Teoretyczne podstawy informatyki

Wykład 8a:
Relacyjny model danych

http://hibiscus.if.uj.edu.pl/~erichter/Dydaktyka2009/TPI-2009

Relacyjny model danych

- Jednym z najważniejszych zastosowań komputerów jest przechowywanie i przetwarzanie informacji.
- Relacyjny model danych opiera się na idei organizowania danych w zbiory dwuwymiarowych tabel nazywanych „relacjami”.
- Jest to uogólnienie modelu danych opartego na zbiorach, rozszerzającego relacje binarne do relacji o dowolnej krotności.
- Relacyjny model danych został pierwotnie opracowany z myślą o bazach danych oraz o systemach zarządzania bazami danych.
- Obecne zastosowania wykraczają poza ten pierwotny zakres.

Relacje - przypomnienie

- Chociaż założyliśmy, że w ogólności elementy należące do zbiorów są niepodzielne, w praktyce często korzystnym rozwiązaniem jest przypisanie elementom pewnych struktur.
- Ważną strukturą dla elementów jest **lista o stałej długości** zwana **krotką**. Każdy element takiej listy nazywamy składową **krotki**.
- Zbiór elementów, z których każdy jest krotką o takiej samej liczności - powiedzmy **k**- nazywamy **relacją**. Licznością takiej relacji jest **k**. Jeśli **liczność wynosi 2** mówimy o **krotce lub relacji binarnej**.

Iloczyn kartezjański $A \times B$

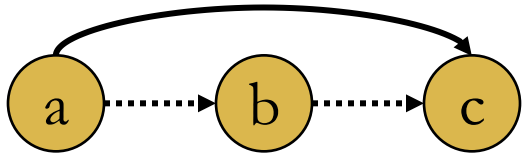
- Jest to zbiór par, z których pierwszy element pochodzi ze zbioru **A**, drugi ze zbioru **B**, czyli

  $$A \times B = \{(a,b) : a \in A \text{ oraz } b \in B\}$$

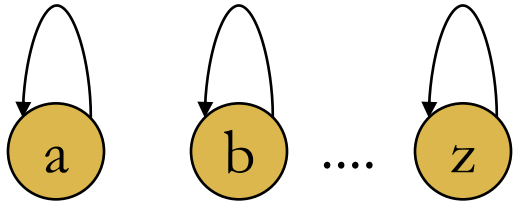
- Iloczyn kartezjański nie ma własności przemienności, $A \times B \neq B \times A$ (dla $A \neq B$)
- **K**-elementowy iloczyn kartezjański $A_1 \times A_2 \times A_3 \ldots \times A_k$ to zbiór **k-tupli** $(a_1,a_2,\ldots,a_n)$

Specyficzne własności relacji binarnych

- **Przechodniość**
  - Niech **R** będzie relacją binarną na dziedzinie **D**.
  - Mówimy, ze **relacja jest przechodnia** jeśli zawsze gdy prawdziwe jest zarówno **aRb** i **bRc**, prawdziwe jest także **aRc**.
    Np. relacja **>**

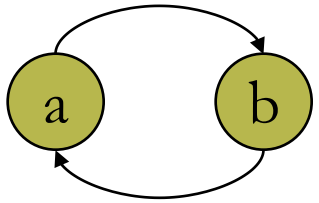


- **Zwrotność**
  - Niech **R** będzie relacją binarną na dziedzinie **D**.
  - Mówimy, ze **relacja jest zwrotną** jeśli dla każdego elementu **a** należącego do dziedziny, relacja zawiera parę **aRa**. Dla tych samych elementów dziedziny mogą też istnieć inne pary **aRb**.
    Np. relacja **≥**

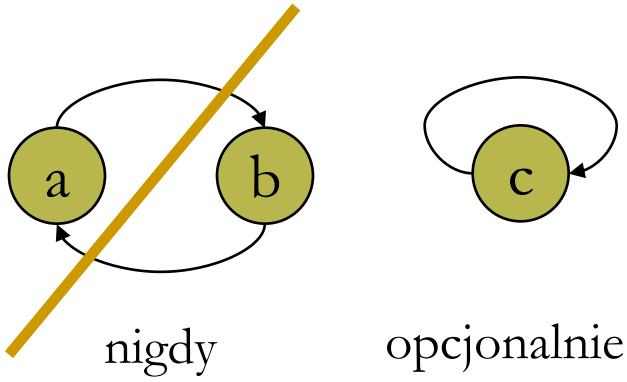


Specyficzne własności relacji binarnych

- **Symetria**
  - Niech **R** będzie relacją binarną na dziedzinie **D**.
  - Mówimy, ze **relacja jest symetryczna** jeśli jest odwrotnością samej siebie tzn. zarówno **aRb** i **bRa**,
    Np. relacja **≠**



