Ponavljanje II

SQL DDL

BAZE PODATAKA II

doc. dr. sc. Goran Oreški
Fakultet informatike,
Sveučilište Jurja Dobrile, Pula

Sadržaj

- ponavljanje prethodnih predavanja
  - sortiranje, grupiranje, agregiranje, filtriranje
  - ugniježđeni podupiti i join operacije
  - pogledi
- katalozi i sheme
- ograničenja tablice
  - ograničenje primanog ključa
  - NOT NULL ograničenje
  - UNIQUE ograničenje
  - CHECK ograničenje
- ograničenje stranog ključa
  - višestruka ograničenja
- kršenje ograničenja stranog ključa

Ponavljanje

- **sortiranje, grupiranje, agregiranje, filtriranje**

sortiranje	
grupiranje	
agregiranje	
filtriranje (nakon grupiranja)	

- logički redoslijed izvršavanja (interpretiranja) SELECT upita?

SELECT, ORDER BY, WHERE, HAVING, FROM, JOIN, GROUP BY, DISTINCT

Ponavljanje

- **sortiranje, grupiranje, agregiranje, filtriranje**

sortiranje	**`ORDER BY (ASC, DESC)`**
grupiranje	**`GROUP BY`**
agregiranje	**`SUM, AVG, COUNT, MIN, MAX`**
filtriranje (nakon grupiranja)	**`HAVING`**

- logički redoslijed izvršavanja (interpretiranja) SELECT upita (*docs.microsoft.com):

  `FROM -> JOIN ->`
  `WHERE -> GROUP BY -> HAVING ->`
  `SELECT -> DISTINCT -> ORDER BY`

  - objekti (npr. alijasi) iz kasnijih koraka nisu dostupni u ranijim

Ponavljanje

- **ugniježđeni podupiti i join operacije**
- ugniježđeni podupiti
  - ugniježđeni podupit je izraz tipa "select – from – where" umetnut unutar drugog upita
  - može biti umetnut unutar WHERE, FROM ili SELECT dijela
- join operacije
  - theta join (INNER JOIN)
  - Kartezijev produkt (CROSS JOIN)
  - outer join (LEFT, RIGHT, FULL OUTER)
  - natural join (NATURAL JOIN)
  - razlika ON i USING?

Ponavljanje

- **pogledi**
- razlozi za korištenje pogleda?
- sintaksa

```
CREATE VIEW view_name AS select_stmt;
```

  - stupci pogleda dolaze iz SELECT naredbe
  - nazivi stupaca moraju biti jedinstveni kao i u tablicama
  - nazivi se mogu definirati i u naredbi stvaranja pogleda

```
CREATE VIEW view_name (att1, att2, …) AS select_stmt;
```

Definiranje baze podataka

- do sada smo veći dio ponavljanja posvetili manipulaciji podataka unutar tablica
- ovo predavanje nastavljamo istraživati značajke definiranja baze podataka:
  - definiranje tablica i pripadajućih stupaca (BP1)
  - definiranje pogleda (BP1)
  - definiranje ograničenja na pojedinačne stupce ili cijele tablice (BP1)
  - stvaranje pohranjenih procedura (u nastavku)
  - definiranje ograničenja pristupa (u nastavku)
  - …

Definiranje baze podataka

- u okviru ovog poglavlja usredotočiti ćemo se na mehaniku definiranja baze podataka
- ignorirati ćemo vrlo bitno pitanje; *koliko je dobra shema baze podataka? (odgovor na to pitanje modeliranje i normalizacija BP1)*
- opći ciljevi dizajniranja baze:
  - shema mora biti u mogućnosti reprezentirati sve detalje i veze modeliranog sustava
  - shema treba pokušati onemogućiti unos neispravnih podataka
  - drugi ciljevi koji se tiču performansi i sigurnosti… (više u nastavku)
- DBMS-ovi podržavaju mnogo različitih ograničenja
  - vrlo je bitno koristiti te mogućnosti s ciljem osiguranja točnosti podataka

Katalozi i sheme

- SQL podržava hijerarhijsko grupiranje za upravljanje kolekcijama tablica
  - također odvaja područje imena za različite kolekcije tablica
- standardni mehanizam ima tri razine:
  - katalog
  - shema
  - tablica
  - sve razine imaju dodijeljeno ime
  - unutar svake razine ime mora biti jedinstveno
- omogućava da više aplikacija koristi isti server

