BAZE PODATAKA

⇒ SQL → „Structure Query Language"

⇒ NoSQL → „Not only SQL"
→ nerelacijski sustavi

⇒ mi ćemo se primarno baviti **relacijskim bazama podataka**

⇒ ORGANIZACIJSKI SUSTAV:
↳ sustav koj sadrži tehničke i humane resurse (smišleno organizirano)

⇒ informacija je neki obrađen i razumljeti **podatak**
(podatak u kontekstu je informacija)

⇒ INFORMACIJSKI SUSTAV:
↳ ukupna infrastruktura i raspolaganje informacijama
↳ središnji dio informacijskog sustava jest upravo BAZA PODATAKA

⇒ <u>BAZA PODATAKA:</u>
↳ skup podataka koj su pohranjeni i organizirani tako da mogu zadovoljit zahtjeve korisnika

⇒ ENTITET:

↳ bilo što, što ima značajke s pomoću kojih se može razlučiti od svoje okoline

↳ nešto o čemu želimo pamtiti podatke (npr. knjiga, osoba, vozilo, objekt, ...)

↳ entitet može biti i poveznost među nekim entitetima

⇒ ATRIBUT / SVOJSTVO:

↳ karakterizira entitet

↳ skup atributa koj uzimamo za entitet ovisi o primjeni

⇒ slične entitete možemo svrstati u skupove entiteta (npr. skup entiteta „predmet")

⇒ IDENTIFIKATOR (ključ):

↳ jedan atribut il skup atributa kojeg jednoznačno određuje entitet

⇒ MODEL PODATAKA (Data Model):

↳ precizna i pojednostavljena slika stvarnog svijeta

↳ sastoj se od:

- skupa objekata
- skupa operacija
- skupa integritetskih ograničenja

⇒ povijesni razvoj modela podataka:

- hijerarhijski model („file system")
- mrežni model (hijerarhijski, ali svaki čvor može imati više roditelja)
- relacijski model (najčešće baze)
- Entity Relationship model (ER Model)
- objektni model (pojavile su se s razvojem objektnih programskih jezika)
- objektno - relacijski model (hibridni model)
- NoSQL model (koristi se za „Big Data", ogromne količine kaotičnih podataka)

⇒ RELACIJSKI MODEL:

↳ relacija je skup n-torki

↳ objekt su „tablice"

↳ integritetska ograničenja - to su pravila za unos odredenog sadržaja (npr. ocjena je između 1 i 5)

⇒ ER MODEL:

↳ služi za prikaz i dokumentaciju baze podataka (najčešće u relacijskom modelu)

↳ ovo je najkorisnije načiniti prije izrade same baze

ARHITEKTURA BAZE PODATAKA

⇒ postoje 3 sheme:

a) INTERNA

b) KONCEPTUALNA

c) EKSTERNA

⇒ KONCEPTUALNA (logička) SHEMA:

↳ najčešće prikazuje koncept (ER dijagram) baze podataka (ta shema koristi korisniku)

⇒ INTERNA SHEMA:

↳ brine se o fizičkoj pohrani podataka (o tome se brine administrator baze)

⇒ EKSTERNA SHEMA:

↳ opisuje „pogled" na dio baze koji je namijenjen specifičnoj grupi korisnika

↳ opisuje ograničenja na pristup baze za određene korisnike

RELACIJSKI MODEL PODATAKA

⇒ temelj ovakvog modela su operacije nad skupovima

⇒ MATEMATIČKA RELACIJA:

↳ nad skupovima $D_1, D_2, \ldots, D_m$, relacija je neki podskup kartezijevog produkta $D_1 \times D_2 \times \ldots \times D_m$

⇒ RELACIJSKA SHEMA:

↳ to je skup atributa $R = \{A_1, A_2, \ldots, A_m\}$

↳ vizualno, to je vrh tablice

⇒ n-torka nad relacijskom shemom:

$t = \{A_1 : v_1, A_2 : v_2, \ldots, v_m\}$

↳ uoči da redoslijed u ovoj notaciji nije bitan

↳ ukoliko pišemo skraćeno $t = \langle v_1, v_2, \ldots v_m \rangle$, onda je redoslijed bitan kako bi znali na koji se atribut referenciramo

⇒ **RELACIJA:**

↳ relacija „$r(R)$" je skup n-torki koje su definirane nad relacijskom shemom R

⇒ svojstva relacija:

- relacija posjeduje jedinstveno ime
- atribut unutar relacijske sheme imaju jedinstvena imena
- u jednoj relaciji ne postoje identične n-torke
- poredak atributa u n-torki nije bitan

⇒ A - vrijednost n-torke:

↳ „uzmem jedan atribut n-torke i gledam njegovu vrijednost"

⇒ X - vrijednost n-torke:

↳ „uzmem jedan skup atributa i gledam njihove vrijednosti"

⇒ stupanj relacije ⇒ broj atributa relacije
⇒ kardinalitet relacije ⇒ broj n-torki relacije

⇒ SHEMA BAZE PODATAKA:

↳ skup relacijskih shema baze podataka

OPERACIJE U RELACIJSKOM MODELU PODATAKA

⇒ karakteristika operacijske algebre je PROCEDURALNOST:
↳ postoj redoslijed izvršavanja operacija te se treba evaluirati cijel izraz (uvijek?)

⇒ PREDIKATNI RAČUN: radbrzava „n"-torke koje zadovoljavaju neki predikat
↳ NAPOMENA: predikatni račun je neproceduralan
PR: $A = B \wedge C \wedge D$
↳ ako znamo da je B false, odmah znamo da je i A false, pa ne trebamo evaluirati cijel izraz

⇒ operacijska algebra i predikatni računi mogu ostvariti iste stvari
↳ analizator upita bira koja mu je opcija optimalnija za pokretanje upita

STRUCTURED QUERY LANGUAGE

⇒ SQL je **deklarativan**, a ne proceduralan

⇒ kreira nove instance baze podataka:

```
CREATE DATABASE studAdmin
```

⇒ brisanje baze podataka (oprez!):

```
DROP DATABASE studAdmin
```

⇒ kreiranje relacije:

```
CREATE TABLE mjesto (
    ptt         INTEGER
  , nazMjesto   CHAR(30)
  , sifZup      SMALLINT
);
```

⇒ upisivanje n-torki u relaciju:

```
INSERT INTO mjesto
    VALUES(42000, 'Varaždin', 7);
```

⇒ dohvat odredenih podataka:

```
SELECT * FROM mjesto
    WHERE sifZup = 7;
```

⇒ ažuriranje podataka:

```
UPDATE mjesto
    SET nazMjesto = 'VARAŽDIN'
    WHERE sifZup = 7;
```

RELACIJSKA ALGEBRA

⇒ razlikujemo relacije:

a) unarne (npr. selekcija, preimenovanje, ...)

b) binarne (npr. presjek, kartezijev produkt, ...)

⇒ obavljanje operacije ne utječe na operande

$r_3 = r_1 \cup r_2$

↳ „r_1" i „r_2" su ovdje neproizvoljno

⇒ UNIJSKA KOMPATIBILNOST:

↳ dvije relacije su unijski kompatibilne ako su ⇒ relacije istog stupnja

⇒ korespodentni atributi su definirani nad istim domenama

↳ domena je npr. kuća, pecivo, zrakoplov - nešto što predstavlja nešto iz stvarnog svijeta, a ne kako to računalo interpretira (skup znakova)

↳ primjet da sada čovjek treba biti svjestan što je unijski kompatibilno

↳ unijska kompatibilnost znači da možemo raditi operaciju presjeka i unije

⇒ DIJELJENJE:

↳ imamo dvije relacije: $r(R)$ $s(S)$

↳ vrijedi da je $S \subseteq R$

$t = r \div s$

} vidi primjer sa slajdova!

