

# Ponavljanje I

*Napredni SQL*

BAZE PODATAKA II

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

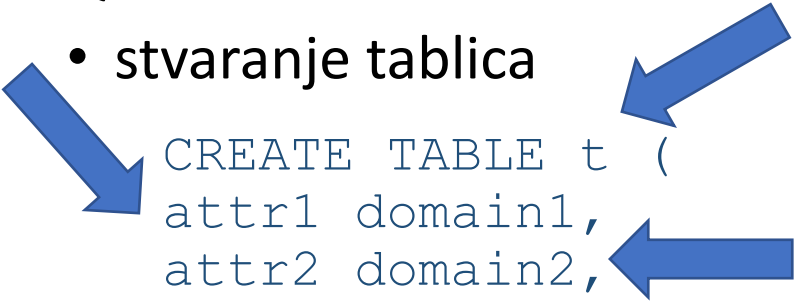
# Sadržaj

- ponavljanje prethodnog gradiva
  - osnove SQL-a
    - stvaranje tablica
    - domene
    - dodavanje redova
  - SELECT naredba
    - dijelovi naredbe
    - primjer
- sortiranje rezultata
- grupiranje i agregiranje rezultata
- filtriranje rezultata
- ugniježđeni podupiti
- NULL vrijednosti u SQL-u
- join operacije
  - theta join, outer join, natural join
- pogledi

# Ponavljjanje

- **SQL osnovne**

- stvaranje tablica



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

- tipovi domena

CHAR(N), VARCHAR(N), INT, NUMERIC(P, D), DOUBLE PRECISION, ...

- dodavanje redova

```
INSERT INTO nastavnik  
VALUES ('I-337', 'Marić', 2876.10);
```

# Ponavljjanje

- naredba **SELECT**

- sintaksa

```
SELECT A1, A2, ...  
FROM r1, r2, ...  
WHERE P;
```

- dijelovi upita i njihova sličnost relacijskoj algebri

```
SELECT A1, A2, ... ->  
FROM r1, r2, ... ->  
WHERE P; ->
```

# Ponavljjanje

- naredba **SELECT**

- sintaksa

```
SELECT A1, A2, ...  
FROM r1, r2, ...  
WHERE P;
```

- dijelovi upita i njihova sličnost relacijskoj algebri

```
SELECT A1, A2, ... -> projekcija  
FROM r1, r2, ... ->  
WHERE P; ->
```

# Ponavljjanje

- naredba **SELECT**

- sintaksa

```
SELECT A1, A2, ...  
FROM r1, r2, ...  
WHERE P;
```

- dijelovi upita i njihova sličnost relacijskoj algebri

```
SELECT A1, A2, ... -> projekcija  
FROM r1, r2, ...   -> Kartezijev produkt  
WHERE P;            ->
```

# Ponavljjanje

- naredba **SELECT**

- sintaksa

```
SELECT A1, A2, ...  
FROM r1, r2, ...  
WHERE P;
```

- dijelovi upita i njihova sličnost relacijskoj algebri

```
SELECT A1, A2, ... -> projekcija  
FROM r1, r2, ...   -> Kartezijev produkt  
WHERE P;            -> selekcija
```

# Ponavljjanje

- naredba **SELECT**

- sintaksa

```
SELECT A1, A2, ...  
FROM r1, r2, ...  
WHERE P;
```

- dijelovi upita i njihova sličnost relacijskoj algebri

```
SELECT A1, A2, ... -> projekcija  
FROM r1, r2, ...   -> Kartezijev produkt  
WHERE P;            -> selekcija
```

- primjer upita

```
SELECT nastavnik_sifra FROM nastavnik  
WHERE prezime = 'Marić';
```



# SQL upiti

- upiti u SQL-u se zadaju pomoću naredbe `SELECT`
- općenita forma `SELECT` naredbe:

```
SELECT A1, A2, ...  
FROM r1, r2, ...  
WHERE P;
```

- $r_i$  su relacije (tablice)
  - $A_i$  su atributi (stupci)
  - $P$  je predikat selekcije
- ekvivalentno izrazu:  $\Pi_{A_1, A_2, \dots}(\sigma_P(r_1 \times r_2 \times \dots))$

# Sortiranje rezultata

- rezultati SQL upita mogu biti sortirani prema atributima
- prema sortiranju razlikuju se sljedeći upiti
  - upiti bez korištenja sortiranja
    - n-torke se pojavljuju u neodređenom redoslijedu
  - upiti sortirani prema atributima  $A_1, A_2, \dots$ 
    - n-torke su sortirane prema zadanim atributima
    - rezultati su prvo sortirani prema  $A_1$
    - unutar svake vrijednosti  $A_1$ , rezultati su sortirani prema  $A_2$
    - itd.
- sortiranje se zadaje pomoću ključne riječi `ORDER BY` na kraju `SELECT` naredbe

# Sortiranje rezultata

- stvorena je i popunjena tablica definirana prema shemi:

*nastavnik(id, ime, prezime, zvanje, fakultet, primanja)*

- *pronadite prezime i primanja nastavnika fakulteta fizike*

```
SELECT prezime, primanja FROM nastavnik  
WHERE fakultet='Fizika';
```

- sortirajte podatke prema primanjima uzlazno

```
SELECT prezime, primanja FROM nastavnik  
WHERE fakultet='Fizika'  
ORDER BY primanja;
```

- zadano sortiranje je uzlazno (*engl. ascending*)

| prezime   | primanja |
|-----------|----------|
| Vidaković | 11805.00 |
| Topić     | 15223.00 |
| Špoljarić | 13369.00 |
| Barić     | 11805.00 |

| prezime   | primanja |
|-----------|----------|
| Vidaković | 11805.00 |
| Barić     | 11805.00 |
| Špoljarić | 13369.00 |
| Topić     | 15223.00 |

# Sortiranje rezultata

- da bi se odredio smjer sortiranja dodaje se ključna riječ ASC ili DESC nakon naziva atributa
  - ASC je zadano ali može poboljšati čitljivost izraza
- kada se navodi više atributa svaki atribut se može sortirati po drugom smjeru
  - *ispišite prezimena svih nastavnika i fakultete na kojima su zaposleni tako da su fakulteti navedeni silazno a prezimena uzlazno*

```
SELECT prezime, fakultet FROM nastavnik  
ORDER BY fakultet DESC, prezime ASC;
```

| +-----+-----+ |             |
|---------------|-------------|
| prezime       | fakultet    |
| +-----+-----+ |             |
| Bašić         | Strojarstvo |
| Horvat        | Strojarstvo |
| Jurišić       | Strojarstvo |
| Tadić         | Strojarstvo |
| Galić         | Medicina    |
| Jelić         | Medicina    |

...

