Napredni modeli i baze podataka

Predavanja
Listopad 2015

3. Objektno orijentirane i objektno-relacijske baze podataka

Pregled

- Objektno orijentirane baze podataka
  - Načela objektno orijentiranih baza podataka
  - Objektno orijentirani sustavi za upravljanje bazama podataka
  - ODMG standard

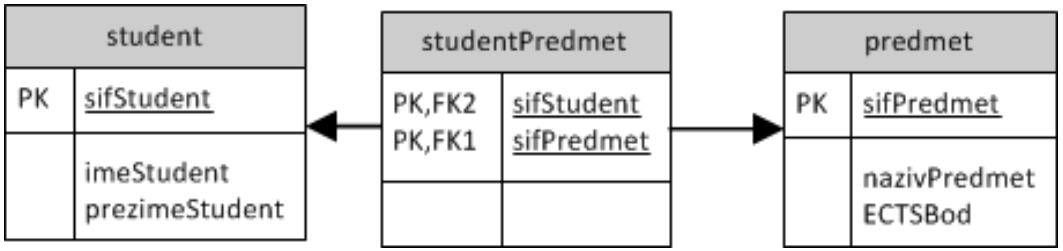
- Objektno-relacijske baze podataka
  - Objektno-relacijski model podataka
  - Objektno-relacijske mogućnosti prema SQL standardu
  - Objektno-relacijska proširenja u PostgreSQL SUBP-u

Zašto objektno orijentirane baze podataka? (1)

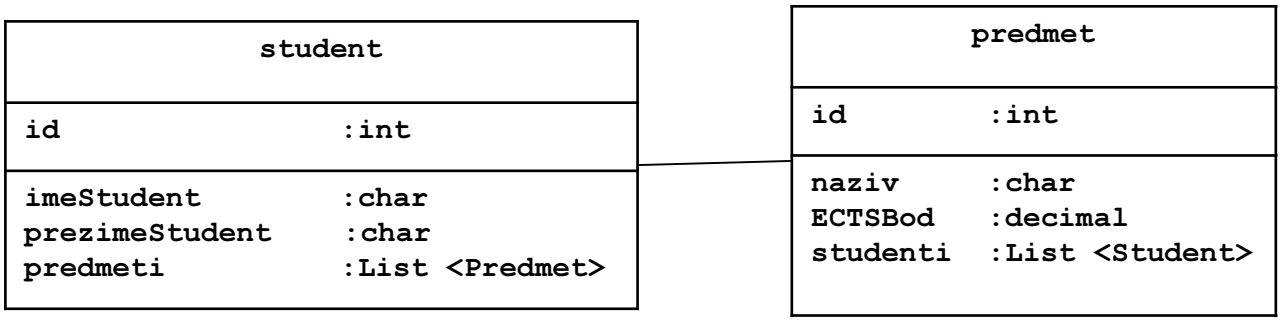
- Relacijske baze podataka nisu prikladne za aplikacije koje koriste složene tipove podataka ili nove tipove podataka za velike nestrukturirane objekte (nestrukturirani tekst, slike, multimedija, GIS objekti,…)
- Model relacijskih baza podataka bitno se razlikuje od objektnog modela aplikacija realiziranih objektno orijentiranim jezicima (Java, C#)

ER model

Relacijski model



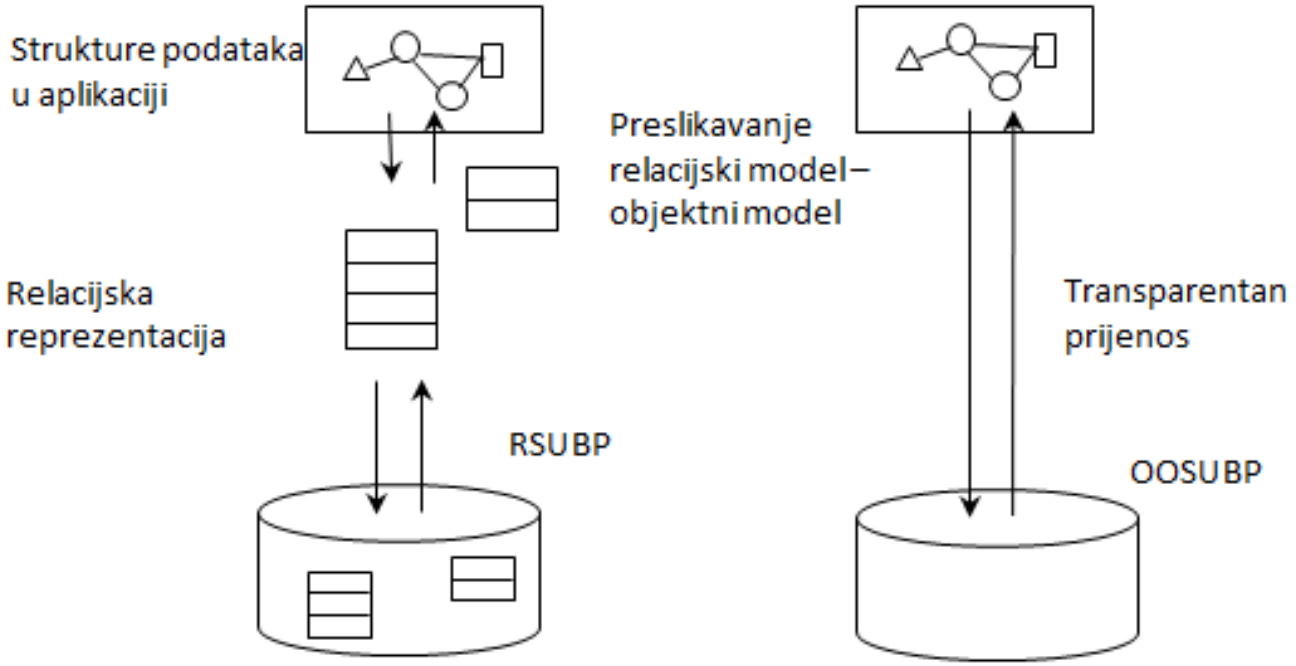
Objektni model



- **Objektno relacijska neusklađenost (*object relational impedance mismatch*)**

Zašto objektno orijentirane baze podataka? (2)

- **Objektno relacijska neusklađenost** (*object relational impedance mismatch*)
- Preslikavanje između dva modela je zahtjevan posao – potreba za transparentnim rukovanjem podacima iz baze, koristeći paradigme objektno orijentiranih jezika



Zašto objektno orijentirane baze podataka? (3)

- **Objektno relacijska neusklađenost** (*object relational impedance mismatch*) (primjer):
  - **Veze između entiteta**:

    Relacijski model: relacije student, studentPredmet, predmet
    - Primarnim i stranim ključevima

    Objektni model: `student.getPredmeti()` ili `predmet.getstudenti()`
    - Referencama na druge objekte
  - **Dohvat podataka**:

    Relacijski model:
    ```
    SELECT predmet.nazivPredmet
      FROM predmet, studentPredmet, predmet
     WHERE ......
    ```
    - SQL (DDL, DML)

    Objektni model: OQL (Object Query Language), SODA (Simple Object Data Access), pomoću objektnog grafa (`student.getPredmeti().get(0).getNaziv()`)
  - **Nasljeđivanje nije podržano u relacijskom modelu**

Objektno-orijentirani model – relacijski model

Objektni pristup	**Relacijski pristup**
Razred	Relacijska shema
Objekt	Entitet, n-torka
Varijable razreda	Atributi
Metoda	Procedura
-	Primarni ključ
OID	-

Čemu bi odgovarala relacija?

Objektno-orijentirana baza podataka

- Objektno-orijentirane baze podataka nazivaju se još i *bazama objekata* (*object databases*)
  - U bazu se pohranjuju objekti – model u bazi ne razlikuje se od onoga u aplikaciji
  - Implementacije OOSUBP uglavnom su programirane za specifičan programski jezik i međusobno se bitno razlikuju
- Objektno orijentirani sustavi za upravljanje bazama podataka *OO SUBP je sustav za upravljanje bazama podataka koji implementira objektno orijentirani model podataka*
- **Manifest o objektno-orijentiranim sustavima baza podataka** (*The Object-oriented Database System Manifesto), Atkinson i drugi,* 1989. – znanstveni članak o svojstvima koje mora zadovoljavati OOSUBP
  - Koncepti objektno-orijentiranog sustava
  - Koncepti sustava za upravljanje bazama podataka

Osnovna načela objektno orijentiranih baza podataka/sustava za upravljanje bazama podataka

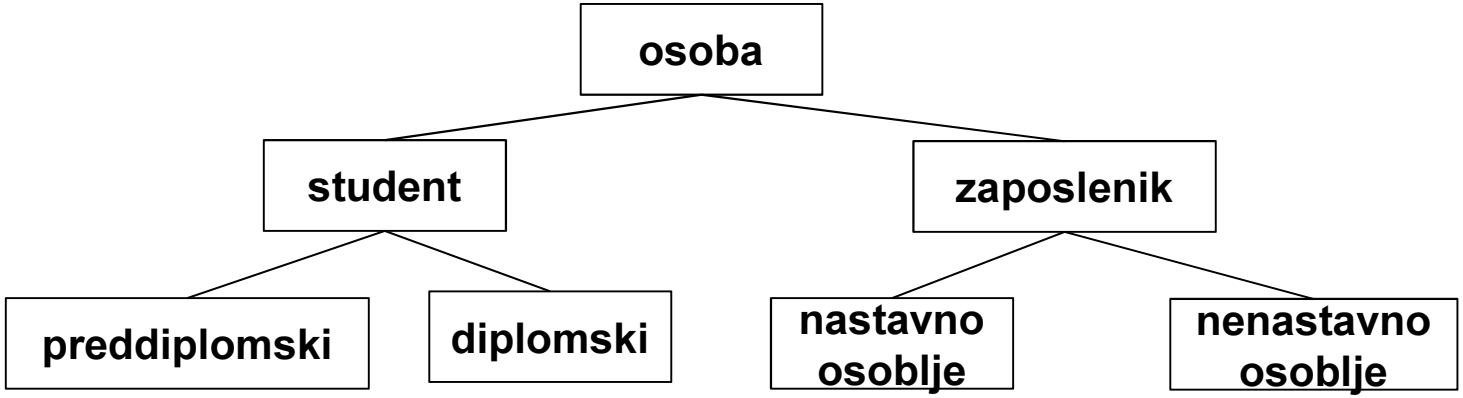
- Koncepti objektno-orijentiranog sustava
  - Razredi (klase)
  - Složeni objekti
  - Identitet objekta
  - Hijerarhije klasa
  - Učahurivanje (*encapsulation*)
  - Nadjačavanje (*overriding*), preopterećivanje (*overloading*) i kasno vezivanje (*late binding*)
- Koncepti sustava za upravljanje bazama podataka
  - Perzistencija podataka
  - Fizička organizacija (*secondary storage management*)
  - Paralelni rad (*concurrency*)
  - Oporavak baze podataka (*recovery*)
  - Ad hoc upitni jezik (*Ad Hoc Query Facility*)