⇒ SELEKCIJA:

↳ operacija selekcije radi nad nekakvim predikatom F i nad nekom relacijom r

↳ oznaka: $\sigma_F(r)$

↳ pisanje selekcijske naredbe:

SELECT (ovdje se radi projekcija - ovo se radi redove)

FROM (ovdje se selektira od kud - ovo se radi prvo)

WHERE (ovdje se radi selekcija - slobodno razmišljaj da se ovo radi drugo)

⇒ PROJEKCIJA:

$\Pi_{A_2 A_4 A_6}(r)$

↳ projekcija eliminira stupce (tj. radimo samo one atribute koje nabrojimo - umjesto nabrajanja možemo koristit * ⇒ time uzimamo sve predikate)

↳ ključna riječ DISTINCT ⇒ eliminira duple „n"-torke u projekciji

BITNO!

⇒ KARTEZIJEV PRODUKT:

↳ imamo dvije relacije $r(R)$ i $s(S)$, pri čemu je $R \cap S = \emptyset$

$$deg(r \times s) = deg(r) + deg(s)$$
$$card(r \times s) = card(r) \cdot card(s)$$

↳ zapisivanje kartezijevog produkta:

a) SELECT *
FROM student, predmet

b) SELECT *
FROM student CROSS JOIN predmet
↳ ovo je ANSII notacija, pa se preporuča

↳ NAPOMENA: ne možemo napraviti kartezijev produkt dvije tablice koje imaju barem jedan stupac istog imena
↳ stoga, koristimo relaciju preimenovanja

r	
A	B
1	3
2	4

s	
B	C
1	5
7	8

} moramo preimenovati tablicu s
↳ preimenovanje preimenuje sve (ono što ne želimo preimenovati, ostavimo isto)

$\rho_{s(B2, C)}(s)$ →

B2	C
1	5
7	8

↳ preimenovanje se radi u SELECT dijelu i radi se „skroz" na kraju

↳ stoga, pisali bi:

```
SELECT * A.B AS B2
FROM T CROSS JOIN A
```

⇒ **PRIRODNO SPAJANJE:**

↳ oznaka: ⋈

mjesto

pbr	ime	sif Zup
10000	Zagreb	1
10200	Zaprešić	1
31000	Osijek	4

županija

sif Zup	ime Zup
1	Grad Zagreb
2	Splitsko-Dalmatinska
4	Slavonska

↳ kako radi naredba „mjesto ⋈ županija"?

1) traži zajedničke atribute dviju tablica (radi $R \cap S$)

2) kombinira n-torke s obzirom na zajedničke atribute

3) na kraju, eliminira stupce jednakog naziva (kako bi dobili relaciju)

mjesto ⋈ županija

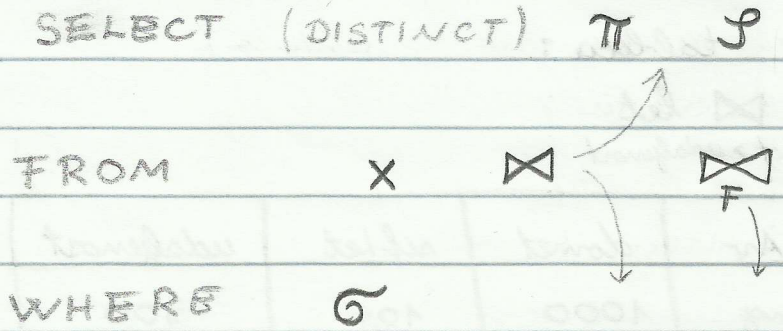
pbr	ime	sif Zup	ime Zup
10000	Zagreb	1	Grad Zagreb
31000	Osijek	4	Slavonska

↳ spajanje se (u većini slučajeva) vrši u FROM dijelu upita

↳ napišimo upit za „mjesto ⋈ županija":

```
SELECT mjesto.*,
       naz_Zup
FROM mjesto JOIN zupanija
     ON mjesto.sif_Zup = zupanija.sif_Zup
WHERE
```

⇒ shema pamćenja operacija:



⇒ NAPOMENA: ukoliko prirodno spajanje ne nađe zajedničke atribute, ono radi kartezijev produkt

⇒ THETA SPAJANJE:

↳ oznaka υ

$$r \underset{F}{\bowtie} s = \sigma_F (r \bowtie s)$$

theta spajanje - prirodno spajanje uz predikat

PR: Imamo dvije relacije, „avion" i „let", želimo dobit sve avione koj mogu preletit letove u jednom letu (bez presjedanja di punjenja spremnika)!

avion

sifAv	narAv	domet
10	Boeing	1000
20	Cesna	200
30	Concord	5000

let

sifLet	udaljenost
101	500
102	1500
103	7500

↳ mi tražimo tablicu:

avion ⋈ let
domet > udaljenost

sifAv	narAv	domet	sifLet	udaljenost
10	Boeing	1000	101	500
30	Concord	5000	101	500
30	Concord	5000	102	1500

⇒ theta spajanje u SQL-u:

```
SELECT *
FROM linija JOIN avion
    ON dolet >= udaljenost;
```

⇒ AGREGACIJA:

↳ „sabija tablicu"

↳ kako radi agregacija:

1) uzme jedan stupac tablice i njega spljošti na jednu vrijednost

2) agregacijska funkcija primijenjuje se na sve vrijednost tog stupca i da nam tu jednu vrijednost

↳ oznaka: $G_{AF(A)}(r)$

↳ AF → agregatna funkcija

↳ agregacija ima problem:

↳ kako interpretirat dobiven rezultat?

- on je bez imena, pa da bi uvijek imali pravilnu agregaciju trebamo vršit preimenovanje:

$$\rho_{REZ(prosjek)}\left(G_{AF(A)}(r)\right)$$

↳ agregacija u SQL-u:

```
SELECT AVG(ocjena) AS prosjOcj
FROM ispit;
```

↳ agregacijske funkcije:

AVG	MIN
SUM	MAX
COUNT	

↳ pogledaj agregacijske funkcije sa DISTINCT!

⇒ **GRUPIRANJE:**

↳ „ranima me agregacija, al grupirana po zadanoj kombinaciji atributa“:

$$A_1 A_2 A_3 \dots G_{AF(A)}(T)$$

↳ naravno, to opet sve treba numerirat, pa treba naći premenovanje nad svime:

$$\rho\left(A_1 A_2 A_3 \dots G_{AF(A)}(T)\right)$$

↳ **grupiranje u SQL-u:**

- uvodimo novi element

4. SELECT
1. FROM
2. WHERE
3. GROUP BY nar Pred, ak God

NEPOTPUNE INFORMACIJE I NULL VRIJEDNOSTI

⇒ SQL se ne drži potpuno pravila relacijske algebre

⇒ IZRAZI → nešto što se može izračunat
→ kombinacija operatora i operanada
↳ izvore najčešće pišemo u:
• SELECT
• WHERE
↳ kada god pišemo izraz, uobičajeno je da stupci koji sme modificirali premenujemo u prikladnu vrijednost

⇒ **NULL vrijednost:**
↳ konstanta koja govori da informacija nije dostupna
• **NULL u izrazima:**
↳ svaka aritmetička operacija (izraz) s jednim od operanada NULL je jednaka NULL - rezultat je nepoznat
• **NULL u usporedbama:**
↳ rezultat usporedbe s NULL vrijednošću je uvijek „unknown"

