Napredni modeli i baze podataka

Predavanja
Listopad 2014

3. Objektno orijentirane i objektno-relacijske baze podataka

Pregled

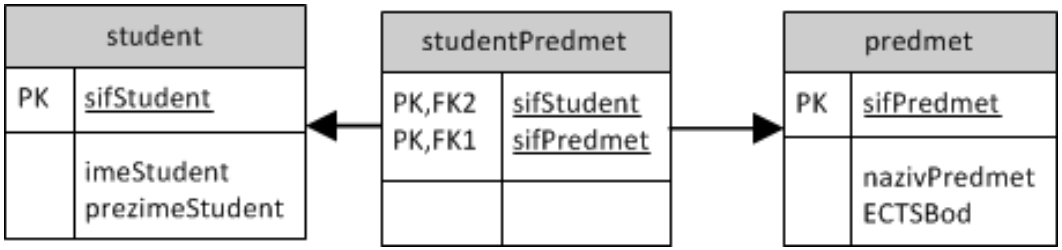
- Objektno orijentirane baze podataka
  - Načela objektno orijentiranih baza podataka
  - Objektno orijentirani sustavi za upravljanje bazama podataka
  - ODMG standard
- Objektno-relacijske baze podataka
  - Objektno-relacijski model podataka
  - Objektno-relacijske mogućnosti prema SQL standardu
  - Objektno-relacijska proširenja u PostgreSQL SUBP-u

Zašto objektno orijentirane baze podataka? (1)

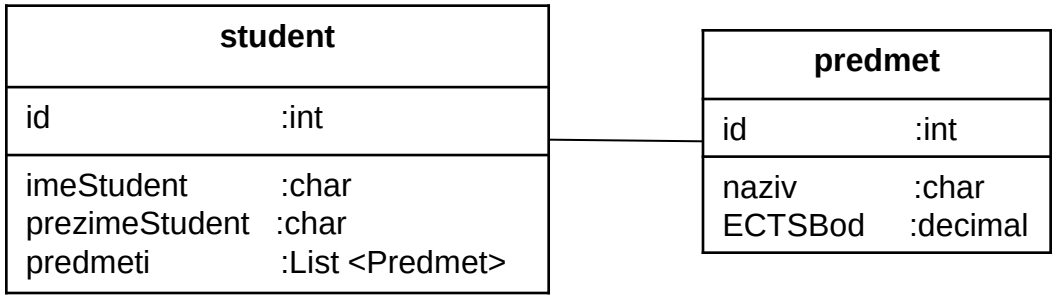
- Relacijske baze podataka nisu prikladne za aplikacije koje koriste složene tipove podataka ili nove tipove podataka za velike nestrukturirane objekte (nestrukturirani tekst, slike, multimedija, GIS objekti,…)
- Model relacijskih baza podataka bitno se razlikuje od objektnog modela aplikacija realiziranih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

ER model

Relacijski model



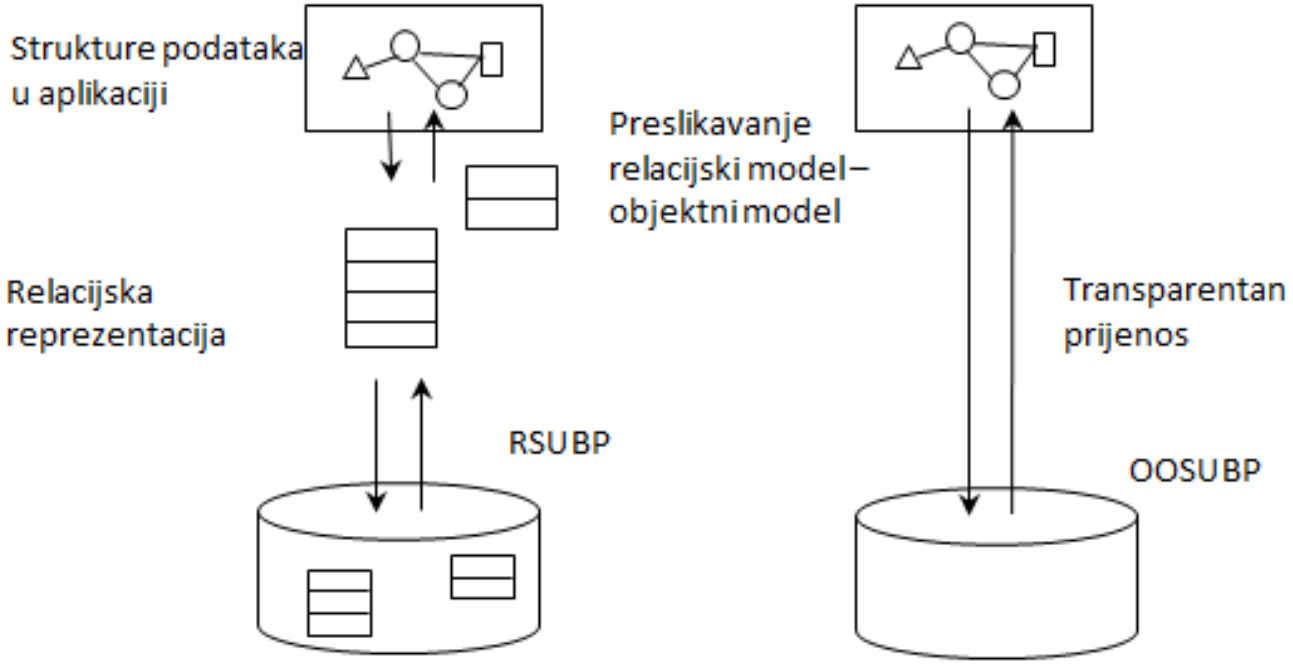
Objektni model



•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()`
    - 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 *OOSUBP 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

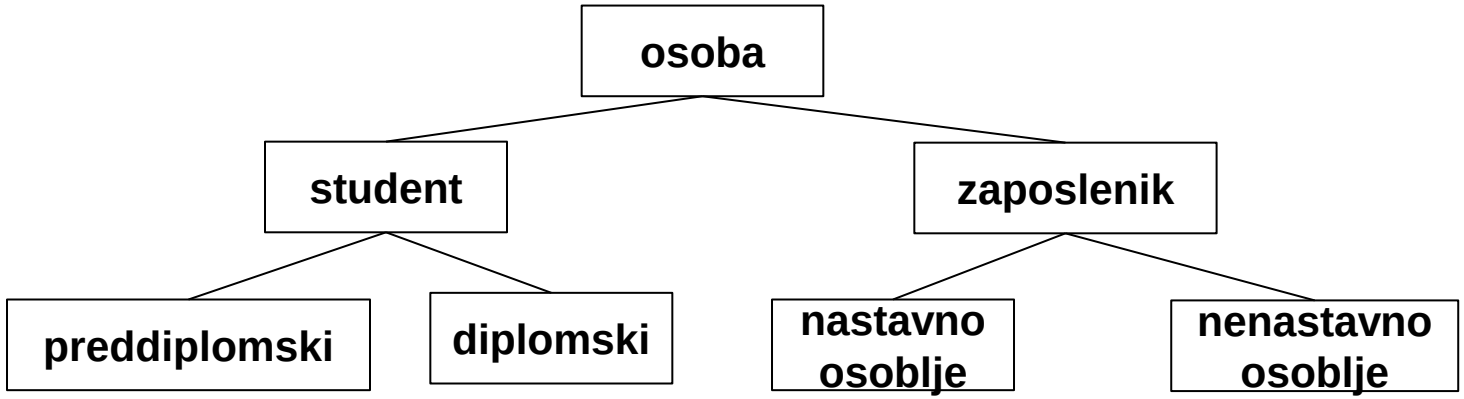
- Koncepti objektno-orijentiranog sustava
  - Razredi (klase)
  - Složeni objekti
  - Identitet objekta
  - Hijerarhije klasa
  - Učahurivanje (*encapsulation*)
  - Nadjačavanje (*overriding*), preopterećivanje (*overloading*) i kasno vezivanje (*late binding*)
- Koncepti sustava za upravljanje bazama 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 float 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*)**

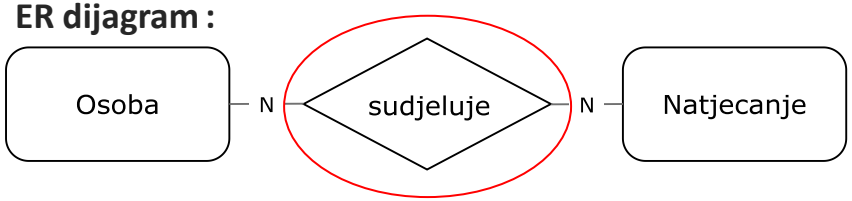
Veze između objekata

- Ostvaruju se **referenciranjem**
- 1:1 veza
  - `kupac.getKosarica()`
  - `kosarica.getKupac()`
- N:1 (1:N) veza
  - `kupac.getMjestoStanovanja()`
  - `mjesto.getKupci()`
- N:N veza
  - `kupac.getArtikli()`
  - `artikl.getKupci()`
- Sve veze mogu biti **dvosmjerne**

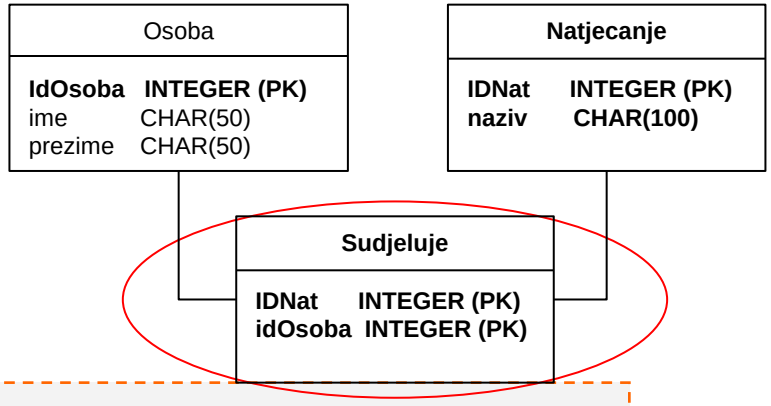
- Dvosmjernost nije nužno izraziti u objektnom modelu, ako nije važna za poslovni proces aplikacije

Veze između objekata (2)

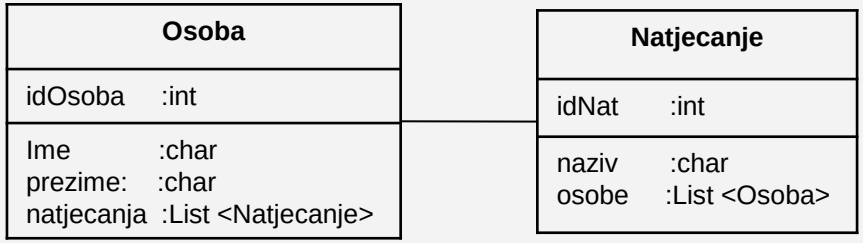
ER dijagram :