# Funkcije agregacije u SQL

- SQL pruža operacije grupiranja i funkcije agregacije kao i relacijska algebra
- funkcije agregacije
  - SUM** zbraja sve vrijednosti u kolekciji
  - AVG** računa prosjek svih vrijednosti u kolekciji
  - COUNT** računa broj elemenata u kolekciji
  - MIN** vraća minimalnu vrijednost iz kolekcije
  - MAX** vraća maksimalnu vrijednost iz kolekcije
- za SUM i AVG ulazni podaci moraju biti numerički

# Primjer agregiranja

- *pronadite prosječna primanja profesora na medicini*

```
SELECT AVG(primanja) FROM nastavnik  
WHERE fakultet = 'Medicina';
```

```
+-----+  
| AVG(primanja) |  
+-----+  
|    9270.600000 |  
+-----+
```

- *pronadite profesora s najvišim primanjem na fakultetu strojarstva*

```
SELECT MAX(primanja) AS max_primanja FROM nastavnik  
WHERE fakultet = 'Strojarstvo';
```

```
+-----+  
| max_primanja |  
+-----+  
|    15264.00 |  
+-----+
```

- rezultat se može preimenovati kao u primjeru

# Primjer agregiranja

- *pronadite prezime osobe s najvišim primanjem na sveučilištu*
- slijedeći upit nije ispravan

```
SELECT prezime, MAX(primanja)  
FROM nastavnik;
```

- funkcije agregacije računaju jednu vrijednost iz multiskupa na ulazu
  - nema smisla kombinirati individualne attribute s funkcijama agregacije kao u primjeru
- u nastavku ćemo se vratiti na primjer

# Eliminiranje duplikata

- ponekad je potrebno eliminirati duplikate u SQL upitima
  - kao i do sada, može se koristiti ključna riječ `DISTINCT` za izbacivanje duplikata
- primjer:
  - *pronadite ukupan broj fakulteta koji zapošljavaju profesore na sveučilištu*  
`SELECT COUNT(fakultet) FROM nastavnik; --rezultat 30`
  - ne daje dobar rezultat jer postoje fakulteti s više zaposlenih profesora  
`SELECT COUNT(DISTINCT fakultet) FROM nastavnik; --rezultat 6`
  - duplikati su eliminirani iz ulaznog multiskupa prije nego što se primijeni funkcija agregacije



# Funkcija COUNT

- prebrojavati se mogu vrijednosti pojedinačnog atributa
  - `COUNT (fakultet)`
  - `COUNT (DISTINCT fakultet)`
- prebrojavati se može i ukupan broj n-torki
  - `COUNT (*)`
    - ako se koristi s grupiranjem, prebrojava se ukupan broj n-torki u grupi
    - ako se koristi bez grupiranja, prebrojava se ukupan broj n-torki
- prebrojavanje vrijednosti pojedinačnog atributa je korisno
  - kada se traži broj (različitih) vrijednosti nekog atributa
  - slučaj kada ulazni podaci u multiskupu mogu biti `NULL`
    - `COUNT` ignorira `NULL` vrijednosti

# Grupiranje i agregiranje

- SQL omogućavanja grupiranje na podacima relacije prije korištenja funkcije za agregaciju
  - navodi se ključna riječ **GROUP BY**  $A_1, A_2, \dots$  na kraju upita
- primjer:
  - *pronađite prosječnu plaću pojedinog fakulteta koji zapošljava profesore*

```
SELECT fakultet, AVG(primanja) AS p_primanja  
FROM nastavnik GROUP BY fakultet;
```

- prvo se n-torke u relaciji *nastavnik* grupiraju po atributu *fakultet*
- potom se primjenjuje funkcija agregacije na pojedinu grupu

| +-----+-----+ |              |
|---------------|--------------|
| fakultet      | p_primanja   |
| +-----+-----+ |              |
| Književnost   | 9524.777778  |
| Fizika        | 13050.500000 |
| Informatika   | 10573.200000 |
| Strojarsstvo  | 11282.250000 |
| Kemija        | 11595.666667 |
| Medicina      | 9270.600000  |
| +-----+-----+ |              |

# Grupiranje i agregiranje

- grupirati se može i po više atributa
  - svaka grupa ima jedinstvene vrijednosti grupiranih atributa u skupu grupiranih atributa
- primjer:
  - *koliko ima pojedinih imena na pojedinom fakultetu?*
  - potrebno je grupirati fakultete i imena
  - potom prebrojati n-torke u svakoj grupi
  - upit?

```
SELECT fakultet, ime, COUNT(*) FROM nastavnik  
GROUP BY fakultet, ime;
```

# Grupiranje i agregiranje

- postoji razlika između sintakse relacijske algebre i SQL-a
- sintaksa relacijske algebre:

- $G_1, G_2, \dots, G_n \mathcal{G}_{F_1(A_1), F_2(A_2), \dots, F_m(A_m)}(E)$ 
  - atribut za grupiranje se pojavljuju samo na lijevo od  $\mathcal{G}$

- SQL sintaksa

```
SELECT  $G_1, G_2, \dots, F_1(A_1), F_2(A_2), \dots$   
FROM  $r_1, r_2, \dots$  WHERE P  
GROUP BY  $G_1, G_2, \dots$ 
```