- **Antysymetria**
  - Niech **R** będzie relacją binarną na dziedzinie **D**.
  - Mówimy, że **relacja jest antysymetryczna** jeśli **aRb** i **bRa** są jednocześnie prawdziwe **tylko** gdy **a=b**.
  - Nie jest konieczne, by prawdziwe było **aRa** dla każdej wartości **a** należącej do dziedziny relacji antysymetrycznej.
    Np. relacja **≥**, **>**



Specyficzne własności relacji binarnych

- **Relacja porządku częściowego i całkowitego**
  - Relacja porządku częściowego jest to relacja binarna spełniająca własność **przechodniości** i **antysymetrii**.
  - Mówimy że jest to **relacja porządku całkowitego** jeśli poza przechodniością i antysymetrią spełnia także warunek, że wszystkie pary elementów należących do jej dziedziny są porównywalne.
  - Oznacza to, że jeśli **R** jest relacją porządku całkowitego oraz jeśli **a** i **b** są dowolnymi elementami tej dziedziny, to albo **aRb**, albo **bRa** jest prawdziwe (mówimy wtedy że relacja jest **spójna**).
  - Należy zauważyć że każdy porządek całkowity jest **zwrotny**, ponieważ możemy przyjąć **a** i **b** będące tym samym elementem – wymaganie porównywalności oznacza że **aRa**.
- **Relacja równoważności**
  - Relacja równoważności to relacją binarną, która jest **zwrotna, symetryczna i przechodnia**.
  - Dzieli ona swoją dziedzinę na **klasy równoważności**.

Relacyjny model danych

- Relacyjny model danych wykorzystuje pojęcie relacji (ang. **relation**) które jest bardzo mocno związane z przedstawioną wcześniej definicją z teorii zbiorów, jednak różni się w kilku szczegółach:
  - W relacyjnym modelu danych informacja jest przechowywana w tabelach.
  - Kolumny tabeli maja nadane konkretne nazwy i są atrybutami relacji.
  - Każdy wiersz w tabeli jest nazywany krotką i reprezentuje jeden podstawowy fakt.
  - Pojęcie relacji odwołuje się do każdej krotki.

Atrybuty relacji: Zajęcia, StudentID, Ocena

Krotki to:
(CS101, 12345, 5.0)
(CS101, 67890, 4.0)
…

zajęcia	student ID	ocena
CS101	12345	5.0
CS101	67890	4.0
EE200	12345	3.0
EE200	22222	4.5
CS101	33333	2.0
PH100	67890	3.5

Relacyjny model danych

- Tabele możemy rozpatrywać w dwóch aspektach:
  - jako zbiór nazw kolumn
  - jako zbiór wierszy zawierających informacje.
- Pojecie „relacji” odwołuje się do zbioru wierszy.
- Każdy wiersz reprezentuje jedną „krotkę” należąca do relacji, ich uporządkowanie nie ma znaczenia.
- Żadne dwa wiersze nie mogą mieć tych samych wartości we wszystkich kolumnach.
- Zbiór nazw kolumn (atrybutów) nazywamy schematem (ang. scheme) relacji.
- Kolejność atrybutów w schemacie relacji nie ma znaczenia, musimy jednak znać powiązania pomiędzy atrybutami i kolumnami w tabeli.

zajęcia	student ID	ocena
CS101	12345	5.0
CS101	67890	4.0
EE200	12345	3.0
EE200	22222	4.5
CS101	33333	2.0
PH100	67890	3.5

Reprezentowanie relacji

- Podobnie jak w przypadku zbiorów istnieje wiele różnych sposobów reprezentowania relacji za pomocą struktur danych.
- **Tabela** postrzegana jako zbiór wierszy **powinna być zbiorem struktur** zawierających pola odpowiadające nazwom kolumn.

```
struct ZSO {
        char Zajecia[5];
        int StudentID;
        char Ocena[3]; }
```

- Sama **tabela** może być reprezentowana za pomocą:
  - **tablicy struktur tego typu**
  - **listy jednokierunkowej złożonej z takich struktur.**
- Możemy identyfikować jeden lub więcej atrybutów jako „dziedzinę" relacji i traktować pozostałe atrybuty jako przeciwdziedzinę.

zajęcia	student ID	ocena
CS101	12345	5.0
CS101	67890	4.0
EE200	12345	3.0
EE200	22222	4.5
CS101	33333	2.0
PH100	67890	3.5