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

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

Primjer 2: Dohvati nazive i cijenu jela, ali preko objekta *restaurant:*

```
    SELECT s.dish.name, s.price
       FROM Restaurants r, r.dishesSold s
      WHERE r.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
- Nema potrebe za primarnim ključem (?)
  - Identifikacija objekata je skrivena od korisnika
- Koristi se samo jedan programski jezik (za aplikaciju i za pristup bazi)
- Nema potrebe za posebnim upitnim jezikom

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 s 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

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
  - prototip objektno-relacijskih sustava: Postgres (PostgreSQL)
  - primjer komercijalnih sustava: Oracle, IBM Informix

DBMS Matrix

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

Ugnježdene relacije (1)

Primjer: informacijski sustav studentske službe

- svaki je predmet opisan:
  - šifrom
  - nazivom
  - skupom izvođača
  - zavodom (katedrom)

predmet

sifPred	nazPred	izvodjaci	zavod(kratZavod, nazZavod)
1	NMBP	{Horvat, Hlapić}	(ZPR, Zavod za …)
2	SBP	{Horvat}	(ZPR, Zavod za …)

  - domene atributa *nositelji* i *zavod* **ne sadrže** jednostavne (nedjeljive) vrijednosti $\Rightarrow$ relacija *predmet* ne zadovoljava 1NF

- 1NF verzija relacije *predmet:*

PREDMET1NF = { sifPred, nazPred, izvodjac, kratZavod, nazZavod }

$K_{PREDMET1NF}$ = { sifPred, izvodjac}

predmet1NF

sifPred	nazPred	izvodjac	kratZavod	nazZavod
1	NMBP	Horvat	ZPR	Zavod za …
1	NMBP	Hlapić	ZPR	Zavod za …
1	SBP	Horvat	ZPR	Zavod za …

  - izgubljeno jedan-na-jedan podudaranje između n-torke i predmeta
  - redundancija

Ugnježdene relacije (2)

- nakon normalizacije dekompozicijom nastaju relacijske sheme:

PREDMET = { sifPred, nazPred, kratZavod }
IZVODJAC = { sifPred, prezOsoba }
ZAVOD = { kratZavod, nazZavod }
OSOBA = { prezOsoba}

$K_{PREDMET}$ = { sifPred }
$K_{IZVODJAC}$ = { sifPred, prezOsoba }
K_{ZAVOD} = { kratZavod }
K_{OSOBA} = { prezOsoba }

predmetN(PREDMET)

sifPred	nazPred	kratZavod
1	NMBP	ZPR
1	SBP	ZPR

zavod(ZAVOD)

kratZavod	nazZavod
ZPR	Zavod za …

osoba (OSOBA)

prezOsoba
Horvat
Hlapić

izvodjac(IZVODJAC)

sifPred	prezOsoba
1	Horvat
1	Hlapić
2	Horvat

- za dobivanje informacija o predmetu potrebno je obaviti spajanje relacija

Objektno-relacijski model podataka (1)

- Temeljen na relacijskom modelu podataka
  - sačuvane su relacijske karakteristike, deklarativan pristup podacima
  - 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 ugnježdene relacije (narušena 1NF)
  - znatno uvećane mogućnosti modeliranja podataka čime je proširen opseg primjene
- Ne postoji jedinstveni model - modeli se razlikuju 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
  - ugnježđene relacije (složeni atributi, kolekcije)

- Objektno-relacijski koncepti nisu potpuno implementirani niti u jednom SUBP-u
  - različiti SUBP-ovi koriste različite pristupe
  - Neke mogućnosti ugrađene u sve glavne komercijalne SUBP

SQL standard: objektno-relacijska proširenja

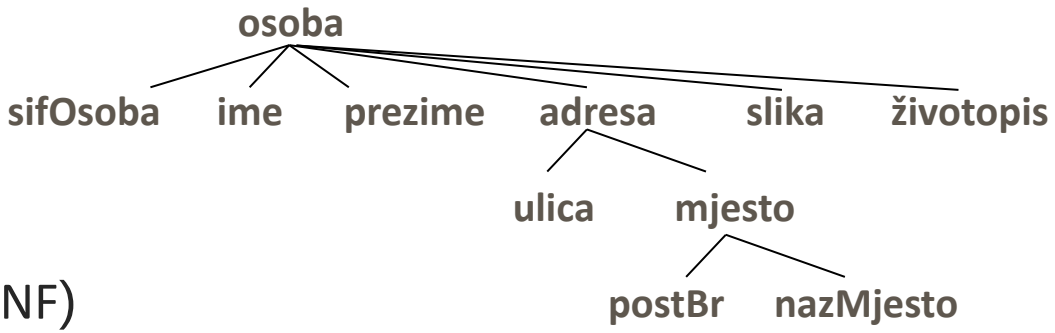
- SQL:1999 uključuje većinu objektno-relacijskih koncepata
  - konstruktori tipova za specificiranje složenih objekata, identitet objekta, učahurivanje, nasljeđivanje
- podjela tipova podataka:
  - **unaprijed definirani tipovi (*predefined types*)**
    - atomaran tip - vrijednost nije izgrađena od 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*)

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)

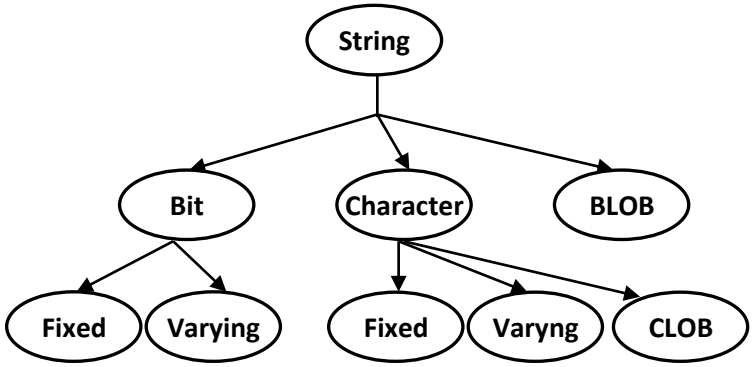
predmet

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*

LOB tip (*Large Object Type*) (1)

- omogućava pohranu velikih vrijednosti (fotografije osoba, medicinske slike u visokoj rezoluciji, video zapisi)
- unaprijed definirani tip (*String*)
  - ***Character Large Object* (CLOB), *National Character Large Object* (NCLOB)**
    - sadrži tekst (*printable characters*, *tabs*, *newlines*, *newpages*)
    - dozvoljene su neke operacije nad znakovnim nizovima (npr. konkatenacija (||), funkcije SUBSTRING, UPPER, TRIM ...), usporedba (LIKE, =, <>)
  - ***Binary Large Object* (BLOB)**
    - bilo kakav niz binarnih podataka



LOB tip (*Large Object Type*) (2)

```
CREATE TABLE osoba (
        sifOsoba  INTEGER,
        ime       VARCHAR(25),
        prezime   VARCHAR(25),
        zivotopis CLOB (50K),
        slika     BLOB (2M)
);
```

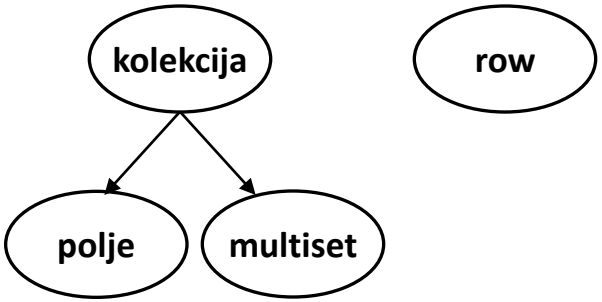
- pristup podatku tipa LOB
  – pomoću *lokatora*

```
EXEC SQL BEGIN DECLARE SECTION:
   SQL TYPE IS BLOB_LOCATOR
       slika_loc;
EXEC SQL END DECLARE SECTION;

EXEC SQL
   SELECT slika
     INTO :slika_loc
     FROM osoba
    WHERE prezime = 'Horvat';

INSERT INTO osoba
   VALUES (..., :slika_loc);
```

- ograničeno korištenje
  - ne može biti dio ORDER BY, GROUP BY klauzule, dio UNIQUE ograničenja, dio stranog ključa ...

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: ROW tip (1)

- izgrađeni kompozitni 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)
    )
```

- može biti korišten za definiranje složenih atributa relacije:

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

SQL standard: ROW tip (2)

- vrijednost ROW tipa sadrži po jednu vrijednost za svaki element tog ROW tipa
- vrijednost elementa mora odgovarati definiranom podatkovnom tipu tog elementa
- 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. ROW tipu:

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

može se pridružiti vrijednost (10000,'Zagreb') pomoću:

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

SQL standard: ROW tip (3)

- upisivanje zapisa u relaciju *osoba*:

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

```
INSERT INTO osoba VALUES(
         34562,
         'Hrvoje',
         'Novak',
          ROW('Ilica 25',
                ROW (10000,'Zagreb')
             )
);
```

- za pristup elementima koristi se *dot* notacija:

```
SELECT o.adresa.mjesto.postBr
  FROM osoba o
 WHERE o.adresa.ulica LIKE 'Ilica%';
```

SQL standard: Kolekcija

- izgrađeni kompozitni tip
- kolekcije vrijednosti homogenog podatkovnog tipa
- struktura podataka kao što je:
  - 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 (1)

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

```
CREATE TABLE predmet (
 sifPred    INTEGER,
 nazPred    VARCHAR(250),
 nositelj   INTEGER REFERENCES nastavnik(sifOsoba),
 izvodjaci  INTEGER ARRAY[10] REFERENCES nastavnik(sifOsoba),
 polaznici  INTEGER ARRAY[1000] REFERENCES student(sifOsoba),
 zavod      ROW(kratZavod CHAR(8), nazZavod VARCHAR(50)),
 PRIMARY KEY (sifPred)
)
```

  - korištenjem polja za pohranu izvođača poznat je i poredak izvođača

- dva polja usporedivih tipova smatraju se identičnim akko su jednake kardinalnosti i na istoj poziciji imaju elemente jednakih vrijednosti

SQL standard: ARRAY (2)

- upisivanje n-torke s atributom tipa ARRAY:

```
INSERT INTO predmet VALUES(1, 'Napredni modeli i baze podataka',
                           123,                    /*nositelj*/
                           ARRAY[123,111,345,24], /*izvođači*/
                           ARRAY[],                /*polaznici*/
                           ROW('ZPR','Zavod za prim.računarstvo')
);
```

```
INSERT INTO predmet (…, ARRAY(SELECT sifOsoba FROM nastavnik WHERE …),…);
```

- postavljanje vrijednosti elemenata polja:

```
UPDATE predmet
   SET polaznici = ARRAY[13503, 14111, 9000]
WHERE sifPred = 1;
```

- dodjeljivanje vrijednosti elementa na pojedinoj poziciji u polju:

```
UPDATE predmet SET polaznici[4] = 14678
WHERE sifPred = 1;
```

SQL standard: ARRAY (3)

