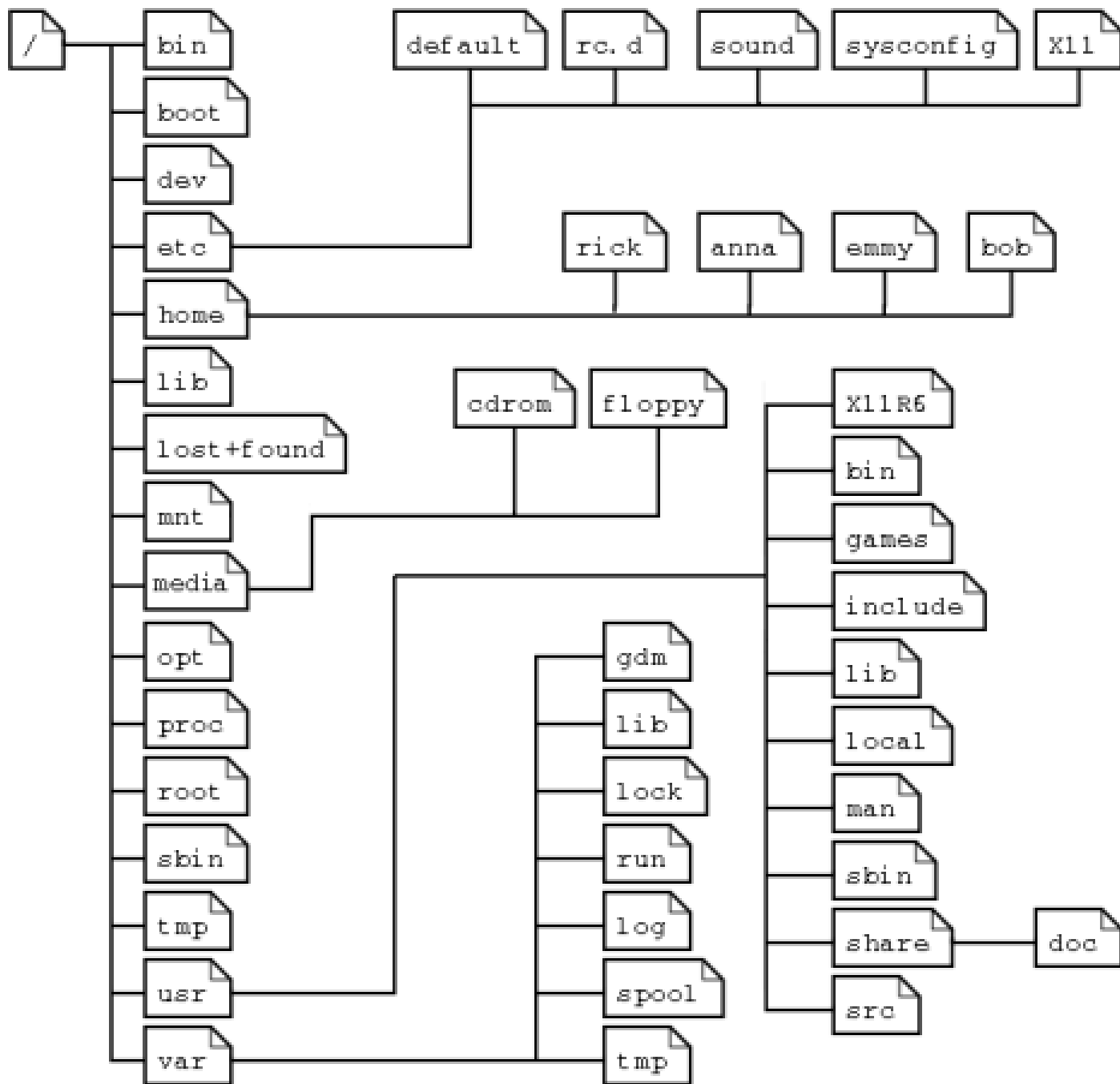
Relacyjne bazy danych są dużo wydajniejsze niż pliki tekstowe, a ich obsługa za pomocą języka zapytań SQL jest prosta, przyjemna i wszechstronna.

Pola są określonego typu, dlatego można w nich łatwo przechowywać teksty, liczby, daty czy obrazki.

Klucze podstawowe umożliwiają jednoznaczną identyfikację danego wiersza.

W relacyjnych bazach danych kolejność wierszy nie ma znaczenia, ponieważ za pomocą języka SQL można nie tylko wybrać interesujące nas wiersze, ale również je sortować.