Identitet objekta - OID

- Jedinstven, nepromjenjiv identifikator objekta generiran od OO sustava
- Neovisan o vrijednostima atributa objekta
- Nevidljiv korisniku
- Koristi se za referenciranje objekata
- Dva objekta su identična ako im je svojstvo koje ih jedinstveno identificira isto – identitet objekta

- U relacijskim bazama podataka
  - identitet entiteta se temelji na vrijednostima podataka
  - primarni ključ se koristi da osigura jedinstvenost
    - primarni ključevi ne osiguravaju vrstu jedinstvenosti koja je potrebna za OO sustave:
      - ključevi su jedinstveni samo u relaciji, a ne u cijelom sustavu
      - ključevi se uglavnom temelje na atributima relacije, što ih čini ovisnima o stanju objekta

Hijerarhija razreda




```
public abstract class Osoba {
    private String IdOsoba;
    private String ime;
    private String prezime;
     ...
    // pristupne metode i konstruktori
     ...
}
```

```
public class Student extends Osoba{
    private String JMBAG;
     ...
    // pristupne metode i konstruktori
     ...
}
public class Zaposlenik extends Osoba{
    private Decimal iznosPlaca;
     ...
    // pristupne metode i konstruktori
     ...
}
...
```

- Dostupni atributi i metode nasljeđuju se od nadređene klase
- **Podklasa može definirati nove atribute i metode**
- **Generalizacija (*osoba*) i specijalizacija (*preddiplomski student*)**

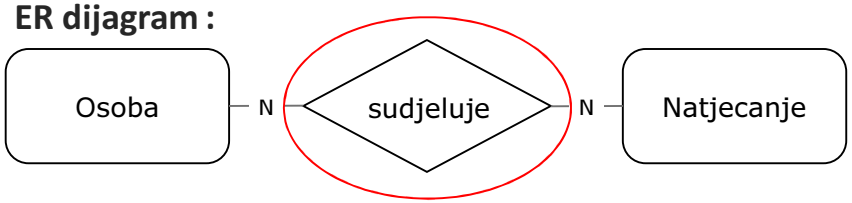
Veze između objekata

- Ostvaruju se **referenciranjem**
- 1:1 veza
  - `osoba.getPutovnica()`
  - `putovnica.getOsoba()`
- N:1 (1:N) veza
  - `kupac.getMjestoStanovanja()`
  - `mjesto.getKupci()`
- N:N veza
  - `kupac.getArtikli()`
  - `artikl.getKupci()`
- Sve veze mogu biti **dvosmjerne**

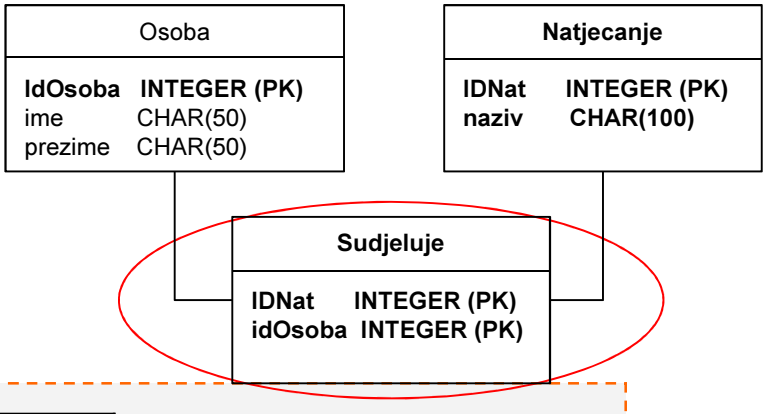
- Dvosmjernost nije nužno izraziti u objektnom modelu, ako nije važna za poslovni proces aplikacije

Veze između objekata (2)

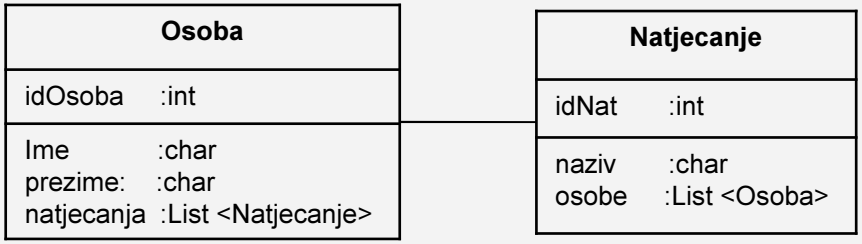
ER dijagram :



relacijski dijagram :



Objektni model



Definicija razreda:

```
public class Osoba {
    private String IdOsoba;
    private String ime;
    private String prezime;
    private List<Natjecanje> natjecanja;
    …
    // pristupne metode i konstruktori
    …
}
```

```
public class Natjecanje {
    private Int sifNat;
    private String naziv;
    private List<Osoba> osobe;
    …
    // pristupne metode i konstruktori
    …
    …
}
```

ODMG standard

- ODMG standard sastoji se od sljedećeg:
  - Objektni model (OM, eng. Object model)
  - Jezik za specificiranje objekata (ODL - Object Definition Language)
  - Objektni upitni jezik (OQL, eng. Object Query Language)
  - Veza na programske jezike
    - uključuje ODL koji je ovisan o odabranom programskom jeziku
    - pruža aplikacijsko programsko sučelje (API) za preslikavanje tipova podataka

OQL - Object Query Language (1)

- Upitni jezik za objektne baze podataka napravljen po uzoru na SQL
- Fleksibilan, ali zbog svoje kompleksnosti niti jedan ga proizvođač nije kompletno implementirao
- Razlike između OQL i SQL:
  - OQL podržava referenciranje na objekte unutar tablica. Objekti mogu ugnježđivati druge objekte.
  - OQL ne podržava sve ključne riječi iz SQL
  - OQL podržava matematičke izračune unutar OQL izraza
- Sintaksa:
  - Upiti oblika *Select-From-Where*

  ili

  - Navigacija kod kompleksnih objekata:

    `knjiga.izdavac.kontakt.email`

OQL - Object Query Language (2)

Rezultat OQL upita je objekt čiji tip ovisi o operandima koji sudjeluju u upitu.

Primjer: Dohvati nazive i cijenu jela na ponudi u restoranu "Snack":

```
public class Restaurant {
  private Int    IdRestaurant;
  private String name;
  ...
  private List<Dish> dish;
  ...
  //metode i konstruktori
}
```

```
public class Dish {
  private Int    IdDish;
  private String name;
  private Decimal price;
  ...
  //metode i konstruktori
}
```

```
public class Sells {
  private Int    IdDish;
  private String name;
  private Decimal price;
 ...
  private Restaurant restaurant;
  private List<Dish> dish;
  ...
  //metode i konstruktori
}
```

```
SELECT s.dish.name, s.dish.price
  FROM Sells s
 WHERE s.restaurant.name = "Snack"
```

OOSUBP – prednosti

- Bolje i brže upravljaju sa složenim objektima i vezama u odnosu na relacijske
- Podržavaju hijerarhiju, klase i nasljeđivanje
- Jedan podatkovni model - objekti u bazi i objekti u aplikaciji su jednaki
  - Nema objektno relacijske neusklađenosti
- Identifikacija objekata je skrivena od korisnika
  - Nema potrebe za primarnim ključem (?)
- Koristi se samo jedan programski jezik (za aplikaciju i za pristup bazi)

OOSUBP – mane

- Nema logičke neovisnosti podataka
  - Izmjene na bazi podataka (evolucija sheme) zahtijevaju izmjene u aplikaciji i obrnuto
- Nedostatak dogovorenih standarda, tj. postojeći standard (ODMG) nije u potpunosti implementiran
- Ovisnost o jednom programskom jeziku. Tipični OOSUBP je svojim programskim sučeljem vezan za samo jedan programski jezik
- Nedostatak interoperabilnosti s velikim brojem alata i mogućnosti koje se koriste u SQL-u
- Nedostatak Ad-Hoc upita (upiti na novim tablicama koje se dobiju spajanjem postojećih s *join*)

OOSUBP u stvarnom svijetu

- Chicago Stock Exchange - upravljanje trgovinom dionica (Versant)
- Radio Computing Services – automatiziranje radio stanica (POET)
- Ajou University Medical Center u Južnoj Koreji – sve funkcije bolnice, uključujući one kritične poput patologije, laboratorija, banke krvi, ljekarne i rendgena
- CERN – veliki znanstveni setovi podataka (Objectivity/DB)
- Federal Aviation Authority – simulacija prometa putnika i prtljage
- Electricite de France – upravljanje elektroenergetskim mrežama

OOSUBP proizvodi

- Versant
- Progress ObjectStore
- Objectivity/DB
- Intersystems Cachè
- POET fastObjects
- **db4o**
- Computer Associates Jasmine
- GemStone

Objektno-relacijske baze podataka

Motivacijski primjer 1
Predstavljanje geografskih podataka u relacijskoj bazi podataka

- Za određivanje položaja točke na Zemlji koriste se geografske koordinate:
  - geografska širina (latitude)
  - geografska dužina (longitude)
- Udaljenost d u km između dvije geografske koordinate (*lat1, long1*) i (*lat2, long2*), uzevši u obzir zakrivljenost Zemlje, može se izračunati pomoću npr. Haversinove formule:

$$a = \sin^2\left(\frac{lat2-lat1}{2}\right) + \cos(lat1) * \cos(lat2) * \sin^2\left(\frac{long2-long1}{2}\right)$$

$$d = R * 2 * atan2\left(\sqrt{a}, \sqrt{1-a}\right) \qquad R = 6371 \text{ prosječna vrijednost radijusa Zemlje}$$

Pogledaj zadatke za vježbu.

Motivacijski primjer 1
Predstavljanje geografskih podataka u relacijskoj bazi podataka

Želim od SUBP

- ugrađen tip podatka (npr. point) za prezentaciju točke na Zemlji pomoću geografskih koordinata
- ugrađenu funkciju
  - za određivanje udaljenosti između dvije točke na Zemlji
  - koja će pomoću dvije zadane točke znati odrediti liniju koja ih spaja
  - koja će pomoću n zadanih točaka znati odrediti najkraći put koji spaja te točke redom kojim su navedene
  - koja će pomoću n zadanih točaka znati odrediti zatvoreni poligon čiji su vrhovi zadane točke
  - koja će znati izračunati površinu gornjeg poligona
  - …
- da sve brzo radi ☺

Ništa od navedenog tradicionalni relacijski SUBP ne podržavaju

- Točka je složeni tip podatka – sastoji se od 2 elementarna podatka
- Linija kao skup točaka je također složeni tip podatka, jednako kao i poligon

Tradicionalni relacijski SUBP mora podržati nove složene tipove podataka i funkcije za rad s njima. Vjerojatno i nove vrste indeksiranja.