Reprezentowanie relacji

- Zbiór relacji nazywamy **bazą danych**.
- Jedną z decyzji którą należy podjąć przy projektowaniu bazy danych to sposób w jaki przechowywane informacje powinny być rozłożone pomiędzy tabele.
- Najskuteczniejsze operacje na bazie danych polegają na wykorzystaniu wielu relacji do reprezentowania powiązanych ze sobą i wzajemnie skoordynowanych typów danych.
- Wykorzystując właściwe struktury danych możemy efektywnie przechodzić z jednej relacji do drugiej i pozyskiwać w ten sposób informacje z bazy danych której nie moglibyśmy otrzymać z pojedynczej relacji.
- **Zbiór schematów** dla różnych relacji w jednej bazie danych nazywamy schematem bazy danych.
  - **schemat bazy danych** - określa sposób organizowania informacji,
  - **zbiór krotek w każdej relacji** - stanowi właściwe informacje które są przechowywane.

Schemat bazy danych

Zajęcia	Dzień	Godzina
CS101	Pn	9.00
CS101	S	9.00
EE200	Pt	8.30
EE200	W	13.00
CS101	Pt	9.00
PH100	C	8.15

Zajęcia	Wymagania
CS101	CS100
EE200	EE005
EE200	CS100
CS120	CS101
CS121	CS120
CS205	CS101
CS206	CS121
CS206	CS205

Zajęcia	Student ID	Ocena
CS101	12345	5.0
CS101	67890	4.0
EE200	12345	3.0
EE200	22222	4.5
CS101	33333	2.0
PH100	67890	3.5

Zajęcia	Klasa
CS101	Aula
EE200	Hala
PH100	Laborat

Zapytania na bazie danych

- **Operacja insert(t,R)**
  - **Dodajemy krotkę t do relacji R ,** jeśli relacja **R** nie zawiera jeszcze takiej krotki.
  - Operacja działa w podobny sposób jak operacja insert dla słowników i relacji binarnych.
- **Operacja delete(X,R)**
  - W tym przypadku **X** jest specyfikacją kilku krotek.
  - Składa się z elementów, po jednym dla każdego z atrybutów relacji R; każdy element (składowa) może być
    - wartością
    - symbolem *****, co oznacza ze dozwolona jest dowolna wartość.
  - Efektem wykonania tej operacji jest usuniecie wszystkich krotek zgodnych ze specyfikacja X.
    Np. delete((„CS101”,*,*),Zajęcia-StudentID-Ocena)
- **Operacja lookup(X,R)**
  - Wynikiem tej operacji jest zbiór krotek z relacji **R**, które są zgodne ze specyfikacją **X**.

Klucze

- Wiele relacji w bazie danych możemy traktować jak funkcję odwzorowujące jeden zbiór atrybutów na pozostałe atrybuty.
  - Przykładowo, relacje

    Zajęcia – StudentID – Ocena

    możemy traktować jak **funkcję**, której dziedziną jest zbiór par **Zajęcia-StudentID**, a przeciwdziedzina wartość atrybutu **Ocena**.
- Ponieważ funkcje są prostszymi strukturami danych niż relacje, pomocna może być znajomość zbioru atrybutów, które mogą tworzyć dziedzinę funkcji. Taki zbiór atrybutów nazywamy **kluczem**.
- **Klucz relacji**
  - Jest to zbiór złożony z jednego lub większej liczby takich atrybutów, że relacja nigdy nie będzie zawierała dwóch krotek, których wartości będą takie same we wszystkich kolumnach należących do klucza.