↳ kod selekcije, uzimaju se samo „n"-torke koje rade usporedbu uvjet sa true, a „n"-torke sa false i unknown se odbacuju

⇒NAPOMENA: NULL vrijednost ne možemo tražiti pomoću upita:

:

WHERE adresa = NULL

↳ NULL vrijednost tražimo / ispitujemo pomoću posebnih operatora:

IS NULL

IS NOT NULL

• NULL u logičkim sudovima:

↳ ovdje je moguće da operacija s unknown vrijednošću ima neku logičku vrijednost (npr. unknown AND false = false)

• NULL u skupovima:

↳ potrebno je odstupit od stroge matematičke definicije, pa se uvode stroga pravila ponašanja NULL vrijednost u skupovima:

- vrijednost NULL je **kopija** druge NULL vrijednosti, pa su sljedeće „n"-torke **kopija** jedna druge:

NULL Janko NULL 1996

NULL Janko NULL 1996

↳ zbog ovakve definicije kopije „n"-torke (PAZI! Ova nisu jednake „n"-torke, al ih tretiramo kao kopije), možemo raditi normalne, intuitivne operacije s „n"-torkama

• NULL u agregaciji

↳ neke vrijednosti postoje, a neke su NULL:

→ tada se NULL vrijednosti zanemaruju to se broje samo postojeće vrijednosti

↳ sve vrijednosti su NULL:

→ COUNT vraća 0

→ ostale funkcije vraćaju NULL

⇒NAPOMENA: COUNT(*) → nije „prava" agregatna funkcija, već samo broj redaka u tablici bez obzira na sadržaj samih redaka

• NULL u grupiranju:

↳ radi jednako kao i grupiranje bez NULL vrijednosti, samo što vodi brigu o definiciji kopije „n"-torke

VANJSKO SPAJANJE

⇒ prirodno spajanje eliminira „n" - torke koje se nisu uspjele spojiti u konačnu tablicu
↳ zbog tog problema eliminacije „n"-torki, uvedeno je **vanjsko spajanje**

⇒ VANJSKO SPAJANJE:
↳ oznaka: $\overset{*}{\bowtie}$ (lijevo vanjsko spajanje)
↳ one vrijednosti koje se nisu uspjele spojiti se dodaju u tablicu, a neprimat atribut popunjavaju se NULL vrijednostima
↳ ovakva nepotpuna spajanja su nam ponekad bitna za agregacije

⇒ postoje tri vrste vanjskog spajanja, ovisno o tome iz koje tablice uzimamo „n"-torke koje se nisu uspjele spojiti:
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

⇒ vanjska spajanja se također rade u FROM dijelu upita, kao i ostala spajanja

STRUCTURE QUERY LANGUAGE

⇒ koriste se dvije vrste upitnih jezika

- DDL – definicijski jezik („Data definition")
- DML – manipulatorski jezik („Data manipulation")

⇒ možemo uvesti novi element upita:

ORDER BY – samo poredava „M"-torke u krajnjoj tablici, pa se ovaj element izvršava posljednji (nakon GROUP BY)

⇒ **TIPOVI PODATAKA:**

- INTEGER → 4 bajta
- SMALLINT → 2 bajta
- CHAR(m) → maksimalno 32767 znakova
- NCHAR(m) → jednako kao i „char", ali kod sortiranja koristi internacionalne kodne stranice, pa dobro sortira Č, Ć, Đ, Ž, ...
- REAL → analogan floatu (4 bajta)
- DOUBLE PRECISION → analogan doublu (8 bajta)
- DECIMAL(m, n) → m – broj znamenki ukupno
  n – broj znamenki iza točke
  (precizniji, ali sporiji od floata i doublea)
- DATE → pohrana datuma

⇒ SELECT:

↳ možemo ramurlt da kontrolira stupce

↳ odredtuje koje stupce ispisuje, što će biti u njima (podaci modificirani izrazom, nekom funkcijom, ...)

⇒ uvjet usporedbe pomoću LIKE:

↳ služi za ispitivanje je li neki niz znakova sličan / jednak nekom uzorku

↳ uzorak ima specijalne znakove:

⇒ _ → točno bilo koji znak

⇒ % → proizvoljan broj proizvoljnih znakova (nula ili više)

↳ escape char:

↳ služi za poništavanje specijalnih znakova kako bi i „%" i „_" mogli detektirati u nizovima znakova

↳ C ima „\" za taj znak, no SQL daje korisniku da sam postavi taj znak

⇒ uvjetni izrazi (CASE):

↳ koristi se u SELECT dijelu upita

↳ omogućava da se dodaju novi stupci čiji sadržaj ovisi (izračunava se) o ostalim stupcima koji su selektirani

↳ kako pisati **case** izraz:

```
SELECT *
    , CASE
        WHEN   uvjet1 = X  THEN 'onako'
        WHEN   uvjet2 = Y  THEN 'onako'
        :
        ELSE   'nekako'
      END AS kako
FROM tablica1;
```

dvije vrste CASE izraza

↳ ako stalno ispitujemo isti uvjet, koristimo pokraćenu verziju **case** izraza:

```
SELECT *
    , CASE uvjet1
        WHEN   X   THEN 'onako'
        WHEN   Y   THEN 'onako'
        :
        ELSE   'nekako'
      END AS kako
FROM tablica1;
```

dvije vrste CASE izraza

⇒ FROM:

↳ ANSI sintaksa za spajanje uvedena je jer kod ravne sintakse nije moguće odrediti koj dio je selekcija, a koj dio je eliminacija stupaca zbog prirodnog spajanja

PR: Ovaj izraz je stoga problematičan!

$$\sigma_{C=100}(A \bowtie t)$$

↳ ravna sintaksa:

```
SELECT A.*, t.E
FROM A, t
WHERE A.D=t.D
  AND C=100;
```

↳ ansi sintaksa:

```
SELECT A.*, t.E
FROM A JOIN t ON A.D=t.D
WHERE C=100;
```

↳ primjet da u obje sintakse u SELECT dijelu treba parti na poravnjanje stupaca

⇒ NAPOMENA: vanjsko spajanje nije moguće realizirati pomoću ravne sintakse, dok ostala spajanja jest

↳ samim time, kod vanjskog spajanja nije svejedno što stavljamo u ON, a što u WHERE

⇒ NAPOMENA: vanjsko spajanje nije asocijativno, pa neke izraze relacijske algebre treba malo „poredati" jer SQL raspoređuje izraze od lijeva prema desno

PR: stud $*_{plrSt=plr}$ (mjesto $\bowtie_{MjSf=sf}$ županija)

↳ ovaj izraz je identičan onome:

(mjesto $\bowtie_{MjSf=sf}$ županija) $\bowtie_{plrSt=plr}$ $*$ stud

↳ ovaj izraz možemo lagano prebaciti u SQL jezik jer se ide s lijeva prema desno, pa slobodno makremo zagrade iz izraza

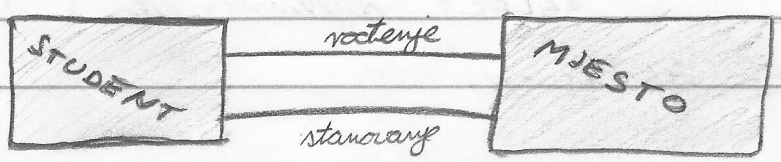
⇒ **preimenovanje tablica unutar upita:**

↳ tablici dodna **alias** - ime koje vrijedi samo unutar upita

↳ ovo se radi u FROM dijelu upita

↳ alias se najčešće koristi kod **paralelnog spajanja:** tada nam ista tablica može imati više uloga, pa trebamo i više imena

PR: Spajanje studenta po mjestu rođenja i stanovanja.