Motivacijski primjer 2

Predstavljanje i pretraživanje dokumenata u relacijskoj bazi podataka

Želim od SUBP

- ugrađen tip podatka pomoću kojeg će se
  - dokument predstaviti listom leksema te za svaki leksem pamtiti pozicije na kojima se pojavljuje u tekstu *
  - uvjet pretrage predstaviti listom leksema povezanih logičkim operatorom & ili || **
  - dokument predstaviti listom q-grama
- ugrađenu funkciju koja će
  - tekst na nekom svjetskom jeziku pretvoriti u ugrađeni tip podatka *
  - uvjet pretrage na nekom svjetskom jeziku pretvoriti u ugrađeni tip podatka **
  - odrediti sličnost između uvjeta i dokumenta temeljem morfologije, sintakse i semantike jezika
  - rangirati dokumente prema sličnosti između uvjeta i dokumenta
  - odrediti sličnost između uvjeta i dokumenta temeljem broja podudarnih q-grama
  - …
- da sve brzo radi ☺

Motivacijski primjer 2

Predstavljanje i pretraživanje dokumenata u relacijskoj bazi podataka

Ništa od navedenog tradicionalni relacijski SUBP ne podržavaju

- Lista leksema je složeni podatkovni tip: lista znakovnih nizova, a uz svaki znakovni niz dodatno je povezana lista pozicija na kojima se niz u tekstu nalazi
- Lista q-grama također je složeni podatkovni tip
- Tradicionalni načini indeksiranja (B-stablo) neće biti prikladni

Tradicionalni relacijski SUBP mora podržati:

- **nove složene tipove podataka**
- **funkcije za rad s njima**
- **nove načine indeksiranja složenih tipova podataka**
- **…**

Objektno-relacijski sustav za upravljanje bazama podataka

- **objektno-relacijski sustav** (*object-relational DBMS* - ORDBMS) ili prošireni relacijski sustav (*enhanced relational systems*)
  - pokušaj spajanja najboljeg iz relacijskog i objektno-orijentiranog pristupa
  - U manjoj ili većoj mjeri objektno-relacijski koncepti ugrađeni su u sve poznatije RSUBP
  - Primjeri na slajdovima - PostgreSQL

DBMS Matrix

M. Stonebreaker, D. Moore: Object-relational DBMSs – the next great wave, Morgan Kaufmann, 1996

Objektno-relacijski model podataka (1)

- Temeljen na relacijskom modelu podataka
  - sačuvane su relacijske karakteristike
  - zadržana kompatibilnost s postojećim relacijskim jezicima
- Proširuje relacijski model
  - uvedena objektna orijentacija i konstrukcije koje omogućuju rukovanje s novim tipovima podataka
  - dozvoljava da atributi u n-torkama imaju složene vrijednosti, uključujući i ugniježđene relacije (narušena 1NF)
  - znatno uvećane mogućnosti modeliranja podataka čime je proširen opseg primjene
- Ne postoji jedinstveni (opće prihvaćen) model - razlikuju se po tome koliko objektnog proširenja uključuju

Objektno-relacijski model podataka (2)

- Proširenja relacijskog modela:
  - apstraktni tipovi podataka (objektni tipovi, strukturirani korisnički-definirani tipovi, ...)
  - identifikatori objekta i reference
  - metode za objektne tipove, učahurivanje
  - korisnički definiran CAST
  - objektne tablice, tipizirane tablice
  - nasljeđivanje tipova i tablica
  - ugniježđene relacije (složeni atributi, kolekcije)

- SQL standardom predviđeni objektno-relacijski koncepti nisu potpuno implementirani niti u jednom SUBP-u

SQL standard: objektno-relacijska proširenja

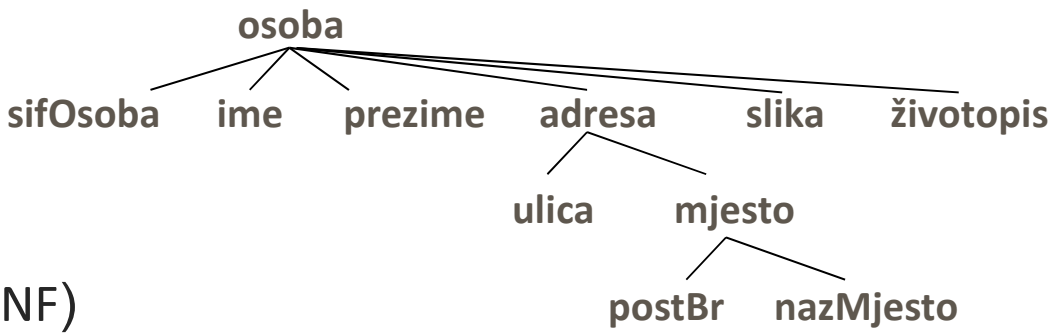
- SQL:1999 uključuje većinu objektno-relacijskih koncepata
  - konstruktori složenih tipova, identitet objekta, učahurivanje, nasljeđivanje
- podjela tipova podataka:
  - **unaprijed definirani tipovi (*predefined types*)**
    - atomarni tip - vrijednost nije izgrađena od vrijednosti drugih podatkovnih tipova: integer, float, character, boolean, datetime, interval ...
  - **izgrađeni tipovi (*constructed types*)**
    - izgrađeni atomarni tipovi (*constructed atomic types*)
      - referenca (*reference*)
    - izgrađeni kompozitni tipovi (*constructed composite types*)
      - kolekcije (*collection*): polje (*array*), multiset
      - *row*
  - **korisnički definirani tipovi (*user-defined types* - UDT)**
    - *distinct type*
    - strukturirani tip (*structured type*)

Primjer: Informacijski sustav studentske službe

- osobe na visokom učilištu (relacija *osoba* nije u 1NF)

osoba

sifOsoba	ime	prezime	adresa			slika	zivotopis
			ulica	mjesto			
				postBr	nazMjesto		
11001	Hrvoje	Novak	Ilica 25	10000	Zagreb		Rođen je .....
78936	Ana	Kolar	Marmontova 18	21000	Split		



- predmeti (relacija *predmet* nije u 1NF)

predmet

sifPred	nazPred	nositelj	izvodjaci	polaznici	zavod (kratZavod, nazZavod)
1	Napredni modeli i baze podataka	123	{123,111, 345,24}	{13503, 14111, 9000, 14678, ...}	(ZPR, Zavod za prim.računarstvo)

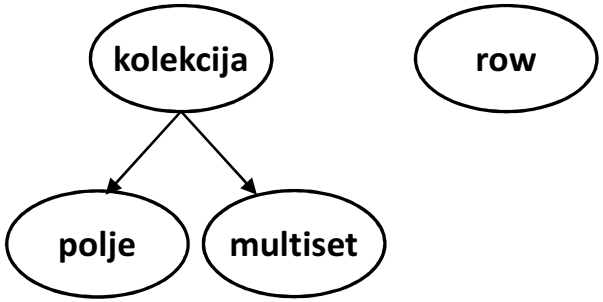
- postoji hijerarhija tablica *osoba*, *nastavnik* i *student*

SQL standard: tipovi podataka

- **unaprijed definirani tipovi (*predefined types*)**
  - atomaran tip - vrijednost nije izgrađena od vrijednosti drugih podatkovnih tipova: integer, float, character, boolean, datetime, interval ...
- **izgrađeni tipovi (*constructed types*)**
  - izgrađeni atomarni tipovi (*constructed atomic types*)
    - referenca (*reference*)
  - izgrađeni kompozitni tipovi (*constructed composite types*)
    - kolekcije (*collection*): polje (*array*), multiset
    - *row*
- **korisnički definirani tipovi (*user-defined types* - UDT)**
  - *distinct type*
  - strukturirani tip (*structured type*)

SQL standard: Izgrađeni tipovi

- dijele se na:
  - **atomarne**
    - referenca (*reference type* - **REF** *type*)
      - tip čija vrijednost pokazuje na lokaciju na kojoj je pohranjena vrijednost referenciranog tipa
      - mogu pokazivati samo na n-torke tablica temeljenih na strukturiranom tipu (*tipizirane tablice*)
  - **kompozitne**
    - svaka vrijednost je složena od jedne ili više vrijednosti koje pripadaju istim (**kolekcija**) ili mogu pripadati različitim podatkovnim tipovima (**ROW**)



- naziv tipa definiran standardom
- specificira se pomoću konstruktora tipa (REF, ARRAY, ROW)

SQL standard: tipovi podataka

- **unaprijed definirani tipovi (*predefined types*)**
  - atomaran tip - vrijednost nije izgrađena od vrijednosti drugih podatkovnih tipova: integer, float, character, boolean, datetime, interval ...
- **izgrađeni tipovi (*constructed types*)**
  - izgrađeni atomarni tipovi (*constructed atomic types*)
    - referenca (*reference*)
  - izgrađeni kompozitni tipovi (*constructed composite types*)
    - kolekcije (*collection*): polje (*array*), multiset
    - *row*
- **korisnički definirani tipovi (*user-defined types* - UDT)**
  - *distinct type*
  - strukturirani tip (*structured type*)

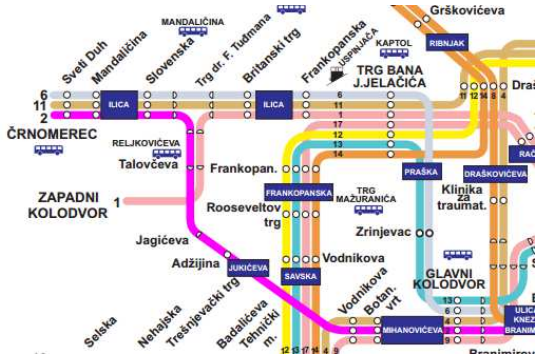
SQL standard: Kolekcija

Kolekcija vrijednosti homogenog podatkovnog tipa:

- polje (ARRAY)
  - jednodimenzionalno polje s maksimalnim brojem elemenata
  - podržano SQL:1999 standardom
- multiskup (MULTISET)
  - neuređena kolekcija koja dozvoljava duplikate
  - podržano SQL:2003 standardom
- skup (SET)
  - neuređena kolekcija koja ne dozvoljava duplikate
- lista (LIST)
  - uređena kolekcija koja dozvoljava duplikate

SQL standard: ARRAY

- indeksirana kolekcija elemenata homogenog podatkovnog tipa
- elementima polja moguće je pristupiti pomoću indeksa
- indeks elemenata polja $\in$ [1, kardinalnost]



Raspored vožnji linije 3

Raspored vožnji za 22.09.2015

Vrijeme	Polazište	Odredište
04:03:55	Ljubljanica	Savišće
04:19:55	Ljubljanica	Savišće
04:35:15	Ljubljanica	Savišće
04:50:55	Ljubljanica	Savišće
05:06:59	Ljubljanica	Savišće

```
CREATE TABLE tramLinija
(sifLinija        INTEGER       PRIMARY KEY,
 kratLinija       CHAR(3),
 nazivLinija      VARCHAR(120),
 tramPostaje      INTEGER ARRAY[50] REFERENCES tramPostaja(sifPostaja),
 vremenaPolaska   TIME ARRAY[300]
);
```