- funkcija CARDINALITY - vraća trenutni broj elemenata u polju

```
SELECT CARDINALITY(izvodjaci) AS brojIzvodjaca
  FROM predmet
```

brojIzvodjaca
4

```
UPDATE predmet
   SET izvodjaci[6] = 555
WHERE sifPred = 1;

SELECT CARDINALITY(izvodjaci) AS brojIzvodjaca
  FROM predmet
 WHERE sifPred = 1;

SELECT predmet.izvodjaci[5] AS petiElement
  FROM predmet
WHERE sifPred = 1;
```

brojIzvodjaca
6

petiElement
NULL

- funkcija UNNEST - obavlja konverziju kolekcije u tablicu
  - kreira se jedna n-torka za svaki element kolekcije

SQL standard: ARRAY (4)

- Primjer: UNNEST u FROM dijelu SELECT naredbe:

p.izvodjaci
ARRAY[123,111,345,24]

```
SELECT i.sifIzvodjac
  FROM predmet AS p,
       UNNEST(p.izvodjaci) AS i(sifIzvodjac)
 WHERE p.sifPred = 1
```

sifIzvodjac
123
111
345
24

```
SELECT p.sifPred, i.sifIzvodjac
  FROM predmet AS p,
       UNNEST(p.izvodjaci) AS i(sifIzvodjac)
```

→ rezultat upita su svi parovi (*sifPred*, *sifIzvodjac*)

```
SELECT p.sifPred,
       i.sifIzvodjac, i.pozicija
  FROM predmet AS p,
       UNNEST(p.izvodjaci) WITH ORDINALITY
           AS i(sifIzvodjac, pozicija)
```

→ prethodni upit uz dohvat pozicije izvođača u polju

- Primjer: UNNEST na mjestu podupita:

```
SELECT nazPred
  FROM predmet
 WHERE 123 IN (UNNEST(izvodjaci))
```

SQL standard: MULTISET (1)

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

```
CREATE TABLE predmet (
 sifPred    INTEGER
 nazPred    VARCHAR(250),
 nositelj   INTEGER REFERENCES nastavnik(sifOsoba),
 izvodjaci  INTEGER MULTISET REFERENCES nastavnik(sifOsoba),
 polaznici  INTEGER MULTISET REFERENCES student(sifOsoba),
 zavod      ROW(kratZavod CHAR(8), nazZavod VARCHAR(50)),
 PRIMARY KEY (sifPred)
)
```

  - korištenjem multiseta za pohranu polaznika, 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*.

SQL standard: MULTISET (2)

- upisivanje n-torke s atributom tipa MULTISET:

```
INSERT INTO predmet VALUES(1, 'Napredni modeli i baze podataka',
                           123,                              /*nositelj*/
                           MULTISET[123,111,345,24],         /*izvođači*/
                           MULTISET[13503, 14111, 9000],     /*polaznici*/
                           ROW('ZPR','Zavod za prim.računarstvo')
);
```

```
INSERT INTO predmet VALUES(1, …
                           MULTISET(SELECT sifOsoba FROM nastavnik
                                    WHERE …), …);
```

- brisanje zadnjeg elementa atributa u tablici rezultira praznim skupom (nije isto što i NULL vrijednost atributa)

- neke dodatne funkcije za rad nad multisetom:
  - SET - uklanja duplikate iz multiseta
  - CARDINALITY – vraća trenutni broj elemenata

SQL standard: MULTISET (3)

- predikati za rad s multisetom:
  - predikat usporedbe (samo jednakost i nejednakost)
  - MEMBER - ispituje postoji li član multiseta s navedenom vrijednosti
    - *value1* [NOT] MEMBER [OF] *multiset_value2*
  - SUBMULTISET – ispituje je li jedan multiset podskup drugog
    - *multiset_value1* [NOT] SUBMULTISET [OF] *multiset_value2*
  - IS [NOT] A SET – provjerava postoje li u multisetu duplikati
    - *multiset_value* IS [NOT] A SET

- dodatne operacije:
  - MULTISET UNION [ALL | DISTINCT] – unija dva multiseta; uz zadržavanje ili izbacivanje duplikata
  - MULTISET INTERSECT [ALL | DISTINCT] – presjek dva multiseta
  - MULTISET EXCEPT [ALL | DISTINCT] – razlika dva multiseta

SQL standard: MULTISET (4)

tabM

id	a	b	c
1	MULTISET [1,1,1,2,2,3,3]	MULTISET [1,1,2,4,4]	MULTISET [5]

```
SELECT SET (a) AS rez FROM tabM
```

rez
MULTISET [1,2,3]

```
ELEMENT (c)
```
→ 5

```
a MULTISET UNION ALL b
```
→ MULTISET [1,1,1,1,1,2,2,2,3,3,4,4]

```
a MULTISET UNION DISTINCT b
```
→ MULTISET [1,2,3,4]

```
a MULTISET INTERSECT ALL b
```
→ MULTISET [1,1,2]

```
a MULTISET INTERSECT DISTINCT b
```
→ MULTISET [1,2]

```
a MULTISET EXCEPT ALL b
```
→ MULTISET [1,2,3,3]

```
a MULTISET EXCEPT DISTINCT b
```
→ MULTISET [1,2,3]

```
b MULTISET EXCEPT ALL a
```
→ MULTISET [4,4]

SQL standard: MULTISET (5)

- agregatne funkcije nad multisetovima:
  - COLLECT – kreira multiset iz vrijednosti svake n-torke iz grupe
  - FUSION – kreira multiset koji se sastoji od svih elemenata multiseta iz svih n-torki iz grupe
  - INTERSECTION – kreira multiset koji se sastoji od presjeka elemenata multiseta iz svih n-torki iz grupe

$\pi_{\text{sifPred, izvodjaci}}(predmet)$

sifPred	izvodjaci
2	MULTISET [145,245]
3	MULTISET [204,145,265]
4	MULTISET [284,145]
5	NULL

```
SELECT COLLECT(sifPred) AS predmeti,
       FUSION(izvodjaci) AS unija,
       INTERSECTION(izvodjaci) AS presjek
  FROM predmet
```

predmeti	unija	presjek
MULTISET [2,3,4,5]	MULTISET [145,145,145,245,204,265,284]	MULTISET [145]

SQL standard: Deugnježđivanje

- deugnježđivanje *(unnesting)* - transformacija ugniježđene relacije u oblik s manje (ili bez) složenih atributa

- primjer: konverzija multiseta u tablicu

```
SELECT SUM (t.c)
  FROM UNNEST (MULTISET [2, 3, 5, 7]) AS t(c)
```

➡ 17

- primjer: deugnježđivanje relacije *predmet*

```
SELECT sifPred, nazPred,
       i.sifIzvodjac
  FROM predmet AS p,
       UNNEST(p.izvodjaci) AS i(sifIzvodjac)
```

➡ predmet1NF

SQL standard: Ugnježđivanje

- ugnježđivanje *(nesting)* - transformacija neugniježđene 1NF relacije u ugniježđenu relaciju
  - grupiranjem - korištenje funkcije COLLECT

```
SELECT sifPred, nazPred,
       COLLECT(izvodjac) AS izvodjaci
  FROM predmet1NF
GROUP BY sifPred, nazPred
```

PREDMET1NF = { sifPred, nazPred, izvodjac, kratZavod, nazZavod }

  - podupitima u SELECT klauzuli

```
SELECT sifPred, nazPred,
       ARRAY(SELECT prezOsoba FROM izvodjac AS a
             WHERE a.sifPred = k.sifPred) AS izvodjaci
  FROM predmetN AS k
```

```
ili
MULTISET
```

IZVODJAC = { sifPred, prezOsoba }

PostgreSQL: ARRAY (1)

- omogućeno je korištenje i višedimenzionalnih polja
- u definiciji tipa atributa se ne navodi veličina polja
  - ako je veličina polja navedena, sustav ne dojavljuje pogrešku, već ignorira navedenu veličinu

```
CREATE TABLE predmet (
 sifPred    INTEGER,
 nazPred    VARCHAR(250),
 nositelj   INTEGER REFERENCES nastavnik(sifOsoba),
 izvodjaci  INTEGER[],              /* ili INTEGER ARRAY [10] */
 polaznici  VARCHAR(25)[][],
 PRIMARY KEY (sifPred)
);
```

```
 …
 izvodjaci INTEGER[] REFERENCES nastavnik(sifOsoba)
 …
```

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

PostgreSQL: ARRAY (2)

- upisivanje n-torke s atributom tipa ARRAY:

```
INSERT INTO predmet VALUES(1, 'Napredni modeli i baze podataka', 123,
     '{123,111,345,24}',
     '{{"Ivan", "Novak"}, {"Ana", "Horvat"}, {"Zrinka", "Kolar"}}');
```

'{ *val1 delim val2 delim ...* }'

ili :

```
INSERT INTO predmet VALUES
    (1, 'Napredni modeli i baze podataka', 123,
     ARRAY[123,111,345,24],
     ARRAY[['Ivan', 'Novak'], ['Ana', 'Horvat'], ['Zrinka', 'Kolar']]);
```

```
SELECT sifPred, izvodjaci, polaznici
  FROM predmet;
```

sifPred	izvodjaci	polaznici
1	{123,111,345,24}	{{Ivan,Novak},{Ana,Horvat},{Zrinka,Kolar}}

- indeksi elemenata polja **izvodjaci** $\in [1, 4]$

- upisivanje polja bez elemenata:

```
INSERT INTO predmet VALUES(1, 'Napredni modeli i baze podataka', 123,
     '{}', '{}');   /* ili ARRAY[]::integer[] */
```

```
SELECT sifPred, izvodjaci, polaznici FROM predmet;
```

sifPred	izvodjaci	polaznici
1	{}	{}

PostgreSQL: ARRAY (3)

- upisivanje n-torke s atributom tipa ARRAY - nastavak

```
INSERT INTO predmet
    VALUES(1, 'Napredni modeli i baze podataka', 123,
           '{123,NULL,345,24}',      /* ili ARRAY[123,NULL,345,24] */
           '{{"Ivan", "Novak"},      /* ili ARRAY[['Ivan', 'Novak'], */
             {NULL, NULL},           /*           [NULL, NULL'],     */
             {"Zrinka", "Kolar"}}'   /*           [Zrinka', Kolar']] */);
```

```
SELECT sifPred, izvodjaci, polaznici
  FROM predmet;
```

sifPred	izvodjaci	polaznici
1	{123,NULL,345,24}	{{Ivan,Novak},{NULL,NULL},{Zrinka,Kolar}}

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

```
INSERT INTO predmet
    VALUES(11, 'Baze podataka', 10,
           ARRAY (SELECT sifOsoba FROM nastavnik WHERE sifOsoba < 5),
           NULL);
```

PostgreSQL: ARRAY (4)

- pristupanje elementu polja - korištenjem indeksa elementa

```
SELECT sifPred, izvodjaci, izvodjaci[2] AS drugi, izvodjaci[3] AS treci
  FROM predmet
 WHERE izvodjaci[1] <> 100;
```

sifPred	izvodjaci	drugi	treci
1	{123,NULL,345,24}	NULL	345

- pristupanje elementima čiji su indeksi unutar navedenih granica

```
  SELECT izvodjaci[1:2] FROM predmet
   WHERE izvodjaci[3:4] = ARRAY [345,24]
```

sifPred	izvodjaci
1	{123,NULL}

```
SELECT sifPred, polaznici, polaznici[1][2] AS p1,
       polaznici[1:2][1] AS p2, polaznici[1:2][2:2] AS p3,
       polaznici[1:2][2] AS p4
  FROM predmet
```