⇒ HAVING:

↳ izvršava se nakon GROUP BY dijela upita

↳ GROUP BY načini nekoliko podtablica iz jedne

↳ HAVING izbacuje sve tablice koje ne zadovoljavaju dani uvjet

- možemo zamislit da radi kao i WHERE, samo što eliminira podtablice, a ne „n"-torke

⇒ STRUKTURA UPITA:

SELECT } izvršava se **peto** (prije sortiranja) (liva atribute koje ostavljamo)

FROM } izvršava se **prvo** (liva relacije i nad neke operacije se njima

WHERE } izvršava se **drugo** (liva „n"-torke i izbacuje one koje ne zadovoljavaju uvjet)

GROUP BY } izvršava se **treće** (stvara podtablice na temelju atributnih vrijednosti

HAVING } izvršava se **četvrto** (liva relacije koje zadovoljavaju uvjet)

ORDER BY } izvršava se **šesto**/posljednje (mogu se koristit i novi nazivi atributa ako su se preimenovali u SELECT dijelu upita)

PODUPIT I

PR: Želimo sva vozila čija je nosivost veća od najtežeg tereta?

vozilo

sifVoz	nosivost
101	2500
102	2000
103	800
104	1000

teret

sifTeret	tezina
1001	1800
1002	1200
1003	1000

```
SELECT *
FROM vozilo
WHERE nosivost > (SELECT MAX(tezina)
                  FROM teret);
```

PODUPIT

↳ NAPOMENA: podupiti su spori jer se za svaku „N" - torku mora izvršiti podupit
↳ SUBP su optimizirani za rad s tablicama (spajanja), a ne za rad s upitima, pa ako problem možemo rješiti bez podupita, to se svakako preporuča
↳ ponekad, ako SUBP shvati da ne treba za svaku „N"-torku ne treba raditi podupit, onda će se to optimizirati

↳ rješimo prethodni primjer bez podupita:

```
SELECT    sifVoz, mosnost
FROM      vozila, teret
GROUP BY  sifVoz, mosnost
HAVING    mosnost > MAX(teret);
```

⇒ podupite najčešće koristimo u:

- WHERE
- SELECT
- HAVING

⇒ VRSTE PODUPITA:

a) skalarni podupit

↳ vraća jednu vrijednost (npr. cijeli broj, datum, ...)

↳ najčešća vrsta podupita

b) vektorski podupit

↳ vraća jedan stupac tablice

c) tablični podupit

↳ vraća tablicu

↳ NAPOMENA: za vektorskim i tabličnim upitima treba biti oprezan (očekuje li se npr. stupac?)

⇒ NAPOMENA:

↳ podupiti u HAVING dijelu upita su najčešća pojava

↳ podupiti u SELECT i WHERE dijelu se često mogu izostaviti i upit se može napisati bez upita

⇒ KORELIRANI PODUPIT:

↳ podupiti koji se baš **za svaku** „n"-torku moraju izvršiti jer se dolina varijat rezultat za svaku „n"-torku (to su skalarni upiti)

↳ takvi podupiti su čest uzrok usporavanja baza podataka (**koriste atribute vanjskog upita**)

⇒ ako se treba preimenovat neka relacija, preporuča se da se preimenuje **tablica vanjskog upita**

⇒ <u>o vektorskim upitima:</u>

↳ ne mogu se pojavit u SELECT dijelu upita

↳ kako bi smo ga koristili u WHERE dijelu, trebamo koristiti identifikatore:

- ALL - svi (npr. „<> ALL" → različito od svih)
- ANY / SOME - barem jedan

↳ također, postoje i operatori za provjeru nalaze li se nešto u nekom skupu:

- IN - u skupu (identično „= ANY")
- NOT IN - nije u skupu (identično „<> ALL")

NAREDBE ZA IZMJENU PODATAKA U BAZI

⇒ INSERT :

↳ služi za ubacivanje „n" - torki u tablicu

↳ piše se ovako:

INSERT INTO tablica [lista atributa]
VALUES (...., ... , ...);

↳ preporuča se pisati listu atributa

↳ ako ne navedemo listu atributa, moramo navesti sve atribute (i one koji su NULL) i moramo ih navesti uredenim redoslijedom

↳ to nije dobro za:

- tablice s puno atributa
- tablice kojima se mijenja relacijska shema

↳ ako želimo neke podatke kopirati iz neke tablice u neku drugu, koristimo sintaksu:

INSERT INTO polaze Fr
SELECT stud. mbr, ime, prez
FROM stud, ispit
WHERE stud. mbr = ispit. mbr

⇒ DELETE:

↳ brisanje „n"-torke iz tablice

↳ piše se ovako:

```
DELETE FROM tablica
WHERE „uvjet";
```

↳ NAPOMENA: ako se ne navede WHERE dio naredbe, briše se sve „n"-torke (OPREZ!)

↳ u WHERE dijelu naredbe možemo koristiti podupite, ali se strogo preporuča da se u podupitu ne koristi tablica iz koje se briše

⇒ UPDATE:

↳ naredba za ažuriranje tablice

↳ piše se ovako:

```
UPDATE tablica
SET [atribut]="vrijednost", ...
WHERE „uvjet"
```

↳ ovdje se WHERE izvršava prvo, a tek onda se postave vrijednosti odabranih „n"-torki

↳ ako se izostavi WHERE, izvršavaju se sve „n"-torke iz tablice

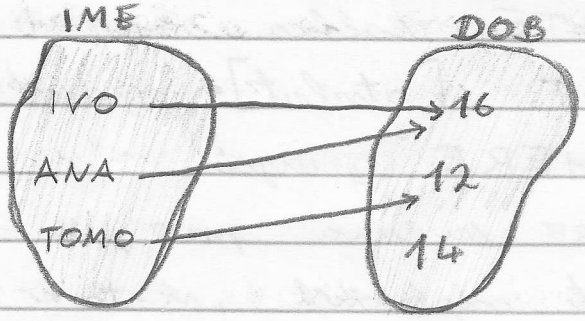
↳ također, preporuča se da se u WHERE dijelu ne koristi podupit koj radi sa tablicom koja se ažurira, jer to dovodi do nekonzistentnosti

OBLIKOVANJE SHEME RELACIJSKE BAZE PODATAKA

⇒ do sada smo sve radili u DML-u (Data Manipulation Language), ali što kada baza još ne postoji?
↳ kako dizajnirat bazu podataka?

⇒ loša baza podataka:
- ima redundanciju
- pojavljuju se lažne „n" - torke

⇒ KAKO DEKOMPONIRATI TABLICU?
↳ koristimo se funkcijskom ravisnosti - to je ravnavo ravisnost dvije domene



↳ funkcijska ravisnost $X \rightarrow Y$:

$$t_1(x) = t_2(x) \Rightarrow t_1(Y) = t_2(Y)$$

- ako u nem stanjma relacij svak par n-torki koji ima jednake X - vrijednosti, također ima jednake Y vrijednosti

PR: Funkcijska ravnost u tablici nekretnina

nekretnina

pbr	mjesto	Ime Prezime
49232	Radoboj	Napolitanke
10000	Zagreb	Bojadera
21000	Split	Dorina
31000	Osijek	Napolitanke

vidimo da su ova dva atributa funkcijski ravisna

↳ NAPOMENA: funkcijska ravisnost se odredjuje „odvarim varimom" poznavanjem domene, a ne analiziranjem relacije

↳ NAPOMENA: ne može se dokazati postojanje funkcijske ravisnosti (može se samo pretpostaviti), ali se može dokazati suprotno

PR: Što znači sljedeća funkcijska ravisnost:

mbr dat_ispit → ocjena

↳ znači da poznavanjem matičnog broja, datuma ispita možemo iz baze znati ocjenu
↳ to znači da je student na dan mogao pisati samo jedan ispit (što nekad ne mora vrijediti!)