- atributi prema kojima se grupira se često navode u SELECT i GROUP BY dijelu upita

# Grupiranje i agregiranje

- SQL ne zahtijeva da se atributi za grupiranje navode u `SELECT` dijelu upita

- jedini je uvjet da se takvi atributi navode nakon `GROUP BY` ključne riječi
- npr. ukoliko su potrebni samo agregirani podaci:

```
SELECT  $F_1(A_1)$  ,  $F_2(A_2)$  , ...  
FROM  $r_1$  ,  $r_2$  , ... WHERE P  
GROUP BY  $G_1$  ,  $G_2$  , ...
```

- korištenje grupiranja i agregiranja se može kombinirati s izrazima
- vrlo neuobičajeno ali ispravno

```
SELECT  $MIN(a+b) - MIN(c)$   
FROM  $t$  GROUP BY  $d * e$ ;
```

# Usporedba znakovnog niza i GROUP BY

- usporedba znakovnog niza korištenjem = nije case-sensitive po zadanim postavkama

```
SELECT 'HELLO' = 'hello'; --vraća istinu
```

- to može dovesti do neočekivanih poteškoća kod grupiranja i agregiranja
- bez obzira na *case* u znakovima iste riječi će se grupirati u jednu grupu
- primjer:

```
SELECT fakultet, AVG(primanja) AS p_primanja  
FROM nastavnik GROUP BY fakultet;
```

- npr. "ekonomija" i "Ekonomija" će završiti u istoj grupi
  - nekada je to poželjno a nekada ne

# Filtriranje rezultata

- prilikom izvršavanja upita, WHERE se primjenjuje prije grupiranja

```
SELECT  $G_1, G_2, \dots, F_1(A_1), F_2(A_2), \dots$   
FROM  $r_1, r_2, \dots$  WHERE P  
GROUP BY  $G_1, G_2, \dots$ 
```

- prevodi se u izraz relacijske algebre
  - $\Pi_{\dots(G_1, G_2, \dots, G_{F_1(A_1), F_2(A_2), \dots})}(\sigma_P(r_1 \times r_2 \times \dots))$
- WHERE ograničava skup n-torki na kojima se primjenjuje grupiranje i agregiranje
  - redoslijed izvršavanja *WHERE – GROUP BY – FUNKCIJE AGREGACIJE*

# Filtriranje rezultata

- da bi se rezultati filtrirali nakon grupiranja i agregiranja potrebno je koristiti ključnu riječ `HAVING`
  - funkcionira identično kao i `WHERE` samo se primjenjuje nakon grupiranja i agregiranja

```
SELECT  $G_1, G_2, \dots, F_1(A_1), F_2(A_2), \dots$   
FROM  $r_1, r_2, \dots$  WHERE  $P_W$   
GROUP BY  $G_1, G_2, \dots$   
HAVING  $P_H$ 
```

- prevodi se u:
  - $\Pi_{\dots}(\sigma_{P_H}(G_1, G_2, \dots \mathcal{G}_{F_1(A_1), F_2(A_2), \dots}(\sigma_{P_W}(r_1 \times r_2 \times \dots))))$



# Ključna riječ HAVING

- HAVING može koristiti funkcije agregacije u predikatu
  - primjenjuje se nakon agregiranja stoga su te vrijednosti dostupne
  - WHERE iz očitog razloga to ne može
- primjer:
  - *koliko ima pojedinih imena na pojedinom fakultetu koja se pojavljuju više od jednom?*

```
SELECT fakultet, ime, COUNT(*) FROM nastavnik  
      GROUP BY fakultet, ime  
      HAVING COUNT(*)>1;
```

|   |          |        |          |
|---|----------|--------|----------|
| + | -----+   | -----+ | -----+   |
|   | fakultet | ime    | COUNT(*) |
| + | -----+   | -----+ | -----+   |
|   | Medicina | Mislav | 2        |
| + | -----+   | -----+ | -----+   |

# Ugniježđeni podupiti

- SQL pruža široku podršku za korištenje ugniježđenih podupita
  - SQL upit je izraz tipa "select – from – where"
  - ugniježđeni podupit je izraz tipa "select – from – where" umetnut unutar drugog upita
  - upit može biti umetnut unutar `WHERE` dijela
    - složena selekcija
  - upit može biti umetnut unutar `FROM` dijela
    - zadavanje upita na generiranoj relaciji
  - upit može biti umetnut unutar `SELECT` dijela
    - od SQL:2003 standarda, mnoge baze podržavaju
    - omogućava jednostavnije pisanje nekih upita, može usporiti izvođenje

# Tipovi podupita

- podupiti mogu kao rezultat vratiti jednu vrijednost  
`SELECT MAX(primanja) FROM nastavnik;`
  - nazivaju se skalarni podupiti
  - to je relacija s jednim atributom i jednom n-torkom
- većina podupita kao rezultat vraća relaciju koja sadrži više n-torki
  - ugniježđeni upiti često produciraju relaciju s jednim atributom...
    - vrlo često vrijedi za podupite u `WHERE` dijelu upita
  - ili s više atributa
    - vrlo često vrijedi za podupite u `FROM` dijelu upita
    - ponekad i za upite u `WHERE` dijelu

# Podupiti unutar WHERE

- često korišteno za:
  - uspoređivanje s vrijednosti skalarnog podupita
  - provjera pripadnosti skupu: `IN`, `NOT IN`
  - provjera praznog skupa: `EXISTS`, `NOT EXISTS`
- ponekad se koristi za:
  - usporedbe sa skupom: `ANY`, `SOME`, `ALL`
  - testove jedinstvenosti: `UNIQUE`, `NOT UNIQUE`
- navedeni primjeri vrijede i za `HAVING` dio upita

# Usporedba s rezultatom skalarnog podupita

- vrijednosti se mogu uspoređivati sa skalarnim podupitima
- primjer:
  - *želimo pronaći fakultet s najnižom plaćom nastavnika*
  - minimalnu plaću možemo vrlo lagano pronaći

```
SELECT MIN(primanja) FROM nastavnik;
```

