Uniwersytet Wrocławski

Bazy danych 2022

Wykład 5

Modelowanie: diagramy encji i związków, SQL — data definition language

Bartosz Brzostowski

Wydział Fizyki i Astronomii UWr
semestr letni r. akad. 2021/22

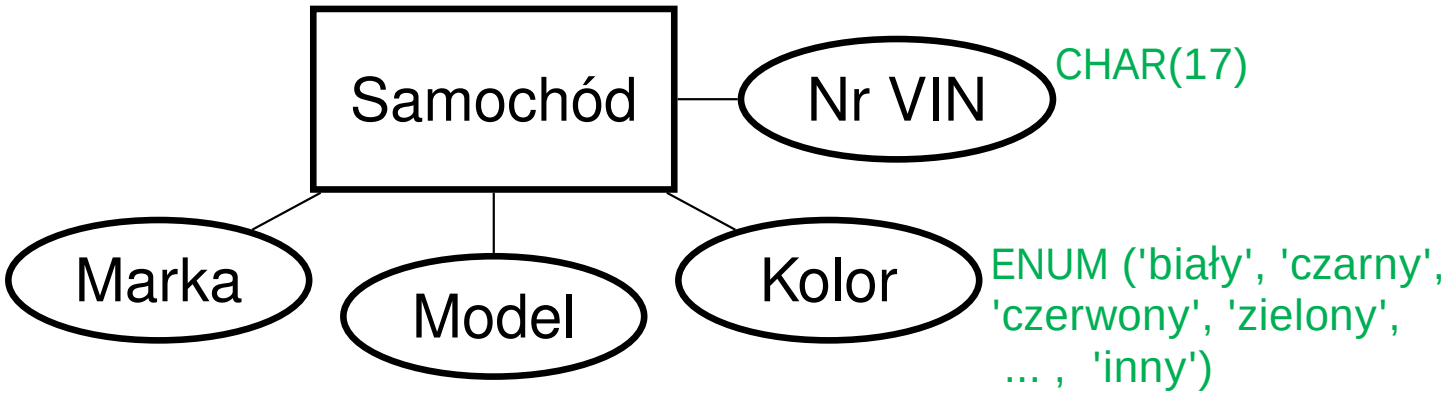
31 marca 2022

Modelowanie

- Tworzenie schematu bazy danych odpowiedniego do danego zastosowania
- (Co najmniej) dwa etapy
- Modelowanie *konceptualne* i *fizyczne*
- Model konceptualny: dość abstrakcyjny opis informacji, które zamierzamy gromadzić
- Diagram encji i związków (*entity-relationship*) — sformalizowany:
  - notacja Chena — prezentowana na wykładzie
  - notacja Barkera
  - UML, inne
- SQL (DDL) jest dla modeli fizycznych i na chwilę możemy o nim zapomnieć
- Na potrzeby tego przedmiotu wystarczą diagramy rysowane na tablicy / kartce

Encje i atrybuty

- *Encja* (czasem: zbiór encji) — zbiór bytów, obiektów, zdarzeń… istniejących w modelowanej rzeczywistości
- Cechy encji to *atrybuty*
- Nie wszystkie cechy będą należeć do modelu: w bazie studentów nie trzymamy koloru oczu (ale na portalu randkowym tak)
- Diagram: encja to prostokąt, atrybuty — owalne „dymki”