⇒ ARMSTRONGOVI AKSIOMI:

1) A-1 Refleksivnost
   - ako je $Y \subseteq X$, tada vrijedi $X \to Y$

2) A-2 Utvrđanje
   - ako u shemi R vrijedi $X \to Y$, tada vrijedi i $XZ \to Y$

3) A-3 Tranzitivnost
   - ako u shemi R vrijedi $X \to Y$, $Y \to Z$, tada vrijedi $X \to Z$

⇒ ARMSTRONGOVA PRAVILA:

1) P-1 Pravilo unije
   - ako u shemi R vrijedi $X \to Y$, $X \to Z$, tada vrijedi, $X \to YZ$

2) P-2 Pravilo dekompozicije
   - ako u shemi R vrijedi $X \to YZ$, tada vrijedi i $X \to Y$

3) P-3 Pravilo o pseudo-tranzitivnosti
   - ako vrijedi $X \to Y$, i vrijedi $VY \to Z$, onda vrijedi i $VX \to Z$

PR: Ako vrijedi $A \to BD$, $B \to C$, $D \to E$, onda dokaži da vrijedi $AE \to AC$?

1) $A \to BD \Rightarrow A \to B$ (P2)

2) $A \to B \wedge B \to C \Rightarrow A \to C$ (A3)

3) $A \to A$ (A1)

4) $A \to A \wedge A \to C \Rightarrow A \to AC$ (P1)

5) $A \to AC \Rightarrow AE \to AC$ (A2) //

⇒ PRAVILO O AKUMULACIJI:

↳ ako u shemi R vrijedi:

$X \to VZ$ i $Z \to W$, tada vrijedi i $X \to VZW$

⇒ ako znamo pravilo o akumulaciji, tada poznavanjem A-1 i P-2 možemo na lakši način riješiti probleme dokazivanja funkcijske ravnosti nekih atributa

PR: Dokaži da vrijedi $AE \to AC$, ako vrijedi $A \to BD$, $B \to C$, $D \to E$. To dokaži principom R. A. D. (refleksivnost, akumulacija dekompozicija)?

1) $AE \to AE$ (A-1) (refleksivnost)

2) $AE \to AE \wedge A \to BD \Rightarrow AE \to ABDE$ (akumulacija)

3) $AE \to ABDE \wedge B \to C \Rightarrow AE \to ABCDE$ (akumulacija)

4) $AE \to ABCDE \Rightarrow AE \to AC$ (P-2) (dekompozicija)

↳ da u nekom trenutku nismo uspjeli dalje akumulirati (uočavati) desnu stranu, mogli smo zaključiti da funkcijska ovisnost ne vrijedi

⇒ KLJUČ RELACIJE

- minimalan skup atributa koji funkcijski određuje sve ostale atribute tablice
  ↳ to je npr. mbr u tablici student

⇒ može se dogoditi da imamo više kandidata za ključeve u jednoj relaciji:

- JMBAG
- matBr
- JMBG

↳ tada odabiremo PRIMARNI KLJUČ
- to npr. za fakultet JMBAG

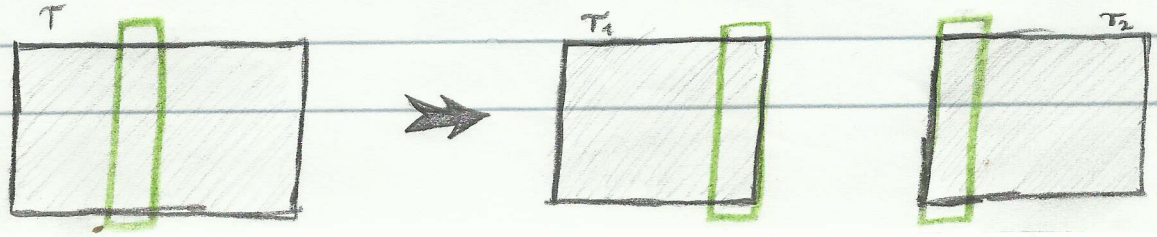
NORMALIZACIJA

⇒ postupak drugovranja baze podataka kojem je glavni cilj ukloniti redundanciju

⇒ postoje NORMALNE FORME : baza podataka
↳ to su razine „dobrote" baze podataka

⇒ glavni postupak normalizacije je DEKOMPOZICIJA
↳ složenu veliku tablicu i rastavljamo tablicu je na način da ne gubimo informaciju
↳ postoji još proces i sintere, al mi se time nećemo baviti

⇒ DEKOMPOZICIJA:
↳ ako relaciju dekomponiramo bez gubitka informacija, prirodnim spajanjem tih dekompozita moramo dobiti početnu relaciju
↳ kako pravilno dekomponirati?
BITNO:
- dekompoziti moraju imati zajedničke atribute
- zajednički atribut mora biti ključ u barem jednom od dekompozita



⇒ PRVA NORMALNA FORMA:

↳ za prvu normalnu formu treba postojat neki ključ i u svakoj „kućici" se nalazi samo jedan podatak (nema podataka odvojenih zarezom unutar kućica)

↳ odredivanje ove forme se najčešće svodi na traženje ključa.

⇒ DRUGA NORMALNA FORMA:

↳ relacija mora biti u prvoj normalnoj formi te ravnsn dio (onoj koj nije u ključu) relacije je potpuno funkcijski ravisan o ključu relacije

↳ potpuna funkcijska ravisnost:

- Y je potpuno funkcijski ovisan o X ako ne postoji pravi podskup od X koji funkcijski odredjuje Y
- matematički pisano:

$$X \rightarrow Y \quad \nexists X' \subset X,\ X' \rightarrow Y$$

↳ „svaki atribut koji ovisi samo o dijelu ključa i izvuci ga van na način da stavimo taj dio ključa i taj atribut (il stup atributa) u novu tablicu"

⇒ TREĆA NORMALNA FORMA:

↳ relacija mora biti u prvoj normalnoj formi i neki jedan atribut nekršnog dijela nije **tranzitivno** ovisan o ključu

↳ tranzitivna funkcijska ovisnost:

- matematički zapis:

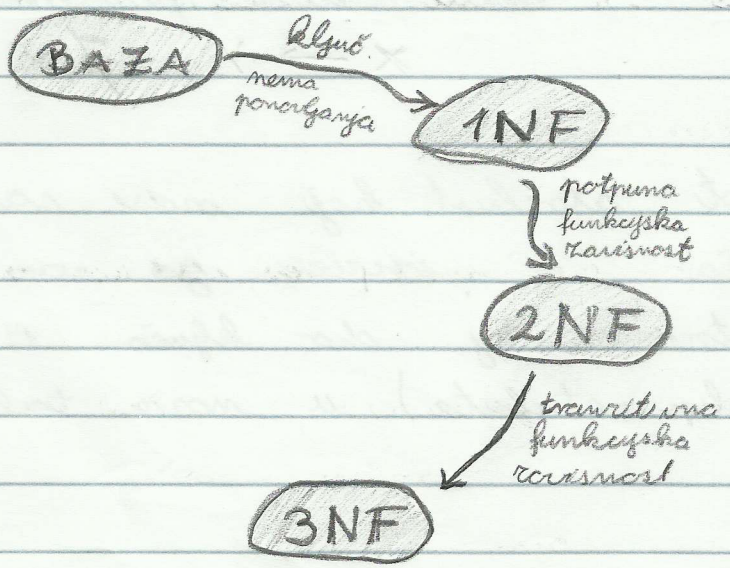
$$X \rightarrow Y \rightarrow Z$$

$$\not X \rightarrow X$$

ne smije postojati ova povratna veza

↳ „nek atribut koji uzrokuje tranzitivnu funkcijsku ovisnost i premjest ga u novu tablicu sa pripadnim ključem"