- M
- M
- C



u

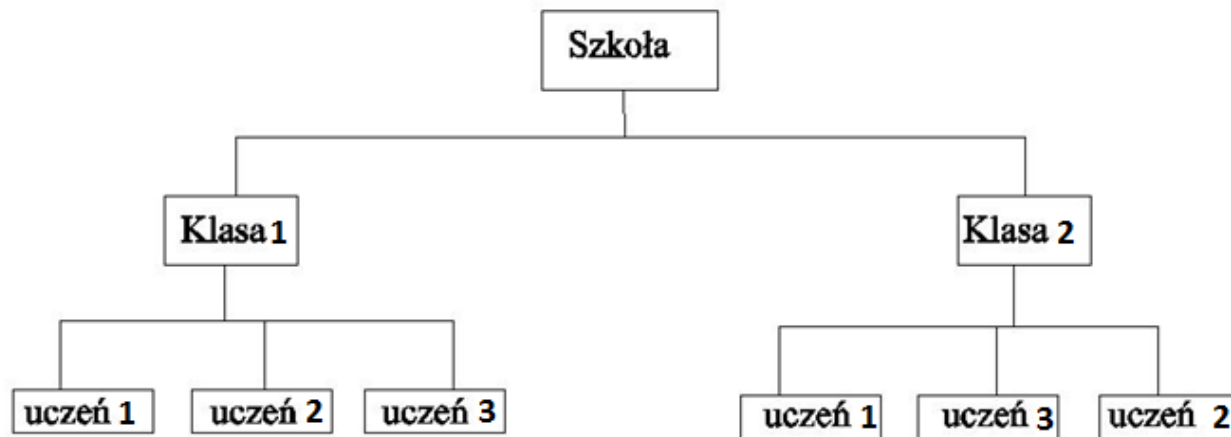
ch się

w

ki, ale

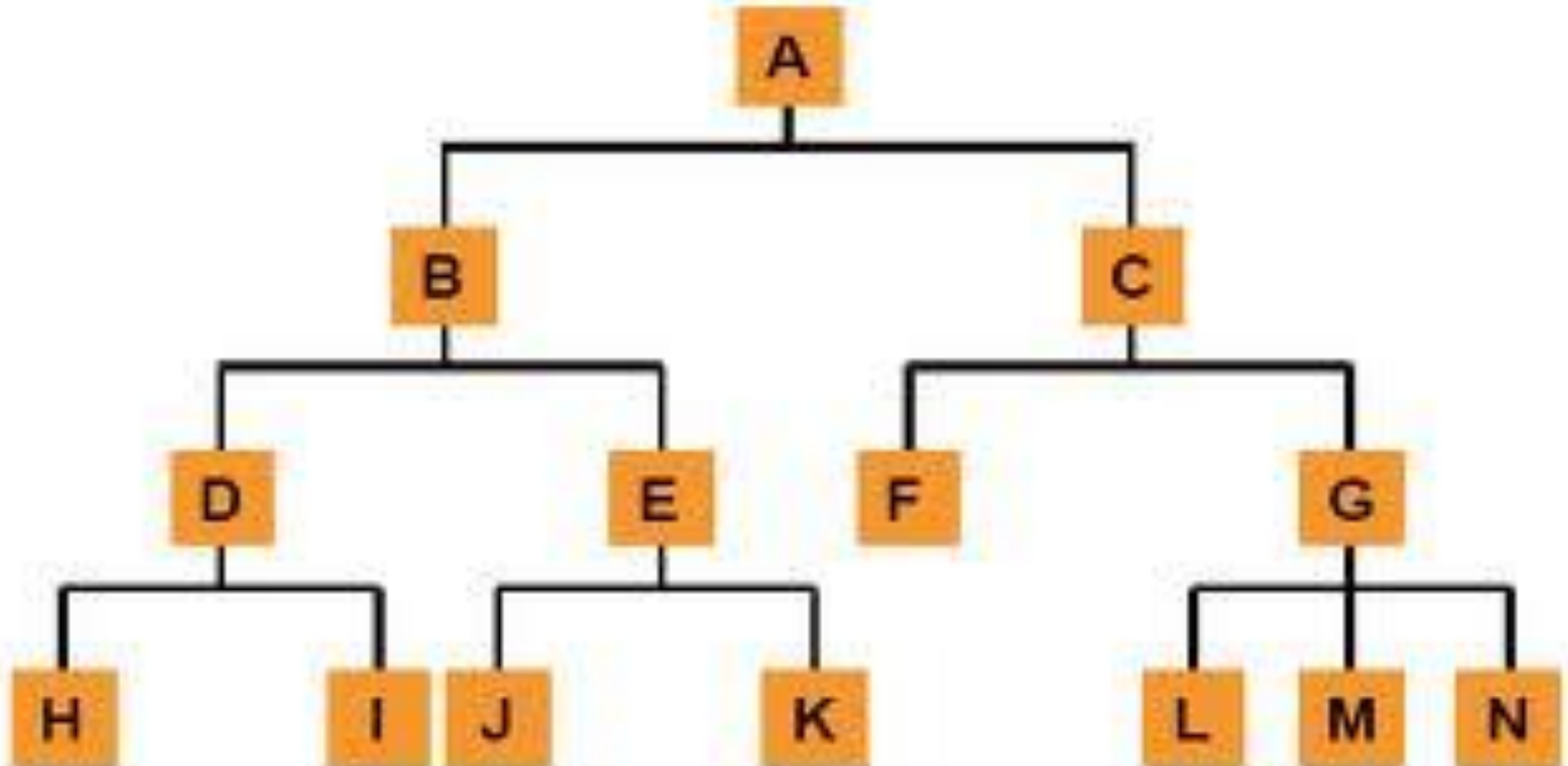
# Model hierarchiczny

- Model hierarchiczny baz danych przypomina strukturę odwróconego drzewa – istnieje jeden korzeń (tabela nadrzędna) oraz dzieci (tabele podrzędne). Jeden ojciec może mieć wiele dzieci, ale każde dziecko ma tylko jednego ojca.



BAZA HIERARCHICZNA

# Model hierarchiczny



# Model hierarchiczny

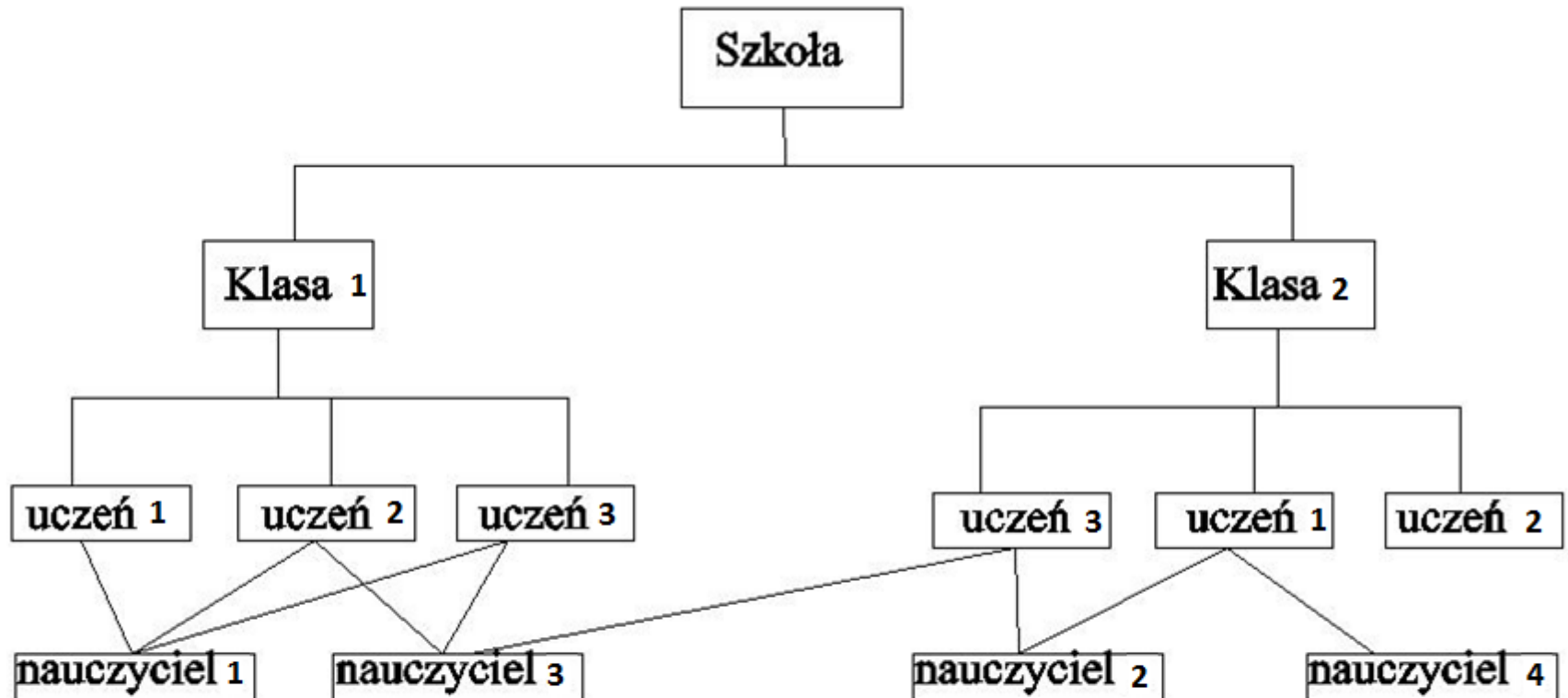
- W modelu hierarchicznym rekord składa się ze zbioru nazwanych pól, każde pole służy do zapisu pojedynczego atrybutu obiektu opisywanego przez rekord, i charakteryzuje się określonym typem danych, np. liczba całkowita, napis, data, itp.
- Na ogół jedno z pól danego typu rekordu wyróżnia się jako klucz, tj. unikalny identyfikator rekordu wśród rekordów danego typu (często przydzielany dość arbitralnie, podobnie jak np. nr słuchacza lub nr PESEL w ewidencji ludności) oraz zakłada się uporządkowanie rekordów wg. wartości jednego z pól.