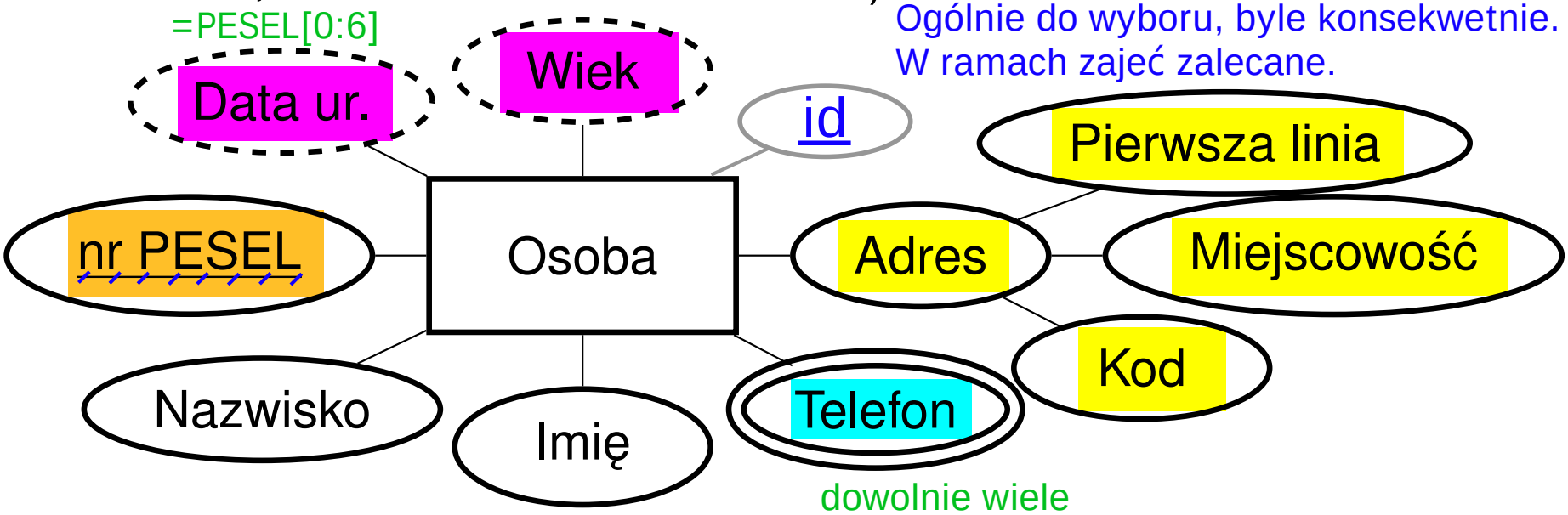


- Dziedzina atrybutu: np. komentarz w / obok dymka

Rodzaje atrybutów

- Prosty
- Złożony — z własną strukturą (dymek z dymkami)
- Wielokrotny — gdy może wystąpić więcej niż raz (podwójna obwódka, krotność w komentarzu)
- Wyliczany — zależy jednoznacznie od innych (przerywana obwódka, np. wzór w komentarzu)
- Główny — jeśli należy do *klucza głównego* (podkreślona nazwa, inne klucze w komentarzu)

Czy podawać klucze syntetyczne?
Ogólnie do wyboru, byle konsekwetnie.
W ramach zajeć zalecane.



Związki

Funkcje przypisują argumentom wartość w sposób jednoznaczny ozn. f(x) = y.
W przykładzie f(post) = autor

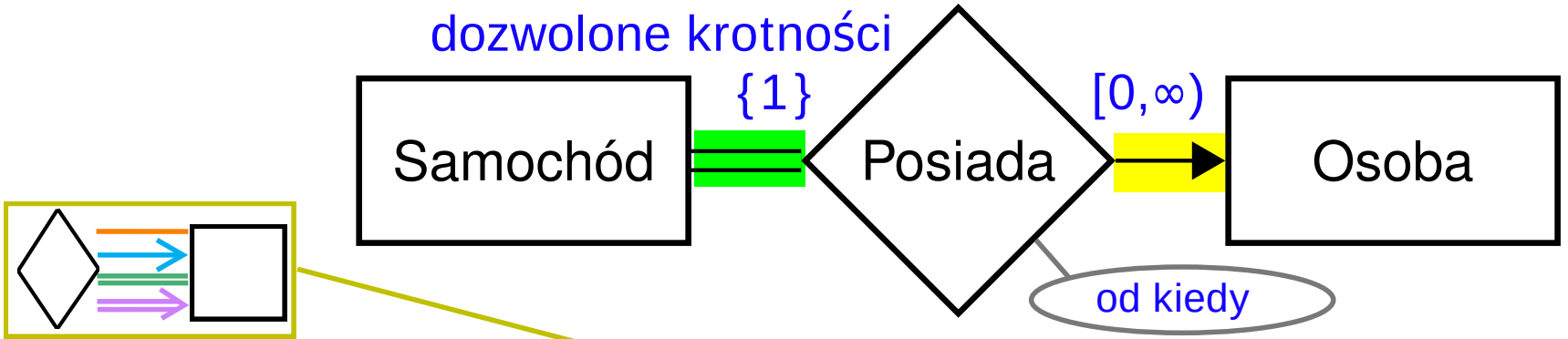
- Typy
  - Wiele-do-wielu (wieloznaczne)
  - Jeden-do-wielu (jednoznaczne / funkcyjne)
  - Jeden-do-jednego (wzajemnie jednoznaczne)
- Przykłady
  - Użytkownik *X* lajkuje posta *Y*
  - Użytkownik *X* jest autorem posta *Y*
  - . . . Wypadek *X* jest opisany w protokole *Y*
- Związki opcjonalne / obowiązkowe (wymuszone) — z każdej „strony” może być inaczej (post pewnie musi mieć autora, użytkownik na pewno może nie mieć postów)
- Strona związku może wchodzić w niego określoną liczbę razy — typowe (choć nie jedyne) możliwości to $[0,\infty), [1,\infty), \{0,1\}, \{1\}$