→ `polaznici[1:2][1:2]`

sifPred	polaznici	p1	p2	p3	p4
1	{{Ivan,Novak}, {Ana,Horvat}, {Zrinka,Kolar}}	Novak	{{Ivan}, {Ana}}	{{Novak}, {Horvat}}	{{Ivan,Novak}, {Ana,Horvat}}

PostgreSQL: ARRAY (5)

- postavljanje vrijednosti elemenata polja:

```
UPDATE predmet SET izvodjaci = ARRAY[123,10,333,24]
WHERE sifPred = 1;
```

  - indeks elementa polja **izvodjaci** ∈ [1, 4]

- dodjeljivanje vrijednosti elementa na pojedinoj poziciji u polju:

```
UPDATE predmet SET izvodjaci[2] = 222
 WHERE sifPred = 1;
```

```
UPDATE predmet SET izvodjaci[-2] = 587
 WHERE sifPred = 1;
```

  - indeks elementa polja **izvodjaci** ∈ [-2, 4]

- dodjeljivanje vrijednosti uzastopnim elementima:

```
UPDATE predmet SET izvodjaci[2:3] = ARRAY[123,10]
 WHERE sifPred = 1;
```

PostgreSQL: ARRAY (6)

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`
&&	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}

PostgreSQL: ARRAY (7)

Funkcija	Opis	Primjer
unnest (*polje*)	kreira jednu n-torku za svaki element polja	`unnest(ARRAY[3,2])` → dvije n-torke: 3, 2
generate_subscripts (*polje*, *dim* [, *desc*])	generira vrijednosti koje odgovaraju indeksima elemenata polja polje (za zadanu dimenziju *dim*); uz *desc* = *true*, poredak vrijednosti je obrnut	`generate_subscripts(ARRAY[6,3,4],1)` → tri n-torke: 1, 2, 3

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

- Primjer: za svakog nositelja dohvatiti polje koje sadrži šifre svih predmeta kojima je nositelj - predmeti u polju moraju biti poredani po šifri predmeta silazno

```
SELECT nositelj, array_agg(sifPred ORDER BY sifPred DESC)
                 AS predmeti
  FROM predmet
 GROUP BY nositelj
```

nositelj	predmeti
145	{5,3,2}
333	{4}

$\pi_{sifPred,nositelj}(predmet)$

sifPred	nositelj
2	145
4	333
5	145
3	145

PostgreSQL: ARRAY (8)

Primjeri korištenja operatora i funkcija (1):

- predmeti s istim izvođačima (uz isti poredak izvođača)

```
SELECT p1.*, p2.* FROM predmet p1, predmet p2
 WHERE p1.sifPred > p2.sifPred
   AND p1.izvodjaci = p2.izvodjaci
```

- predmeti čiji je izvođač nastavnik sa šifrom 145

```
SELECT * FROM predmet
 WHERE izvodjaci && ARRAY[145];
```

ili

```
SELECT * FROM predmet
   WHERE 145 = ANY (izvodjaci)
```

- za predmet čiji nositelj nije evidentiran i kao izvođač, ispisati šifru predmeta, šifru nositelja i polje šifri izvođača

```
SELECT sifPred, nositelj, izvodjaci
  FROM predmet
  WHERE NOT (nositelj = ANY(izvodjaci))
```

- za vježbu: predmeti čiji je izvođač samo nastavnik sa šifrom 145 - koristiti funkciju ALL

PostgreSQL: ARRAY (9)

- primjeri korištenja operatora i funkcija (2):

```
SELECT izvodjaci,
       array_length(izvodjac,1) AS brojIzv
  FROM predmet
WHERE sifPred = 1;
```

izvodjaci	brojIzv
{123,111,345,24}	4

```
UPDATE predmet SET izvodjaci[6] = 555
 WHERE sifPred = 1;

SELECT izvodjaci,
       array_length(izvodjaci,1) AS brojIzv,
       izvodjaci[5] AS petiElement
  FROM predmet
WHERE sifPred = 1;
```

izvodjaci	brojIzv	petiElement
{123,111,345,24,NULL,555}	6	NULL

```
UPDATE predmet SET izvodjaci = izvodjaci[1:2] || ARRAY[NULL,22]
 WHERE sifPred = 1;
```

izvodjaci	brojIzv	petiElement
{123,111,NULL,22}	4	NULL

PostgreSQL: ARRAY (10)

predmet.izvodjaci
ARRAY[123,111,345,24]

- deugnježđivanje - korištenje funkcije UNNEST (konverzija polja u tablicu)

```
SELECT UNNEST(izvodjaci) AS sifIzv
  FROM predmet
 WHERE sifPred = 1
```

```
SELECT i.sifIzv
  FROM predmet,
       UNNEST(predmet.izvodjaci) AS i(sifIzv)
 WHERE predmet.sifPred = 1
```

sifIzv
123
111
345
24

→ ime (virtualne) relacije i imena atributa u njoj određena su s: *AS i(sifIzv)*

- dohvat svih parova (*sifPred*, *sifIzv*):

```
SELECT sifPred, UNNEST(izvodjaci) AS sifIzv
  FROM predmet
```

ili

```
SELECT sifPred, sifIzv
  FROM predmet, UNNEST(izvodjaci) AS i(sifIzv)
```

- za predmete, za koje su navedeni izvođači koji nisu evidentirani kao osobe, ispisati šifre predmeta i izvođača u obliku: *šifra predmet*, *šifra izvođač*

```
SELECT sifPred, sifIzv
  FROM predmet, UNNEST(predmet.izvodjaci) AS i(sifIzv)
 WHERE sifIzv NOT IN (SELECT sifOsoba FROM nastavnik)
```

PostgreSQL: ARRAY (11)

predmet.izvodjaci
ARRAY[123,111,345,24]

- dohvat šifre izvođača i pozicije izvođača u polju
  - **WITH ORDINALITY** je podržan od verzije 9.4.

```
SELECT i.sifIzv, i.poz
  FROM predmet,
       UNNEST(izvodjaci) WITH ORDINALITY AS i(sifIzv, poz)
 WHERE sifPred = 1
 ORDER BY i.sifIzv
```

sifIzv	poz
24	4
111	2
123	1
345	3

  - verzije < 9.4 - korištenje funkcije **generate_subscripts**:

```
SELECT izvodjaci[poz], poz
  FROM (SELECT sifPred, izvodjaci, generate_subscripts(izvodjaci, 1) AS poz
          FROM predmet
         WHERE sifPred = 1) AS i
 ORDER BY izvodjaci[poz];
```

- za predmet za koji nisu evidentirani svi izvođači (tj. postoji element polja izvođači čija je vrijednost NULL) ispisati šifru predmeta i broj neevidentiranih izvođača

```
SELECT sifPred, izvodjaci, array_length(izvodjaci,1)-COUNT(*)
  FROM predmet, UNNEST(izvodjaci) AS sifIzvE
 WHERE sifIzvE IS NOT NULL
 GROUP BY sifPred, izvodjaci, array_length(izvodjaci,1)
HAVING array_length(izvodjaci,1)-COUNT(*)> 0
```

PostgreSQL: ARRAY (12)

- ugnježđivanje - pretvaranje rezultata upita u kolekciju

```
ARRAY(SELECT izraz... FROM ...)
```

```
INSERT INTO predmet (sifPred, izvodjaci) VALUES (555
                           , ARRAY(SELECT sifOsoba
                                     FROM nastavnik
                                    WHERE prezime LIKE 'P%'));
```

```
funkcija array_agg (izraz)
```

- Primjer: dohvatiti sve nastavnike koji su bili izvođači bilo kojeg predmeta - šifre svih izvođača smjestiti u polje naslovljeno sa *sviIzvodjaci*, poredane uzlazno

```
SELECT ARRAY(SELECT DISTINCT sifIzv
               FROM predmet, UNNEST(izvodjaci) AS sifIzv
              WHERE sifIzv IS NOT NULL
              ORDER BY sifIzv) AS sviIzvodjaci;
```

¡|¡

```
SELECT array_agg(DISTINCT sifIzv ORDER BY sifIzv) AS sviIzvodjaci
  FROM predmet, UNNEST(izvodjaci) AS i(sifIzv)
 WHERE sifIzv IS NOT NULL
```

PostgreSQL: ARRAY (13)

- proširenje: **intarray** modul - dodatni operatori i funkcije za rad s **cjelobrojnim poljima** koja **ne sadrže NULL elemente** te mogućnost kreiranja indeksa nad array atributima - `CREATE EXTENSION intarray;`
- neki dodatni operatori i funkcije:

Operator	Opis	Primjer
polje1 - *polje2*	izbacivanje iz polja elemenata polja *polje2*	`ARRAY[3,1,3,2] - 3 → {1,2}` `ARRAY[4,3,4,5] - ARRAY[4,3,1] → {5}`
polje | *element* *polje* | *polje2*	unija elemenata polja i elementa *element*/elemenata polja *polje2*	`ARRAY[3,3,4,1] | 4 → {1,3,4}` `ARRAY[3,3,4,1] | ARRAY[5,1,1] → {1,3,4,5}`
polje1 & *polje2*	presjek dva polja	`ARRAY[3,1,4,1] & ARRAY[5,1,1] → {1}`

Funkcija	Opis	Primjer
icount (*polje*)	broj elemenata polja	`icount(ARRAY[[1,3,0], [4,6,4]]) → 6`
sort_asc (*polje*) sort_desc (*polje*)	sortiranje elemenata polja	`sort_asc(ARRAY[3,8,3,4]) → {3,3,4,8}` `sort_desc(ARRAY[3,8,3,4]) → {8,4,3,3}`
idx (*polje*, *element*)	indeks prvog pojavljivanja elementa	`idx(ARRAY[2,1,1,2,1,2,3], 1) → 2`
subarray (*polje*,*poc* [, *duljina*])	segment polja od indeksa *poc*, duljine *duljina*	`subarray(ARRAY[2,1,1,2,1,3],4) → {2,1,3}` `subarray(ARRAY[2,1,1,2],2,5) → {1,1,2}`

PostgreSQL: ARRAY (14)

- proširenje - nastavak
  - 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 izvodjaciIdx ON predmet USING GIST (izvodjaci gist__intbig_ops);
```

```
CREATE INDEX izvodjaciIdx ON predmet USING GIN (izvodjaci);
```

Primjeri:

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

```
CREATE TABLE zavrsni (
   sifOsoba 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: Korisnički definirani tipovi