# Model hierarchiczny

- Typy rekordów tworzą strukturę drzewa, tzn. każdy typ wiersza związany jest dokładnie z jednym typem nadrzędnym (nie dotyczy to wiersza najwyższego w hierarchii).
- Jednocześnie każdy określony wiersz podrzędny jest związany z określonym wierszem typu nadrzędnego.
- Wyszukiwanie odpowiednich wierszy odbywa się poprzez sprawdzanie spełniania określonych warunków i w ten sposób odbywa się wędrowanie po drzewie.

# Model sieciowy

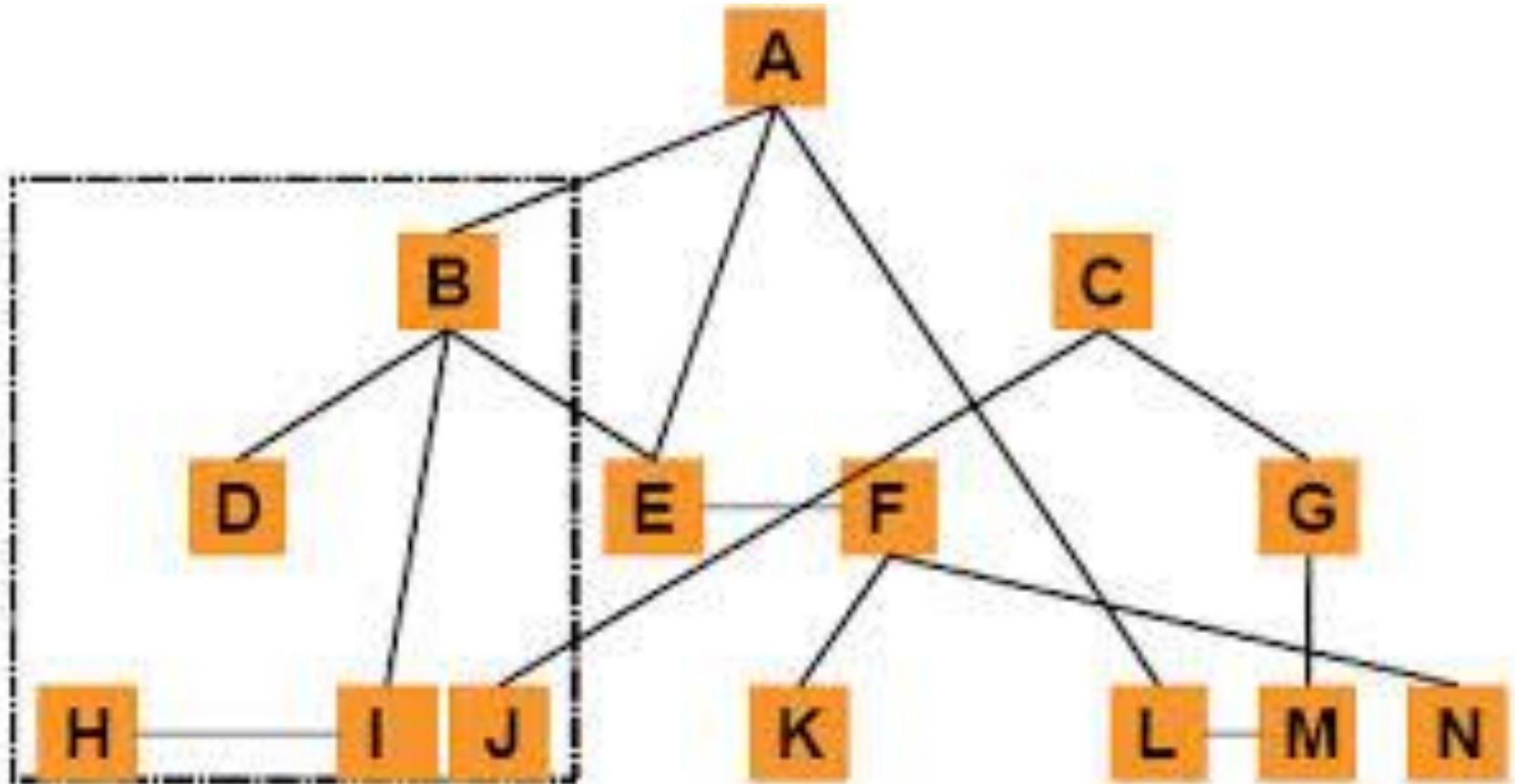


Baza sieciowa

# Model sieciowy

- Model sieciowy jest podobny do modelu hierarchicznego. Zamiast związku nadrzędny-podrzędny między rekordami obowiązuje tutaj tzw. typ kolekcji, który jest specjalnym, złożonym typem danych. W tym typie danych zawarte są odniesienia do innych rekordów określonego typu. Jest to więc podobny schemat do struktury w języku C.
- Integralność danych jest zapewniona dzięki zgodności zawartości pól wierszy z określeniem typu wierszy i oczywiście unikalności pól kluczowych.

# Model sieciowy



# Jaki jest cel normalizacji

- Normalizacja relacji (tabeli) ma na celu takie jej przekształcenie, by nie posiadała ona cech niepożądanych.
- W przypadku tablicy należy ją tak zaprojektować, żeby można było łatwo, szybko i wygodnie operować danymi w niej zawartymi, oraz zapewnić, żeby projektowana tablica była uniwersalna.

# Jaki jest cel normalizacji

- Najczęściej projektuje się bazę danych składającą się z wielu tablic, które zawierają informacje powiązane ze sobą logicznie, i konieczne jest aby można było uzyskać informacje z wielu tablic nie powielając jednocześnie niepotrzebnie tych samych danych.
- Proces normalizacji, prowadzony w oparciu o analizę relacji, dostarcza cennych informacji o poprawnej strukturze projektowanej bazy danych, jednak w praktyce konieczne jest korzystanie z zasad normalizacji w sposób dobrze przemyślany.

# Jaki jest cel normalizacji

- Uzyskanie większej elastyczności w przechowywaniu danych w bazie danych
- Zapewnienie, że kolumny będą umieszczone w odpowiednich tabelach
- Redukcja nadmiarowości (redundancji) danych
- Poprawienie efektywności programowania
- Niższe koszty zarządzania aplikacją
- Maksymalizacja stabilności struktury danych

# Anomalie

- Głównym powodem, dla którego normalizuje się bazę danych jest uniknięcie anomalii, które mogą wystąpić przy nieprawidłowo skonstruowanej strukturze bazy.