↳ NAPOMENA: primjeti da je nepotpuna funkcijska ovisnost ujedno i tranzitivna, ali se preporuča postupak:



Pr: Normalizirraj relaciju:

ISPIT = {matBr, prez, ime, sifPred, nazPred, datIsp, ocj, sifNast, prezNast}

1.) odredimo ključ + najlakši korak

K_{ISPM} = {matBr, sifPred, datIsp}

1.NF ↳ pretpostavka: student na isti dan može pisat više različitih ispita

2.) odredimo atribute koji ovise o dijelu ključa i izravnajmo ih van

STUDENT = {matBr, ime, prezime}

2.NF PREDMET = {sifPred, nazPred}

ISPIT 2 = {matBr, sifPred, datIsp, ocj, sifNast, prezNast}

3.) odredimo tranzitivne funkcijske ovisnosti

NASTAVNIK = {sifNast, prezNast}

3.NF ISPIT 3 = {matBr, sifPred, datIsp, ocj, sifNast}

↳ relacije STUDENT, PREDMET su već u trećoj normalnoj formi jer nema tranzitivne ovisnosti

PRIMJERI NORMALIZACIJE

ZADATAK 2 $F = \{AB \to CD,\ AB \to EF,\ A \to F,\ D \to E\}$

$R = ABCDEF$

1) Odredimo ključ:

$(AB) \longrightarrow CDEF$

2) Izvodimo van one ovisne o dijelu ključa

$R_1 = \underline{A}F$ $\qquad$ $R_2 = \underline{AB}CDE$

3) Odredimo funkcijsku ovisnost (tranzitivnu)

$AB \to D \longrightarrow E$

$R_1 = \underline{A}F$ $\qquad$ $R_2 = \underline{AB}CD$ $\qquad$ $R_3 = \underline{D}E$

ZADATAK 3 Relacijska shema:

POSUDBA

sif Clv, prez Clv, ime Clv, plv, nar My, adr Clv, inv Br Prim, dat Pos, dat Vr, sif Knj, nar Knj, sif Trd, nar Trd

1) Odred ključ pomoću pravila

$K_{POSUDBA} = \{$ sif Clv, inv Br Pos, dat Pos $\}$

↳ napomena: sif Knj - klasa knjige (npr. sve „Zagor" - i imaju istu šifru knjige)

inv Br Prim - šifra instance knjige (svaki primjerak „Zagor"-a ima vlastiti inventarski broj primjerka)

2) ČLAN = { sif Clv, prez Clv, ime Clv, nar My, adr Clv, plv }

KNJIGA = { inv Br Prim, sif Knj, nar Knj, sif Trd, nar Trd }

POSUDBA = { sif Clv, dat Pos, inv Br Prim, dat Vr }

3) MJESTO = { plv, nar My }

↳ odredimo i ostale tranzitivne ovisnosti

FIZIČKA ORGANIZACIJA PODATAKA

⇒ ova tema obuhvaća:
- strukture podataka za pohranu
- metode pristupa već pohranjenim podacima

⇒ podaci se pohranjuju u sekundarnu memoriju u blokovima:
- na **hard-disk**
- na **solid-state disk**

↳ cilj je minimizirat broj U/I operacija pri pohrani i dohvatu podataka

↳ nadalje, koji je tročlan „pametne" organizacije podataka?

⇒ **ključ pretrage** - nije nužno primarni il alternativni ključ, već samo atribut kojima možemo pretraživati

⇒ načini pohrane podataka:

a) NEPOREDANA (heap) DATOTEKA

↳ zapisuje se na bilo koje slobodno mjesto u datoteci

↳ prosječno imamo $N/2$ čitanja da nađemo postojeći podatak

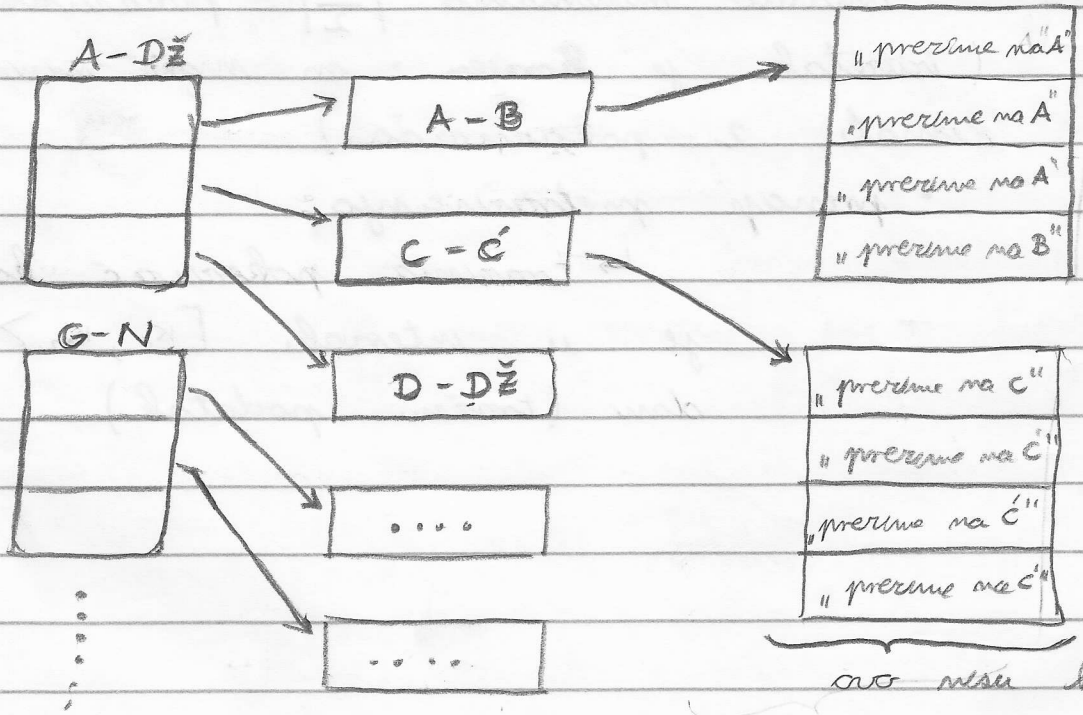
↳ naravno, N/2 čitanja imamo ako tražimo po primarnom ključu

↳ onakav način se koristi za:
- male relacije
- relacije koje uvijek čitamo cijele (npr. uvijek nešto agregiramo)

mbr	ime
15	Branimir
2	Ana
7	Janko
4	Ana
⋮	⋮

b) B-STABLA („Balanced Tree")

↳ pohranimo neku dodatnu informaciju kako bi „pametnije" organizirali podatke



ovo nisu knjige nego samo listići koji govore gdje se knjiga nalazi

↳ **red stabla**: najveći broj djece koje čvor stabla može imati (oznaka „m")