- Korisnički definirani tipovi (*user-defined types* – UDT)
  - definirani i imenovani od strane korisnika
  - izdržljivi (*persistence*)
  - 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: naredba za kreiranje korisnički definiranih tipova

```
CREATE TYPE <user-defined type body>

<user-defined type body> ::= <user-defined type name> [UNDER <supertype name>]
                             [AS <representation>] [[NOT] INSTANTIABLE]
                             [NOT] FINAL
                             [REF IS SYSTEM GENERATED | REF USING <predefined type> |
                                  REF FROM <attribute name> [{,<attribute name> }...]]
                             [<method specification list>]

<representation> ::= <predefined type> | <member list>

<member list> ::= (<attribute definition> [{,<attribute definition> }...])

<attribute definition> ::= <attribute name> {<data type> | <collection type>}
                           [<reference scope check>] [DEFAULT <default value>]

<data type> ::= <predefined type> | <reference type>

<collection type> ::= <data type> ARRAY [<unsigned integer>]   /* [] dio sintakse */

<method specification list> ::= <method specification> [{,<method specification> }... ]

<method specification> ::=
    <partial method specification> | <overriding method specification>

<overriding method specification> ::= OVERRIDING <partial method specification>

<partial method specification> ::= [CONSTRUCTOR] METHOD <method name>
                                    <SQL parameter declarations> RETURNS <data type>
```

SQL standard: DISTINCT tip (1)

- korisnički definirani tip
- temeljen na atomarnom tipu
- dodjeljuje posebno značenje postojećem atomarnom tipu
  - sprječava miješanje logički nekompatibilnih vrijednosti (npr. usporedbu godina s težinom)
- nad DISTINCT tipovima moguće je definirati metode
- nije moguća hijerarhija

```
CREATE TYPE godineT AS INTEGER FINAL;
CREATE TYPE tezinaT AS INTEGER FINAL;
CREATE TABLE osoba (
        sifOsoba    INTEGER
        prezime     VARCHAR(25),
        godineOsoba godineT
        tezinaOsoba tezinaT
        PRIMARY KEY sifOsoba);
```

SQL standard: DISTINCT tip (2)

- uspoređivanje vrijednosti istog DISTINCT tipa:

```
SELECT o1.sifOsoba
  FROM osoba o1, osoba o2
 WHERE o2.sifOsoba = 123 AND o1.godineOsoba < o2.godineOsoba;
```

- nije dozvoljeno uspoređivanje vrijednosti DISTINCT tipa i atomarnog tipa na kojem je temeljen te različitih DISTINCT tipova temeljenih na istom atomarnom tipu:

```
SELECT sifOsoba FROM osoba
 WHERE (godineOsoba * 2) < tezinaOsoba;
```

→ ERROR

  - uspoređivanje vrijednosti DISTINCT tipa i atomarnog tipa na kojem je temeljen dozvoljeno je uz korištenje funkcije CAST:

```
SELECT sifOsoba FROM osoba
 WHERE CAST(godineOsoba AS INTEGER) * 2 < CAST(tezinaOsoba AS INTEGER);
```

SQL standard: Strukturirani tipovi (1)

- 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

- instanca strukturiranog tipa je vrijednost, ali ima neke karakteristike objekta
  - pohranjeni podatak ⇒ stanje ⇒ atributi
  - ponašanje ⇒ semantika ⇒ metode
- vrijednost strukturiranog tipa sastoji se od određenog broja vrijednosti atributa (*attribute values*)

SQL standard: Strukturirani tipovi (2)

- 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
- kreiranje tablica s atributima koji su strukturiranog tipa:

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

CREATE TYPE adresaT AS (
        ulica       VARCHAR(50),
        mjesto      mjestoT
) NOT FINAL;
```

```
CREATE TABLE osoba (
        sifOsoba    INTEGER
        ime         VARCHAR(25),
        prezime     VARCHAR(25),
        adresa      adresaT
);
```

SQL standard: Tipizirane tablice (1)

- ***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
- za modeliranje veza između entiteta i ponašanje entiteta
- kreiranje tipizirane tablice:

```
CREATE TABLE <table name> OF <user-defined type>
   [UNDER <supertable name>][ <table element list>]

<table element list> ::= (<table element> [{,<table element>}...])

<table element> ::= <table constraint>
   | <self-referencing column specification> | <column options>

<self-referencing column specification > ::=
   REF IS <self-referencing column name> <reference generation>

<reference generation> ::= SYSTEM GENERATED | USER GENERATED | DERIVED

<column options> ::= <column name> WITH OPTIONS <column option list>

<column option list> ::= [SCOPE <table name>[<reference scope check>]]
                  [DEFAULT <default value>] [ <column constraint>... ]
```

SQL standard: Tipizirane tablice (2)

- Primjer: kreiranje tablice *mjesto* na temelju tipa *mjestoT*

1.
```
CREATE TYPE mjestoT AS (postBr     INTEGER,
                        nazMjesto  VARCHAR(40))
                        INSTANTIABLE
                        NOT FINAL
                        REF IS SYSTEM GENERATED;
```

  - mogućnost direktnog kreiranja instance tipa:
    - INSTANTIABLE – moguće je direktno kreirati instance tipa (za tip postoji *constructor* metoda); nužno za tip koji se koristi zajedno s tipiziranom tablicom
    - NOT INSTANTIABLE - za tip ne postoji *constructor* metoda
  - način generiranja vrijednosti reference na objekt (*object reference*) - 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

SQL standard: Tipizirane tablice (3)

2.

```
CREATE TABLE mjesto OF mjestoT
(PRIMARY KEY (postBr),
 REF IS mjestoID SYSTEM GENERATED,
 postBr WITH OPTIONS CONSTRAINT dozvoljeniPbr
                      CHECK (postBr BETWEEN 10000 AND 99999));
```

- OF klauzula - sadrži strukturirani tip nad kojim je definirana tipizirana tablica
- REF IS klauzula - način generiranja vrijednosti reference
  - mora biti konzistentan s načinom generiranja u korištenom strukturiranom tipu
    - SYSTEM GENERATED, USER GENERATED, DERIVED
  - dodjeljuje ime dodatnom stupcu

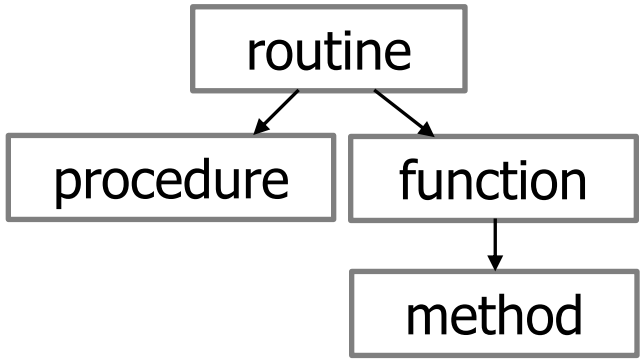
- upisivanje zapisa u tablicu *mjesto*

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

mjesto

mjestoID	postBr	nazMjesto
1023456734	10000	Zagreb

SQL standard: Pohranjene procedure (1)



- pohranjene procedure *(SQL-invoked routines)* - mogu biti pozvane iz SQL kôda
- SQL razlikuje tri tipa pohranjenih procedura:
  - **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
  - **Procedura:**
    - ulazni i izlazni parametri
    - pozivanje CALL naredbom

SQL standard: Pohranjene procedure (2)

- **SQL rutina** - napisana u SQL-u
  - SUBP-ovi podržavaju vlastite proceduralne jezike (npr. Oracle -PL/SQL, Microsoft SQLServer - TransactSQL, u IBM Informix - SPL)

- **eksterna rutina** (*external routines*)
  - napisana u eksternom programskom jeziku (npr. Java, C++)
  - poziva se na isti način kao SQL procedura ili funkcija
  - omogućava korištenje postojećeg koda
  - portabilnost između različitih sustava baza podataka
  - problem nepodudaranja tipova podataka (npr. time, blob, …)
  - primjer eksterne rutine:

```
CREATE FUNCTION malaSlika (IN ulaznaSlika slikaT) RETURNS BOOLEAN
EXTERNAL NAME '/usr/bin/slike/malaSlika'
LANGUAGE C ...
```

SQL standard: Strukturirani tipovi - Metode

- s tipom se manipulira samo kroz metode koje su na njemu definirane
  - tip prvog, implicitnog parametra metode je strukturirani tip uz kojeg je metoda vezana (funkcije i procedure nisu vezane uz strukturirani tip)
- aplikacije *svemu* pristupaju kroz funkcionalno sučelje
  - promjena implementacije ne utječe na aplikacije ukoliko sučelje ostane nepromijenjeno
- vrijednosti atributa su učahurene (*encapsulated*) - može im se pristupiti pozivanjem *observer i mutator* funkcija

SQL standard: Ugrađene metode (1)

- za strukturirane tipove postoje tri vrste ugrađenih metoda:
  - ***constructor***
    - koristi se za kreiranje instanci tipa
    - istog imena kao tip
    - implicitno definirana u trenutku kreiranja tipa
      - funkcija bez argumenata
      - vraća pretpostavljene (*default*) ili NULL vrijednosti atributa
    - prilikom poziva koristi se izraz *new*
  - ***observer*** i ***mutator***
    - jedna za svaki atribut strukturiranog tipa
    - istog imena kao atribut
    - *observer* - vraća vrijednosti atributa strukturiranog tipa
    - *mutator* - modificira vrijednosti atributa strukturiranog tipa
    - poziva se korištenjem *dot* notacije (**varijabla.imeFunkcije**)

SQL standard: Ugrađene metode (2)

- Primjer:
  - sekvenca kôda iz SQL rutine koja koristi *constructor* metodu i *mutator* metodu za kreiranje instance tipa *mjestoT*

```
CREATE TYPE mjestoT AS (
        postBr     INTEGER,
        nazMjesto  VARCHAR(40)
) NOT FINAL;
CREATE TABLE osoba (
        sifOsoba   INTEGER
        ime        VARCHAR(25),
        prezime    VARCHAR(25),
        mjestoStan mjestoT
);
```

```
BEGIN
   DECLARE iMjesto mjestoT;
   /* poziv constructor funkcije */
   SET iMjesto = new mjestoT();
   /* poziv mutator funkcija */
   iMjesto.postBr(10000);
   iMjesto.nazMjesto('Zagreb');
   ...
   INSERT INTO osoba VALUES(…, iMjesto);
END
```

  - ovdje nova instanca tipa nije objekt
    - da bi bila objekt, mora biti korištena u kombinaciji s *tipiziranim tablicama*

SQL standard: Korisnički-definirane metode (1)

- korisnici mogu za strukturirane tipove definirati vlastite metode
- metode se specificiraju kao dio definicije tipa

```
CREATE TYPE zaposlenikT (imeZaposlenik   VARCHAR(15),
                         prezZaposlenik  VARCHAR(15),
                         placa           INTEGER) NOT FINAL
   METHOD placaUzPovisicu(postotak INTEGER) RETURNS INTEGER;
```

- tijelo metode definira se odvojeno

```
CREATE METHOD placaUzPovisicu (postotak INTEGER) RETURNS INTEGER
       FOR zaposlenikT
  BEGIN
     RETURN (100 + postotak) * SELF.placa / 100;
  END
```

  - pomoću ključne riječi SELF moguće je pristupiti vrijednosti implicitnog parametra u implementaciji metode (instanca tipa za kojeg je metoda definirana)

- Primjer korištenja korisnički-definirane metode: za svakog zaposlenika dohvatiti prezime i iznos plaće uz navedeni postotak povišice na plaću