# Anomalie przykład

Założmy, że mamy następującą strukturę bazy książek w bibliotece:

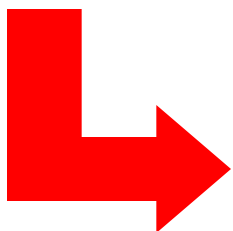
Tytuł książki	Autor	Wypożyczający	Adres wypożyczającego	Data wypożyczenia
---------------	-------	---------------	--------------------------	----------------------

W tej bazie wystąpią następujące anomalie:

Anomalia	Opis
przy <b>aktualizacji</b>	Jeżeli wypożyczający zmienił adres, trzeba przeszukać całą bazę i we wszystkich komórkach, w których występuje, zmienić ten adres
przy <b>usuwaniu</b>	Jeżeli wypożyczający zwróci ostatnią książkę, zostanie utracona informacja na jego temat (adres i inne dane osobowe)
przy <b>wstawianiu</b>	Nowa osoba nie może zapisać się do biblioteki, jeżeli nie wypożyczy książki (a nie musi od razu wypożyczać)
<b>redundacja</b>	Redundacja, czyli powtarzanie tej samej informacji w kilku miejscach w bazie, powoduje niepotrzebne zajmowanie pamięci (wypożyczenie dwóch książek powoduje, że niepotrzebnie adres jest powtarzany dwa razy)

# Pierwsza postać normalna

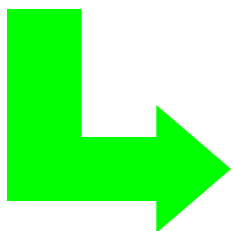
przed



Nr studenta	Opiekun	Pokój opiekuna	Zajęcia 1	Zajęcia 2	Zajęcia 3
1022	Czarnecki	412	101-07	143-01	159-02
4123	Borkowski	216	201-01	211-02	214-01

Tabele powinny mieć tylko dwa wymiary. Ponieważ jeden student może mieć kilka rodzajów zajęć, zajęcia powinny być wymienione w osobnej tabeli. Pola Zajęcia 1, Zajęcia 2 i Zajęcia 3 w powyższych rekordach sygnalizują problemy z projektem.

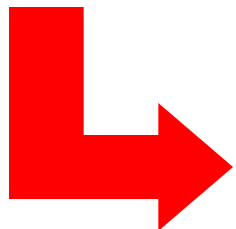
po



Nr studenta	Opiekun	Pokój opiekuna	Nr zajęć
1022	Czarnecki	412	101-07
1022	Czarnecki	412	143-01
1022	Czarnecki	412	159-02
4123	Borkowski	216	201-01
4123	Borkowski	216	211-02
4123	Borkowski	216	214-01

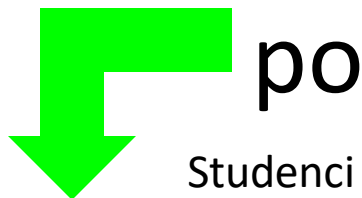
# Druga postać normalna

## przed



Nr studenta	Opiekun	Pokój opiekuna	Nr zajęć
1022	Czarnecki	412	101-07
1022	Czarnecki	412	143-01
1022	Czarnecki	412	159-02
4123	Borkowski	216	201-01
4123	Borkowski	216	211-02
4123	Borkowski	216	214-01

W powyższej tabeli dla każdej wartości Nr studenta występuje wiele wartości Nr zajęć. Wartości Nr zajęć nie są funkcjonalnie zależne od klucza podstawowego Nr studenta, więc ta relacja nie jest w drugiej formie normalnej.

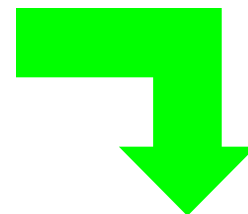


po

Studenci

Nr studenta	Opiekun	Pokój opiekuna
1022	Czarnecki	412
4123	Borkowski	216

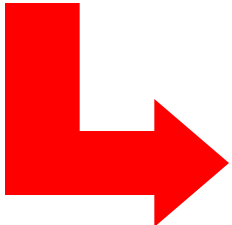
po



Rejestracja

Nr studenta	Nr zajęć
1022	101-07
1022	143-01
1022	159-02
4123	201-01
4123	211-02
4123	214-01

# przed Trzecia postać normalna

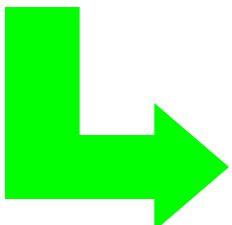


Nr studenta	Opiekun	Pokój opiekuna
1022	Czarnecki	412
4123	Borkowski	216

Nr studenta	Nr zajęć
1022	101-07
1022	143-01
1022	159-02
4123	201-01
4123	211-02
4123	214-01

W ostatniej tabeli wartości Pokój opiekuna są funkcjonalnie zależne od atrybutu Opiekun. Rozwiązaniem jest przeniesienie tego atrybutu z tabeli Studenci do tabeli Wykładowcy, jak pokazano poniżej:

po



Studenci	
Nr studenta	Opiekun
1022	Czarnecki
4123	Borkowski

Wykładowcy		
Nazwisko	Pokój	Wydział
Czarnecki	412	42
Borkowski	216	42

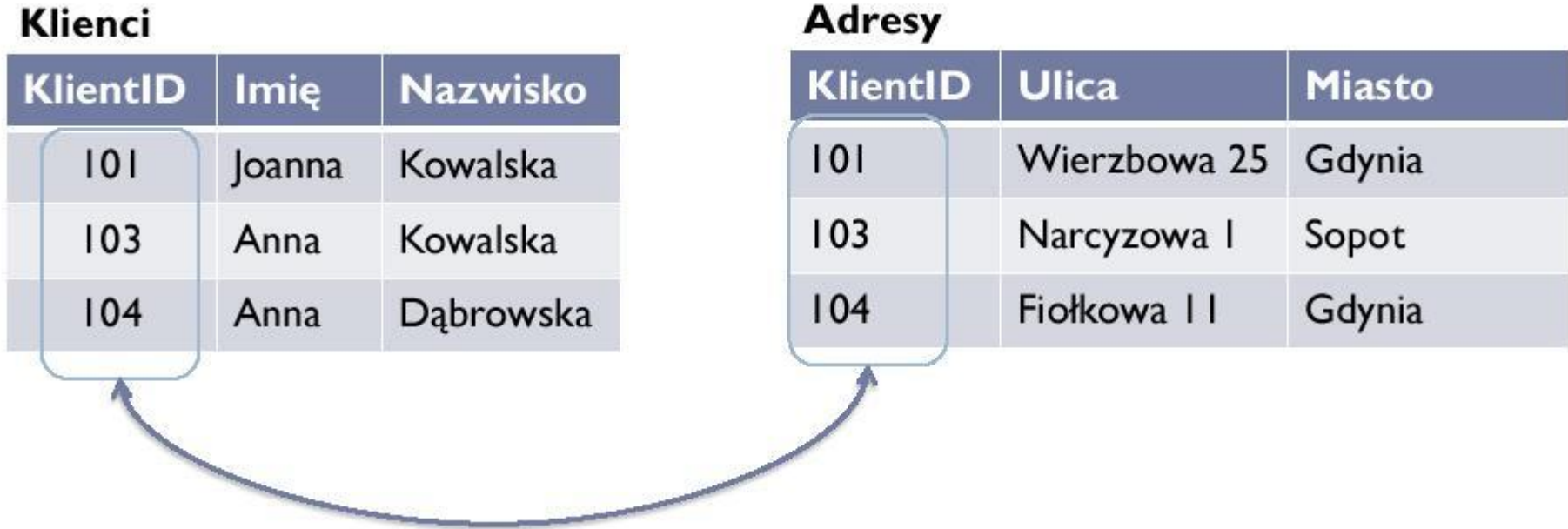
# Model relacyjny

- Podczas przeprowadzania normalizacji tablice (lub encje) nazywa się **relacjami**.
- Podział danych na tabele opiera się na relacjach między obiektami.
- Można wyróżnić trzy możliwe relacje:
  - Jeden do jednego,
  - Jeden do wiele,
  - Wiele do wiele.