Główne struktury przechowywania danych w relacjach

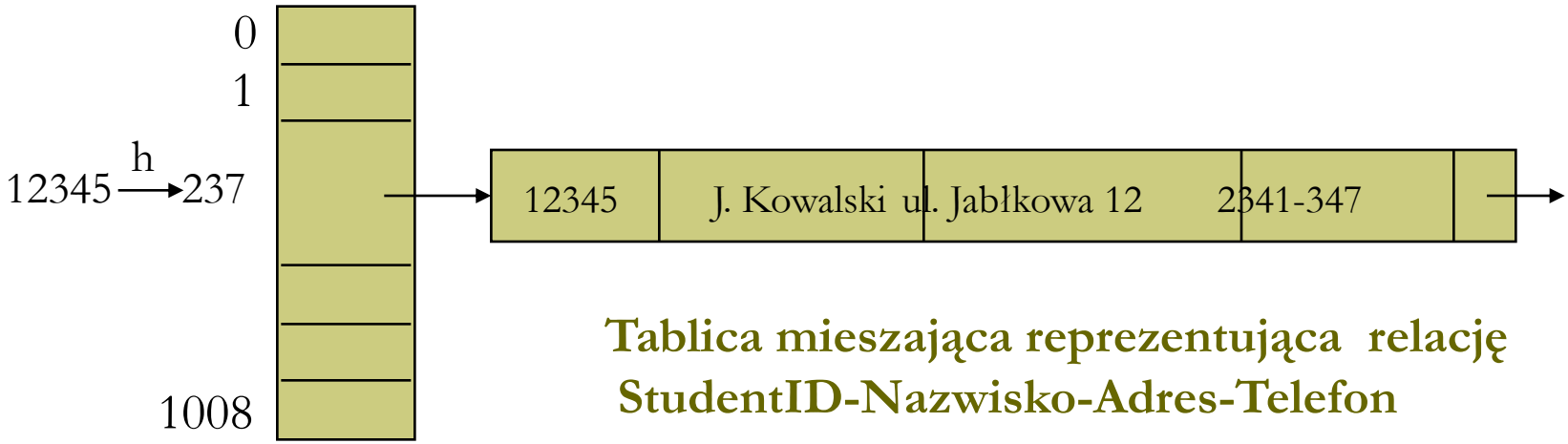
1. Drzewo przeszukiwania binarnego z relacją „mniejszy od” na wartościach dziedziny, która wyznacza pozycje krotek w drzewie. Struktura może znacznie ułatwić wykonywanie operacji, w których daną jest wartość z dziedziny.
2. Tablica wykorzystywana jako wektor własny z wartościami z dziedziny pełniącymi funkcję indeksu tablicy.
3. Tablica mieszająca, w której mieszamy wartości z dziedziny w celu wyznaczenia właściwej komórki.

- Wybraną strukturę nazywamy **strukturą indeksu głównego** (ang. primary index structure) relacji.
  - **Główny** bo lokalizacja komórek jest wyznaczana przez tę strukturę.
  - **Index** jest strukturą danych ułatwiającą znajdowanie komórek dla danej wartości jednej lub kilku składowych szukanej komórki.

Struktura indeksu głównego

- Kluczem jest atrybut StudentID, będzie on dziedziną.
- Musimy wybrać funkcję mieszającą, np. h(x) = x%1009.
- Tablica złożona z 1009 nagłówków zwiera listę jednokierunkową struktur.

```
typedef struct TUPLE * TUPLELIST;
struct TUPLE {
     int StudentID;
     char Nazwisko[30];
     char Adres[60];
     char Telefon[8];
     TUPLELIST next;
};
typedef TUPLELIST HASHTABLE[1009];
```