```
{ CREATE TABLE zaposlenik OF zaposlenikT; }
SELECT prezZaposlenik , placaUzPovisicu (7)
  FROM zaposlenik
```

SQL standard: Korisnički-definirane metode (2)

- Primjer: nadjačavanje *constructor* metode
  - nadjačana constructor funkcija koristi se za postavljanje vrijednosti atributa u trenutku kreiranja instance tipa
  - u definiciji tipa mora biti specificirana ključna riječ OVERRIDING

```
CREATE TYPE mjestoT AS (
        postBr     INTEGER,
        nazMjesto  VARCHAR(40)) NOT FINAL
OVERRIDING constructor METHOD /* novi konstruktor s parametrima */
   mjestoT(postBr INTEGER, nazMjesto VARCHAR(40)) RETURNS mjestoT;
```

```
CREATE METHOD mjestoT (pPostBr INTEGER, pNazMjesto VARCHAR(40))
       RETURNS mjestoT FOR mjestoT
  BEGIN
     SET SELF.postBr = pPostBr;
     SET SELF.nazMjesto = pNazMjesto
     RETURN SELF;   /* modificirana instanca tipa mjestoT */
  END
```

```
...
DECLARE novoMjesto mjestoT;
SET novoMjesto = new mjestoT (10000, 'Zagreb');
...
```

SQL standard: Korištenje metoda

- Primjer: korištenje *constructor* i *mutator* metoda

```
CREATE TYPE mjestoT AS (
      postBr      INTEGER,
      nazMjesto   VARCHAR(40)
) NOT FINAL;
CREATE TABLE osoba (
      sifOsoba    INTEGER
      ime         VARCHAR(25),
      prezime     VARCHAR(25),
      mjestoStan mjestoT
);
```

```
INSERT INTO osoba
     VALUES (12345, 'Pero', 'Perić'
                NEW mjestoT(10000, 'Zagreb'))
```

```
UPDATE osoba SET mjestoStan
      = NEW mjestoT(10010, mjestoStan.nazMjesto)
 WHERE sifOsoba = 12345
```

```
...SET mjestoStan = mjestoStan.postBr(10010) ...
```

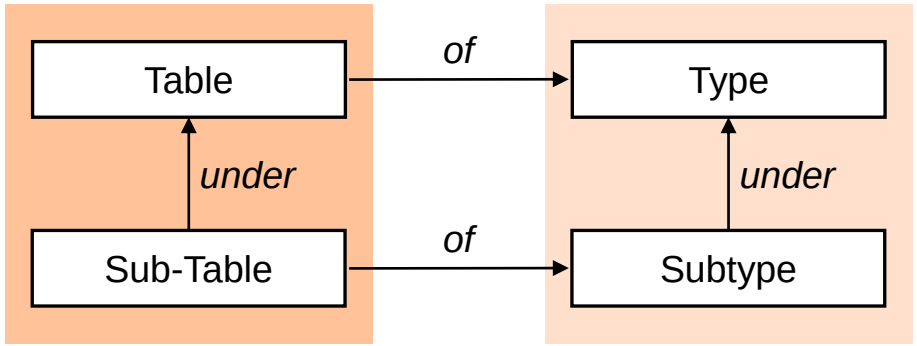
```
...SET mjestoStan.postBr = 10010 ...
```

- korištenje *observer* metoda

```
SELECT o.mjestoStan.postBr
  FROM osoba o
 WHERE o.mjestoStan.nazMjesto LIKE '%Zagreb%';
```

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



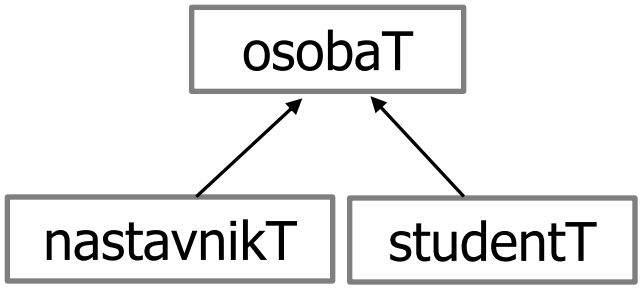
SQL standard: Nasljeđivanje tipova (1)

- podtip nasljeđuje atribute i metode nadređenog tipa
- hijerarhija strukturiranih tipova definira se klauzulom UNDER u definiciji tipa
- iz tipa može biti kreiran drugi tip, ako je u njegovoj definiciji navedena ključna riječ NOT FINAL
- podtip može redefinirati učinak metode ponovnim deklariranjem metode, korištenjem nadjačavanja metoda:
  - u definiciji podtipa, prilikom deklaracija metoda, umjesto METHOD koristi se OVERRIDING METHOD

SQL standard: Nasljeđivanje tipova (2)

- Primjer:
  - čuvaju se dodatne informacije o osobama koje su studenti i osobama koje su nastavnici

```
CREATE TYPE osobaT AS (
        sifOsoba    INTEGER,
        jmbg        CHAR(13)
        prezime     VARCHAR(25))
        INSTANTIABLE NOT FINAL
        REF IS SYSTEM GENERATED
METHOD starost(jmbg CHAR(13)) RETURNS INTEGER;

CREATE TYPE studentT UNDER osobaT AS (
        razinaStudij CHAR(20),
        zavod        VARCHAR(100))
        INSTANTIABLE NOT FINAL;

CREATE TYPE nastavnikT UNDER osobaT AS (
        placa       INTEGER,
        zavod       VARCHAR(100))
        INSTANTIABLE NOT FINAL;
```



- *studentT* i *nastavnikT* su podtipovi tipa *osobaT*
- *osobaT* je nadređen tipovima *studentT* i nastavnik*T*
- *studentT* i *nastavnikT* od *osobaT* nasljeđuju:
  - atribute *sifOsoba, jmbg* i *prezime*
  - metodu *starost*

SQL standard: Nasljeđivanje tablica

- omogućeno je samo za tipizirane tablice
- tipovi podtablica moraju biti podtipovi tipa nadređene tablice
- svaki atribut koji postoji u nadređenoj tablici postoji i u podtablicama
- svaka n-torka iz podtablice implicitno postoji i u nadređenoj tablici
  - n-torka u podtablici odgovara n-torki u nadređenoj tablici, ako ima iste vrijednosti svih naslijeđenih atributa
  - korespondentne n-torke u podtablici i nadređenoj tablici predstavljaju isti entitet

Primjer: kreiranje tablica *student* i *nastavnik* kao podtablica tablice *osoba*

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

CREATE TABLE student OF studentT
             UNDER osoba;

CREATE TABLE nastavnik OF nastavnikT
             UNDER osoba;
```

- prethodno mora biti definirana hijerarhija tipova *osobaT*, *studentT* i *nastavnikT*
- atributi *sifOsoba*, *jmbg* i *prezime* postoje i u tablicama *student* i *nastavnik*
- svaka n-torka iz *student* i *nastavnik* postoji u tablici *osoba*

SQL standard: Pravila vezana uz nasljeđivanje

- nije dozvoljeno višestruko nasljeđivanje - svaki podtip/podtablica ima samo jednu temeljnu (korijensku) nadklasu/nadtablicu
- primarni ključ definira se samo za temeljnu (korijensku) nadtablicu, a nasljeđuju ga sve njezine podtablice
- REF IS klauzula se definira samo na razini temeljnog (korijenskog) nadtipa/nadtablice. Identifikator objekta nasljeđuju sve podtablice.
- integritetska ograničenja navedena u definiciji tablice navode se samo uz atribute koje je u tablicu uveo strukturirani tip nad kojim je tablica definirana, a ne na naslijeđenim atributima
- hijerarhiju tipova moguće je definirati neovisno o hijerarhiji tablica
  - NOT INSTANTIABLE klauzula može biti korištena u hijerarhiji tipova koji nisu vezani uz tipizirane tablice
- svi tipovi vezani uz hijerarhiju tipiziranih tablica moraju biti definirani kao INSTANTIABLE, kako bi bilo podržano korištenje INSERT naredbi na tipiziranim tablicama

SQL standard: Hijerarhija tablica - upisivanje n-torki

- u INSERT naredbi navode se vrijednosti i za naslijeđene atribute
- vrijednost identifikatora objekta se automatski generira prilikom izvođenja INSERT naredbe
- najspecifičnija tablica (*most-specific table*) n-torke - tablica u koju je n-torka direktno upisana
- najspecifičniji tip (*most-specific type*) - odgovara tipu tablice u koju je n-torka direktno upisana

```
INSERT INTO osoba VALUES (111, '120196', 'Kolar');
INSERT INTO osoba VALUES (222, '231196', 'Novak');
INSERT INTO nastavnik VALUES (555, '300176, 'Jurak', 5000, 'ZPM');
```

osoba

osobaID	sifOsoba	jmbg	prezime
123456	111	120196	Kolar
234567	222	231196	Novak
564790	555	300176	Jurak

nastavnik

osobaID	sifOsoba	jmbg	prezime	placa	zavod
564790	555	300176	Jurak	5000	ZPM

SQL standard: Hijerarhija tablica - dohvat n-torki

- upit postavljen nad tablicom:
  - pristupa samo atributima koji postoje u tablici (ne u podtablicama)
  - n-torke koje su rezultat upita mogu uključivati:
    - n-torke direktno upisane u tablicu i n-torke upisane u podtablice ili
    - samo n-torke direktno upisane u tablicu
      - ključna riječ **ONLY** uz naziv tablice u FROM dijelu SELECT naredbe

```
SELECT jmbg
     , prezime
  FROM osoba
```

jmbg	prezime
111	Kolar
222	Novak
555	Jurak

```
SELECT jmbg
     , prezime
  FROM nastavnik
```

jmbg	prezime
555	Jurak

- dohvat samo n-torki koje su direktno upisane u tablicu:

```
SELECT jmbg
     , prezime
  FROM ONLY(osoba)
```

jmbg	prezime
111	Kolar
222	Novak

SQL standard: REF tip (1)

- 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
- SQL sintaksa za definiranje REF tipa:

```
<reference type> ::= REF (<referenced type>) [ <scope clause> ]
          [ARRAY [<unsigned integer>]] /* [] je dio sintakse */
          [ <reference scope check> ]
<referenced type> ::= <user-defined type name>
<scope clause> ::= SCOPE <table name>
<reference scope check> ::= REFERENCES ARE [ NOT ] CHECKED
                            [ ON DELETE <action> ]
```

- 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

SQL standard: REF tip (2)

```
CREATE TYPE predmetT (
   sifPred      INTEGER,
   nazPred      VARCHAR(250),
   nositelj     REF (nastavnikT),
   izvodjaci    REF (nastavnikT) ARRAY[10],
   polaznici    REF (studentT) MULTISET,
   zavod        zavodT)
   INSTANTIABLE NOT FINAL
   REF IS SYSTEM GENERATED
```

Napomena: prethodno je kreirana hijerarhija tipiziranih tablica *osoba*, *nastavnik* i *student*, kao i strukturirani tip *zavodT*

```
CREATE TABLE predmet OF predmetT
(PRIMARY KEY (sifPred)
 REF IS predmetID SYSTEM GENERATED)
```