# Relacja jeden do jeden

- Relacja **jeden do jednego** zachodzi, gdy jeden element zależy od drugiego.
- W przypadku tej relacji elementy umieszcza się zwykle w jednej tabeli. Można też rozdzielić elementy do osobnych tabel, ale wówczas należy wprowadzić wspólny łącznik (wspólny klucz).
- Na przykład projektując tabelę, w której mają być przechowywane dane personalne osób, umieszczamy w tabeli najczęściej takie kolumny jak: imię, nazwisko, data urodzenia, PESEL.
- Wymienione powyżej dane nie rozdziela się do innych tabel, ponieważ nie ma sensu umieszczać informacji o dacie urodzenia czy PESELu w osobnej tabeli, gdyż imię i nazwisko jednoznacznie przekładają się na datę urodzenia czy PESEL.

# Relacja jeden do jeden



W przypadku tabel **Klienci** i **Adresy** zachodzi relacja jeden do jednego, ponieważ pojedynczy wiersz przechowujący informację o imieniu i nazwisku klienta w tabeli **Klienci** jest związany z tylko jednym wierszem przechowującym informację o adresie tego klienta w tabeli **Adresy**.

Łącznikiem między tymi dwoma tabelami jest kolumna oznaczona jako **KlientID**.

W prezentowanym przykładzie nastąpiło rozdzielenie informacji o imieniu i nazwisku z informacją o adresie klienta, ale równie dobrze można było by utworzyć z dwóch przedstawionych tabel jedną tabelę, w której zawarte były by wyszczególnione informacje osobowe klienta. Niemniej, oba rozwiązania są poprawne.

# Relacja jeden do wiele

- Relacja **jeden do wiele** zachodzi, gdy jeden element z pierwszej tabeli powiązany jest z wieloma innymi elementami z drugiej tabeli.
- W takim przypadku elementy umieszcza się w osobnych tabelach.
- Wiąże się ze sobą za pomocą **pary kluczy: klucza głównego i klucza obcego**.
- Klucz główny znajduje się w tabeli, gdzie element, który jest powiązany z innymi elementami jest unikalny i niepowtarzalny (taki tylko jeden), natomiast klucz obcy znajduje się w tabeli, w której znajduje się wiele elementów.
- Klucz obcy też jest unikalny w obrębie tabeli.
- W bazie ze uczniami taką relacją może być powiązanie osoby ucznia z oceną ucznia - każdy uczeń ma wiele ocen, jednak każda pojedyncza ocena jest przyporządkowana jednemu uczniowi



# Relacja jeden do wielu

**Klienci**

KlientID	Imię	Nazwisko	Adres
101	Joanna	Kowalska	Wierzbowa 25

**Zamowienia**

ZamowienieID	KlientID	Wartosc	Data
302	101	255,2	2006-01-12
304	101	12,25	2006-01-14
305	101	1321,1	2006-01-14

W przypadku tabel **Klienci** i **Zamówienia** zachodzi relacja jeden do wielu, ponieważ pojedynczemu wierszowi przechowującemu informację o danych personalnych klienta w tabeli **Klienci** odpowiada kilka wierszy przechowujących informację o zamówieniach zrealizowanych przez danego klienta i zapisanych w tabeli **Zamówienia**.

Zastosowanie relacji jeden do wiele w przedstawionym przykładzie wynika z faktu, że jeden klient ma przecież prawo zamówić (zakupić) więcej niż jeden produkt.

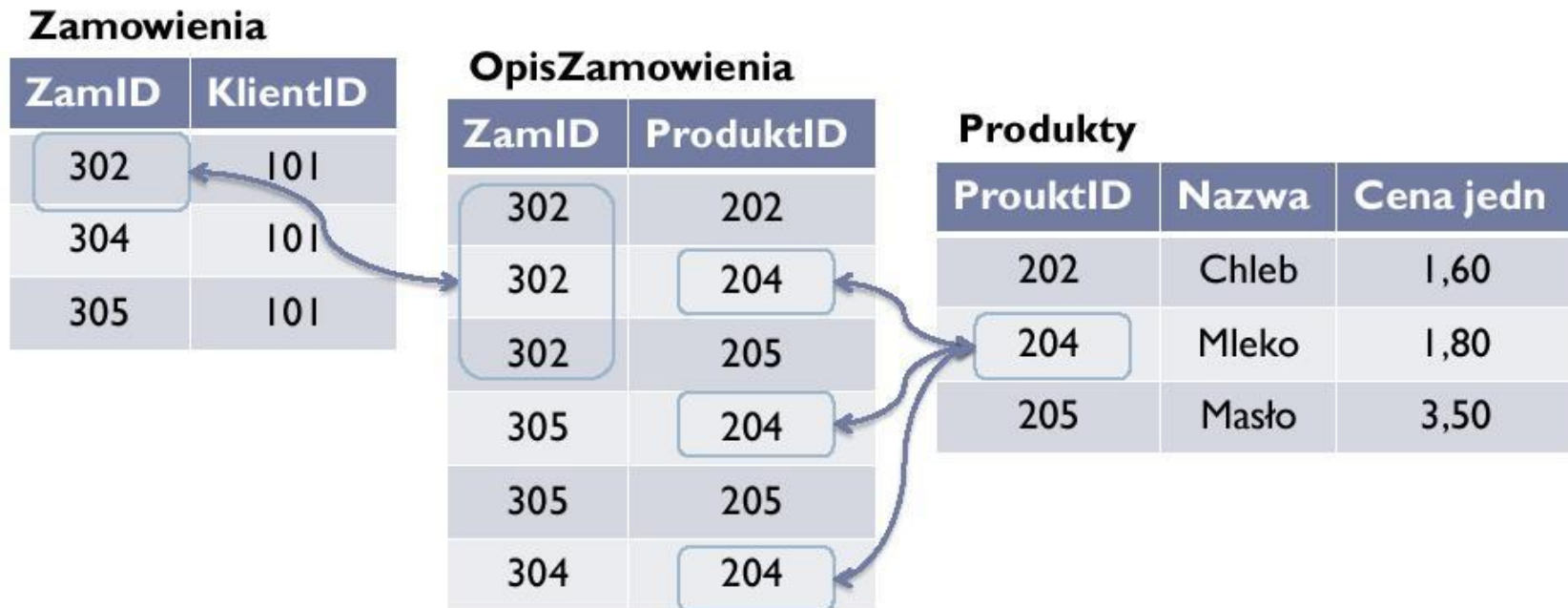
Łącznikiem w tej relacji jest kolumna **KlientID**, która w tabeli **Klienci** stanowi **klucz główny**, natomiast w tabeli **Zamówienia** stanowi **klucz obcy**.