  - korištenjem polja za pohranu tramvajskih postaja poznat je i poredak postaja
- dva polja usporedivih tipova smatraju se identičnim akko su jednake kardinalnosti i na istoj poziciji imaju elemente jednakih vrijednosti

SQL standard: MULTISET

- neuređena i neograničena kolekcija elemenata homogenog podatkovnog tipa u kojoj se iste vrijednosti mogu ponavljati

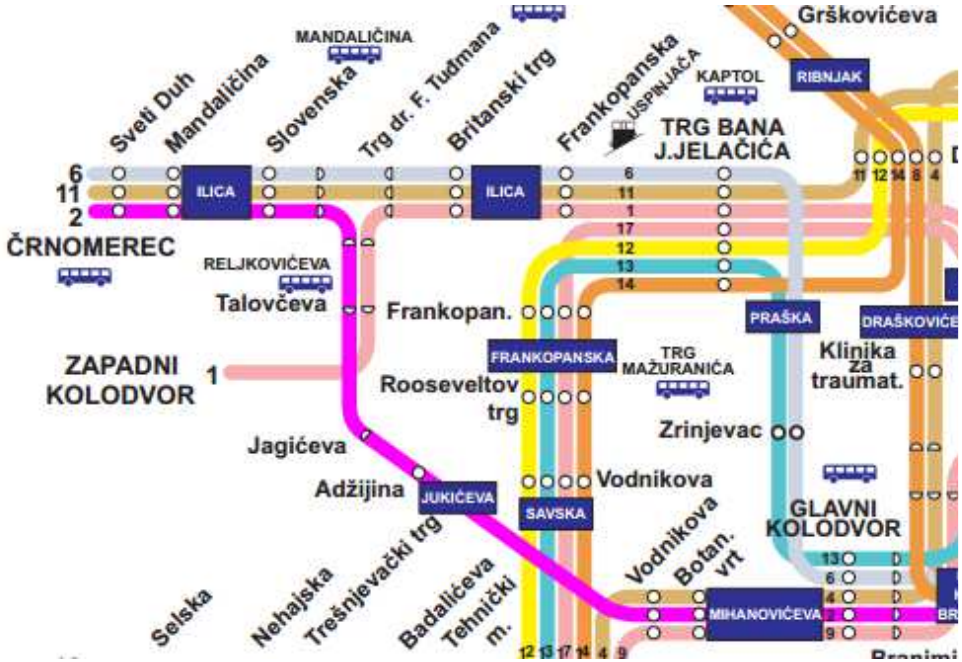
```
CREATE TABLE tramLinija
(sifLinija        INTEGER       PRIMARY KEY,
 kratLinija       CHAR(3),
 nazivLinija      VARCHAR(120),
 tramPostaje      INTEGER MULTISET REFERENCES tramPostaja(sifPostaja),
 vremenaPolaska   DATETIME MULTISET
);
```

  - korištenjem multiseta za pohranu tramvajskih postaja, nije poznat njihov poredak

- dva multiseta, *A* i *B*, usporedivih tipova elemenata, smatraju se identičnim akko imaju jednaku kardinalnost i za svaki je element *x* iz *A*, broj elemenata iz *A* koji su identični elementu *x* (uključujući), jednak broju elemenata iz *B* koji su jednaki elementu *x*.

PostgreSQL: ARRAY (1)

- u definiciji tipa atributa se ne navodi veličina polja
  - ako je veličina polja navedena, sustav ne dojavljuje pogrešku, već je ignorira




```
CREATE TABLE tramLinija
(sifLinija   SERIAL       PRIMARY KEY,
 kratLinija  CHAR(3)      NOT NULL UNIQUE,
 nazivLinija VARCHAR(120) NOT NULL UNIQUE,
 tramPostaje INTEGER[]);
```

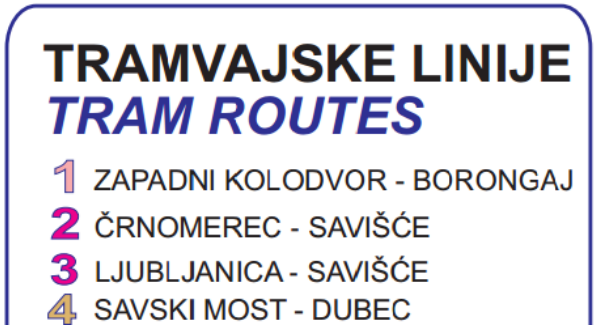
```
...
tramPostaje INTEGER[]
      REFERENCES tramPostaja(sifPostaja)
...
```

```
CREATE TABLE tramPostaja
(sifPostaja  SERIAL PRIMARY KEY,
 nazivPostaja VARCHAR(120) NOT NULL
                           UNIQUE);
```

```
INSERT INTO tramPostaja (nazivPostaja)
        VALUES ('Zapadni kolodvor');
INSERT INTO tramPostaja (nazivPostaja)
        VALUES ('Talovčeva');
...
```

- strani ključ (još) nije moguće definirati nad elementima polja, proširenje?!: **izvodjaci INTEGER[] ELEMENT REFERENCES tramPostaja**

PostgreSQL: ARRAY (2)



- upisivanje n-torke s atributom tipa ARRAY:

```
INSERT INTO tramLinija (kratLinija, nazivLinija, tramPostaje)
 VALUES ('1', 'Zapadni kolodvor - Borongaj',
        '{1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ,11, 12, 13, 14, 15, 16}')
```

Redoslijed postaja prati indeks elementa polja

ili :

```
INSERT INTO tramLinija (kratLinija, nazivLinija, tramPostaje)
 VALUES ('2', 'Črnomerec - Savišće',
        ARRAY[17, 18, 19, 20, 3 , 2,  21, 22, NULL, NULL, NULL])
```

ili :

```
INSERT INTO tramLinija (kratLinija, nazivLinija, tramPostaje)
 VALUES ('3', 'Ljubljanica - Savišće',
        '{}')     /* ili ARRAY[]::integer[] */
```

```
SELECT * FROM tramLinija
```

sifLinija integer	kratLinija character(3)	nazivLinija character varying(120)	tramPostaje integer[ ]
1	1	Zapadni kolodvor – Borongaj	{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}
2	2	Črnomerec - Savišće	{17, 18, 19, 20, 3 , 2, 21, 22, NULL, NULL, NULL}
3	3	Ljubljanica – Savišće	{}

PostgreSQL: ARRAY (3)

- upisivanje n-torke s atributom tipa ARRAY - elementi polja dobiveni SELECT naredbom:

```
INSERT INTO tramLinija (kratLinija, nazivLinija, tramPostaje)
 VALUES ('4', 'Savski most - Dubec',
         ARRAY (SELECT sifPostaja FROM tramPostaja
                  WHERE sifPostaja BETWEEN 56 AND 70)
         )
```

- pristupanje elementima polja

korištenjem indeksa elementa

array_length (polje,dim) – broj elemenata u polju (za zadanu dimenziju dim)

```
SELECT nazivLinija,
       tramPostaje[1] AS prva,
       tramPostaje[2] AS druga,
       tramPostaje[array_length (tramPostaje, 1)] AS zadnja
  FROM tramLinija
 WHERE tramPostaje[2] = 2 OR tramPostaje[1:2] = ARRAY [17,18]
```

pristupanje elementima čiji su indeksi unutar navedenih granica

nazivLinija character varying(120)	prva integer	druga integer	zadnja integer
Zapadni kolodvor – Borongaj	1	2	16
Črnomerec - Savišće	17	18	

PostgreSQL: ARRAY (4)

Operator	Opis	Primjer
=, <>	usporedba	ARRAY[1,2,3] = ARRAY[1,2,3] → true ARRAY[1,2,3,4] = ARRAY[1,2,4,3] → false
@>	sadrži	ARRAY[3,1,4,2] @> ARRAY[2,1] → true ARRAY[3,1,4,2] @> ARRAY[4,4] → true
<@	sadržan je u	ARRAY[3,2] <@ ARRAY[4,7,1,2,3] → true
&&	imaju li polja zajedničke elemente	ARRAY[4,3] && ARRAY[2,1,4] → true ARRAY[4,3] && ARRAY[2,1] → false
||	konkatenacija	ARRAY[1,2] || 3 → {1,2,3} ARRAY[1,2] || ARRAY[3,4] → {1,2,3,4}

Funkcija	Opis	Primjer
array_length (*polje*,*dim*)	broj elemenata u polju (za zadanu dimenziju *dim*)	array_length(ARRAY[1,2,3], 1) → 3
array_lower (*polje,dim*) array_upper (*polje*,*dim*)	donja/gornja granica za zadanu dimenziju polja	array_lower(ARRAY[0,1,8], 1) → 1 array_upper(ARRAY[0,1,8], 1) → 3
array_append (*polje*, *el*) array_prepend (*el*, *polje*)	dodavanje elementa na kraj/početak polja	array_append(ARRAY[1,2], 3) → {1,2,3} array_prepend(3, ARRAY[1,2]) → {3,1,2}
array_remove (*polje*, *el*)	brisanje elemenata vrijednosti *el* iz jednodimenzionalnog polja	array_remove(ARRAY[3,1,3,2],3) → {1,2}

Motivacijski primjer: odgnježđivanje

Za tramvajsku liniju s kraticom „1” potrebno je ispisati redni broj i naziv postaje od završne do polazne. Poredati ih prema redoslijedu postaja (desc).

tramLinija

sifLinija integer	kratLinija character(3)	nazivLinija character varying(120)	tramPostaje integer[ ]
1	1	Zapadni kolodvor – Borongaj	{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}
2	2	Črnomerec - Savišće	{17, 18, 19, 20, 3 , 2, 21, 22, NULL, NULL, NULL}
3	3	Ljubljanica – Savišće	{}

tramPostaja

sifPostaja integer	nazivPostaja varchar(120)

- Tramvajske postaje su pohranjene u kolekciji ***tramPostaje INTEGER[ ]***.
- Naziv postaje treba dobiti spajanjem vrijednosti svakog elementa cjelobrojnog polja s relacijom ***tramPostaja***.
- Sortiranje treba obaviti prema indeksu elemenata polja.

Kako to izvesti u SQL-u?

Motivacijski primjer: odgnježđivanje

Za tramvajsku liniju s kraticom „1” potrebno je ispisati redni broj i naziv postaje od završne do polazne. Poredati ih prema redoslijedu postaja (desc).