- navedeni upit ćemo koristiti kao podupit unutar WHERE dijela

```
SELECT fakultet FROM nastavnik  
WHERE primanja = (SELECT MIN(primanja) FROM nastavnik);
```

```
+-----+  
| fakultet |  
+-----+  
| Književnost |  
+-----+
```

# Provjera pripadnosti skupu

- za provjeru pripadnosti skupu koriste se `IN (...)` i `NOT IN (...)` ključne riječi
- primjer:
  - *pronadite prezimena koja se pojavljuju u relaciji nastavnik i student*
    - za zadatak se može koristiti `INTERSECT` operacija
  - isti problem se može riješiti korištenjem podupita...
  - *pronadite prezimena u relaciji nastavnik koja se pojavljuju negdje u relaciji student*

```
SELECT DISTINCT prezime FROM nastavnik
WHERE prezime IN (SELECT prezime FROM student);
```

# Provjera pripadnosti skupu

- `IN (...)` i `NOT IN (...)` se mogu koristiti za podupite koji kao rezultat vraćaju više atributa

- *pronadite n-torke profesora s najvišim primanjima na svakom fakultetu*

- prvi korak potrebno je pronaći najveća primanja na svakom fakultetu

```
SELECT fakultet, MAX(primanja) FROM nastavnik GROUP BY fakultet;
```

- drugi korak, koristimo rezultate iz prvog koraka za dohvat preostalih podataka

```
SELECT * FROM nastavnik  
WHERE (fakultet, primanja) IN  
(SELECT fakultet, MAX(primanja) FROM nastavnik GROUP BY  
fakultet);
```

# Provjera praznog skupa

- za testiranje postojanja bilo kojeg zapisa u podupitu koriste se

`EXISTS (...)`

`NOT EXISTS (...)`

- primjer:

- *pronađite studente koji nemaju isto prezime kao bilo koji profesor na sveučilištu*

```
SELECT DISTINCT prezime  
FROM student s WHERE NOT EXISTS  
(SELECT * FROM nastavnik n WHERE n.prezime = s.prezime);
```

- rezultati uključuju sva prezimena koja se pojavljuju u tablici student a ne pojavljuju se u tablici nastavnik



# Provjera praznog skupa

- *pronadite studente koji nemaju isto prezime kao bilo koji profesor na sveučilištu*

```
SELECT DISTINCT prezime  
  FROM student s WHERE NOT EXISTS  
    (SELECT * FROM nastavnik n WHERE n.prezime = s.prezime);
```

- unutarnji upit referencira attribute iz relacije vanjskog upita
- ugniježđeni podupiti mogu referencirati relaciju iz vanjskog upita, kao u primjeru
- obrtno referenciranje nije dozvoljeno

# Korelirani podupit

- kada ugniježđeni podupit referencira attribute iz relacije vanjskog upita, to se naziva **korelirani podupit**
- unutarnji upit se izvršava jednom za svaki red iz vanjskog upita
- u većini slučajeva treba izbjegavati!
  - vrlo spori upiti
- osim u WHERE dijelu, mogu se pojaviti i u SELECT dijelu upita

```
SELECT prezime, fakultet, (SELECT AVG(primanja) FROM  
    nastavnik WHERE fakultet = n.fakultet) AS fakultet_prosjek  
FROM nastavnik n;
```

# Korelirani podupit

- mnogi korelirani podupiti se mogu drugačije definirati koristeći join ili Kartezijev produkt
  - u većini slučajeva join operacija je puno brža
  - napredniji DBMS-ovi će automatski dekorelirati takve upite (ali ne svi!)
- određeni uvjeti unutar upita, npr. `EXISTS` ili `NOT EXISTS`, najčešće su indikator postojanja koreliranih podupita
- ukoliko ih je jednostavno dekorelirati, učinite to
- u suprotnom potrebno je testirati performanse upita
  - ponekad je dobro u izmijeniti pristup dohvata podataka

# Usporedbe sa skupom

- neka vrijednost se može usporediti sa skupom vrijednosti
  - da li je veća, manja, ... od neke vrijednosti u skupu
- primjer
  - *pronadite sve profesore koji imaju plaću veću od barem jednog nastavnika s medicinskog fakulteta*

```
SELECT * FROM nastavnik
WHERE primanja > SOME(
  SELECT primanja FROM nastavnik WHERE fakultet='Medicina');
```

# Usporedbe sa skupom

- opći oblik usporedbe

***atribut*** *operacija\_usp* **SOME** ( *podupit* )

- može se koristiti bilo koja operacija usporedbe

= SOME je isto kao i IN

- ANY je sinonim za SOME

- vrijednost se može usporediti i sa svim vrijednostima u skupu

- koristi se ALL umjesto SOME

<> ALL je isto kao i NOT IN

# Usporedbe sa skupom

- primjer:

- *pronadite profesore s primanjima koja su veća od primanja svih profesora s medicinskog fakulteta*

```
SELECT * FROM nastavnik  
WHERE primanja > ALL(  
    SELECT primanja FROM nastavnik WHERE fakultet='Medicina');
```

- isto se može napisati i sa skalarnim podupitom

```
SELECT * FROM nastavnik WHERE primanja > (  
    SELECT MAX(primanja) FROM nastavnik WHERE  
    fakultet='Medicina');
```

# Test jedinstvenosti

- korištenjem se može provjeriti dali podupit generira duplikate

`UNIQUE (...)`

`NOT UNIQUE (...)`

- nisu svugdje implementirani

- zahtjevna operacija

- mogu se preformulirati na mnogo načina npr.