Katalozi i sheme

- svaka tablica ima puno ime
  - katalog.shema.tablica
- sustavi baza podataka se jako razlikuju po implementaciji tih značajki
  - definicija kataloga nije pokrivena SQL specifikacijom
  - razine shema i tablica su definirane
  - većina DBMS-a nudi neku vrstu grupiranja
- uobičajeno ponašanje
  - "baze podataka" generalno odgovaraju katalozima

    ```
    CREATE DATABASE unipu;
    ```
  - grupiranje na razini sheme je često dostupno

    ```
    CREATE SCHEMA fipu;
    ```

Korištenje baze podataka

- da bi koristili bazu potrebno je spojiti se na server baze podataka
  - navesti korisničko ime, lozinku i druge podatke
- svaka konekcija na bazu podataka ima svoju okolinu
  - sesija je povezana s korisnikom
  - može se navesti shema ili katalog koji se koristi
    - npr. `USE unipu;` za korištenje baze podataka unipu
      - u MySql-u DATABASE i SCHEMA su sinonimi
  - sve operacije će koristiti navedenu bazu podataka kao zadanu
  - može se zaobići korištenjem punih naziva tablica
    - npr. `SELECT * FROM unipu.nastavnik;`

Stvaranje tablica

- opći oblik naredbe:

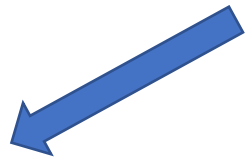
```
CREATE TABLE t (
attr1 domain1,
attr2 domain2,
...);
```

- SQL podržava mnoštvo različitih tipova podataka
  - `INT, CHAR(N), VARCHAR(N), DATE…`
  - (*ponovite BP1*)
- tablice i stupci moraju pratiti specifična pravila
- tablica mora imati jedinstveno ime unutar sheme
- stupci moraju imati jedinstveno ime unutar tablice

Ograničenja tablice

- nakon stvaranja, SQL tablice nemaju ograničenja
  - mogu se dodati višestruke kopije bilo kojeg reda
  - red može sadržavati `NULL` vrijednosti u bilo kojem stupcu
- unutar tablice mogu se definirati stupci koji čine primarni ključ

```
CREATE TABLE nastavnik (
    nastavnik_sifra CHAR(10),
    prezime         VARCHAR(20),
    primanja        NUMERIC(10,2),
    PRIMARY KEY (nastavnik_sifra)
);
```



- dva reda ne mogu imati istu vrijednost PK-a
- unutar tablice može biti definiran samo jedan PK

Ograničenja primarnog ključa

- alternativna sintaksa

```
CREATE TABLE nastavnik (
   nastavnik_sifra CHAR(10) PRIMARY KEY ,
   prezime         VARCHAR(20),
   primanja        NUMERIC(10,2));
```

  - može se koristiti samo za primarne ključeve koji sadrže jedan stupac
- ukoliko koristi više stupaca, PK se mora navesti nakon stupaca

```
CREATE TABLE predaje (
   nastavnik_sifra CHAR(10),
   kolegij_naziv   VARCHAR(40),
   PRIMARY KEY (nastavnik_sifra, kolegij_naziv));
```

Ograničenje null vrijednosti

- svaka domena atributa sadrži null vrijednost
  - isto vrijedi i za SQL; svaki stupac se može postaviti na `NULL` ukoliko nije dio primarnog ključa
- često, `NULL` nije prihvatljiva vrijednost
  - npr. saldo računa u banci, ocjena studenta, …
- da bi se isključile `NULL` vrijednosti za određeni stupac potrebno ga je definirati kao `NOT NULL`
  - navodi se prilikom deklaracije stupca
- primarni ključ se ne mora navoditi kao `NOT NULL` jer se to podrazumijeva

Nastavnik relacija

- *šifra* nastavnika je primarni ključ
  - ne može biti NULL
- *ime, prezime, titula* i *primanja* moraju uvijek biti navedena
  - dodajemo NOT NULL ograničenje na te stupce
- SQL

```
CREATE TABLE nastavnik (
   nastavnik_sifra CHAR(10) PRIMARY KEY ,
   ime             VARCHAR(30) NOT NULL,
   prezime         VARCHAR(30) NOT NULL,
   zvanje          VARCHAR(40) NOT NULL,
   primanja        NUMERIC(10,2) NOT NULL);
```