**Kluczem głównym** tabeli **Zamówienia** jest kolumna oznaczona jako **ZamowienieID**.

# Relacja wiele do wiele

- Relacja **wiele do wiele** zachodzi, gdy istnieją dwie grupy elementów, które mogą łączyć się ze sobą w taki sposób, że zarówno dowolny element z pierwszej grupy może łączyć się z wieloma elementami grupy drugiej, jak również dowolny element grupy drugiej może łączyć się z wieloma elementami grupy pierwszej.
- W praktyce, relację taką tworzy się **przez wprowadzenie dodatkowej tabeli pełniącej rolę łącznika między dwoma tabelami**, które zawierają specyfikacje elementów obu grup powiązanych relacją.
- Na przykład w bazie z ocenami uczniów taka relacja może powstać, jeśli wydzielimy obiekt „forma sprawdzania wiedzy” (sprawdzian, odpowiedź ustna, praca kontrolna). W takim przypadku powiązanie uczniów z „formą sprawdzenia wiedzy” jest relacją **wiele do wiele**. Ponieważ wiedza w przypadku każdego ucznia była sprawdzana w różny sposób (odpytywanie, sprawdzian, itp), ale też stosując wybraną formę sprawdzania wiedzy sprawdzano wielu uczniów.

# Relacja wiele do wielu



W przypadku tabel **Zamowienia** i **Produkty** zachodzi relacja wiele do wielu, ponieważ kilku wierszą przechowującym informację o zamówieniach zrealizowanych przez danego klienta może odpowiadać również kilka wierszy przechowujących informację o produktach w tabeli **Produkty**.

Łącznikiem między tymi dwoma tabelami są kolumny **ZamID** i **ProduktID** zawarte w tabeli **OpisZamowienia**.

Kolumny wchodzące w skład tabeli **OpisZamowienia** są jednocześnie **kluczami głównymi** w swoich „macierzystych” tabelach (w **Zamowienia** jest to **ZamID**, w **Produkty** jest to **ProduktID**).

Relacje między tabelą **Zamowienia** i **OpisZamowienia** oraz **OpisZamowienia** i **Produkty** jest

# Przykłady notacji związków

(notacja Martina – wronie łapki)



Związek typu - małżeństwo konwencjonalne



Związek typu - poligamia



Związek typu - poliandria



Związek typu - małżeństwo grupowe

## Instrukcja SELECT:

Instrukcja `SELECT` jest podstawową instrukcją języka SQL, służącą głównie (choć nie tylko) do pobierania danych z tabeli lub tabel na podstawie zadanych warunków. Wynikiem jej wywołania (o ile nie wystąpi błąd) jest zawsze pewna tabela. Składnia tej instrukcji jest dość złożona; należy przy tym pamiętać, że kolejność klauzul (tj. odpowiednich słów kluczowych) jest istotna.

`SELECT` *wyrażenie*

W tej postaci instrukcja `SELECT` zwróci po prostu wartość podanego wyrażenia (zbudowanego z wykorzystaniem stałych i funkcji). Dokładniej mówiąc, zwróci ona tabelę składającą się z jednego wiersza i jednej kolumny, zawierającą wartość tego wyrażenia.

Szczególny przypadek to

```
SELECT * FROM tabela:
```

co spowoduje wypisanie całej tabeli.

**Inna postać z warunkiem:**

```
SELECT wyrażenie1, wyrażenie2, ... FROM  
tabela WHERE warunek
```

Warunek podany po słowie kluczowym `WHERE` ogranicza działanie instrukcji `SELECT` do wierszy spełniających ten warunek.

Powinien on być wyrażeniem logicznym, zbudowanym z wykorzystaniem nazw kolumn tabeli.

## Proste przykłady:

```
mysql> SELECT 2+2;
```

```
+-----+  
| 2+2 |  
+-----+  
|    4 |  
+-----+
```

```
mysql> SELECT SQRT(16);
```

```
+-----+  
| SQRT(16) |  
+-----+  
|          4 |  
+-----+
```

```
mysql> SELECT IF(5>6,1,2);
```

```
+-----+  
| IF(5>6,1,2) |  
+-----+  
|           2 |  
+-----+
```

## Składnia:

jest najczęściej używaną instrukcją SQL i ma następującą postać ogólną:

```
SELECT    [ALL | DISTINCT]  { * | [wyrażenie_kolumnowe  
    [AS nowa_nazwa]] , [...] }  
  
FROM      NazwaTabeli [alias] , [...]   
  
[WHERE     warunek_selekcji_wierszy]  
  
[GROUP BY lista_kolumn] [HAVING warunek_selekcji_grup]  
  
[ORDER BY lista_kolumn];
```



**SELECT** – wskazuje, które kolumny powinny pojawić się w wyniku;

**DISTINCT** – eliminuje powtórzenia po wykonaniu rzutowania na wybrane kolumny;

**FROM** – określa tabelę (lub tabele), z których będziemy korzystać;

**WHERE** – pozwala wybrać wiersze spełniające zadany warunek selekcji wierszy;

**GROUP BY** – tworzy grupy wierszy o tej samej wartości wskazanej kolumny;

**HAVING** – pozwala wybrać grupy ze względu na podany warunek selekcji grup;

**ORDER BY** – określa uporządkowanie wyniku.

# Wyszukiwanie wszystkich wierszy:

1. wszystkie kolumny i wszystkie wiersze:

```
SELECT *  
FROM Klient;
```

2. wybrane kolumny i wszystkie wiersze:

```
SELECT pracownikNr, imie, nazwisko, pensja  
FROM Personel;
```

3. wybrane kolumny i wszystkie wiersze uporządkowane malejąco:

```
SELECT pracownikNr, imie, nazwisko, pensja  
FROM Personel  
ORDER BY pensja DESC;
```

#### 4. Porządkowanie według wielu kolumn:

```
SELECT nieruchomoscNr, typ, pokoje, czynsz  
FROM Nieruchomosc  
ORDER BY typ, czynsz DESC;
```

#### 5. eliminacja powtórzeń – podaj numery nieruchomości odwiedzone przez klientów:

```
SELECT DISTINCT nieruchomoscNr  
FROM Wizyta;
```

#### 6. wyliczanie pensji rocznej:

```
SELECT pracownikNr, imie, nazwisko,  
pensja*12 AS pensjaRoczna  
FROM Personel;
```

## Wybieranie wierszy:

### 7. warunek selekcji: porównanie pensja do stałej

```
SELECT pracownikNr, imie, nazwisko,  
stanowisko, pensja  
FROM Personel  
WHERE pensja>1000;
```

### 8. złożony warunek selekcji: porównanie – podaj adresy biur w Łomży lub w Białymstoku

```
SELECT biuroNr, ulica, miasto, kod  
FROM Biuro  
WHERE miasto=„Łomża" OR miasto="Białystok";
```

## 9. Selekcja pensji z zakresu:

```
SELECT pracownikNr, imie, nazwisko,  
stanowisko, pensja
```

```
FROM Personel
```

```
WHERE pensja BETWEEN 2000 AND 3000;
```

lub

```
WHERE pensja>=2000 AND pensja<=3000;
```

## 10. Selekcja czynszu z zakresu oraz miast

```
SELECT nieruchomoscNr, miasto, typ,  
czynsz
```

```
FROM Nieruchomosc
```

```
WHERE miasto='Białystok' AND czynsz  
Between 350 AND 450 OR miasto='Augustów' ;
```