U relacijskom modelu kolekcija ***tramPostaje*** bila bi predstavljena relacijom ***postajaLinija***.

tramLinija

sifLinija	kratLinija	nazivLinija

tramPostaja

sifPostaja	nazivPostaja

linijaPostaja

sifLinija	sifPostaja	rbrPostaja

Rješenje je trivijalno:

```
SELECT rbrPostaja, nazivPostaja
  FROM tramPostaja
  JOIN linijaPostaja ON linijaPostaja.sifPostaja = tramPostaja.sifPostaja
  JOIN tramLinija    ON tramLinija.sifLinija     = linijaPostaja.sifLinija
 WHERE kratLinija = '1'
ORDER BY rbrPostaja
```

Ideja: kolekciju transformirati u „privremenu” relaciju i koristiti u SQL upitima na uobičajeni način.

PostgreSQL: odgnježđivanje

Funkcija	Opis	Primjer
unnest (polje) [WITH ORDINALITY] [AS] table_alias [(column_alias [, ...])]]	kreira jednu n-torku za svaki element polja Uz specificiran WITH ORDINALITY, generira se redni broj elementa polja počevši od 1.	`unnest(ARRAY[3,2])` → dvije n-torke: 3 2

tramLinija

sifLinija	kratLinija	nazivLinija	tramPostaje Integer[ ]

tramPostaja

sifPostaja	nazivPostaja

```
SELECT nazivLinija,
       UNNEST(tramPostaje) AS sifPostaja
  FROM tramLinija
WHERE kratLinija = '1'
```

ili

```
SELECT nazivLinija,
       linijaPostaja.sifPostaja
  FROM tramLinija,
  UNNEST(tramPostaje) AS linijaPostaja (sifPostaja)
WHERE kratLinija = '1'
```

nazivLinija	sifPostaja
Zapadni kolodvor – Borongaj	1
Zapadni kolodvor – Borongaj	2
…	…
Zapadni kolodvor – Borongaj	16

PostgreSQL: odgnježđivanje

Za tramvajsku liniju s kraticom „1” potrebno je ispisati redni broj i naziv postaje od završne do polazne. Poredati ih prema redoslijedu postaja (desc).

tramLinija

sifLinija	kratLinija	nazivLinija	tramPostaje Integer[ ]

tramPostaja

sifPostaja	nazivPostaja

```
SELECT linijaPostaja.rbrPostaja,
       tramPostaja.nazivPostaja
  FROM tramLinija,
       UNNEST(tramLinija.tramPostaje) WITH ORDINALITY
           AS linijaPostaja (sifPostaja, rbrPostaja),
       tramPostaja
 WHERE tramPostaja.sifPostaja = linijaPostaja.sifPostaja
   AND kratLinija = '1'
ORDER BY rbrPostaja DESC
```

rbrPostaja	nazivPostaja
16	Borongaj
15	Svetice
14	Harambašićeva
…	…
2	Talovčeva
1	Zapadni kolodvor

Motivacijski primjer: ugnježđivanje

Relaciju ***tramPostaja*** potrebno je proširiti atributom ***tramLinije*** u kojem će se u obliku kolekcije pohraniti nazivi svih linija koje kroz postaju prolaze. Atribut ažurirati u skladu s podacima u relaciji ***tramLinija***.

tramPostaja

sifPostaja	nazivPostaja	tramLinije character varying(120) [ ]
1	Zapadni kolodvor	{"Zapadni kolodvor - Borongaj"}
2	Talovčeva	{"Zapadni kolodvor - Borongaj", "Črnomerec - Savišće"}
…	…	

```
ALTER TABLE tramPostaja ADD tramLinije VARCHAR(120)[];
```

```
UPDATE tramPostaja
   SET tramLinije =
  (SELECT nazivLinija
     FROM tramLinija
    WHERE tramLinija.tramPostaje ... tramPostaja.sifPostaja)
```

3. pretvoriti u polje čiji je svaki element znakovni niz

2. pretvoriti u element cjelobrojnog polja

1. operator „sadrži”

1. Operator „sadrži” je @>

2. Cjelobrojno polje možemo dobiti sa ARRAY [sifPostaja]

```
SELECT ARRAY[sifPostaja]
 FROM tramPostaja
```

array Integer [ ]
{1}
{2}
…

Motivacijski primjer: ugnježđivanje

```
UPDATE tramPostaja
   SET tramLinije =
  (SELECT nazivLinija
     FROM tramLinija
    WHERE tramLinija.tramPostaje ... tramPostaja.sifPostaja)
```

3. pretvoriti u polje znakovnih nizova

3. Polje možemo dobiti sa ARRAY

```
SELECT ARRAY[nazivLinija]
  FROM tramLinija
 WHERE tramPostaje @> ARRAY[3]
```

array bpchar [ ]
{"Zapadni kolodvor - Borongaj"}
{"Črnomerec - Savišće"}

Trebam jednu n-torku, a ne 2 ili više.

array character varying(120) [ ]
{"Zapadni kolodvor - Borongaj", "Črnomerec - Savišće"

```
SELECT tramPostaja.*,
       (SELECT ARRAY[nazivLinija]
          FROM tramLinija
         WHERE tramPostaje @> ARRAY[sifPostaja])
  FROM tramPostaja
 WHERE sifPostaja = 3
```

Correlated subquery

```
ERROR:  more than one row returned by a subquery used as an expression
```

Trebam „nešto” (npr. funkciju) što će više zasebnih elemenata polja transformirati u kolekciju.

SQL standard: ugnježđivanje

- ugnježđivanje *(nesting)* - transformacija neugnježđene relacije u ugniježđenu relaciju
  - grupiranjem - korištenje funkcije COLLECT

tramLinija

sifLinija	kratLinija	nazivLinija

tramPostaja

sifPostaja	nazivPostaja

linijaPostaja

sifLinija	sifPostaja	rbrPostaja

```
SELECT sifLinija, nazivLinija,
       COLLECT(sifPostaja) AS tramPostaje
  FROM tramLinija, linijaPostaja
 WHERE tramLinija.sifLinija = linijaPostaja.sifLinija
GROUP BY sifLinija, nazivLinija
```

  - podupitima u SELECT klauzuli

```
SELECT sifLinija, nazivLinija,
       ARRAY(SELECT sifPostaja
               FROM linijaPostaja
              WHERE tramLinija.sifLinija = linijaPostaja.sifLinija)
             AS tramPostaje
  FROM tramLinija
```

PostgreSQL: ugnježđivanje

Funkcija	Opis
array_agg (*izraz*)	kreira polje iz vrijednosti svake n-torke iz grupe, uključujući NULL vrijednost (vraća NULL vrijednost, ako nije dohvaćena niti jedna n_torka)

```
UPDATE tramPostaja
  SET tramLinije =
 (SELECT array_agg (nazivLinija)
    FROM tramLinija
   WHERE tramLinija.tramPostaje @> ARRAY[sifPostaja])
```

```
SELECT *
  FROM tramPostaja;
```

sif Postaja	nazivPostaja	linije character varying(120) [ ]
1	Zapadni kolodvor	{"Zapadni kolodvor - Borongaj"}
2	Talovčeva	{"Zapadni kolodvor - Borongaj", "Črnomerec - Savišće"}

Rezultat izraza (argumenta funkcije array_agg) ne može npr. biti n-torka. Zbog toga ne mogu pomoću fukcije array_agg stvoriti kolekciju s više od 1 dimenzije.

```
SELECT ARRAY_agg[kratLinija, nazivLinija]
  FROM tramLinija
 WHERE tramPostaje @> ARRAY[3]
```

```
ERROR:  syntax error at or near "," LINE 1: SELECT ARRAY_agg[kratLinija, nazivLinija]
```

Je li moguće da PostgreSQL ima toliko skromnu podršku za ugnježđivanje?
Nije. Elementi kolekcije mogu biti složenog tipa. Što je složeni tip - kasnije.

PostgreSQL: indeksiranje atributa tipa ARRAY

- indeks nad atributima tipa array - koriste se pri obavljanju operacija: &&, @>, @@ ,…
  - GIST indeks (*Generalized Search Tree*) : gist__int_ops - za manje skupove podataka (pretpostavljeno), gist__intbig_ops
  - GIN indeks (*Generalized Inverted Index*)

```
CREATE INDEX postajeIdx ON tramLinija USING GIST (tramPostaje gist__intbig_ops);
```

```
CREATE INDEX postajeIdx ON tramLinija USING GIN (tramPostaje);
```

Primjeri:

- osigurati da vrijednosti ocjena koje su pohranjene u atributu tipa array, mogu poprimiti samo cjelobrojne vrijednosti između 1 i 5

```
CREATE TABLE rezIspit (
   sifNastavnik integer,
   ocjene       integer[] CHECK (ocjene <@ ARRAY[1,2,3,4,5]) );
```

- osigurati da mogu biti upisane najviše 3 ocjene

```
... CHECK (icount(ocjene) <= 3)
```

SQL standard: tipovi podataka

- **unaprijed definirani tipovi (*predefined types*)**
  - atomaran tip - vrijednost nije izgrađena od vrijednosti drugih podatkovnih tipova: integer, float, character, boolean, datetime, interval ...
- **izgrađeni tipovi (*constructed types*)**
  - izgrađeni atomarni tipovi (*constructed atomic types*)
    - referenca (*reference*)
  - izgrađeni kompozitni tipovi (*constructed composite types*)
    - ***kolekcije (collection): polje (array), multiset***
    - ***row***
- **korisnički definirani tipovi (*user-defined types* - UDT)**
  - *distinct type*
  - strukturirani tip (*structured type*)

SQL standard: ROW tip

- sekvenca od jednog ili više elemenata (field)
  - broj elemenata: stupanj (*degree*)
- element je definiran parom (i*me_elementa*, *tip_podatka*)

```
ROW ( postBr     INTEGER,
      nazMjesto  VARCHAR(20)
    )
```

- za pridruživanje vrijednosti elementima koristi se ROW konstruktor
- u ROW konstruktoru može biti navedena lista vrijednosti elemenata, a vrijednost elemenata može biti i rezultat upita
- npr. gornjem ROW tipu vrijednost (10000,'Zagreb') može se pridružiti na sljedeći način:

```
ROW(10000, 'Zagreb')
```

SQL standard: ROW tip

ROW tip se može koristiti za definiranje složenih atributa relacije:

osoba

sifOsoba	ime	prezime	adresa		
			ulica	mjesto	
				postBr	nazMjesto
11001	Hrvoje	Novak	Ilica 25	10000	Zagreb
78936	Ana	Kolar	Marmontova 18	21000	Split

```
CREATE TABLE osoba (
  sifOsoba  INTEGER,
  ime       VARCHAR(25),
  prezime   VARCHAR(25),
  adresa    ROW ( ulica VARCHAR(50),
                  mjesto ROW (postBr    INTEGER,
                              nazMjesto VARCHAR(40))
                )
);
```

PostgreSQL: ROW konstruktor

ROW je u PostgreSQL-u konstruktor za instanciranje složenog tipa (*Composite Type*)

```
ROW (literal, ...)
```

```
ROW ('HR', 'Hrvatska')
```

ili

```
'( val1 , val2 , ... )'
```

```
'("HR", "Hrvatska")'
```

- dozvoljeno izostaviti ROW ako se sastoji od više atributa

http://www.postgresql.org/docs/9.4/static/rowtypes.html

Sintaksa CREATE TABLE naredbe s prethodnog slide-a nije podržana u PostgreSQL-u.