opcjonalny wielokrotny | obowiązkowy wielokrotny | opcjonalny jednoznaczny | obowiązkowy jednoznaczny

Związki na diagramach

Nazwa związku powinna się "dobrze czytać" np. Osoba POSIADA Samochód

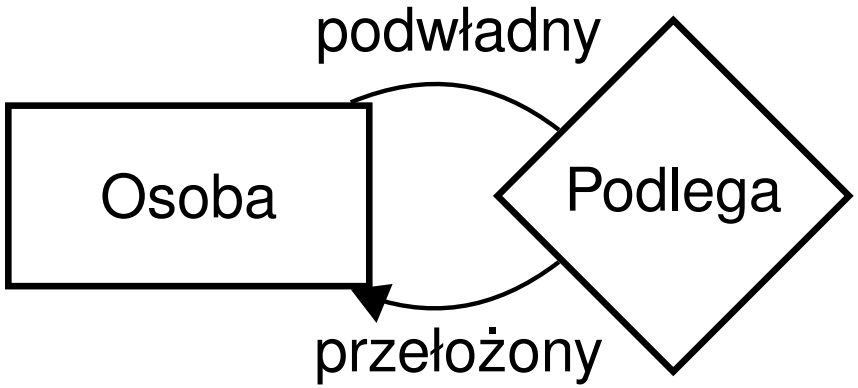
- Romb połączony liniami z encjami, które wiąże
- Związek obowiązkowy dla którejś encji: podwójna linia po jej stronie (lub linia pogrubiona, na zajęciach nie zalecana)
- Związek jeden-do-. . . : strzałka po stronie „jeden” (czyli po przeciwnej stronie niż encja, która najwyżej raz wchodzi w związek!): ***jedna*** *osoba posiada (potencjalnie)* ***wiele*** *samochodów* *(ale samochód jest posiadany przez jedną osobę)*



- Łącznie cztery typy linii, ~~16~~ 10 typów związków binarnych — ale tylko kilka „typowych”
- Związki mogą mieć atrybuty (znowu dymki)

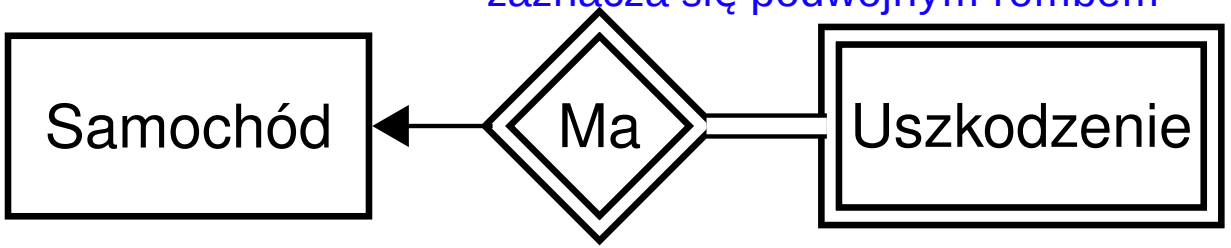
Związki — komplikacje

- Związki mogą dotyczyć więcej niż dwóch encji: zajęcia *X* *odbywają* się w sali *Y* w porze *Z*
- Związek rekurencyjny (raczej nie „unarny”): odwołuje się więcej niż raz do tej samej encji