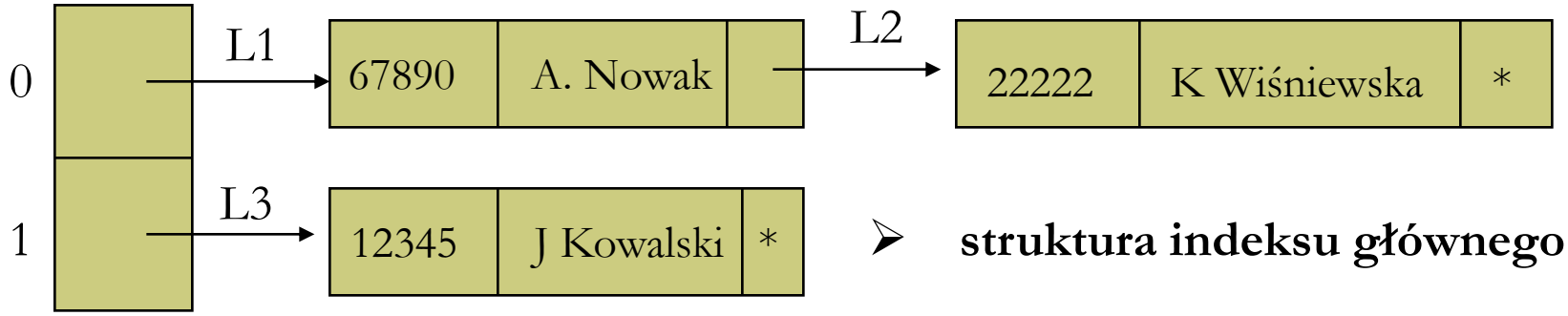


Tablica mieszająca reprezentująca relację StudentID-Nazwisko-Adres-Telefon

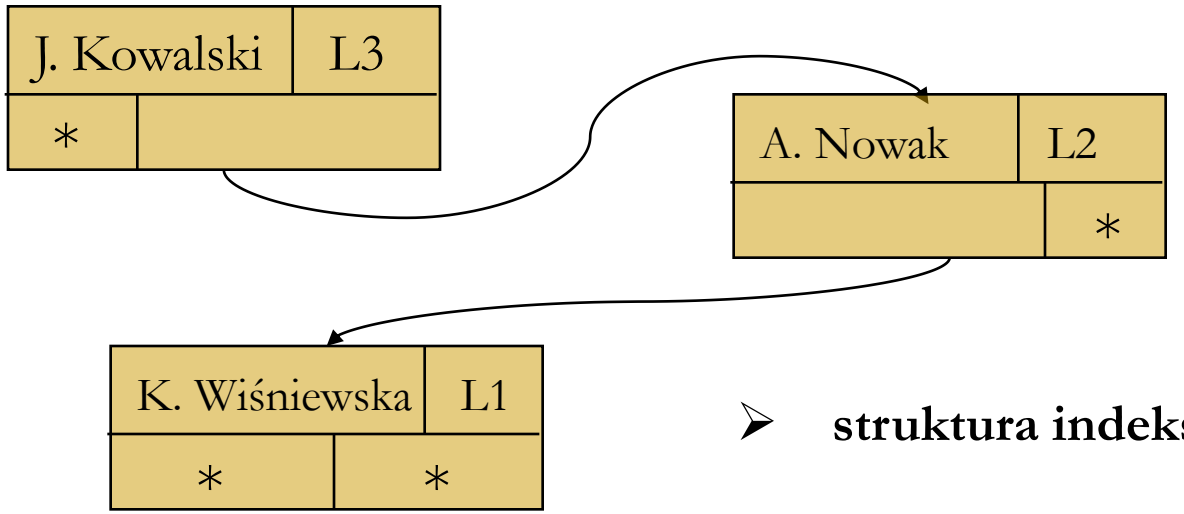
Struktura indeksu drugorzędnego

- Nie służy do pozycjonowania krotek wewnątrz całej struktury tylko do szybkiego znajdowania lokalizacji krotki której jedno z pól ma wartość zgodną z poszukiwaną.
- **Indeksem drugorzędnym jest relacja binarna.**
  - Indeks drugorzędny na atrybucie **A** relacji **R** jest zbiorem par **(n, p)**, gdzie:
    - **n** jest wartością atrybutu **A**
    - **p** jest wskaźnikiem do jednej z krotek ze struktury indeksu głównego dla relacji **R**, w której składowa **A** ma wartość **n**.