Kao konstruktor, ROW se koristi u radu sa korisnički definiranim tipom - složenim tipom podataka (objašnjeno kasnije).

SQL standard: tipovi podataka

- **unaprijed definirani tipovi (predefined types)**
  - atomaran tip - vrijednost nije izgrađena od vrijednosti drugih podatkovnih tipova: integer, float, character, boolean, datetime, interval ...
- **izgrađeni tipovi (*constructed types*)**
  - izgrađeni atomarni tipovi (*constructed atomic types*)
    - referenca (*reference*)
  - izgrađeni kompozitni tipovi (*constructed composite types*)
    - ***kolekcije (collection): polje (array), multiset***
    - ***row***
- **korisnički definirani tipovi (*user-defined types* - UDT)**
  - *distinct type*
  - strukturirani tip (*structured type*)

SQL standard: Korisnički definirani tipovi

- Korisnički definirani tipovi (*user-defined types* – UDT)
  - definirani i imenovani od strane korisnika
  - nazivaju ih i *apstraktnim tipovima podataka* (*abstract data types*)

(1) **distinct tip**

- temeljen na unaprijed definiranom tipu koji se naziva izvorni tip (*source type*)
- nema nasljeđivanja tipova

(2) **strukturirani tip**

- iskazan kao lista atributa
- podržano nasljeđivanje tipova

SQL standard: DISTINCT tip

- temeljen na atomarnom tipu
- dodjeljuje posebno značenje postojećem atomarnom tipu
  - sprječava miješanje logički nekompatibilnih vrijednosti (npr. usporedbu visine s težinom)
- nad DISTINCT tipovima moguće je definirati metode
- nije moguća hijerarhija (FINAL znači da nije dozvoljeno kreirati podtip tipa)

```
CREATE TYPE visinaT AS DECIMAL (5,2) FINAL;
CREATE TYPE tezinaT AS DECIMAL (5,2) FINAL;
```

```
CREATE TABLE osoba (
  sifOsoba     INTEGER PRIMARY KEY,
  prezime      VARCHAR(25),
  visina       visinaT,
  tezina       tezinaT);
```

- nije dozvoljeno uspoređivanje vrijednosti DISTINCT tipa i atomarnog tipa na kojem je temeljen te različitih DISTINCT tipova temeljenih na istom atomarnom tipu. Potrebno je obaviti konverziju (CAST).

```
SELECT sifOsoba,
       tezina/visina/visina AS BMI,
      'Pretilost'
  FROM osoba
 WHERE tezina > visina*visina*30;
```

```
SELECT sifOsoba,
       CAST(tezina AS DECIMAL (5,2))/
       CAST((visina*visina) AS DECIMAL (5,2)) AS BMI,
      'Pretilost'
  FROM osoba
 WHERE CAST(tezina          AS DECIMAL (5,2)) >
       CAST((visina*visina) AS DECIMAL (5,2)) *
       CAST(30              AS DECIMAL (5,2));
```

ERROR

BMI = tezina/visina$^{2;}$ <20 Pothranjenost; >30 Pretilost

PostgreSQL: DISTINCT

- CREATE DOMAIN - skalarni tip temeljen na ugrađenom tipu podatka - alias za ugrađeni tip podatka uz mogućnost navođenja DEFAULT, NOT NULL i CHECK ograničenja
  - pojednostavljena sintaksa:

```
CREATE DOMAIN domName [AS] dataType
   [ DEFAULT expression ][ NOT NULL ]
   [ CHECK expression ]
```

Primjer:

```
CREATE DOMAIN dCelsius AS DECIMAL(4,1)
   CHECK (VALUE BETWEEN -100 AND 100);

CREATE TABLE euroTemp
(datum   DATE,
 mjesto  CHAR(20),
 temp    dCelsius);

INSERT INTO euroTemp VALUES
('03.01.2015', 'Rovaniemi', -20.0);
```

```
CREATE DOMAIN dFahrenheit AS DECIMAL(4,1)
    CHECK (VALUE BETWEEN -140 AND 210);

CREATE TABLE USATemp
(datum   DATE,
 mjesto  CHAR(20),
 temp    dFahrenheit);

INSERT INTO USATemp VALUES
('03.01.2015', 'Anchorage', -20.0);
```

```
SELECT USAtemp.datum,
       euroTemp.mjesto euro,
       USATemp.mjesto usa
euroTemp.temp tempC, USATemp.temp tempF
  FROM USAtemp, eurotemp
 WHERE USAtemp.temp = eurotemp.temp
   AND USAtemp.datum = eurotemp.datum
```

datum	euro	usa	tempC numeric(4,1)	tempF numeric(4,1)
03.01.2015	Rovaniemi	Anchorage	-20	-20

Nije u skladu sa SQL standardom.
Provodi se implicitna konverzija u ugrađeni tip.
+ rezultat nema smisla!

SQL standard: Strukturirani tipovi

- imenovani, korisnički-definiran podatkovni tip
- proizvoljno složene strukture
- atribut – imenovana komponenta strukturiranog tipa
  - za atribut se navodi ime i podatkovni tip
    - primjer: definicija strukturiranog tipa *mjestoT*

```
CREATE TYPE mjestoT AS (
        postBr     INTEGER,
        nazMjesto  VARCHAR(40))
  NOT FINAL;
```

- **NOT FINAL** - dozvoljeno je kreirati podtip tipa

- strukturirani tipovi se mogu koristiti kao:
  - tip atributa drugog strukturiranog tipa
  - tip parametra funkcija, metoda i procedura
  - tip SQL varijable
  - tip atributa i tip n-torke relacije

SQL standard: Tipizirane tablice

- ***tipizirana tablica*** (*typed table*) - tablica definirana na temelju strukturiranog tipa
  - atributi tipa postaju stupci tablice
  - dodatni stupac sadrži jedinstveni identifikator objekta (*self-referencing column)*
    - identifikator objekta je jedinstven samo unutar tipizirane tablice
- Primjer: kreiranje tablice *mjesto* na temelju tipa *mjestoT*

```
CREATE TYPE mjestoT AS
(  postBr      INTEGER,
   nazMjesto   VARCHAR(40)
 ) NOT FINAL
   REF IS SYSTEM GENERATED;
```

način generiranja vrijednosti reference na objekt (*object reference*) - REF IS klauzula

```
CREATE TABLE mjesto OF mjestoT
(PRIMARY KEY (postBr),
 REF IS mjestoID SYSTEM GENERATED);
```

```
INSERT INTO mjesto VALUES
            (10000, 'Zagreb');
```

mjestoID	postBr	nazMjesto
1023456734	10000	Zagreb

- REF IS klauzula:
  - SYSTEM GENERATED - generiranje obavlja sustav
  - USING - kreiranje vrijednosti reference obavlja korisnik
  - FROM - korisnik specificira listu atributa iz strukturiranog tipa, koja će biti korištena za izvođenje jedinstvene reference na objekt

PostgreSQL: Složeni tip (*Composite Type*)

- struktura koja se sastoji od atributa (elemenata, polja) koji se mogu razlikovati po tipu
  - za atribut se navodi naziv i podatkovni tip; nije moguće definiranje integritetskih ograničenja (npr. NOT NULL, CHECK, DEFAULT)
    - atributi mogu biti bilo kojeg tipa (uključujući druge složene tipove i ARRAY tip)

- kreiranje tipa:
  - naredbom CREATE TYPE

```
CREATE TYPE typeName AS ( atribName atribType [, ... ] ] )
```

npr:

```
CREATE TYPE drzavaT AS (oznDrzava CHAR(2)
                      , nazDrzava VARCHAR(20));
```

  - automatski prilikom kreiranja tablice

```
CREATE TABLE drzava (oznDrzava CHAR(2)
                   , nazDrzava VARCHAR(20));
```

→ kreiran je i tip *drzava*

PostgreSQL: Složeni tip podatka (*Composite Type*)

- kreirani tip moguće je koristiti prilikom kreiranja:
  - drugog tipa (kao tip atributa drugog tipa)

```
CREATE TYPE drzavaT AS
 (oznDrzava    CHAR(2),
  nazDrzava
VARCHAR(20));
```

```
CREATE TYPE mjestoT AS
 (postBroj     INTEGER,
  nazMjesto    VARCHAR(20),
  drzava       drzavaT);
```

  - relacije (kao tip atributa relacije)

```
CREATE TABLE poduzece (naziv CHAR(20),
                       sjediste mjestoT);
```

- Unos n-torke:

```
INSERT INTO poduzece
     VALUES ( 'INA',
              ROW ( 10000, 'Zagreb',
                    ROW ('HR', 'Hrvatska')
                  )
            );
```

Može se izostaviti kada složeni tip ima više od jednog atributa

- pristup atributima složenog tipa - *dot* notacija
  - naziv stupca koji je složenog tipa mora biti naveden u zagradama

```
SELECT poduzece.sjediste,
       (poduzece.sjediste).postBroj,
       (sjediste).drzava,
       (sjediste.drzava).nazDrzava
  FROM poduzece;
```

sjediste mjestoT	postBroj integer	drzava drzavaT	nazDrzav varchar(2
(10000,Zagreb,"(HR,Hrvatska)")	10000	(HR,Hrvatska)	Hrvatsk

Motivacijski primjer – referenciranje iz složenog tipa

mjesto

postBr	nazMjesto
10000	Zagreb
21000	Split

osoba

sifOsoba	ime	prezime	adresa	
			ulicaiKbr	postBr?? mjestoREF??
11001	Hrvoje	Novak	Ilica 25	
78936	Ana	Kolar	Marmontova 18	

- Ne želim u adresi osobe pohranjivati vrijednosti svih atributa iz tablice mjesto. Želim čuvati vrijednost koja će mi omogućiti jednostavno pristupanje atributima tablice mjesto.
- U relacijskom modelu sličan zahtjev je riješen pomoću stranog ključa.
- SQL standard gornji zahtjev rješavaju pomoću tzv. REF tipa

```
CREATE TYPE mjestoT AS
(  postBr      INTEGER,
   nazMjesto   VARCHAR(40)
 ) NOT FINAL;
```

```
CREATE TABLE mjesto OF mjestoT
(PRIMARY KEY (postBr)
 REF IS mjestoID SYSTEM GENERATED)
```

```
CREATE TYPE adresaT AS
(  ulicaiKbr      VARCHAR(40),
   mjestoStan     REF(mjestoT)
) NOT FINAL;
```

veza s objektima tipa *mjestoT* ostvarena je korištenjem REF tipa

```
CREATE TYPE osobaT AS
(  sifOsoba      INTEGER,
   ime           VARCHAR(25)
   prezime       VARCHAR(25)
   adresaStan    adresaT)
 ) NOT FINAL;
```

```
CREATE TABLE osoba OF osobaT
(PRIMARY KEY (sifOsoba)
 REF IS osobaID SYSTEM GENERATED)
```