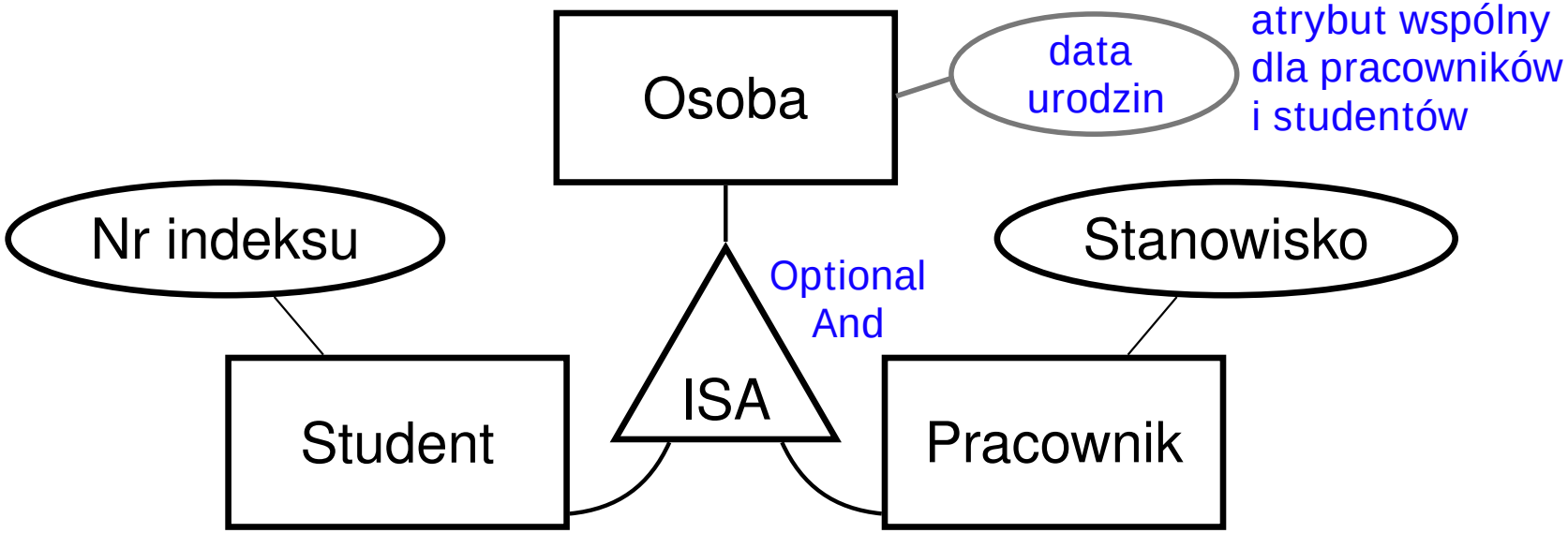
- Słaba encja: instancje rozróżnialne dopiero w kontekście któregoś ze związków

związek, który ujednoznacznia słabą encję, zaznacza się podwójnym rombem



Hierarchia encji (*is a*)

- Encje mogą mieć „podklasy” z dodatkowymi atrybutami
- Podencji może być więcej z różnymi zestawami atrybutów, ale wszystkie dziedziczą atrybuty z nadencji
- Instancja podencji zawsze jest instancją nadencji



- Należenie do podencji może być opcjonalne lub obowiązkowe — komentarz odp. *Optional* / *Mandatory*
- Należenie do kilku podencji naraz może być dozwolone lub nie — komentarz odpowiednio *And* / *Or*