↳ **balansirano stablo**:

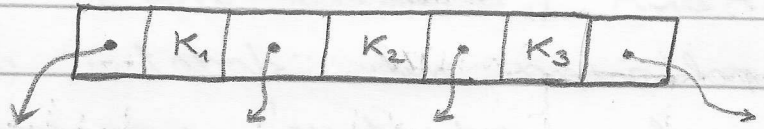
- stablo kod kojeg je put od svakog lista do korijena jednak

↳ **dubina stabla**:

- najdulji put od korijena do nekog lista stabla

↳ **struktura internog čvora**:

- umjesto se pokazači - ključ



- imamo maksimalno „m" pokazača
- imamo minimalno $\lceil \frac{m}{2} \rceil$ pokazača

(iznimka je **korijen** - on može minimalno imati 2 pokazača)

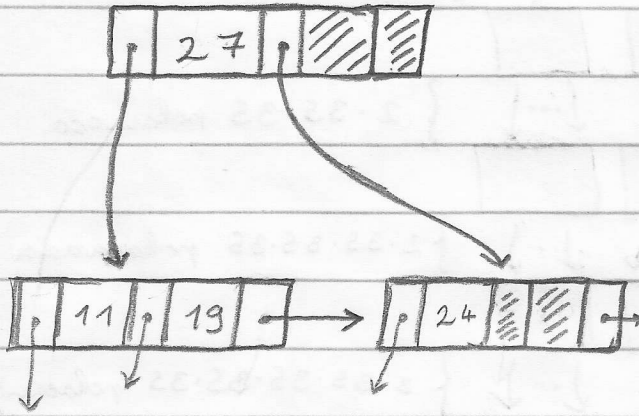
- princip pretraživanja:

↳ tražimo pokazač koji je u intervalu $[K_1, K_2>$ za dan (traženi podatak)

↳ **struktura lista:**

- maksimalno „n-1" pokazivača
  (jedan pokazivač je rezerviran)
- minimalno $\lceil \frac{n-1}{2} \rceil$ pokazivača

↳ jedan rezervirani pokazivač služi za povezivanje listova
(jedan list pokazuje na sljedeći)
↳ to nam koristi za slijedno čitanje više podataka (nađemo najmanji zapis, onda se slijedno prošetamo po svim listovima, ive dok je potrebno - uoči da smo time dobili i sortirane podatke

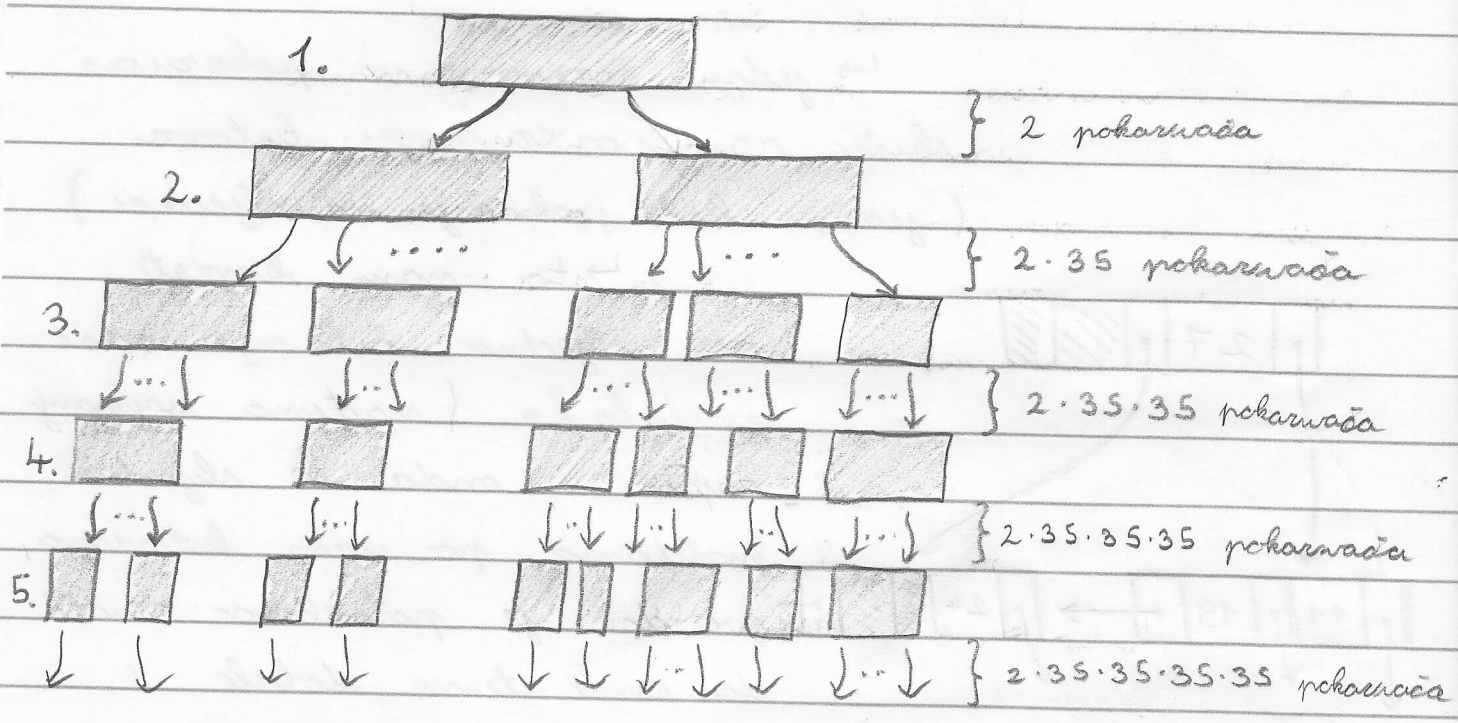


⇒ NAPOMENA: pisanje, brisanje i ažuriranje može inducirat drastičnu izmjenu b-stabla, to zatim može usporiti čitav podataka

↳ učinkovitost B-stabla: BITNO!!

PR: Imamo 10^6 n-torki, imamo red stabla 70!

↳ izgradimo najlošije stablo:



↳ radnja razina je previše jer ne možemo radovoljit uvjet $\lceil \frac{n-1}{2} \rceil$, a imamo preko 3 milijuna pokazivača (pokazivač ne može pokazivat na ništa, pa trebamo uzet razinu 4 (formalna razina 3, neubrojivši korijen))

↳ stoga trebamo obavit 5 U/I operacija da bi dobavili podatak:

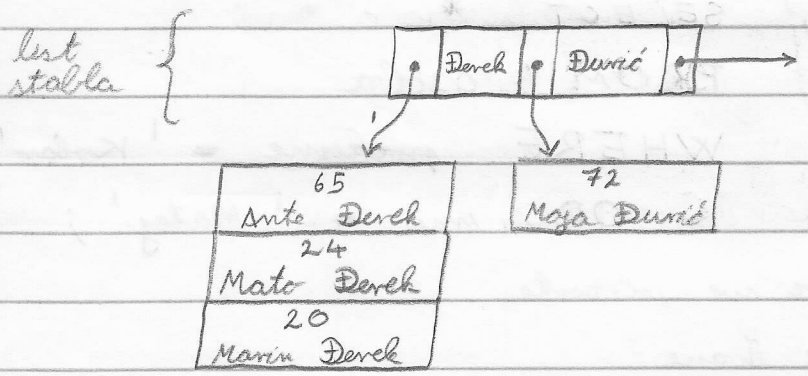
- čitanje korijena
- čitanje 2., 3., 4.
- čitanje podatka