`GROUP BY ... HAVING COUNT (*) = 1`

`GROUP BY ... HAVING COUNT (*) > 1`

# Podupiti u FROM dijelu upita

- često se rezultati moraju računati u više koraka
- u SQL-u je moguće vršiti upite na rezultatima podupita
  - to se naziva **izvedena relacija (tablica)**
  - koristi se kao obična relacija
- primjer primjene
  - `HAVING` ključna riječ se može implementirati pomoću ugniježđenog podupita u `FROM` dijelu



# HAVING vs ugniježđeni upit

- promijenimo shemu nastavnika; dodajmo atribut *grad* koji sadrži podatak o mjestu prebivanja profesora

- *pronađite gradove u kojima žive više od dva profesora*

```
SELECT grad, COUNT(*) AS broj_profesora  
FROM nastavnik GROUP BY grad HAVING COUNT(*) > 2;
```

- ili možemo napisati

```
SELECT grad, broj_profesora FROM (  
    SELECT grad, COUNT(*) FROM nastavnik GROUP BY grad)  
AS broj(grad, broj_profesora) WHERE broj_profesora > 2;
```

- grupiranje i agregiranje se izvršava u unutarnjem upitu
- vanjski upit radi selekciju podataka iz izvedene relacije

# Sintaksa izvedene relacije

- podupit unutar FROM dijela upita mora imati dodijeljeno ime
  - mnogi DBMS-ovi zahtijevaju i definiranje imena atributa

```
SELECT * FROM (  
    SELECT grad, COUNT(*) FROM nastavnik GROUP BY grad)  
    AS broj(grad, broj_profesora)  
    WHERE broj_profesora > 2;
```

- ugniježđeni upit se naziva **broj** i specificira dva atributa
- sintaksa varira od pojedinog DBMS-a
- \* unutar SELECT-a podrazumijeva oba atributa definirana unutar izvedene relacije *broj*

# Korištenje izvedene relacije

- češće je zadavanje upita na agregiranim vrijednostima
- primjer
  - *pronađite fakultet s najvećim ukupnim izdatcima za plaće na sveučilištu*
  - potrebno je izračunati ukupne izdatke za plaće za svaki fakultet...

```
SELECT fakultet, SUM(primanja) AS ukupni_izdatak  
FROM nastavnik GROUP BY fakultet;
```

- slijedeći korak je jednostavan...

```
SELECT MAX(ukupni_izdatak) AS max_izdatak  
FROM (SELECT fakultet, SUM(primanja)  
FROM nastavnik GROUP BY fakultet)  
AS primanja (fakultet, ukupni_izdatak);
```

# Agregiranje agregata

- potrebno je biti pažljiv kada se računaju agregacije više razine
  - *pronađite fakultet s najvećim ukupnim izdatcima za plaće na sveučilištu*
  - dvije agregacije: *max of sums*
- vrlo česta greška:

```
SELECT fakultet, SUM(primanja) AS ukupna_primanja
FROM nastavnik GROUP BY fakultet
HAVING ukupna_primanja = MAX(ukupna_primanja);
```

- SELECT upit može izvoditi samo jednu razinu agregiranja
- potreban je drugi SELECT za izračun najvećih ukupnih primanja
- neki sustavi će izvršiti ovaj upit i vratiti neispravne rezultate

# Dodatne operacije za manipulaciju podataka

- SQL pruža mnoštvo operacija za umetanje, izmjenu i brisanje podataka iz baze
- sve naredbe podržavaju sintaksu tipa SELECT
- u tablicu se mogu umetnuti pojedinačni zapisi...

```
INSERT INTO table VALUES (1, 'foo', 5);
```

- ili podaci kao rezultat upita

```
INSERT INTO table SELECT...;
```

- jedino ograničenje je da generirani rezultati moraju imati kompatibilnu shemu s tablicom u koju se pune

# Brisanje redova

- SQL DELETE naredba može koristiti WHERE ključnu riječ

`DELETE FROM table;`

- brišu se svi redovi iz tablice

`DELETE FROM table WHERE ...;`

- brišu se samo redovi koji zadovoljavaju uvjet
- WHERE može koristiti sve što podržava WHERE unutar SELECT-a
  - uključujući i ugniježdene upite

# Izmjena podataka

- za modificiranje postojećeg reda u tablici koristi se UPDATE naredba
- opći oblik naredbe

```
UPDATE table  
SET att1 = value1, att2 = value2,...  
WHERE condition;
```

- moraju se odrediti atributi kojima se mijenjaju podaci
- WHERE dio naredbe je opcionalan
  - ako se ne navede mijenjaju se svi redovi
- WHERE može sadržavati ugniježdene upite itd.

# Izmjena podataka

- unutar UPDATE-a mogu se zadavati aritmetički izrazi
  - aritmetički izrazi se mogu primjenjivati na bilo koji atribut relacije
- primjer
  - *dodajte 5% povišice svim profesorima s primanjima manjim od 12000 kuna*

```
UPDATE nastavnik  
  SET primanja = primanja * 1.05  
  WHERE primanja < 12000;
```



# Null vrijednost u SQL-u

- kao i u relacijskoj algebri, SQL reprezentira nedostajuće podatke s null
  - `NULL` je ključna riječ u SQL-u
  - kao i sve ključne riječi tipično se piše velikim slovima
- da bi se `NULL` vrijednosti provjerile koristi se `IS NULL` i `IS NOT NULL`
  - `attr = NULL` nije nikada istina! (nepoznato)
  - `attr <> NULL` isto nije nikada istina! (nepoznato)
- operacije agregiranja ignoriraju `NULL` vrijednost u ulaznim podacima
  - `COUNT` vraća 0 za prazni ulazni multiskup
  - ostale funkcije vraćaju `NULL` za prazan ulaz

# Null vrijednost u SQL-u

- relacijska algebra prepoznaje *unknown* istinitu vrijednost
  - true, false, unknown
  - nastaje usporedbom s NULL vrijednosti
- SQL ima test za nepoznate vrijednosti
  - comp IS UNKNOWN*
  - comp IS NOT UNKNOWN*
  - *comp* je operacija usporedbe

# NULL u INSERT i UPDATE naredbi

- NULL vrijednost se može koristiti prilikom umetanja ili izmjene redova u tablici

```
INSERT INTO nastavnik
```

```
VALUES ('1', 'Ćosić', 'Marko', 'asistent', NULL, '11760');
```