Struktura indeksu głównego i drugorzędnego

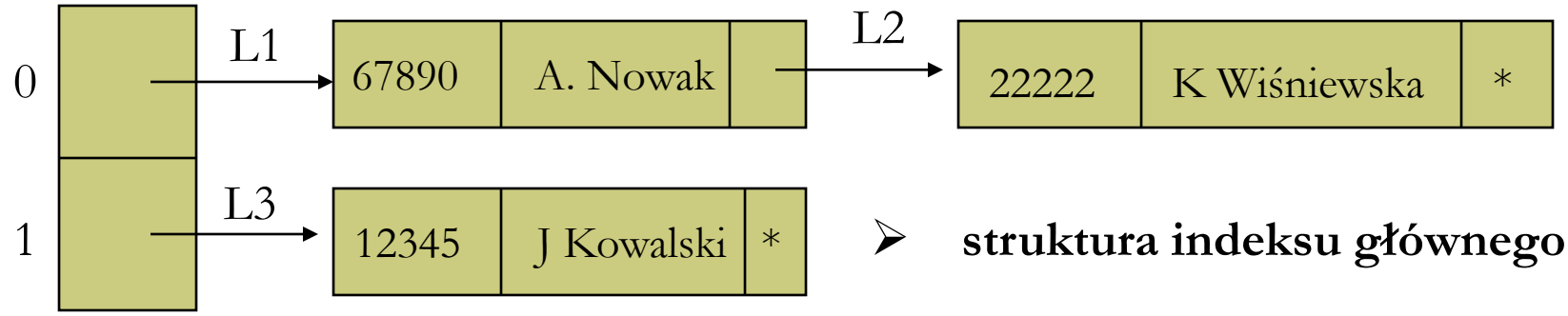


➢ **struktura indeksu głównego**



➢ **struktura indeksu drugorzędnego**

Struktura indeksu głównego



➢ **struktura indeksu głównego**

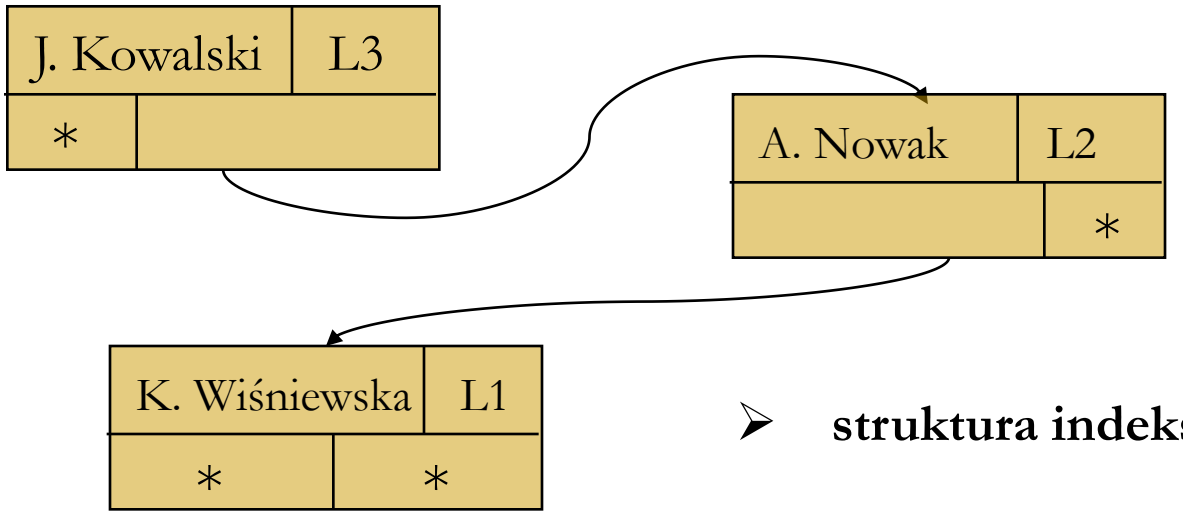
```
typedef struct KROTKA * KROTKALIST;
struct KROTKA {
    int StudentID;
    char Nazwisko[30];
    char Adres[60];
    char Telefon[8];
    KROTKALIST next;
};
typedef KROTKALIST HASHTABLE[2];
```

- **Tablica mieszająca** o **atrybucie** StudentID, pełniąca funkcje **indeksu głównego**.
- Krotki z informacja dotyczącą studenta przechowywane w formie struktur **KROTKA** w liście jednokierunkowej zajmującej pojedyncza komórkę tablicy mieszającej.

Struktura indeksu drugorzędnego

- **NODE jest węzłem** drzewa binarnego z dwoma polami, **Nazwisko** i **toKrotka**, czyli wartość elementu nazwisko i wskaźnik do krotki gdzie jest przechowywana inna informacja dotycząca tego studenta.
- Pozostałe dwa pola to **wskaźniki do lewego** i **prawego dziecka węzła**.

```
typedef struct NODE * TREE;
struct NODE {
    char Nazwisko[30];
    KROTKALIST toKrotka;
    TREE leftChild;
    TREE rightChild;
};
```



➢ **struktura indeksu drugorzędnego**

Analizowanie struktury indeksu drugorzędnego

- Jeżeli dla danej relacji istnieje jeden lub więcej indeksów drugorzędnych, operacje wstawiania i usuwania krotek stają się nieco trudniejsze.
  - **Wstawianie:**
    Jeśli wstawiamy nową krotkę z wartością **n** atrybutu **A**, musimy utworzyć parę **(n, p)**, gdzie **p** wskazuje na nowy element w strukturze indeksu głównego.
    Następnie, musimy wstawić tę sama parę **(n, p)** do struktury indeksu drugorzędnego.
  - **Usuwanie:**
    Kiedy usuwamy krotkę z wartością **n** atrybutu **A**, musimy najpierw zachować wskaźnik –nazwijmy go **p** – do usuwanej krotki.
    Następnie przechodzimy do struktury indeksu drugorzędnego i sprawdzamy wszystkie pary z pierwszą składową zawierającą wartość **n**, aż znajdziemy tę, której druga składowa ma wartość **p**.
    Znaleziona w ten sposób para jest teraz usuwana ze struktury indeksu drugorzędnego.

Poruszanie się wśród wielu relacji

- Do tej pory rozważaliśmy wyłącznie operacje na pojedynczych relacjach, takie jak znajdowanie krotki dla danych wartości jednej lub kilku jej składowych.
- Możliwości modelu relacyjnego można jednak w pełni docenić w momencie, gdy rozważamy operacje wymagające „poruszania się", lub „przechodzenia" z jednej relacji do drugiej.