→ veza s objektima tipa *nastavnikT* i *studentT* ostvarena je korištenjem REF tipa

predmet

predmetID	sifPred	nazPred	nositelj	...
1463144246	1	Napredni ...		

prema *nastavnikT* objektu

SQL standard: REF tip (3)

- upisivanje n-torke s NULL vrijednosti atributa tipa REF:

```
INSERT INTO PREDMET (sifPred, nazPred, nositelj)
       VALUES (1, 'Napredni modeli i baze podataka', NULL);
```

- dodjeljivanje vrijednosti atributu definiranom kao REF tip:

```
UPDATE predmet
   SET nositelj = (SELECT osobaID
                     FROM nastavnik
                    WHERE sifOsoba = 12343)
 WHERE nazPred = 'Napredni modeli i baze podataka'
```

  - pri dodjeljivanju vrijednosti atributa REF tipa koristi se ime dodatnog atributa tipizirane tablice (identifikatora objekta) u kojoj se nalazi objekt na kojeg REF tip pokazuje

```
CREATE TABLE osoba OF osobaT (PRIMARY KEY (sifOsoba),
       REF IS osobaID SYSTEM GENERATED;
CREATE TABLE student OF studentT UNDER osoba;
CREATE TABLE nastavnik OF nastavnikT UNDER osoba;
```

SQL standard: REF tip (4)

- doseg REF tipa:
  - određuje na koje se objekte može referencirati
  - definiran SCOPE klauzulom u specifikaciji REF tipa u CREATE TYPE ili CREATE TABLE naredbi
    - navedena SCOPE klauzula - dozvoljeno referenciranje samo na objekte navedene tipizirane tablice
    - nije navedena SCOPE klauzula - dozvoljeno referenciranje na objekte iz bilo koje tablice temeljene na navedenom tipu
- kontrola dosega:
  - klauzula REFERENCES ARE CHECKED u specifikaciji REF tipa - nisu dozvoljene neispravne vrijednosti reference
- akcije pri pokušaju narušavanja ograničenja dosega:
  - klauzula ON DELETE u specifikaciji REF tipa
    - NO ACTION, SET NULL, SET DEFAULT, CASCADE

SQL standard: REF tip (5)

- primjer navođenja dosega, kontrole dosega i akcije pri pokušaju narušavanja ograničenja dosega
  - u CREATE TYPE naredbi:

```
CREATE TYPE predmetT (
 ...
 nositelj REF (nastavnikT) SCOPE nastavnik
                           REFERENCES ARE CHECKED
                           ON DELETE SET NULL
 ...)
```

  - u CREATE TABLE naredbi:

```
CREATE TABLE predmet OF predmetT
                  (nositelj WITH OPTIONS SCOPE nastavnik
                                REFERENCES ARE CHECKED
                                ON DELETE SET NULL)
```

SQL standard: REF tip (6)

```
CREATE TYPE predmetT (... nositelj REF (nastavnikT) ...);
CREATE TABLE predmet OF predmetT ...
```

- postavljanje upita nad tablicama s atributima REF tipa
  - korisnike zanimaju vrijednosti atributa referencirane n-torke
  - → *(dereference operator)* - omogućava "prijelaz" do referencirane n-torke
    - do referencirane n-torke dolazi se *implicitnim spajanjem*
    - upit postavljen na jednoj tablici vraća vrijednost atributa referencirane n-torke iz druge tablice

```
SELECT nositelj -> prezime, nositelj -> placaUzPovisicu(7)
  FROM predmet
 WHERE nazPred = 'Napredni modeli i baze podataka';
```

→ metoda tipa *nastavnikT*

  - funkcija *deref* - vraća referenciranu n-torku
    - tip dohvaćenih n-torki odgovara tipu na temelju kojeg je definirana tipizirana tablica u kojoj se nalaze

```
SELECT deref(nositelj)
  FROM predmet p
 WHERE p.zavod.kratZavod = 'ZPR';
```

→ rezultat upita je tablica s jednim stupcem tipa *nastavnikT*

PostgreSQL: Korisnički definirani tipovi

- naredbe za kreiranje korisnički definiranih tipova:
  - 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 dCelsTemp AS DECIMAL(4,1)
       CHECK (VALUE BETWEEN -100 AND 60);

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

INSERT INTO euroTemp VALUES ('2003-01-03', 'Rovaniemi',
                             -20.0::tCelsTemp);

SELECT AVG(temp) FROM usTemp
```

    - CAST iz/u DOMAIN tip trenutno nema učinka - koriste se oni vezani uz tip na kojem je DOMAIN tip temeljen
  - CREATE TYPE - složeni tip podatka (*Composite Type*)

PostgreSQL: Složeni tip podatka (*Composite Type*) (1)

- 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)
- definicija je pohranjena u rječniku podataka (*pg_type*)
- 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 AS (oznDrzava CHAR(2)
                        , nazDrzava VARCHAR(20));
```

→ kreiran je i tip *drzava*

PostgreSQL: Složeni tip podatka (*Composite Type*) (2)

- 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);
```

- uništavanje tipa: `DROP TYPE [ IF EXISTS ] typeName [ CASCADE | RESTRICT ]`

```
DROP TYPE IF EXISTS drzavaT;
```
→ ERROR

- pretpostavljeno: RESTRICT - odbijanje izvođenja ako je tip korišten u definiciji drugog tipa ili relacije

```
DROP TYPE IF EXISTS drzavaT CASCADE;
```

```
→ drop cascades to composite type mjestot column drzava
```

- izmjena definicije tipa: naredba **ALTER TYPE**

PostgreSQL: Složeni tip podatka (*Composite Type*) (3)

- instanciranje vrijednosti (literal):

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

ili

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

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

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

- dozvoljeno izostaviti ROW ako se sastoji od više atributa

- Primjer:

```
CREATE TYPE drzavaT AS (oznDrzava CHAR(2)
                      , nazDrzava VARCHAR(20));
CREATE TYPE mjestoT AS (postBroj  INTEGER,
                        nazMjesto VARCHAR(20),
                        drzava    drzavaT);
CREATE TABLE poduzece (naziv    CHAR(20)
                     , sjediste mjestoT);
```

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

PostgreSQL: Složeni tip podatka (*Composite Type*) (4)

```
CREATE TYPE drzavaT AS (oznDrzava CHAR(2)
                      , nazDrzava VARCHAR(20));
CREATE TYPE mjestoT AS (postBroj  INTEGER,
                        nazMjesto VARCHAR(20),
                        drzava    drzavaT);
CREATE TABLE poduzece (naziv    VARCHAR(20)
                     , sjediste mjestoT);
```

- pristup atributima složenog tipa:

- *dot* notacija :

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

(naziv stupca koji je složenog tipa mora biti naveden u zagradama)

- funkcijska notacija:

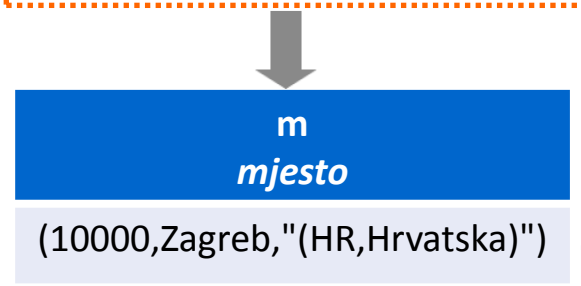
```
SELECT sjediste(poduzece),
       postBroj(poduzece.sjediste),
       drzava(sjediste),
       nazDrzava(drzava(sjediste))
  FROM poduzece
```

sjediste *mjestoT*	postBroj *integer*	drzava *drzavaT*	nazDrzava *varchar(20)*
(10000,Zagreb,"(HR,Hrvatska)")	10000	(HR,Hrvatska)	Hrvatska

PostgreSQL: Složeni tip podatka (*Composite Type*) (5)

- ime relacije ili *alias* ime relacije može biti navedeno u SELECT listi:

```
SELECT m FROM mjesto m;
```



→ vrijednost tipa mjesto

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

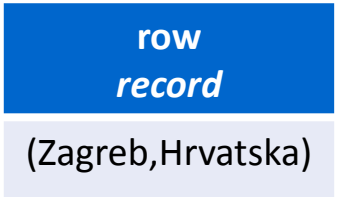
```
INSERT INTO mjesto VALUES (10000, 'Zagreb',
                           (SELECT drzava FROM drzava WHERE oznDrzava = 'HR') );
```

- promjena vrijednosti atributa složenog tipa

```
UPDATE mjesto SET drzavaM.oznDrzava = UPPER((drzavaM).oznDrzava);
```

- primjer: jednostupčani upit (*single column query*) sa stupcem čija je vrijednost tipa ROW

```
SELECT DISTINCT ROW (nazMjesto, (drzavaM).nazDrzava)
  FROM mjesto;
```



PostgreSQL: Složeni tip podatka (*Composite Type*) (6)

- funkcije koje koriste složeni tip podatka:

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

```
CREATE OR REPLACE FUNCTION mjestoS(mjesto) RETURNS char AS $$
    SELECT $1.postBroj || ' ' || $1.nazMjesto  || ', ' || ($1.drzavaM).nazDrzava;
$$ LANGUAGE SQL;
```

```
SELECT mjestoS(ROW(10000, 'Zagreb'
                , ROW('HR', 'Hrvatska'))::mjesto)
```

mjestoS *bpchar*
10000 Zagreb, Hrvatska

```
SELECT mjestoS(mjesto) FROM mjesto;
SELECT mjesto.mjestoS FROM mjesto;
```

```
CREATE OR REPLACE FUNCTION selMjesto(pPbr int, OUT pbr int, OUT nazM varchar) AS $$
    SELECT postBroj As pbr, nazMjesto AS nazM FROM mjesto
      WHERE postBroj = pPbr;
$$ LANGUAGE SQL;
```

```
SELECT selMjesto(10000);
```

selMjesto *record*
(10000,Zagreb)

```
SELECT *
  FROM selMjesto(10000);
```

pbr *integer*	nazm *varchar*
10000	Zagreb

```
SELECT (selmjesto(10000)).nazM;
SELECT nazM(selmjesto(10000));
```

nazm *varchar*
Zagreb

PostgreSQL: Složeni tip podatka (*Composite Type*) (7)

- funkcije koje koriste složeni tip podatka - nastavak:

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

```
CREATE OR REPLACE FUNCTION mjestaIzDr(char) RETURNS SETOF mjesto AS $$
    SELECT * FROM mjesto
      WHERE (drzavaM).oznDrzava = $1;
$$ LANGUAGE SQL;
```

```
SELECT mjestaIzDr('HR');
```

mjestaIzDr *mjesto*
(10000,Zagreb,"(HR,Hrvatska)")
(44000,Sisak,"(HR,Hrvatska)")

```
SELECT * FROM mjestaIzDr('HR');
```

postBroj *integer*	nazMjesto *varchar*	*drzavaM* *drzava*
10000	Zagreb	(HR,Hrvatska)
44000	Sisak	(HR,Hrvatska)

PostgreSQL: Složeni tip podatka (*Composite Type*) (8)

- funkcije koje koriste složeni tip podatka - nastavak:

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

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

postBroj *integer*	nazMjesto *varchar(20)*	*oznDrzava char(2)*	*drzavaT drzava*
10000	Zagreb	HR	(HR,Hrvatska)
44000	Sisak	HR	(HR,Hrvatska)