- treba izbjegavati jer može dovesti do različitih problema
- atributi primarnog ključa ne dozvoljavaju unos NULL vrijednosti
  - zašto?
- postoje načini da se ograniči unos NULL vrijednosti za željene attribute

# Dodatne join operacije

- SQL-92 je predstavio dodatne join operacije
  - natural join
  - left/right/full outer join
  - theta join
- theta join – relacijska algebra
  - generalizirana join operacija
    - ponekad se naziva i condition join
  - relacijska algebra:
    - $r \bowtie_{\theta} s$  ili  $\sigma_{\theta}(r \times s)$
    - ne sadržava projekciju
    - ne sadržava null dodane vrijednosti kao outer join

# SQL theta join

- SQL podržava operaciju theta join
- primjer
  - *povežite profesore i kolegije na kojima predaju*

```
SELECT * FROM nastavnik INNER JOIN predaje
ON nastavnik.id = predaje.nastavnik_id;
```

| id | prezime   | ime      | zvanje              | fakultet    | primanja | grad   | nastavnik_id | kolegij_naziv |
|----|-----------|----------|---------------------|-------------|----------|--------|--------------|---------------|
| 10 | Petrić    | Gabriel  | viši asistent       | Književnost | 9399.60  | Zagreb | 10           | MR            |
| 12 | Bašić     | Michele  | redoviti profesor   | Kemija      | 11025.00 | Sisak  | 12           | PIS           |
| 12 | Bašić     | Michele  | redoviti profesor   | Kemija      | 11025.00 | Sisak  | 12           | SPI           |
| 13 | Dujmović  | Martino  | redoviti profesor   | Književnost | 11760.00 | Zagreb | 13           | MR            |
| 16 | Šimunović | Nikolina | asistent            | Književnost | 9399.60  | Pula   | 16           | STAT          |
| 17 | Novak     | Patrick  | izvanredni profesor | Medicina    | 11025.00 | Pula   | 17           | STAT          |

...

# SQL theta join

- sintaksa se piše u FROM dijelu

```
table1 INNER JOIN table2 ON condition
```

- ključna riječ `INNER` je opcionalna; koristi se da bi se stvorila razlika u odnosu na outer join
- ne brišu se dupli atributi s istim imenom
  - mogu se specificirati naziv relacije i atributi

```
table1 INNER JOIN table2 ON condition  
AS rel (aat1, att2, ...)
```

- slično kao izvedene relacije

# Theta join na više tablica

- koristeći istu sintaksu može se spojiti više tablica
- primjer: join nastavnik, predaje, kolegij tablica

```
SELECT * FROM nastavnik AS n
      INNER JOIN predaje AS p
            ON n.id = p.nastavnik_id
      INNER JOIN kolegij AS k
            ON p.kolegij_naziv = k.kolegij_naziv;
```

- upiti se u pravilu izvode s lijeva na desno
- redoslijed se može odrediti korištenjem zagrada
- redoslijed u pravilu ne utječe na performanse (za razliku od outer join-a)

# Theta join na više tablica

- primjer: join nastavnik, predaje, kolegij tablica – *pokušaj 2*
  - jedan Kartezijev produkt i jedan theta join

```
SELECT * FROM nastavnik AS n
  JOIN predaje AS p JOIN kolegij AS k
    ON n.id = p.nastavnik_id
   AND p.kolegij_naziv = k.kolegij_naziv;
```

- rezultat jednak kao i u prošlom primjeru
- nisu dva theta join-a
- sustav će sam optimizirati ovakav upit



# Join uvjeti

- bilo koji uvjeti se mogu specificirati koristeći ključnu riječ `ON` (i ugniježđeni podupiti)
  - čak i uvjeti koji nisu vezani uz join
- smjernice:
  - koristite `ON` za join uvjete
  - koristite `WHERE` za selekciju redova
  - miješanje smjernica može dovesti do zabune

# Kartezijev produkt

- Kartezijev produkt se može definirati s CROSS JOIN
  - za CROSS JOIN nije dozvoljeno navoditi ON uvjete

- Kartezijev produkt tablica nastavnik i predaje

```
SELECT * FROM nastavnik CROSS JOIN predaje;
```

- isto kao i theta join bez ON uvjeta

```
SELECT * FROM nastavnik INNER JOIN predaje ON TRUE;
```

- ili jednostavnije

```
SELECT * FROM nastavnik JOIN predaje;
```

```
SELECT * FROM nastavnik, predaje;
```

# Outer join

- SQL podržava outer join

```
SELECT * FROM table1
    LEFT OUTER JOIN table2 ON condition;
SELECT * FROM table1
    RIGHT OUTER JOIN table2 ON condition;
SELECT * FROM table1
    FULL OUTER JOIN table2 ON condition;
```

- OUTER se podrazumijeva nakon LEFT/RIGHT/FULL i može se izostaviti

```
SELECT * FROM table1
    LEFT JOIN table2 ON condition;
```

# Zajednički atributi

- ponekad je shema tako dizajnirana da zajednički atributi imaju ista imena
  - npr. predaje.kolegij\_naziv i kolegij.kolegij\_naziv
- `USING` je pojednostavljeni oblik `ON` ključne riječi
  - `ON` sintaksa je nepotrebna za jednostavne join operacije
    - i omogućava uvjete koji bi trebali biti smješteni u `WHERE`
- otprilike isto kao

```
SELECT * FROM t1 LEFT OUTER JOIN t2
    USING (a1, a2, ...);
```

```
SELECT * FROM t1 LEFT OUTER JOIN t2
    ON (t1.a1 = t2.a1 AND t1.a2 = t2.a2 AND ...);
```

# Zajednički atributi

- `USING` eliminira duple join attribute
  - rezultat join-a s `USING ( a1, a2, ...)` će imati samo po jedan join atribut
  - eliminacija je moguća jer join podrazumijeva identične vrijednosti join atributa
- primjer: tablice  $r(a, b, c)$  i  $s(a, b, d)$   

```
SELECT * FROM r JOIN s USING (a) ;
```