SQL standard: REF tip

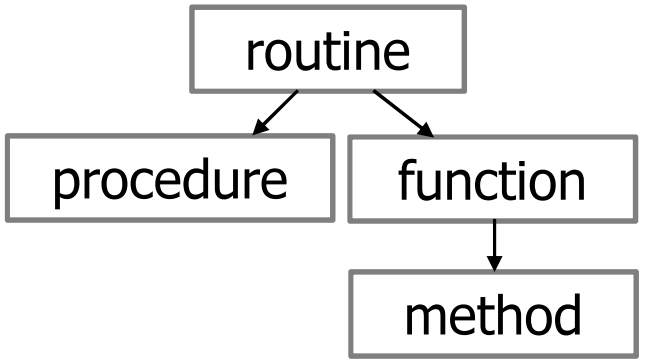
- tip čija vrijednost pokazuje na lokaciju na kojoj je pohranjena vrijednost referenciranog tipa, tj.
  - ako je *T* tip, tada je REF *T* pokazivač na objekt tipa *T*
- može pokazivati samo na n-torke tipizirane tablice
- koristi se za definiranje:
  - atributa u tablici
  - atributa strukturiranog tipa
    - modeliranje povezanosti objekata u tipiziranim tablicama, temeljene na identitetu objekta, umjesto korištenja stranih ključeva
  - SQL varijable, parametra

sifOsoba	ime	prezime	adresa	
			ulicaiKbr	postBr?? mjestoREF??
11001	Hrvoje	Novak	Ilica 25	

REF tip nije implementiran u PostgreSQL-u.
Koristi ga npr. Oracle.

```
INSERT INTO osoba (sifOsoba, ime, prezime, adresaStan)
           VALUES (11001, 'Hrvoje', 'Novak',
                     ROW ('Ilica 25', (SELECT mjestoID
                                         FROM mjesto
                                        WHERE postBr = 21000)
                          )
                  );
```

SQL standard: Pohranjene rutine (1)



- pohranjene rutine *(SQL-invoked routines)* - mogu biti pozvane iz SQL kôda
- SQL razlikuje tri tipa pohranjenih rutina:
  - **Funkcija:**
    - samo ulazni parametri (izlazni je vraćen kao "vrijednost" funkcije)
    - poziva se korištenjem notacije: **`imeFunkcije(parametri)`**
  - **Metoda:**
    - specijalni slučaj funkcije - čvrsto vezana uz jedan strukturirani tip
    - prvi parametar je implicitan: tip parametra je strukturirani tip uz koji je metoda vezana
    - s tipom se manipulira samo kroz metode koje su na njemu definirane - učahurivanje (encapsulation)
  - **Procedura:**
    - ulazni i izlazni parametri
    - pozivanje CALL naredbom

PostgreSQL: Funkcije koje koriste složeni tip

Primjer

```
CREATE TABLE drzava (oznDrzava CHAR(2) PRIMARY KEY
                   , nazDrzava VARCHAR(20));

CREATE TABLE mjesto (postBroj  INTEGER,
                     nazMjesto VARCHAR(20),
                     oznDrzava CHAR(2)
                     REFERENCES drzava(oznDrzava));

INSERT INTo state VALUES ('HR', 'Hrvatska');
INSERT INTo city  VALUES (10000, 'Zagreb', 'HR');
```

```
CREATE OR REPLACE FUNCTION drzavaMjesta(mjesto) RETURNS drzava AS $$
    SELECT * FROM drzava
      WHERE oznDrzava = $1.oznDrzava;
$$ LANGUAGE SQL;
```

```
SELECT m.*, drzavaMjesta(m) FROM mjesto m;

Ili funkcijska notacija:

SELECT m.*, m.drzavaMjesta  FROM mjesto m;
```

postBroj integer	nazMjesto varchar(20)	oznDrzava char(2)	drzavaT drzava
10000	Zagreb	HR	(HR,Hrvatska)

PostgreSQL: Funkcije koje koriste složeni tip

Primjer

```
CREATE TABLE drzava (oznDrzava CHAR(2) PRIMARY KEY
                   , nazDrzava VARCHAR(20));
CREATE TABLE mjesto (postBroj  INTEGER,
                     nazMjesto VARCHAR(20),
                     oznDrzava CHAR(2)
                     REFERENCES drzava(oznDrzava));
```

```
CREATE FUNCTION mjesto(int)
RETURNS mjesto AS $$
    SELECT * FROM mjesto
      WHERE postBroj = $1;
$$ LANGUAGE SQL;
```

```
CREATE FUNCTION mjesto(text)
RETURNS mjesto AS $$
  SELECT * FROM mjesto
   WHERE nazMjesto = $1;
$$ LANGUAGE SQL;
```

```
SELECT * FROM mjesto('Zagreb');
SELECT * FROM mjesto('10000'::int);
```

```
SELECT * FROM mjesto('10000');
```

postBroj integer	nazMjesto varchar(20)	oznDrzava char(2)
10000	Zagreb	HR

postBroj integer	nazMjesto varchar(20)	oznDrzava char(2)

- pretvorba cjelobrojne vrijednosti u vrijednost tipa *mjesto* npr: 10000 u **(10000,Zagreb,HR)**

```
CREATE CAST (int AS mjesto)
  WITH FUNCTION mjesto(int);
```

```
SELECT 10000::mjesto
ili
SELECT CAST(10000 AS mjesto)
```

mjesto Mjesto
(10000, Zagreb, HR)

PostgreSQL: korisnički definiran CAST (za DOMAIN i složeni tip)

°C = (°F -32) / 1.8
°F = °C *1.8 + 32

Primjer

```
CREATE DOMAIN dCelsius
    AS DECIMAL(4,1)
 CHECK (VALUE BETWEEN -100 AND 100);
```

```
CREATE DOMAIN dFahren
    AS DECIMAL(4,1)
 CHECK (VALUE BETWEEN -140 AND 210);
```

Omogućiti pretvorbu iz Celsiusa u Fahrenheite i obratno na razini SUBP => CAST

```
CREATE FUNCTION FToC(dFahren)
    RETURNS dCelsius AS $$
    SELECT CAST(($1-32)/1.8 AS dCelsius);
$$ LANGUAGE SQL;
```

```
CREATE FUNCTION CToF (dCelsius)
  RETURNS dFahren AS $$
  SELECT CAST(($1*1.8 +32) AS dFahren);
$$ LANGUAGE SQL;
```

```
CREATE CAST (dFahren AS dCelsius)
  WITH FUNCTION FToC(dFahren);
```

```
CREATE CAST (dCelsius AS dFahren)
  WITH FUNCTION CToF(dCelsius);
```

```
WARNING:  cast will be ignored because the source data type is a domain
```

```
SELECT FtoC(32), CAST(CAST (32 AS dFahren) AS dCelsius)
```

ftoc numeric(4,1)	dcelsius numeric(4,1)
0	32.0

Zaključak: za DOMAIN tip u PostgreSQL-u korisnički definiran CAST nije moguće postići.

PostgreSQL: CAST za složeni tip

Primjer

```
CREATE TYPE tFahren AS
(temperature DECIMAL(4,1) );
```

°C = (°F -32) / 1.8

°F = °C *1.8 + 32

```
CREATE TYPE tCelsius AS
(temperature DECIMAL(4,1) );
```

Omogućiti pretvorbu iz Celsiusa u Fahrenheite i obratno na razini SUBP => CAST

```
CREATE OR REPLACE FUNCTION FToCType (tFahren)
RETURNS tCelsius AS $$
    SELECT CAST( ROW (($1.temperature-32)/1.8) AS tCelsius);
$$ LANGUAGE SQL;
```

```
CREATE CAST (tFahrenheit AS tCelsius)
  WITH FUNCTION FToCType(tFahren);
```

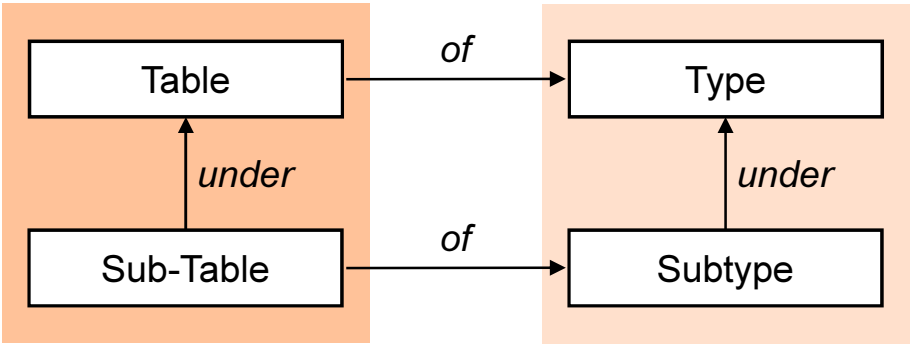
```
SELECT FToCType((ROW(32))::tFahren),
       (ROW (32) :: tFahren) :: tCelsius
```

ftocType tcelsius	row tcelsius
(0.0)	(0.0)

Zaključak: za složeni tip, korisnički definiran CAST JE moguće postići.
Za pretvorbu Celsiusa u Fahrenheite i obratno ovo rješenje nije prikladno jer jednostavan podatak kao što je temperatura nije prikladno pohranjivati u složenom tipu.
Bolji primjer CAST-a za složeni tip može se vidjeti u zadacima za vježbu.