Data definition language

- Część języka służąca do określenia schematu bazy
- Podstawowe elementy schematu: tabele, perspektywy
- `CREATE TABLE [IF NOT EXISTS]` *`[nazwa_tabeli]`* `(`
  `[nazwa_kol1] [typ_danych1] [więzy_kolumnowe1],`
  `[nazwa_kol2] [typ_danych2] [więzy_kolumnowe2],`
  `...,`
  `[CONSTRAINT` *`[nazwa_więzu]`*`]` *`[więz_tabelowy]`*`,`
  `...`
  `);`
- Więzy tabelowe — wielokolumnowe warianty dla `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY`

Typy danych

- Liczbowe (modyfikator `[UN]SIGNED`, *po* nazwie typu):
  - Całkowitoliczbowe (długości 1, 2, 3, 4, 8 bajtów):
    `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, `BIGINT`
    — dla nich parametr wpływa tylko na wyświetlanie
  - Zmiennopozycyjne: `FLOAT`, `DOUBLE` ← mają argumenty, ale innego sensu niż DECIMAL – zmniejszają precyzję!
  - Stałopozycyjne:
    `DECIMAL(`*`[#wszystkich_cyfr]`*`,`*`[#cyfr_po_kropce]`*`)`
- Daty i czasu: 1 sty 31 gru + "data zero" 0000-00-00
  - `DATE`: od 1000 do 9999 r., `DATETIME`: analogicznie z czasem
  - `TIMESTAMP`: ograniczony do epoki Uniksa 1970–2038
  - `TIME`: sam czas (ze znakiem) ± ok 840:00:00 ≃ 2 tygodnie
  - Wszystkie czasowe z opcjonalną precyzją do 6 (µs)
  - `YEAR`: rok od 1901 do 2155 lub 0000 ← to jest TINYINT (256 możliwych wartości) w przebraniu
- Tekstowe:
  - `CHAR` o stałej długości (parametr do 255)
  - `VARCHAR` o zmiennej długości (parametr do 65535)
  - `TEXT`, `BLOB` (także `TINY-`, `MEDIUM-`, `LONG-`)
  - `ENUM('x','y','z',...)`: zawsze dopuszcza wartość pustą

do rozmiaru odpowiednio 65535, 256, 16MB, 4GB

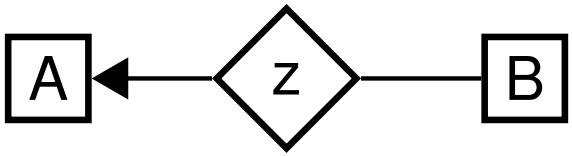
1-2 dodatkowe bajty na długość napisu

Własności kolumn i więzy

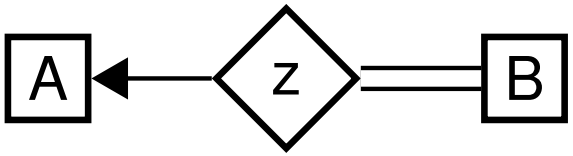
- `NULL` (najczęściej domyślnie) albo `NOT NULL`: kolumna (nie)nullowalna
- `UNIQUE`: niedozwolone powtórzenia wartości
- Ale: jeśli brak `NOT NULL`, to NULLe mogą się powtarzać — nie są wartościami; zdanie (NULL = NULL) jest nie jest prawdziwe, tylko niewiadome
- `PRIMARY KEY`: najwyżej jeden raz; klucze kandydujące — `UNIQUE NOT NULL`
- `DEFAULT` *[stała]* : wartość domyślna
- Dla `DATETIME`, `TIMESTAMP` (ale nie `DATE`!) także `NOW`
- Gdy brak: NULL, gdy także `NOT NULL`: zależnie od typu
- Specyficzne zachowanie *pierwszej* kolumny `TIMESTAMP`: automatycznie zachowuje datę modyfikacji wiersza
- Dla tekstowych: deklaracja kodowania, ~~kolacji~~ sortowania
- `AUTO_INCREMENT`: dobre dla kluczy „syntetycznych”