  - shema rezultata je:  $a(a, r.b, c, s.b, d)$
- `USING` se može koristiti s `INNER` ili `OUTER` join-om
  - `CROSS` join ne dozvoljava uvjete

# Natural join

- SQL natural join operacija

```
SELECT * FROM table1 NATURAL INNER JOIN table2;
```

- INNER je opcionalno
- nema ON ili USING uvjeta

- svi zajednički atributi se koriste u natural join operaciji

- ukoliko je potrebno spojiti tablice na podskupu zajedničkih atributa koristi se običan INNER JOIN i USING ključna riječ

```
SELECT * FROM kolegij NATURAL JOIN predaje;
```

- ili...

```
SELECT * FROM kolegij JOIN predaje USING (kolegij_naziv);
```

# Natural outer join

- može se definirati i natural outer join
  - `NATURAL` ključna riječ definira kako se spajaju tablice
  - koriste se svi zajednički atributi
  - zapisima koji nisu upareni dodaju se `NULL` vrijednosti (način ovisi o tipu outer joina) i uključuju u rezultat
- primjer

```
SELECT * FROM table1
    NATURAL LEFT OUTER JOIN table2;
SELECT * FROM table1
    NATURAL LEFT JOIN table2;
```

# Outer join i agregacije

- outer join može generirati `NULL` vrijednosti
- funkcije agregacije ignoriraju `NULL` vrijednosti od kojih `COUNT` ima najkorisnije ponašanje
- primjer
  - *pronadite koliko pojedini profesor drži kolegija*
  - *uključite i profesore bez kolegija*
    - njima pridodajte 0
  - potrebno je koristiti tablicu nastavnik i kolegij
  - potrebno je koristiti outer join da bi se uključili profesori bez kolegija



# Outer join i agregacije

- prvi korak: left outer join tablice nastavnik i predaje

```
SELECT n.prezime, e.kolegij_naziv
FROM nastavnik n LEFT OUTER JOIN predaje e
ON (n.id = e.nastavnik_id);
```

- rezultat
  - profesori bez kolegija imaju pridruženu vrijednost NULL

| prezime   | kolegij_naziv |
|-----------|---------------|
| Petrić    | MR            |
| Bašić     | PIS           |
| Bašić     | SPI           |
| Dujmović  | MR            |
| Šimunović | STAT          |
| Ćosić     | NULL          |
| Bašić     | NULL          |
| ...       |               |

# Outer join i agregacije

- drugi korak

```
SELECT n.prezime, COUNT(e.kolegij_naziv) AS broj
FROM nastavnik n LEFT OUTER JOIN predaje e
ON (n.id = e.nastavnik_id)
GROUP BY n.prezime
ORDER BY COUNT(e.kolegij_naziv) DESC;
```

nije točno za profesore s istim prezimenom

- rezultat

|               |      |               |   |
|---------------|------|---------------|---|
| +-----+-----+ |      | ...           |   |
| prezime       | broj | Vučković      | 1 |
| +-----+-----+ |      | Barić         | 0 |
| Tadić         | 3    | Marušić       | 0 |
| Bašić         | 2    | Bilić         | 0 |
| Janković      | 2    | Burić         | 0 |
| Galić         | 2    | Jurišić       | 0 |
| Ćosić         | 1    | Mihaljević    | 0 |
| Horvat        | 1    | Šimić         | 0 |
| ...           |      | +-----+-----+ |   |

# Outer join i agregacije

- drugi korak – drugi pokušaj
  - jedan od načina

```
SELECT    n.id
,         COUNT(e.kolegij_naziv) AS broj
FROM      nastavnik n
LEFT JOIN predaje e
ON        (n.id = e.nastavnik_id)
GROUP BY  n.id)
```

# Outer join i agregacije

- drugi korak – drugi pokušaj
  - jedan od načina

```
SELECT    nastavnik.prezime
,         nastava.broj_kolegija AS broj
FROM      nastavnik
JOIN      (SELECT    n.id
,                COUNT(e.kolegij_naziv) AS broj
FROM          nastavnik n
LEFT JOIN     predaje e
ON            (n.id = e.nastavnik_id)
GROUP BY     n.id) AS nastava(id, broj_kolegija)
USING     (id)
ORDER BY  nastava.broj_kolegija DESC;
```

# Outer join i agregacije

- drugi korak – drugi pokušaj
  - jedan od načina

```
SELECT    nastavnik.prezime
,         nastava.broj_kolegija AS broj
FROM      nastavnik
JOIN      (SELECT    n.id
,                 COUNT(e.kolegij_naziv) AS broj
FROM          nastavnik n
LEFT JOIN     predaje e
ON            (n.id = e.nastavnik_id)
GROUP BY     n.id) AS nastava(id, broj_kolegija)
USING     (id)
ORDER BY  nastava.broj_kolegija DESC;
```

|                |                |
|----------------|----------------|
| -----+-----+   | ...            |
| prezime   broj | Jurišić   0    |
| -----+-----+   | Marić   0      |
| Galić   2      | Marušić   0    |
| Janković   2   | Mihaljević   0 |
| Bašić   2      | Ćosić   0      |
| Tadić   2      | Šimić   0      |
| Horvat   1     | Bilić   0      |
| Dujmović   1   | Horvat   0     |
| -----+-----+   | -----+-----+   |
| ...            |                |

# Pogledi

- *engl. views*
- do sada smo koristili SQL na logičkoj razini
  - upiti u pravilu koriste stvarne relacije
  - ali nije nužno!
  - upiti se mogu pisati i na izvedenim relacijama
    - ugniježđeni podupiti ili join u `FROM` dijelu upita
- SQL puža operacije na razini pogleda
- omogućava definiranje pogleda na logičku razinu
  - upiti se mogu pisati direktno na pogledima