Ključevi kandidati

- neke relacije imaju više ključeva kandidata
- ključevi kandidati se mogu definirati s `UNIQUE` ograničenjem
  - kao i kod primarnog ključa, jednostavan kandidati ključ se može navesti u deklaraciji kolone ili ako je složen, nakon svih stupaca
- za razliku od primarnih ključeva, `UNIQUE` ograničenja ne isključuju `NULL` vrijednosti
  - ograničenje smatra `NULL` vrijednosti kao nejednake
    - ovisi o implementaciji!
  - ako neki atributi unutar `UNIQUE` ograničenja dozvoljavaju `NULL`, tada će baza dozvoliti višestruke redove s istom vrijednosti

UNIQUE ograničenja

- primjer: nastavnik relacija

```
CREATE TABLE nastavnik (
   id                INT PRIMARY KEY
   nastavnik_sifra CHAR(10) NOT NULL UNIQUE ,
   ime               VARCHAR(30) NOT NULL,
   prezime           VARCHAR(30) NOT NULL,
   zvanje            VARCHAR(40) NOT NULL,
   primanja          NUMERIC(10,2) NOT NULL
   );
```

  - *id* je primarni ključ
  - svaki nastavnik mora imati šifru, dva nastavnika ne smiju imati istu šifru
  - svi nastavnici moraju imati ime, dva nastavnika smiju imati isto ime

UNIQUE i NULL

- primjer:

```
CREATE TABLE partner (
   naziv    VARCHAR(40) NOT NULL,
   adresa   VARCHAR(60),
   UNIQUE (naziv, adresa));
```

  - ukoliko pokušamo dodati redove

```
INSERT INTO partner VALUES ('Ina d.d.', 'Dubrovačka 15');
INSERT INTO partner VALUES ('Ina d.d.', 'Dubrovačka 15');
```

  - rezultat je

```
Duplicate entry 'Ina d.d.-Dubrovačka 15' for key 'naziv'
```

UNIQUE i NULL

- primjer:

```
CREATE TABLE partner (
   naziv    VARCHAR(40) NOT NULL,
   adresa   VARCHAR(60),
   UNIQUE (naziv, adresa));
```

  - ukoliko pokušamo dodati redove

```
INSERT INTO partner VALUES ('Ina d.d.', NULL);
INSERT INTO partner VALUES ('Ina d.d.', NULL);
```

    - insert uspješan

- treba biti oprezan s korištenjem stupaca s NULL vrijednosti unutar UNIQUE ograničenja (tipično svi trebaju biti NOT NULL)

CHECK ograničenja

- potreba za postavljenjem dodatnih ograničenja na vrijednosti
- CHECK ograničenja se koriste kada je potrebno postaviti uvjete na vrijednosti podataka koji se dodaju u bazu podataka
  - glavna primjena je ograničavanje domene stupca i sprječavanje unosa očito neispravnih vrijednosti
- CHECK ograničenje se navodi nakon specifikacije svih stupaca
- u teoriji, ograničenje je bilo koji izraz koji kao rezultat generira Boolean vrijednost
  - to uključuje i ugniježđene podupite
  - o implementaciji ovisi podrška ovog ograničenja; često je dosta limitirana

Primjer CHECK ograničenja

- ograničiti se mogu vrijednosti u određenom kolonama

```
CREATE TABLE nastavnik (
   id                INT PRIMARY KEY,
   nastavnik_sifra CHAR(10) NOT NULL UNIQUE ,
   ime               VARCHAR(30) NOT NULL,
   prezime           VARCHAR(30) NOT NULL,
   zvanje            VARCHAR(40) NOT NULL,
   primanja          NUMERIC(10,2) NOT NULL,
   CHECK (primanja > 3200.00)
   );
```

  - osigurava se da profesor prima barem minimalnu plaću

Primjer CHECK ograničenja

- ograničiti se mogu vrijednosti u određenom kolonama

```
CREATE TABLE nastavnik (
   id                INT PRIMARY KEY,
   nastavnik_sifra CHAR(10) NOT NULL UNIQUE ,
   ime               VARCHAR(30) NOT NULL,
   prezime           VARCHAR(30) NOT NULL,
   zvanje            VARCHAR(40) NOT NULL,
   primanja          NUMERIC(10,2) NOT NULL,
   CHECK (primanja > 3200.00),
   CHECK (zvanje IN
              ('asistent', 'viši asistent', 'docent' ,…))
   );
```