SQL standard: Nasljeđivanje

- nasljeđivanje tipova
  - moguće je samo za strukturirane tipove
  - hijerarhija tipova
- nasljeđivanje tablica
  - moguće je samo za tipizirane tablice
  - hijerarhija tablica
  - odgovara E-R pojmu specijalizacije/generalizacije
  - omogućava više tipova istog objekta, dozvoljavajući istovremeno postojanje entiteta u više od jedne tablice



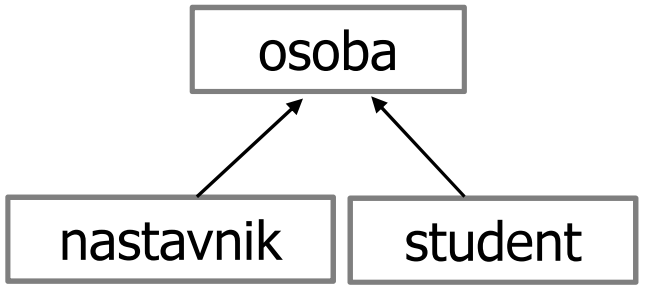
PostgreSQL: Nasljeđivanje (1)

- omogućeno je samo nasljeđivanje (temeljnih) tablica (ne i složenih tipova)
  - nasljeđuju se:
    - osnovne definicije stupca (naziv, tip, NULL ograničenje)
    - pretpostavljene (*default*) vrijednosti stupca
    - CHECK ograničenja (ne mogu biti nadjačana u djeca tablicama)
    - metode tablica
  - ne nasljeđuju se:
    - indeksi
    - UNIQUE, PRIMARY KEY, FOREIGN KEY ograničenja
    - okidači
- dozvoljeno je višestruko nasljeđivanje

PostgreSQL: Nasljeđivanje

Primjer:

```
CREATE TABLE osoba(
    sifra          INTEGER PRIMARY KEY,
    ime            VARCHAR(25),
    prezime        VARCHAR(25)) ;
```




```
CREATE TABLE nastavnik (
    brTekRacun    CHAR(11),
    zaposlenOd    DATE NOT NULL,
    zaposlenDo    DATE
)  INHERITS (osoba);
```

```
CREATE TABLE student (
    JMBAG      CHAR(10),
    datumUpis DATE
) INHERITS (osoba);
```

student i n*astavnik* od *osoba* nasljeđuju atribute *sifra, ime* i *prezime*

```
INSERT INTO osoba     VALUES (1, 'Ana'   , 'Ban');
INSERT INTO student   VALUES (2, 'Mia'   , 'Nel', '0036000001', '01.07.2013');
INSERT INTO student   VALUES (3, 'Ivo'   , 'Puž', '0036000002', '21.07.2014');
INSERT INTO nastavnik VALUES (4, 'Petar', 'Par', '23456187902', '01.05.2001', NULL);
```

```
SELECT * FROM osoba;
```

sifra	ime	prezime
1	Ana	Ban
2	Mia	Nel
3	Ivo	Puž
4	Petar	Par

```
SELECT * FROM ONLY osoba;
```

sifra	ime	prezime
1	Ana	Ban

```
SELECT * FROM nastavnik;
```

sifra	ime	prezime	brTekRavun	datumOd	datumDo
1	Ana	Ban	23456187902	01.05.2001	

PostgreSQL: Nasljeđivanje metoda

```
CREATE OR REPLACE  FUNCTION description (osoba)
RETURNS VARCHAR(250) AS $$
    SELECT ime || ' ' || prezime FROM osoba
      WHERE sifra = $1.sifra;
$$ LANGUAGE SQL;
```

student i n*astavnik* nasljeđuju metodu description od *osoba*

```
SELECT nastavnik, nastavnik.description
  FROM nastavnik;

SELECT student, student.description
  FROM student
```

nastavnik	description
(4,Petar,Par, 23456187902 ,01.05.2001, null)	Petar Par

student	description
(2, Mia, Nel, 0036000001, 01.07.2013)	Mia Nel
(3, Ivo, Puž, 003600002, 12.07.2014)	Ivo Puž

Polimorfizam – redefiniranje metode (funkcije) na razini tablice:

```
CREATE OR REPLACE FUNCTION description (student)
RETURNS VARCHAR(250) AS $$
    SELECT JMBAG|| ', ' || datumUpis FROM student
      WHERE sifra = $1.sifra;
$$ LANGUAGE SQL;
```

```
SELECT student, student.description
  FROM student
```

student	description
(2, Mia, Nel, diplomski, Psihologija)	0036000001, 01.07.2013
(3, Ivo, Puž, diplomski, Medicina)	0036000002, 12.07.2014

PostgreSQL: Nasljeđivanje

Problemi s primarnim ključem (UNIQUE ograničenjem, indeksima)

- podtablica ne nasljeđuje primarni ključ
- podtablica može sadržavati duplikat vrijednosti primarnog ključa iz nadtablice

```
CREATE TABLE osoba(
   sifra          INTEGER PRIMARY KEY,
   ime            VARCHAR(25),
   prezime        VARCHAR(25)) ;
```

```
CREATE TABLE student (
   JMBAG      CHAR(10),
   datumUpis DATE
) INHERITS (osoba);
```

```
INSERT INTO osoba     VALUES (1, 'Ana'    , 'Ban');
INSERT INTO osoba     VALUES (1, 'Tena'   , 'Pale');                                  → ERROR

INSERT INTO student   VALUES (1, 'Mia'   , 'Nel', '0036000001', '01.07.2013'); → OK
INSERT INTO student   VALUES (1, 'Ivo'   , 'Puž', '0036000002', '12.07.2014'); → OK
```

```
ALTER TABLE student ADD CONSTRAINT studentPk PRIMARY KEY (sifra);
```

Je li problem riješen?

U zadacima za vježbu vidi kako se problem može riješiti.

PostgreSQL: Nasljeđivanje

Problemi sa stranim ključem

1. strani ključevi se ne nasljeđuju

```
CREATE TABLE skola(
   sifSkola    INTEGER PRIMARY KEY,
   nazivSkola VARCHAR(250));
```

```
CREATE TABLE osoba(
   sifra         INTEGER PRIMARY KEY,
   ime           VARCHAR(25),
   prezime       VARCHAR(25),
   sifSkola      INTEGER REFERENCES
                  skola(sifSkola));

CREATE TABLE student (
   JMBAG      CHAR(10),
   datumUpis DATE
) INHERITS (osoba);
```

```
INSERT INTO skola VALUES (1, 'Gimnazija');

INSERT INTO osoba     VALUES (1, 'Ana'    , 'Ban', 2);                                  → ERROR
INSERT INTO student   VALUES (2, 'Mia'    , 'Nel', 2, '0036000001', '01.07.2013');→ OK
```

Rješenje:

```
ALTER TABLE student ADD CONSTRAINT studentSkolaFk
                    FOREIGN KEY (sifSkola) REFERENCES skola(sifSkola);
```

Prethodno treba obrisati n-torku koja ne zadovoljava referencijski integritet.

PostgreSQL: Nasljeđivanje

Problemi sa stranim ključem - nastavak

2. nemogućnost stvaranja stranih ključeva koji se referenciraju na sve n-torke iz hijerarhije (iz nadtablice i svih njezinih podtablica)
   - u obzir se uzimaju samo n-torke kojima je tablica u kojoj je definiran strani ključ najspecifičnija tablica (*most-specific table*)

```
CREATE TABLE osoba(
   sifra        INTEGER PRIMARY KEY,
   ime          VARCHAR(25),
   prezime      VARCHAR(25));
```

```
CREATE TABLE student (
   JMBAG      CHAR(10),
   datumUpis DATE
) INHERITS (osoba);
```

```
CREATE TABLE ulazakUDvoranu(
   sifra        INTEGER REFERENCES osoba(sifra),
   dvorana      VARCHAR(10),
   trenutak     TIMESTAMP);
```

```
INSERT INTO osoba   VALUES (1, 'Ana'   , 'Ban');
INSERT INTO student VALUES (2, 'Mia'   , 'Nel', '0036000001', '01.07.2013');

INSERT INTO ulazakUDvoranu VALUES (1, 'B5', '2016-10-22 10:50');   → OK
INSERT INTO ulazakUDvoranu VALUES (2, 'B5', '2016-10-22 10:50');   → ERROR
```

ERROR: insert or update on table "roomattendance" violates foreign key constraint roomattendance_personid_fkey
DETAIL: Key (personid)=(2) is not present in table "person".

PostgreSQL: OID (*Object identifier*) (1)

- jedinstveni identifikator objekta; skriveni stupac
- nije ga moguće promijeniti (npr. UPDATE naredbom)
- PostgreSQL ga koristi kao primarne ključeve u sistemskim tablicama
- u korisničkim tablicama postoji:
  - ako je tablica kreirana s parametrom: `WITH (OIDS=TRUE)`, ili
  - postavljena konfiguracijska varijabla (*default_with_oids*)

```
CREATE TABLE predmet (sifPred    INTEGER PRIMARY KEY,
                      nazPred    VARCHAR(250))
                      WITH (OIDS=TRUE);
INSERT INTO predmet VALUES (1, 'Baze podataka');
```

```
SELECT * FROM predmet;
```

sifPred integer	nazPred varchar(250)
1	Baze podataka

oid je skriveni stupac

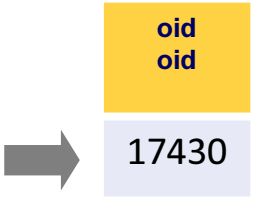
```
SELECT oid, * FROM predmet;
```

oid oid	sifPred integer	nazPred varchar(250)
17376	1	Baze podataka

PostgreSQL: OID (*Object identifier*) (2)

- implementiran kao unsigned integer (4 byte)
  - nedovoljno za pružanje jedinstvenosti na razini baze podataka
  - kada dosegne najveću vrijednost ponovo započinje od 1 - mogući duplikati
- OID kao primarni i/ili strani ključ relacije:

```
CREATE TABLE predmet (nazPred VARCHAR(250))
                      WITH (OIDS=TRUE);
ALTER TABLE predmet ADD PRIMARY KEY (oid);
INSERT INTO predmet VALUES ('Baze podataka') RETURNING oid;
```




```
-- upisan predmet
CREATE TABLE upPredmet (oidPred INT REFERENCES predmet(oid),
                        sifStud INT REFERENCES student(sifOsoba),
                        PRIMARY KEY (oidPred, sifStud);
INSERT INTO upPredmet VALUES (17430, 100);
-- ili
INSERT INTO upPredmet VALUES ((SELECT oid FROM predmet
                               WHERE nazPred = 'Baze podataka') -- ???
                              , 100);
```

ORDBMS prednosti i nedostaci

- prednosti
  - zadržane sve mogućnosti relacijskih baza podataka
    - proširenim relacijskim pristupom očuvana znanja i iskustva uložena u razvoj aplikacija temeljenih na relacijskom modelu
  - mogućnost ponovnog korištenja i dijeljenja funkcionalnosti
    - proširenjem poslužitelja SUBP-a funkcionalnost dostupna svima

- nedostaci
  - složenost
  - nezadovoljstvo pobornika relacijskog modela
    - izgubljena osnovna jednostavnost i čistoća relacijskog modela
    - performanse lošije u odnosu na trenutnu relacijsku tehnologiju
  - nezadovoljstvo pobornika objektno-orijentiranog modela
    - nezadovoljstvo korištenom terminologijom i pristupom objektnim konceptima

Literatura

- J. A. Hoffer, R. Venkataraman, H.Topi: **Modern Database Management (11th Edition)**, Prentice Hall, 2013
  - Chapter 13: Overview: Object-Oriented Data Modeling
    http://wps.prenhall.com/wps/media/objects/14735/15089538/M13_HOFF2253_11_SE_C13WEB.pdf
- Pearson Education: **Advanced Database Topics**: **Object-Relational Databases**
  - http://wps.pearsoned.co.uk/wps/media/objects/10977/11240737/Web%20chapters/Chapter%2014_WEB.pdf
- PostgreSQL 9.4 Documentation - http://www.postgresql.org/docs/9.4/static/index.html
- S.W. Dietrich, S.D. Urban: **An Advanced Course in Database Systems : Beyond Relational Databases**, Prentice Hall, 2005