# Pogledi

- razlozi za korištenje pogleda:
  - performanse i jednostavnost
  - sigurnost
- performanse i jednostavnost
  - pogled se može definirati za često korištenu izvedenu relaciju
  - upiti postaju jednostavniji
  - DBMS automatski računa sadržaj pogleda koji se koristi u upitu
- neke baze podatka podržavaju *materijalizirane poglede*
  - sadržaj pogleda se računa unaprijed i sprema na disk
  - DBMS osigurava da je sadržaj ažuran
    - frekvencija osvježavanja može biti neposredna ili periodička

# Pogledi

- sigurnost
  - mogu se definirati ograničenja pristupa na tablice i pogledе
  - na tablicama se mogu definirati stroga ograničenja pristupa s ciljem zaštite osjetljivih podataka
  - dok se pogledima mogu odvojiti podaci koji nisu zaštićeni
- primjer
  - tablica *nastavnik* može sadržavati OIB, adresu, primanja i druge privatne podatke
  - pogled *nastavnik\_info* može sadržavati ime i prezime, službene kontakt informacije...



# Stvaranje pogleda

- SQL sintaksa za stvaranje pogleda

- temelji se na naredbi SELECT

```
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;
```

- brisanje pogleda

```
DROP VIEW view_name;
```

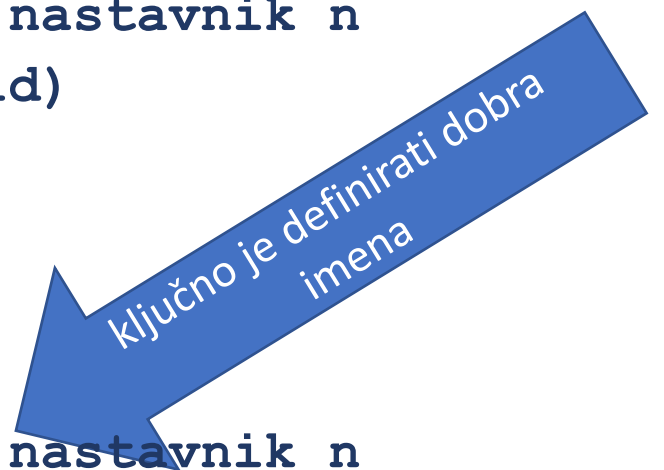
# Primjer pogleda

- *napravite pogled koji prikazuje broj ukupnih ects-a koji pojedini profesor predaje*

```
SELECT n.id, SUM(k.ects) AS ukupni_ects FROM nastavnik n
      JOIN predaje p ON (n.id = p.nastavnik_id)
      NATURAL JOIN kolegij k
      GROUP BY n.id;
```

- stvaranje pogleda

```
CREATE VIEW nastavnik_ects AS
SELECT n.id, SUM(k.ects) AS ukupni_ects FROM nastavnik n
      JOIN predaje p ON (n.id = p.nastavnik_id)
      NATURAL JOIN kolegij k
      GROUP BY n.id;
```



# Izmjena podataka

- pogled je izvedena relacija...
- što napraviti ukoliko se javi potreba za `INSERT` ili `UPDATE` upita?
- jednostavno rješenje je zabraniti takve akcije
- odluka može biti prepuštena dizajneru baze podataka
  - koje operacije su dozvoljene prilikom pokušaja promjene podatka unutar definiranog pogleda
  - zadani pristup je zabrana izmjena
  - izmjena je uvjetovana određenim pravilima

# Izmjena podataka

- za određene poglede može se definirati izmjena podataka (*engl. updatable views*)
  - podaci se mogu izmijeniti ukoliko su zadovoljeni slijedeći uvjeti
    - unutar FROM-a se koristi samo jedna relacija
    - SELECT isključivo koristi attribute iz relacije i ne izvodi računanja na atributima
    - atributi koji se ne koriste unutar SELECT-a moraju se moći postaviti na NULL
    - pogled ne koristi grupiranje ili agregiranje unutar upita
- ako su uvjeti zadovoljeni, na pogledu se mogu izvoditi operacije
  - INSERT, UPDATE, DELETE
- *vrlo rijetko postoji potreba za korištenjem ove mogućnosti*

# Dodavanje podataka

- napravimo jednostavan upit na pogledu koji sadrži sve profesore iz Karlovca

```
SELECT * FROM nastavnik_karlovac;
```

| id | ime     | prezime | grad     |
|----|---------|---------|----------|
| 3  | Karlo   | Bogdan  | Karlovac |
| 6  | Bernard | Bašić   | Karlovac |

- dodajmo novi red

```
INSERT INTO nastavnik_karlovac  
VALUES (31, 'Marko', 'Marić', 'Umag');
```

- ponovno napravimo isti upit

```
SELECT * FROM nastavnik_karlovac;
```

| id | ime     | prezime | grad     |
|----|---------|---------|----------|
| 3  | Karlo   | Bogdan  | Karlovac |
| 6  | Bernard | Bašić   | Karlovac |

- gdje je dodani red?

# Provjera umetnutih redova

- u deklaraciju pogleda može se dodati ključna riječ `WITH CHECK OPTION`
  - redovi koji se dodaju se provjeravaju prema `WHERE` uvjetima
  - ukoliko red ne zadovoljava `WHERE` uvjet, odbacuje se
- primjer

```
CREATE VIEW nastavnik_karlovac AS (  
    SELECT id, ime, prezime, grad  
    FROM nastavnik WHERE grad = 'Karlovac')  
WITH CHECK OPTION;
```

- kad se `INSERT` ponovi  
`CHECK OPTION failed 'unipu.nastavnik_karlovac'`

# Literatura

- Pročitati
  - [DSC] poglavlje 3.6. – 3.10.
  - [DSC] poglavlje 4.1. – 4.2.
  - Caltech CS121
  - **Vježbe: [DSC] – 3.1. – 3.24.**
  - **Vježbe: [DSC] – 4.1. – 4.4.**
- Slijedeće predavanje
  - [DSC] poglavlje 3.2.2.
  - [DSC] poglavlje 4.4.
  - Caltech CS121 - 7