```
SELECT DISTINCT (m.drzavaT).nazDrzava
  FROM mjesto m
WHERE (m.drzavaT).nazDrzava LIKE 'H%';
```

nazDrzava varchar(20)
Hrvatska

- koristiti samo u slučaju malog broja n-torki u relaciji mjesto

PostgreSQL: Složeni tip podatka (*Composite Type*) (9)

- primjer konstruktora:

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

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

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

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

```
SELECT (mjesto(10000)).drzavaT.nazDrzava;
```

nazDrzava varchar(20)
Hrvatska

(*drzavaT* - funkcija s prethodnog slajda)

PostgreSQL: Složeni tip podatka (*Composite Type*) (10)

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

```
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;
```

- konstruktor

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

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

mjesto *mjesto*
(10000,Zagreb,HR)

```
SELECT (10000::mjesto).nazMjesto;
```

nazMjesto *varchar(20)*
Zagreb

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:
  - 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;
```

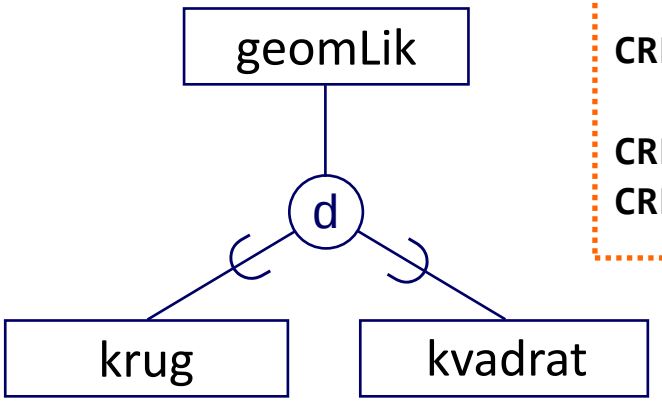
oid *oid*
17430

```
-- 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);
```

PostgreSQL: Nasljeđivanje (1)

- omogućeno je samo nasljeđivanje (baznih) tablica
  - 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
    - rules i okidači
- dozvoljeno je višestruko nasljeđivanje

PostgreSQL: Nasljeđivanje (2)




```
CREATE TABLE geomLik (id   INT
                    , boja CHAR(20));
CREATE TABLE krug (radijus FLOAT) INHERITS (geomLik);
CREATE TABLE kvadrat (duljina FLOAT) INHERITS (geomLik);
```

```
INSERT INTO geomLik VALUES (1, 'žuta');
INSERT INTO krug VALUES(2, 'crvena', 2);
INSERT INTO krug VALUES(3, 'zelena', 4);
INSERT INTO kvadrat VALUES(4, 'crvena', 5);
```

```
SELECT * FROM geomLik;
```

id	boja
1	žuta
2	crvena
3	zelena
4	crvena

```
SELECT * FROM ONLY geomLik;
```

id	boja
1	žuta

```
SELECT * FROM krug;
```

id	boja	radijus
2	crvena	2
3	zelena	4

```
SELECT * FROM geomLik*;
```

PostgreSQL: Nasljeđivanje (3)

- najspecifičniju tablicu n-torke moguće je saznati iz rječnika podataka:

```
SELECT p.relname, g.*
  FROM geomLik g, pg_class p
 WHERE g.tableoid = p.oid;
```

relname	id	boja
geomlik	1	žuta
krug	2	crvena
krug	3	zelena
kvadrat	4	crvena

- pretvorba tipa n-torke podtablice u tip n-torke nadtablice:

```
SELECT krug::geomLik
  FROM krug;
```

krug *geomLik*
(2,"crvena")
(3,"zelena")

- nasljeđivanje metode:

```
CREATE OR REPLACE FUNCTION bojaU(geomLik) RETURNS text
LANGUAGE SQL AS $$
SELECT upper($1.boja);
$$;
```

```
SELECT *, krug.bojaU
  FROM krug;
```

id	boja	radijus	bojaU
2	crvena	2	CRVENA
3	zelena	4	ZELENA

PostgreSQL: Nasljeđivanje (4)

Problemi s primarnim ključem (UNIQUE ograničenjem, indeksima)

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

```
CREATE TABLE geomLik (id INT PRIMARY KEY, boja VARCHAR(20));
CREATE TABLE krug (radijus FLOAT) INHERITS (geomLik);
```

```
INSERT INTO geomLik VALUES (100, 'crvena');
INSERT INTO geomLik VALUES (100, 'bijela');    → ERROR
INSERT INTO krug VALUES (100, 'žuta', 2.3);    → OK
INSERT INTO krug VALUES (100, 'zelena', 1.8);  → OK
```

```
DELETE FROM geomLik WHERE id = 100;       → obrisane 3 n-torke
DELETE FROM ONLY geomLik WHERE id = 100;  → obrisana 1 n-torka
```

```
ALTER TABLE krug ADD CONSTRAINT krugPk PRIMARY KEY (id);
```

- je li problem riješen?

PostgreSQL: Nasljeđivanje (5)

Problemi s primarnim ključem (UNIQUE ograničenjem, indeksima)

- rješenje: kreiranje pomoćnih funkcija i okidača

```
CREATE TABLE geomLik (id INT PRIMARY KEY, boja VARCHAR(20));
CREATE TABLE krug (radijus FLOAT) INHERITS (geomLik);
```

```
/* osiguravanje jedinstvenosti primarnog ključa */
CREATE FUNCTION geomLikPkChk() RETURNS TRIGGER AS $geomLikPkChk$
BEGIN
   IF (EXISTS (SELECT * FROM geomLik
               WHERE geomLik.id = NEW.id)) THEN
       RAISE EXCEPTION 'Postoji zapis s identifikatorom geom. lika %.', NEW.id;
   END IF;
   RETURN NEW;
END
$geomLikPkChk$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER geomLikPkChk
BEFORE INSERT ON krug
  FOR EACH ROW
    EXECUTE PROCEDURE geomLikPkChk();
-- isto i za tablicu kvadrat
```

```
CREATE TRIGGER geomLikPkChk
BEFORE INSERT ON geomLik
   FOR EACH ROW
      EXECUTE PROCEDURE geomLikPkChk();
```

PostgreSQL: Nasljeđivanje (6)

Problemi sa stranim ključem

1. strani ključevi se ne nasljeđuju

```
CREATE TABLE boja (idBoja SMALLINT PRIMARY KEY,
                   opis VARCHAR(20));
CREATE TABLE geomLik (id INT PRIMARY KEY
                    , idBoja SMALLINT REFERENCES boja (idboja));
CREATE TABLE krug (radijus FLOAT) INHERITS (geomLik);
```

```
INSERT INTO boja VALUES (1, 'bijela');
INSERT INTO geomLik VALUES (100, 2);    → ERROR
INSERT INTO krug VALUES (100, 2, 2.3);  → OK
```

```
ALTER TABLE krug ADD CONSTRAINT krugBojaFk
                     FOREIGN KEY (idBoja) REFERENCES boja (idBoja));
```

PostgreSQL: Nasljeđivanje (7)

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 geomLik (id INT PRIMARY KEY, boja VARCHAR(20));
CREATE TABLE krug (radijus FLOAT) INHERITS (geomLik);

CREATE TABLE tijelo(idTijelo INT, idLik INT REFERENCES geomLik(id));
```

```
INSERT INTO geomLik VALUES (100, 'crvena');
INSERT INTO geomLik VALUES (101, 'crvena');
INSERT INTO krug VALUES (102, 'žuta', 2.3);

INSERT INTO tijelo VALUES (10001, 100);   → OK
INSERT INTO tijelo VALUES (10001, 101);   → OK
INSERT INTO tijelo VALUES (10001, 102);   → ERROR
```

PostgreSQL: Nasljeđivanje (8)

- primjer: izračunati površinu svakog objekta

```
CREATE FUNCTION povrsina (krug) RETURNS float AS $$
   SELECT $1.radijus * $1.radijus * 3.1415926;
$$ LANGUAGE SQL;
```

```
CREATE FUNCTION povrsina (kvadrat) RETURNS float AS $$
   SELECT $1.duljina * $1.duljina::float;
$$ LANGUAGE SQL;
```

```
SELECT geomLik.id, geomLik.boja, povrsina(geomLik)
  FROM geomLik
```

ERROR: function povrsina(geomlik) does not exist

```
SELECT id, povrsina(krug) FROM krug
 UNION ALL
SELECT id, povrsina(kvadrat) FROM kvadrat;
```

id	boja	povrsina
2	crveno	12.5663704
3	zeleno	50.2654816
4	crveno	25

- funkcija *povrsina* se na razne načine evaluira za razne tipove objekata (polimorfizam)

PostgreSQL: Višestruko nasljeđivanje

- atributi podtablice: unija atributa nadređenih tablica
  - ako nadređene tablice sadrže istoimene atribute, podtablica zadržava samo jedan atribut tog imena
    - istoimeni atributi iz nadređenih tablica moraju biti istog podatkovnog tipa
    - naslijeđena su sva CHECK, NOT NULL ograničenja definirana u svim nadređenim tablicama

```
CREATE TABLE r1 (a11 INTEGER, a2 INTEGER CHECK (a2>0));
CREATE TABLE r2 (a21 INTEGER, a2 INTEGER NOT NULL CHECK (a2>5));
CREATE TABLE d3 (a31 INTEGER) INHERITS (r1,r2);
```

```
INSERT INTO r1 VALUES(1,11);
INSERT INTO r2 VALUES(2,21);
INSERT INTO d3 (a11, a2, a21, a31) VALUES(3,2,4,5);  -- ERROR
INSERT INTO d3 (a11, a2, a21, a31) VALUES(3,10,4,5); -- OK
```

```
SELECT * from r1;
```

a11	a2
1	11
3	10 300

```
SELECT * from r2;
```

a21	a2
2	21
4	**10 300**

```
SELECT * from d3;
```

a21	a2	a21	a31
2	10 300	4	5

```
UPDATE r2 SET a2 = 300 WHERE a21 = 4;
```

ORDBMS prednosti i nedostaci

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

- S.W. Dietrich, S.D. Urban: **An Advanced Course in Database Systems : Beyond Relational Databases**, Prentice Hall, 2005
- Jim Melton: **Advanced SQL: 1999 - Understanding Object-Relational and Other Advanced Features**, Morgan Kaufmann, 2002
- T. Connolly, C. Begg: **Database Systems: A Practical Approach to Design, Implementation, and Management**, 4th Edition, Pearson Education , 2005
- A. Silberschatz, H.F. Korth, S. Sudarshan: **Database Systems Concepts**, 5th Edition, McGraw-Hill, 2005.
- M. Stonebraker, D. Moore: **Object-Relational DBMSs: The Next Great Wave**, Morgan Kaufmann Publichers, 1996
- R. Ramakrishnan, J. Gehrke: **Database Management Systems**, McGraw-Hill, 2003
- **PostgreSQL 9.3.5 Documentation -** http://www.postgresql.org/docs/9.3/static/index.html