Klucze obce, modelowanie związków

- *[kol1]* ... REFERENCES *[tab]*(*[kol2]*)
  ON UPDATE ... ON DELETE ...
  lub: potomna ↔ macierzysta
- Tabela z tą definicją — podrzędna, *[tab]* — nadrzędna
- Akcje referencyjne dotyczą zmian w tabeli nadrzędnej:
  - CASCADE: analogiczna zmiana w tabeli podrzędnej
  - RESTRICT = NO ACTION: zablokowanie zmiany (domyślne)
  - SET NULL: wstawienie NULL w tabeli podrzędnej (kolumna musi być NULLowalna)
  - SET DEFAULT: ~~ustawia wartość domyślną~~ niezaimplementowane
- Typy danych powiązanych kolumn muszą być podobne
- Związki funkcyjne modeluje się przez klucze obce:



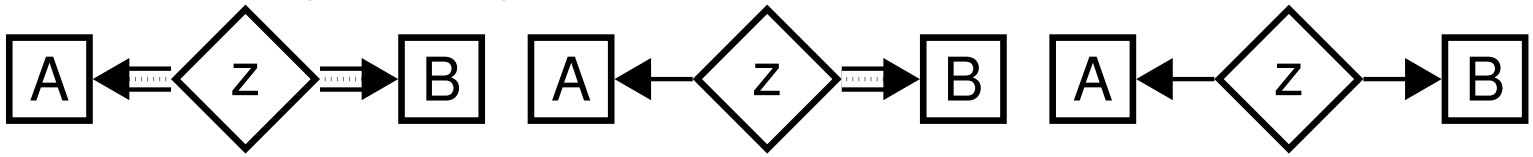
klucz obcy w tabeli B



(tu dodatkowo z NOT NULL)

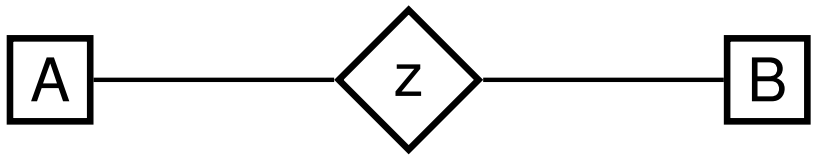
Modelowanie związków cd.

- Związki wzajemnie jednoznaczne



sklejenie tabel | klucz obcy `UNIQUE` `NOT NULL` w B | klucz obcy `UNIQUE` w A **albo** B

- Związki wieloznaczne



Dodatkowa tabela z *dwoma* kluczami obcymi `NOT NULL` odnoszącymi się do A i B

- Pozostałe związki nie modelują się łatwo — najczęściej „zamienia się” linie podwójne na pojedyncze

DDL cd.

- Kolumny generowane (`[kol1] AS [wyrażenie]`)
  — modelują atrybuty wyliczane, ale mogą zależeć tylko od innych kolumn w tym samym wierszu
- Usuwanie tabel: `DROP TABLE [IF EXISTS] [t1], [t2];`
  — w *dobrych* DBMS także deklaracja `RESTRICT` / `CASCADE`
- Modyfikowanie tabel: `ALTER TABLE [tabela] ...`
  - `RENAME TO`: zmienia nazwę tabeli
  - `DROP` / `ADD` / `CHANGE COLUMN`: operacje na kolumnach, składnia oczywista lub taka jak w `CREATE TABLE`
  - `DROP PRIMARY KEY`, `DROP FOREIGN KEY [nazwa_fk]` — jeśli więzy nie były jawnie nazwane przy definiowaniu, to DBMS nadaje nazwy automatycznie

brak w MySQL

Czyli jak od tabeli zależą inne obiekty (perspektywy, inne tabele przez klucz obcy) to czy usunąć je też, czy zablokować operację?