## 11. warunek selekcji – dopasowanie do wzorca

```
SELECT wlascicielNr, imie, nazwisko,  
adres, telefon
```

```
FROM Wlasciciel
```

```
WHERE adres LIKE '%Białystok%';
```

lub

```
WHERE adres LIKE "*Białystok*";
```

**12. policz z ograniczeniem – w ilu nieruchomościach miesięczny czynsz jest wyższy od 350:**

```
SELECT Count (*) AS liczba
```

```
FROM Nieruchomosc
```

```
WHERE czynsz>350;
```

**13. policz i sumuj – oblicz ilu jest dyrektorów i jaka jest ich sumaryczna pensja:**

```
SELECT COUNT (pracownikNr) AS liczba,
```

```
SUM (pensja) AS suma
```

```
FROM Personel
```

```
WHERE stanowisko='dyrektor' ;
```

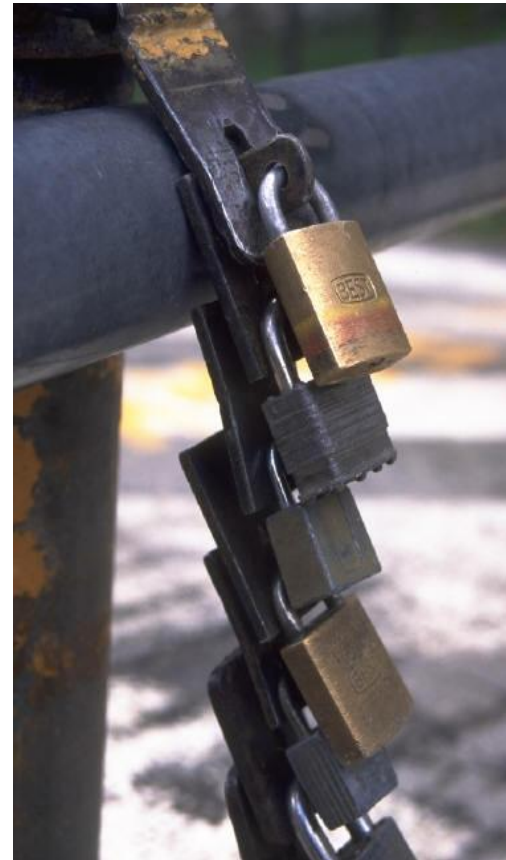


# Programy i ich formaty

- Prawie każda aplikacja wprowadza swój wewnętrzny format.
- Nowe wersje tej samej aplikacji wprowadzają zmiany do używanego formatu:
  - wsteczna kompatybilność,
  - brak możliwości zapisu do formatu poprzednich wersji.
- Aplikacje dostarczają konwerterów:
  - tylko do najpopularniejszych formatów,
  - możliwość utraty danych podczas konwersji.

# Standardy

- Nie istnieją uznane standardy.
- Istnieją substandardy w różnych dziedzinach:
  - dokumenty biurowe: Microsoft Word,
  - teksty naukowe: Postscript, TeX,
  - Internet: HTML, GIF, JPG,
  - elektroniczna wymiana danych: EDIFACT.
- Standard musi być:
  - własnością publiczną,
  - otwarty i jawny,
  - niezależny od konkretnego producenta oprogramowania.



# Potrzeba struktury

- Masa informacji cyfrowej powoduje potrzebę struktury:
  - jeden format dokumentu nie wystarczy dla 5 miliardów ludzi,
  - ale nie możemy operować milionami niekompatybilnych formatów.

# Rozwój języków uogólnionego znakowania tekstu

- 1969: GML – Generalized Markup Language (IBM; Goldfarb, Mosher, Laurie).
- 1986: SGML – Standard Generalized Markup Language, ISO 8879:1986.
- 1991: powstaje World Wide Web.
- 1994: HTML 2.0 zdefiniowany jako zastosowanie SGML-a.
- 1998: XML – Extensible Markup Language, World Wide Web Consortium.



# World Wide Web Consortium (W3C)

- Kuźnia standardów internetowych, np.:
  - HTML – Hyper Text Markup Language,
  - HTTP – Hyper Text Transfer Protocol,
  - CSS – Cascading StyleSheets,
  - ...
- XML – Extensible Markup Language:
  - najważniejsza rekomendacja ostatnich lat,
  - twórcy: Tim Bray (Netscape), Jean Paoli (Microsoft), C.M. Sperberg-McQueen (University of Illinois).
- Obecne dominują prace nad standardami związanymi z XML-em.

# Idea SGML/XML (1)

**Oddzielenie znaczenia tekstu  
od sposobu prezentacji**

<OSOBA MÓWIĄCA>Hamlet</OSOBA MÓWIĄCA>  
<WYPOWIEDŹ>Być albo nie być.  
Oto jest pytanie.</WYPOWIEDŹ>



# Idea SGML/XML (2)

**Stworzenie najodpowiedniejszego modelu  
dla naszych własnych dokumentów.**

<OSOBA MÓWIĄCA>Hamlet</OSOBA MÓWIĄCA>  
<WYPOWIEDŹ> <NUDA> Być albo nie być.  
Oto jest pytanie.</NUDA> </WYPOWIEDŹ>

# Najodpowiedniejszy model

- Przykłady:
  - encyklopedia: <nazwisko>, <imie>, <ur>, <zm>, <wymowa>, <etymologia>, <liczba-mieszk>
  - prawo: <promulgator>, <rocznik>, <poz>, <art> <sąd>, <sygn-wyroku>, <teza>
  - dokument techniczny: <part-number>, <function-name>
  - patenty: <wynalazca>, <nr-zgłoszenia>
  - ubezpieczenia: <data-polisy>, <wartość-polisy>



# Język – metajęzyk

- Stan wyjściowy:
  - Wieża Babel (brak wspólnego języka),
  - czy w ogóle możliwy jeden wspólny język?
- Wspólny metajęzyk:
  - znana gramatyka,
  - jednolita metodologia,
  - takie same narzędzia.
- Dowolnie wiele języków specyficznych dla zastosowań.



# Co to jest XML?

- XML to nie język programowania.
- XML to sposób zapamiętywania danych wraz z ich strukturą w dokumencie tekstowym:
  - otwarty,
  - elastyczny,
  - bezpłatny,
  - niezależny od platformy sprzętowej.
- XML to rama składniowa do tworzenia języków specyficznych dla zastosowań.
- Użycie XML-a nie zwalnia od myślenia (analizy, projektowania, ...)