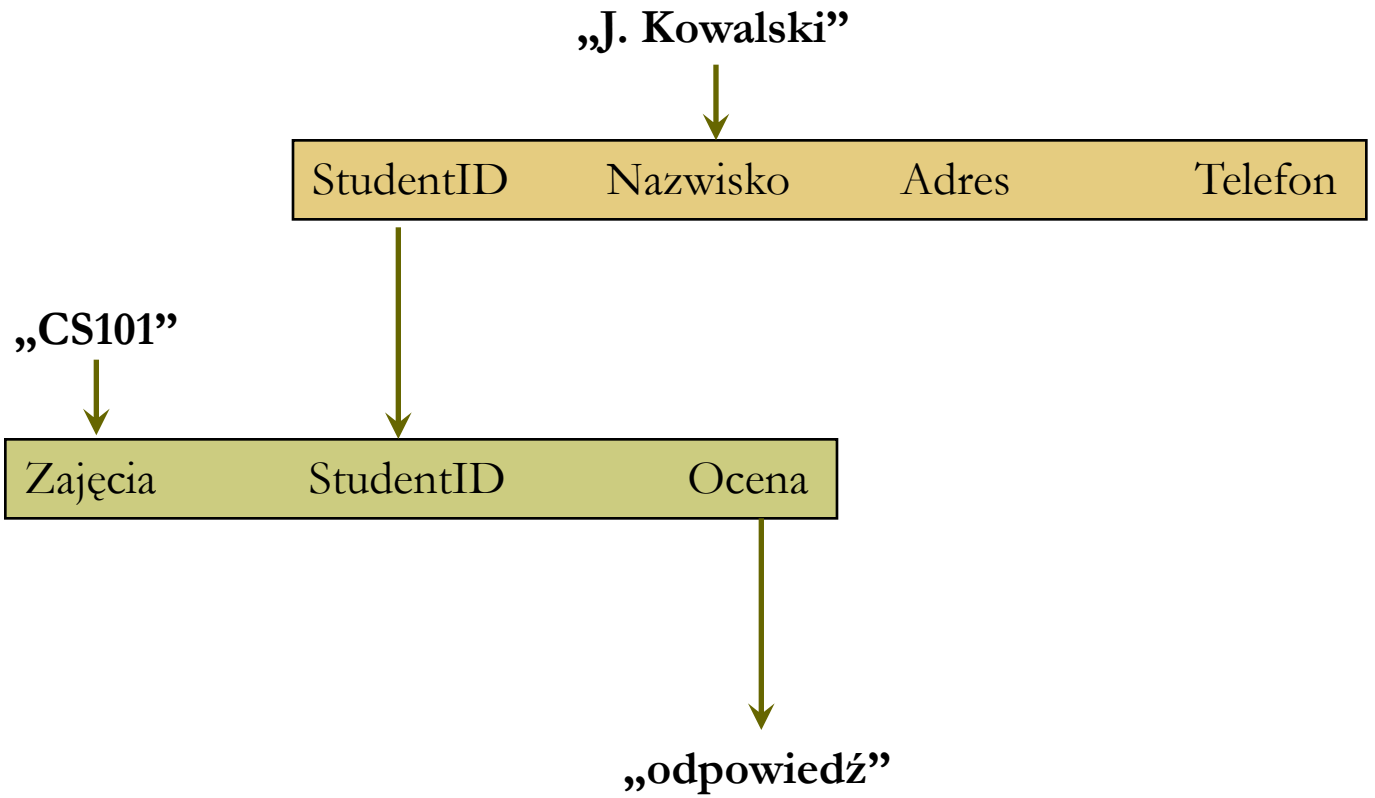
- Aby znaleźć odpowiedz na pytanie:

  „Jaką ocenę uzyskał J. Kowalski z przedmiotu CS101?"

  musimy:

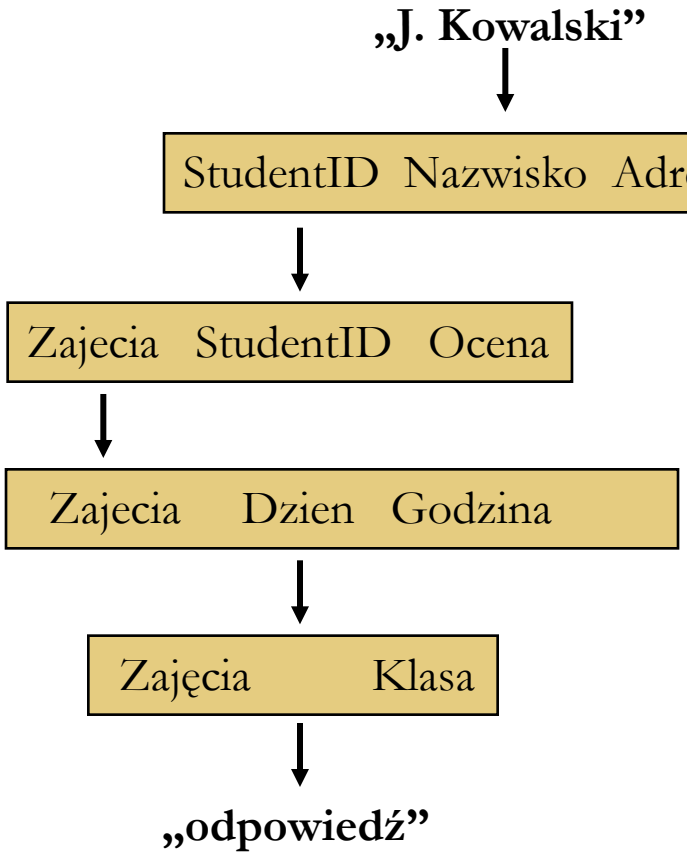
  1. odwołać się do relacji StudentID-Nazwisko-Adres-Telefon i przełożyć dane nazwisko „J. Kowalski" na odpowiedni numer indeksu (możliwość istnienia duplikatu nazwiska ale nie numeru indeksu),
  2. odwołać się do relacji Zajęcia-StudentID-Ocena i wyznaczyć krotkę mającą w polu Zajęcia wartość „CS101" a w polu numer indeksu wyznaczoną poprzednio wartość,
  3. odczytać wartość umieszczoną w polu Ocena.

Diagram wykonania zapytania:
„Jaka ocenę uzyskał J. Kowalski z przedmiotu CS101? ”



- **Czas realizacji zapytania będzie dużo krótszy jeżeli wykorzystamy indeksowanie drugorzędne → patrz ćwiczenia.**

Diagram wykonania zapytania: „Gdzie przebywa J. Kowalski w poniedziałek o 9-tej rano? ”



Zakładając że jest zapisany na tylko jedne zajęcia które odbywają się o tej porze…

- **Indeksowanie drugorzędowe bardzo przyspiesza czas wykonania.**

Projektowanie

- **Projektowanie I : wybór schematu baz danych**
  - rozdzielamy informacje budując kilka relacji (krotek) zamiast umieszczać je w jednej dużej krotce,
  - nie należy rozdzielać atrybutów reprezentujących powiązane ze sobą informacje.
- **Projektowanie II : wybór klucza**
  - jeden z ważniejszy aspektów projektowania bazy danych,
  - nie istnieje „jedyna" właściwa metoda wybierania klucza.
- **Projektowanie III: wybór indeksu głównego**
  - ma zdecydowany wpływ na szybkość z jaka możemy wykonywać „typowe" zadanie.
- **Projektowanie IV: kiedy tworzyć indeks drugorzędny?**
  - utworzenie ułatwia wykonywanie operacji wyszukiwania krotki dla danej wartości jednej lub więcej składowych,
  - każdy indeks drugorzędny wymaga dodatkowego czasu wstawiania i usuwania informacji z relacji.

Podsumowanie

- Istnieje wiele istotnych własności **relacji binarnych**.
  Do najważniejszych należą: **zwrotność, przechodniość, symetria i antysymetria**.
  Relacja porządku częściowego, porządku całkowitego oraz relacja równoważności to specyficzne rodzaje relacji binarnych;
- Dwuwymiarowe tabele zwane relacjami, są uniwersalnym sposobem przechowywania informacji.
  Wiersze relacji nazywamy **krotkami**, zaś kolumny noszą nazwę **atrybutów**.
- „**Indeks główny**" reprezentuje krotki relacji w formie struktury danych i rozdziela je w taki sposób, by ułatwić (przyśpieszyć) operacje wykorzystujące dane wartości należące do „dziedziny" indeksu;

Podsumowanie

- „**Kluczem**” relacji jest zbiór atrybutów, które jednoznacznie określają wartości wszystkich pozostałych atrybutów tej samej relacji.
  Klucz jest często wykorzystywany jako dziedzina indeksu głównego;
- „**Indeksy drugorzędowe**” są strukturami danych ułatwiającymi operacje, w których dane są wartości konkretnych atrybutów nie będących zazwyczaj częścią indeksu głównego.
  Ułatwiają szybkie odczytanie lub zmodyfikowanie informacji zawartych w tabeli.