Modelowanie hierarchii *is a*

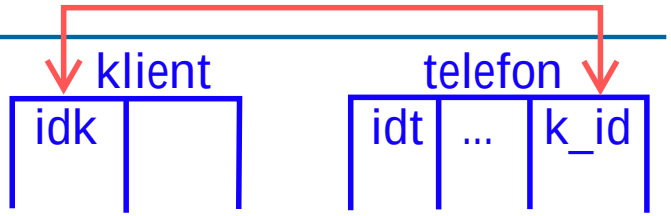
- Trzy sposoby: za pomocą 1, n lub $n+1$ tabel (n podencji)
- Jedna tabela: atrybuty z nadencji i wszystkich podencji
  - Jeśli hierarchia *Or*: dodatkowy atrybut (np. typu `ENUM`) — do jakiej podencji należy dany wiersz?
  - Jeśli hierarchia *And*: n dodatkowych atrybutów (boolowskich) — czy dany wiersz należy do danej podencji?
  - Atrybuty pochodzące z innych podencji niż ta, do której należy dany wiersz powinny być zawsze NULL *
  - Co jeśli dana podencja ma atrybuty nieNULLowalne?
- Po jednej tabeli na podencję: atrybuty jej i nadencji
  - Trudno wymusić hierarchię *Or** a jeszcze gorzej *Optional**
  - Konieczność użycia `UNION` w roli nadencji w zapytaniach
- Jedna tabela z atrybutami nadencji, po jednej tabeli z atrybutami każdej podencji + klucz obcy do nadencji
  - Trudno wymusić hierarchie *Or** albo *Mandatory**
  - Więcej złączeń w zapytaniach
  - Najlepsza dla modelowania związków (innych dla nadencji, innych dla podencji)

* Da się rozwiązać po stronie DBMS, choć może nie łatwo

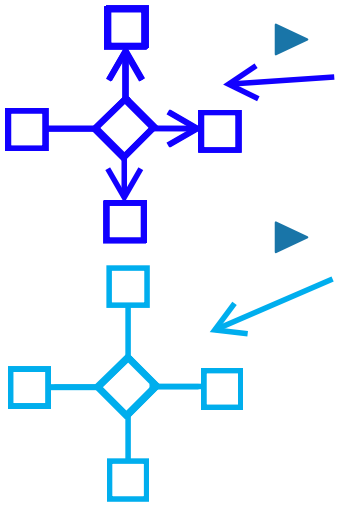
Gdzie zapisać coś, co nie należy do żadnej podencji?

Np. gdy nadencja wchodzi w związek. Można użyć perspektywy. Ale to już praktycznie III wariant.

Modelowanie cd.



- Atrybut wielokrotny: dodatkowa tabela z kluczem obcym do tabeli modelującej encję
- Słaba encja: kolumna klucza obcego jest jednocześnie składową (złożonego) klucza głównego
- Związek rekurencyjny: klucz „obcy” odwołujący się do innej kolumny tej samej tabeli
- Związek n-arny:
  - Jeden-do-jednego-. . . -do-jednego-do-wielu: kilka kolumn z kluczami obcymi w tabeli po stronie „wiele” (tj. jak przy kilku osobnych związkach jeden-do-wielu)
  - Wiele-do-wielu-. . . -do-wielu-do-wielu: dodatkowa tabela z n kluczami obcymi



Data modification language

Czyli to co zostało z SQL poza DQL i DDL, odpowiada to trzem z czterech liter C R U D

- `INSERT INTO`: wstawianie wierszy; dwie składnie:
  - `INSERT INTO` *`[tab]`* `VALUES (...), (...) ...;`
    — liczba i kolejność wartości taka, jak w definicji tabeli
  - `INSERT INTO` *`[tab]`*`(`*`[kol1]`*`, `*`[kol2]`*`, ...)`
    `VALUES (...), (...) ...;`
    — liczba i kolejność taka, jak zadeklarowana; brakujące wartości zastępowane NULL / domyślnymi / na podstawie `AUTO_INCREMENT`...
- Wiersze łamiące więzy nie są wstawiane
- `REPLACE` (rozszerzenie MySQL): analogicznie, ale wiersze, które łamałyby `PRIMARY KEY` lub `UNIQUE`, zastępują „stare”
- `DELETE FROM` *`[tab]`* `WHERE` *`[warunek]`*`;` — usuwa wybrane wiersze (gdy brak `WHERE`: usuwamy wszystko)
- `UPDATE` *`[tab]`* `SET` *`[kol1]`* `=` *`[wyr1]`*`, `*`[kol2]`* `=` *`[wyr2]`*`, ...` `WHERE` *`[warunek]`*`;` — modyfikuje wiersze; wyrażenia zależą od innych kolumn danego wiersza, stałych, funkcji bezargumentowych; alternatywnie: `DEFAULT`