# Jak wygląda XML?

```
<?xml version="1.0"?>
```

```
<zeznanie-sprawcy nr="1313/2001">
```

```
<autor>st. asp. Jan Łapówka</autor>
```

```
<miejsce>Dołowice Górne</miejsce>
```

```
<treść>Wypadek dnia
```

```
<data>13.10.2001r</data>
```

```
o godzinie <godzina>13:13</godzina>
```

```
(<dzien-tygodnia>piątek
```

```
</dzien-tygodnia>) miał miejsce nie
```

```
z mojej winy. <poszkodowany>Alojzy
```

```
M.</poszkodowany> nie miał żadnego
```

```
pomysłu w którą stronę uciekać, więc
```

```
go przejechałem.</treść>
```

```
</zeznanie-sprawcy>
```

Deklaracja XML

Element główny

Atrybut

Element

Znacznik początkowy

Znacznik końcowy

Zawartość tekstowa

# HTML ↔ XML

- Znaczenie elementów i ich atrybutów z góry określone.
- Interpretację elementów określa standard, a w praktyce przeglądarki internetowe.
- To, co jest poprawne również określają przeglądarki internetowe.
- Znaczenie elementów i ich atrybutów określa użytkownik lub aplikacja.
- `<p>` może w jednym dokumencie oznaczać **paragraf**, w drugim **pomoc**, a w trzecim **pismo odręczne**.
- Poprawność XML-a jest ściśle określona przez specyfikację.

# SGML ↔ XML

- Filozofia: jeden duży system zarządzania treścią.
- Konieczność definiowania struktury.
- Skomplikowana składnia, wiele opcji.
- Trudność tworzenia parserów.
- Bardzo drogie narzędzia.
- Filozofia: wiele małych komunikujących się ze sobą modułów.
- Opcjonalne definiowanie struktury.
- Uproszczona składnia.
- Łatwość tworzenia parserów.
- Darmowe narzędzia.

# Klasy zastosowań XML-a

## Zarządzanie dokumentami, treścią, wiedzą:

- Pierwotne zastosowanie SGML-a.
- Dokumenty tworzone przez człowieka i przeznaczone dla człowieka.
- Długi czas życia dokumentów.
- Typowy model mieszany zawartości.

## Elektroniczna wymiana danych, integracja aplikacji:

- Nowa klasa zastosowań XML-a.
- Dokumenty tworzone oraz przetwarzane automatycznie
- Dokumenty tworzone tylko na czas komunikacji.
- Konieczność dokładnego kontrolowania struktury i zawartości.

# Dwie twarze XML-a

## Dokument tekstowy:

```
<zeznanie-sprawcy>
Wypadek dnia <data>
13.01.2001 r.</data>
o godzinie <godzina>13.13
</godzina> (<dzien-
tygodnia>piątek
</dzien-tygodnia>) miał
miejsce nie z mojej winy.
<poszkodowany>Alojzy
M.</poszkodowany> nie miał
żadnego pomysłu w którą
stronę uciekać, więc go
przejechałem.
</zeznanie-sprawcy>
```

## Baza danych:

```
<zamowienie>
  <pozycja>
    <nazwa>Papier</nazwa>
    <jednostka>ryza
    </jednostka>
    <ilosc>3</ilosc>
  </pozycja>
  <zamawiajacy id="123456">
    <imie>Szymon</imie>
    <nazwisko>Zioło
    </nazwisko>
    <firma>ABG Ster-Projekt
    </firma>
  </zamawiajacy>
</zamowienie>
```

# JSON

JSON to sposób na przechowywanie i przekazywanie danych za pomocą zwykłego tekstu. JSON to nic więcej niż łańcuch znaków. Ma on oczywiście pewne zasady, składnię.

```
1 {  
2   "dogs": [{  
3     "id": 0,  
4     "name": "maja",  
5     "race": "pies"  
6   }, {  
7     "id": 1,  
8     "name": "milus",  
9     "race": "pies"  
10  }],  
11  "cats": [{  
12    "id": 0,  
13    "name": "puszek",  
14    "race": "kot"  
15  }, {  
16    "id": 1,  
17    "name": "greebo",  
18    "race": "kot"  
19  }]  
20 }
```

JSON korzysta ze składni JavaScriptu. JSON to skrót od JavaScript Object Notation. Czyli JavaScriptowy zapis obiektowy.



# Do czego używamy JSON

- JSON, podobnie jak inne struktury służące do przechowywania danych takich jak np. XML, ma szerokie zastosowanie.
- Najczęściej jednak jest wykorzystywany do przekazywania i odbierania danych z serwera przez aplikacje na stronie internetowej.
- Na przykład kiedy użytkownik loguje się na do swojego konta w grze przeglądarkowej. Poprzez AJAX przeglądarka wysyła do serwera ID i hasło użytkownika. Skrypt po stronie serwera, napisany na przykład w PHP, sprawdza czy hasło jest poprawne. Jeżeli tak jest, w odpowiedzi skrypt wysyła do przeglądarki w formie JSON wszystkie dane użytkownika. Na podstawie tak otrzymanych danych, w przeglądarce wyświetlane są wszystkie informacje na temat konta użytkownika.

# Jak zamieniać JSONowe łańcuchy znaków na obiekty JavaScript?

Jako przykład wykorzystam konto gracza, w grze przeglądarkowej. Założmy, że mamy obiekt JSON, zawierający wszystkie potrzebne dane. Oto struktura:

```
1 {  
2   "name":["Jan","Kowalski"],  
3   "email" : "jan@Kowalski.pl",  
4   "settings" : {  
5     "sound" : "off",  
6     "fullScreen": true,  
7     "newsDisplayed" : 5  
8   },  
9   "Characters" :[{ "Conen":{  
10     "level":10,  
11     "items":["topor"],  
12     "hungry": true  
13   }  
14   },{  
15     "Druzst":{  
16       "level":4,  
17       "items":["sejmitar","sejmitar"],  
18       "hungry": true  
19     }  
20   },{  
21     "Garelt":{  
22       "level":10,  
23       "items":["miecz","miecz"],  
24       "hungry": true  
25     }  
26   }  
27 }]
```

# Jak zamieniać JSONowe łańcuchy znaków na obiekty JavaScript?

Zmiana łańcucha znaków JSON na obiekt JavaScriptowy. Zamiast obiektu, dostajemy taki łańcuch znaków:

```
1 '{"name":["Jan","Kowalski"],"email":"jan@Kowalski.pl","settings":{"sound":"off","fullScreen":true,"newsDisplayed":5},"Characters":[{"Conen":{"level":10,"items":["topor"],"hungry":true}},{"Druztt":{"level":4,"items":["sejmitar","sejmitar"],"hungry":true}},{"Garelt":{"level":10,"items":["miecz","miecz"],"hungry":true}}]}'
```

JavaScript przychodzi z pomocą. A dokładniej, wbudowany obiekt *JSON* i jego metoda *parse*. Jako argument metoda ta przyjmuje łańcuch znaków, zawierający poprawny JSON, a zwraca obiekt JavaScriptowy.

```
1 var userDataString = '{"name":["Jan","Kowalski"],"email":"jan@Kowalski.pl","settings":{"sound":"off","fullScreen":true,"newsDisplayed":5},"Characters":[{"Conen":{"level":10,"items":["topor"],"hungry":true}},{"Druztt":{"level":4,"items":["sejmitar","sejmitar"],"hungry":true}},{"Garelt":{"level":10,"items":["miecz","miecz"],"hungry":true}}]}'
2
3 var userData = JSON.parse(userDataString);
4 console.log(userData.name[0]+ " " +userData.name[1]); //Jan Kowalski
5 console.log(userData.email); //jan@Kowalski.pl
6 console.log(userData.settings.fullScreen); //true
```