Primjer CHECK ograničenja

- predaje relacija

```
CREATE TABLE predaje (
   nastavnik_id INT,
   kolegij_sifra CHAR(6),
   PRIMARY KEY (nastavnik_id, kolegij_sifra),
   CHECK (nastavnik_id IN
      (SELECT id FROM nastavnik))
);
```

  - tablica može sadržavati samo nastavnike koji su uneseni u tablicu nastavnik
  - ovo je primjer ograničenja *referencijalnog integriteta*

Primjer CHECK ograničenja

- predaje relacija

```
CREATE TABLE predaje (
   nastavnik_id INT,
   kolegij_sifra CHAR(6),
   PRIMARY KEY (nastavnik_id, kolegij_sifra),
   CHECK (nastavnik_id IN
      (SELECT id FROM nastavnik))
);
```

  - kada se ovo ograničenje mora provjeravati?
    - kada se rade promjene na *predaje* tablici
    - također kada se rade promjene na *nastavnik* tablici

CHECK ograničenje

- vrlo je jednostavno pisati zahtjevna `CHECK` ograničenja
- `CHECK` ograničenja se ne koriste često
  - nije široko podržano; korištenje ograničava prenosivost
  - kada se koriste, u pravilu su jednostavni izrazi
    - kao u primjerima
  - ne koristi se radi utjecaja na performanse sustava
- `CHECK` ograničenje se ne koristi za osiguravanje referencijalnog integriteta
  - postoji bolji način

Ograničenja referencijalnog integriteta

- ograničenja referencijalnog integriteta su vrlo bitna
  - ograničenja obuhvaćaju više tablica
  - omogućavaju povezivanje podataka između tablica
  - vrijednosti u jednoj tablici su ograničena vrijednostima iz druge tablice
- relacija može (mora) definirati primarni ključ
  - skup atributa koji jedinstveno definiraju svaku n-torku
- relacija također može sadržavati primarni ključ neke druge relacije
  - naziva se *strani ključ*
  - vrijednosti stranog ključa mora postojati u referenciranoj relaciji

Ograničenja referencijalnog integriteta

- ako je dana relacija *r(R)*
  - $K \subseteq R$ je primarni ključ za *R*
- druga relacija *s(S)* referencira *r*
  - $K \subseteq S$
  - $\langle \forall t_s \in s : \exists t_r \in r : t_s[K] = t_r[K] \rangle$
- odnos se naziva i zavisnost podskupa
  - $\Pi_K(s) \subseteq \Pi_K(r)$
  - vrijednosti stranog ključa u *s* moraju biti podskup vrijednosti primarnog ključa u *r*

Ograničenja stranog ključa

- kao i kod primarnog, deklaracija stranog ključa je moguća na više načina
- stani ključ od jednog stupca se može navesti zajedno s deklaracijom stupca

```
CREATE TABLE predaje (
   nastavnik_id INT REFERENCES nastavnik,
   kolegij_sifra CHAR(6) REFERENCES kolegij,
   PRIMARY KEY (nastavnik_id, kolegij_sifra));
```

- ograničenje stranog ključa ne podrazumijeva `NOT NULL`
  - ukoliko je potrebno mora se eksplicitno definirati
  - u navedenom slučaju, ograničenje primarnog ključa eliminira `NULL` vrijednosti

Ograničenja stranog ključa

- prilikom deklaracije može se navesti stupac koji se referencira
- korisno kada se referencira kandidat-ključ, koji nije primarni
- primjer:
  - nastavnik relacija ima ID i OIB; oba podatka su `UNIQUE`
  - postoji relacija (npr. zdravstveno osiguranje) koje koristi OIB za povezivanje s relacijom nastavnik
- povezivanje relacija s kandidat-ključem koji nije primarni treba oprezno koristiti

Strani ključ primjer

- tablica nastavnik

```
CREATE TABLE nastavnik (
   id        INT PRIMARY KEY
   OIB       CHAR(11) NOT NULL UNIQUE,
   ime       VARCHAR(30) NOT NULL,
   prezime   VARCHAR(30) NOT NULL,
   …);
```

- tablica zdravstveno_osiguranje

```
CREATE TABLE zdravstveno_osiguranje (
   OIB      CHAR(11) PRIMARY KEY REFERENCES nastavnik (OIB),
   sifra    CHAR(9) NOT NULL UNIQUE,
   …);
```

Višestruka ograničenja

- različita ograničenja se mogu kombinirati

```
OIB   CHAR(11) PRIMARY KEY
   REFERENCES nastavnik (OIB)
```

  - OIB je primarni ključ relacije *zdravstveno_osiguranje*
  - OIB je također strani ključ prema *nastavnik* relaciji
  - strani ključ referencira ključ-kandidat *nastavnik.OIB*

Samo-referencirajući strani ključevi

- relacija može imati strani ključ može referencira atribut iste relacije

```
CREATE TABLE nastavnik (
   id        INT PRIMARY KEY
   OIB       CHAR(11) NOT NULL UNIQUE,
   ime       VARCHAR(30) NOT NULL,
   prezime   VARCHAR(30) NOT NULL,
   …
   asistent INT REFERENCES nastavnik);
```

  - nastavnik id i asistent imaju istu domenu – skup valjanih ID-eva nastavnika
  - dozvoljavamo `NULL` za nastavnike koji nemaju asistente

Sintaksa

- strani ključ se može navesti nakon specifikacije svih stupaca tablice
  - nužno za ključeve s više stupaca
- primjer

```
CREATE TABLE nastavnik (
id          INT
OIB         CHAR(11) NOT NULL,
ime         VARCHAR(30) NOT NULL,
…
asistent INT,
PRIMARY KEY (id),
UNIQUE (OIB),
FOREIGN KEY (asistent) REFERENCES nastavnik);
```

Primjer

- tablica *nastavnik*

```
CREATE TABLE nastavnik (
  id        INT PRIMARY KEY
  OIB       CHAR(11) NOT NULL UNIQUE,
  ime          VARCHAR(30) NOT NULL, …);
```

- tablica *predaje*

```
CREATE TABLE predaje (
   nastavnik_id INT,
   kolegij_sifra CHAR(6),
   PRIMARY KEY (nastavnik_id, kolegij_sifra),
   FOREIGN KEY (nastavnik_id) REFERENCES nastavnik,
   FOREIGN KEY (kolegij_sifra) REFERENCES kolegij
  );
```

Kršenje stranog ključa

- postoji nekoliko načina za kršenje ograničenja stranog ključa
- ako referencirajuća relacija dobije pogrešnu vrijednost stranog ključa, operacija jednostavno nije dopuštena
  - ako se u tablicu *predaje* pokušava dodati red koji ima vrijednost *nastavnik_id* od nepoznatog nastavnika (nije evidentiran u *nastavnik* tablici)
  - ako se u tablicu *predaje* pokušava dodati red koji ima vrijednost *kolegij_sifra* od nepoznatog kolegija (nije evidentiran u *kolegij* tablici)
- kada se rade promjene u referenciranoj relaciji
  - ako se iz tablice nastavnik ukloni red koji je referenciran u tablici predaje?

Primjer - nastavak

- *nastavnik* podaci

```
+----+-------------+------------+----------------------+--------------+----------+----------+
| id | prezime     | ime        | zvanje               | fakultet     | primanja | grad     |
+----+-------------+------------+----------------------+--------------+----------+----------+
…
| 10 | Petrić      | Gabriel    | viši asistent        | Književnost  |  9399.60 | Zagreb   |
| 11 | Bilić       | Lucija     | docent               | Informatika  |  8279.25 | Đakovo   |
| 12 | Bašić       | Michele    | redoviti profesor    | Kemija       | 11025.00 | Sisak    |
| 13 | Dujmović    | Martino    | redoviti profesor    | Književnost  | 11760.00 | Zagreb   |
…
```

- *predaje* podaci

```
+-------------+---------------+
| nastavnik_id | kolegij_naziv |
+-------------+---------------+
…
| 10           | MR            |
| 12           | PIS           |
| 12           | SPI           |
| 13           | MR            |
…
```

Što bi se dogodilo da pobrišemo nastavnika s *id*-em 10 iz tablice *nastavnik*?

Kršenje stranog ključa

- opcija 1: zabraniti brisanje reda iz tablice nastavnik
  - korisnik treba pobrisati prvo sve redove iz relacije predaje koje referenciraju nastavnika
  - potom se briše red iz relacije nastavnik
  - zadano SQL ponašanje, ponekad može biti zamorno ali je dobro rješenje
- opcija 2: kaskadno (*engl. cascade*) brisanje podataka
  - ako korisnik obriše red iz tablice nastavnik brišu se i svi podaci iz istih ili drugih relacija koje referenciraju taj red
  - razumna opcija; stavke računa nemaju smisla ostati u bazi podataka ukoliko je račun obrisan
    - slučaj s profesorom i asistentima?

Kršenje stranog ključa

- opcija 3: postavljanje stranog ključa na `NULL`
  - ako primarni ključ nestane, svi redovi koji referenciraju taj red se postavljaju na `NULL`
  - strani ključ ne smije imati `NOT NULL` ograničenje
  - nema smisla u nekim slučajevima
    - u slučaju nastavnik – predaje?
- opcija 4: postavljanje stranog ključa na zadanu vrijednost
  - može se navesti zadana vrijednost za stupac

```
grad varchar(255) DEFAULT 'Zagreb'
```

Upravljanje promjenama

- ponašanje ograničenja stranog ključa se može definirati

```
CREATE TABLE predaje (
    …
    FOREIGN KEY (nastavnik_id) REFERENCES nastavnik
                               ON DELETE CASCADE
    FOREIGN KEY (kolegij_sifra) REFERENCES kolegij
                                ON DELETE CASCADE);
```

  - čitaj: "Kada se pobriše red u referenciranoj relaciji tada se brišu i odgovarajući redovi iz ove relacije!"
- ponašanje se može definirati i `ON UPDATE`
  - kada se vrijednost primanog ključa mijenja u referenciranoj relaciji

Zadatak

- završite SQL DDL definiciju sveučilišne baze podataka tako da uključite relacije *student*, *takes*, *advisor* i *prereq*. (*zadatak 4.6.*)

```
CREATE TABLE classroom
(building VARCHAR(15),
room_number VARCHAR(7),
capacity NUMERIC(4,0),
PRIMARY KEY (building, room_number))
```

```
CREATE TABLE instructor
```

```
CREATE TABLE department
```

```
CREATE TABLE section
```

```
CREATE TABLE course
```

Zadatak

- završite SQL DDL definiciju sveučilišne baze podataka tako da uključite relacije *student*, *takes*, *advisor* i *prereq*. (*zadatak 4.6.*)

```
CREATE TABLE classroom
(building VARCHAR(15),
room_number VARCHAR(7),
capacity NUMERIC(4,0),
PRIMARY KEY (building, room_number))
```

```
CREATE TABLE department
(dept_name VARCHAR(20),
building VARCHAR(15),
budget NUMERIC(12,2) CHECK(budget > 0),
PRIMARY KEY (dept_name))
```

```
CREATE TABLE course
(course_id VARCHAR(8),
title VARCHAR(50),
dept_name VARCHAR(20),
credits NUMERIC(2,0) CHECK (credits > 0),
PRIMARY KEY(course_id),
FOREIGN KEY(dept_name) REFERENCES department)
```

```
CREATE TABLE instructor
(ID VARCHAR(5),
name VARCHAR(20), NOT NULL
dept_name VARCHAR (20),
salary NUMERIC(8,2), CHECK (salary > 29000),
PRIMARY KEY (ID),
FOREIGN KEY (dept_name) REFERENCES department)
```

```
CREATE TABLE section
(course_id VARCHAR(8),
sec_id VARCHAR(8),
semester VARCHAR(6), CHECK (semester IN
        ('Fall', 'Winter', 'Spring', 'Summer'),
year NUMERIC(4,0), CHECK (year > 1759 and year < 2100)
building VARCHAR(15),
room_number VARCHAR(7),
time_slot_id VARCHAR(4),
PRIMARY KEY (course_id, sec_id, semester, year),
FOREIGN KEY (course_id) REFERENCES course,
FOREIGN KEY (building, room_number) REFERENCES classroom)
```

Zadatak

```
CREATE TABLE student
 (ID VARCHAR(5),
 name VARCHAR(20) NOT NULL,
 dept_name VARCHAR(20),
 tot_cred NUMERIC(3,0) CHECK (tot_cred >= 0),
 PRIMARY KEY (ID),
 FOREIGN KEY (dept_name) REFERENCES department ON DELETE SET NULL);


CREATE TABLE takes
(ID VARCHAR(5),
 course_id VARCHAR(8),
 section_id VARCHAR(8),
 semester VARCHAR(6),
 year NUMERIC(4,0),
 grade VARCHAR(2),
 PRIMARY KEY (ID, course_id, section_id, semester, year),
 FOREIGN KEY (course_id, section_id, semester, year) REFERENCES section ON DELETE CASCADE,
 FOREIGN KEY (ID) REFERENCES student ON DELETE CASCADE);
```

Zadatak

```
CREATE TABLE advisor
 (i_id VARCHAR(5),
 s_id VARCHAR(5),
 PRIMARY KEY (s_ID),
 FOREIGN KEY (i_ID) REFERENCES instructor (ID) ON DELETE SET NULL,
 FOREIGN KEY (s_ID) REFERENCES student (ID) ON DELETE CASCADE);


CREATE TABLE prereq
 (course_id VARCHAR(8),
 prereq_id VARCHAR(8),
 PRIMARY KEY (course_id, prereq_id),
 FOREIGN KEY (course_id) REFERENCES course ON DELETE CASCADE,
 FOREIGN KEY (prereq_id) REFERENCES course);
```

Zadatak

- razmotrite relacijsku bazu podatka navedenu u nastavku
- napišite SQL DDL definiciju baze podataka
- identificirajte ograničenja referencijalnog integriteta koja bi trebala vrijediti i uključite ih u DDL definiciju (*zadatak 4.7*)

employee (employee_name, street, city)
works (employee_name, company_name, salary)
company (company_name, city)
manages (employee_name, manager_name)

Zadatak

```
CREATE TABLE employee
 (person_name CHAR(20),
 street CHAR(30),
 city CHAR(30),
 PRIMARY KEY (person_name) )
```

```
CREATE TABLE works
 (person_name CHAR(20),
 company_name CHAR(15),
 salary INTEGER,
 PRIMARY KEY (person_name))
```

```
CREATE TABLE company
 (company_name CHAR(15),
 city CHAR(30),
 PRIMARY KEY (company_name))
```

```
CREATE TABLE manages
 (person_name CHAR(20),
 manager_name CHAR(20),
 PRIMARY KEY (person_name))
```

Zadatak

```
CREATE TABLE employee
 (person_name CHAR(20),
 street CHAR(30),
 city CHAR(30),
 PRIMARY KEY (person_name) )
```

```
CREATE TABLE works
 (person_name CHAR(20),
 company_name CHAR(15),
 salary INTEGER,
 PRIMARY KEY (person_name),
 FOREIGN KEY (person_name) REFERENCES employee,
 FOREIGN KEY (company_name) REFERENCES company)
```

```
CREATE TABLE company
 (company_name CHAR(15),
 city CHAR(30),
 PRIMARY KEY (company_name))
```

```
CREATE TABLE manages
 (person_name CHAR(20),
 manager_name CHAR(20),
 PRIMARY KEY (person_name),
 FOREIGN KEY (person_name) REFERENCES employee,
 FOREIGN KEY (manager_name) REFERENCES employee)
```

Zadatak

- SQL omogućava da strani ključ referencira istu relaciju, kao u primjeru:

```
CREATE TABLE manager
(employee_name VARCHAR(20) NOT NULL,
 manager_name VARCHAR(20) NOT NULL,
 PRIMARY KEY employee_name,
 FOREIGN KEY (manager_name) REFERENCES manager ON DELETE CASCADE )
```

- primarni ključ je *employee_name*
- strani ključ zahtjeva da je svaki menadžer ujedno i zaposlenik
- što se dogodi kada se obriše red u tablici *manager*?

Literatura

- Pročitati
  - [DSC] poglavlje 3.2.2.
  - [DSC] poglavlje 4.4.
  - Caltech CS121 - 7
  - **Vježbe: [DSC] Poglavlje 4**
- Slijedeće predavanje
  - [DSC] poglavlje 4.5.
  - Caltech CS121 - 8
  - **Vježbe: [DSC] Poglavlje 4**