⇒ IZGRADNJA B-STABLA U SQL-u : –

↳ primjer naredbe:

```
CREATE INDEX osoba_prez
ON osoba (prezime);
```

↳ stvara B-stablo koje će indeksirat osobe po prezimenu (pretpostavka da prezime ne mora bit jedinstveno za osobu – osobe sa istim prezimenom će biti grupirane na način da ista pokazivač pokazuje na zapise sa istim prezimenima)



↳ ključna riječ UNIQUE:

- sprječava ponašanje da, npr. dvije osobe imaju isto prezime – zahtjeva da se indeksira po ključu

↳ analizator upita je svjestan stabala neke tablice i on ih koristi za optimizaciju upita

⇒ glavni razlog NE-korištenja indeksiranja jest usporavanje operacija koje mijenjaju, brišu i dodaju atribute po kojem indeksirano (vidi primjer na prezentaciji)

⇒ SLOŽENI INDEKSI:

↳ indeksiranje po dva il više atributa

↳ primjet da složen indeks po prezimenu i imenu pomaže kod:

- traženja po prezimenu i imenu
- traženja po prezimenu

↳ no, takav indeks ne pomaže kod traženja samo po imenu

↳ također, ne pomaže kod upita sa OR uvjetom, tipa:

```
SELECT *
FROM osoba
WHERE prezime = 'Kolar'
   OR ime = 'Matej';
```

kod ovog upita svakako trebamo pretražiti brez ne „n"-torke da pronađemo sve Ivane

BITNO!!

ZAD: Napisi najmanji broj indeksa tako da se na navedeni upit efikasno izvrši:

1. SELECT * FROM stud ORDER BY ime DESC, prez;
2. SELECT * FROM stud ORDER BY ime DESC, prez DESC;
3. SELECT * FROM stud ORER BY ime, prez, plaStan;
4. SELECT * FROM stud WHERE prez='Novak' AND ime='Ivo';
5. SELECT * FROM stud WHERE plaStan > 51000
   ORDER BY plaStan DESC;

↳ **trebamo indekse koji su po ovome:**

a) ime DESC, prez
b) ime, prez, plaStan
c) plaStan

↳ **naredba izgleda ovako:**

```
CREATE INDEX i1
ON stud (ime DESC, prez);
```

↳ **napomena:** prihvaćeno da za upit broj 4 pomaže i indeks pod „a":

ime DESC, prez

INTEGRITET

⇒ INTEGRITETSKA OGRANIČENJA:

- osiguravaju da izmjene podataka koje obavljaju korisnici ne narušavaju konzistentnost podataka
- npr. netko je greškom unio ocjenu 55

⇒ stoga, shema baze podataka se sastoji od:

- skupa relacijskih shema

$$\mathcal{R} = \{R_1, \ldots, R_N\}$$

- skupa integritetskih ograničenja

$$\mathcal{IC} = \{IC_1, \ldots, IC_N\}$$

⇒ RJEČNIK PODATAKA:

↳ tablica u kojoj su pohranjene:

- relacijske sheme
- integritetska ograničenja
- informacije o tablicama
- ostale informacije o samoj bazi

↳ na taj način, administrator baze podataka pomoću SQL upita može administrirati bazu podataka

⇒ VRSTE INTEGRITETA

↳ Entitetski integritet ⇒ niti jedan od atributa primarnog ključa ne smije biti NULL vrijednost

↳ Integritet ključa ⇒ u relaciji ne smiju postojati dvije „n"-torke s jednakim vrijednostima (bilo kojeg) ključa

↳ Domenski integritet ⇒ atribut mora biti iz definirane domene

↳ Ograničenje NULL vrijednosti ⇒ ograničenje na neke atribute koji nisu ključ, ali ne želimo NULL vrijednost u njima (npr. ime, prezime)

↳ Strani ključ i referencijski integritet ⇒ možemo se ne-referencirati ili se možemo referencirati samo na postojeću n-torku

⇒ DEFINICIJE ENTITETA U SQL-u:

- Entitetski + Integritetski:
  PRIMARY KEY
- Integritetski:
  UNIQUE
- Domenski + opća pravila integriteta
  „tip podatka" + CHECK

- Ograničenje NULL vrijednosti

  NOT NULL

- Referencijski integritet

  FOREIGN KEY ključ

  REFERENCES tablica(ključ)

  ↳ ako dani atribut ne postoji u drugoj (stranoj) tablici, unosi se NULL

⇒ pripaziti da ograničenje stranog ključa može smetati kod brisanja i izmjene:

° npr. želimo obrisati nastavnika koji je otišao u mirovinu, ali je njegova šifra strani ključ u tablici „ispit"

↳ željeno ponašanje definiramo ovako:

ON DELETE NO ACTION

ON DELETE SET NULL

ON DELETE SET DEFAULT

ON DELETE CASCADE

↳ napomena: cascade briše sve referencirajuće „n"-torke (OPREZ!)

⇒ ograničenja integriteta možemo imenovati:

ocj SMALLINT NOT NULL CONSTRAINT ocjOgr

ime ograničenja

VIRTUALNE I PRIVREMENE RELACIJE

⇒ CREATE TABLE radi temeljne relacije koje se trajno pohranjene u bazi

⇒ PRIVREMENA RELACIJA:

↳ relacijska shema čiji je sadržaj privremen (također je i sama shema privremena)

↳ „SQL-Session" je kontekst za vrijeme jednog „log-in"-a korisnika

↳ privremena relacija je privatna za korisnika i ona se briše nakon završetka sjednice (session-a)

↳ stvaranje privremene relacije:

CREATE TEMP TABLE

↳ ako stavimo podatke iz jedne relacije u neku privremenu te ih ratim promjenimo u originalnoj tablici, privremena relacija neće imati te promjene - ona sadržava kopiju originalnih podataka

⇒ VIRTUALNA RELACIJA: (pogled)

↳ stvara se ovako:

CREATE VIEW

↳ to je zapravo pogled na odredenu relaciju

↳ prednost pogleda jest to da se uvijek prikazuju ažurni podaci (za razliku od privremene relacije)

↳ stoga, pogled se može koristit kao normalna relacija koja ispod sebe ima SQL upit, pa se automatski ažurira

↳ u SELECT STATEMENTU create view naredbe ne možemo koristit ORDER BY

↳ virtualna relacija je trajno vidljiva u bazi podataka i to za sve korisnike

⇒ pogledaj kako baza optimizira dva select upita od kojih je jedan u CREATE VIEW, a drugi uzima neki podatak iz pogleda

↳ to se zad kombinacijom dva SELECT upita iz tih naredbi

⇒ može se desit problem „propadanja" kroz pogled

↳ upisujemo u pogled, no zapravo upisujemo u temeljnu tablicu, a upisana „n"-torka ne pripada uvjetu pogleda - u pogledu neće biti umetnuta „n"-torka

↳ taj problem se može javit i sa ažuriranjem tablice preko pogleda

↳ stoga, moramo bit oprezni ako tretiramo pogled kao relaciju

↳ WITH CHECK OPTION:

- ključna riječ koja spriječava propadanje - ako neka „n"-torka nakon izmjena neće proći uvjet pogleda, ta izmjena se neće izvršiti

⇒ neizmjenjive virtualne relacije:

↳ ako nam je virtualna relacija neka agregacija (npr. AVG), onda ne možemo dodat neku „n"-torku s projekcom 4 jer nije jednoznačno određeno kolko i kakve „n"-torke treba ubacit u temeljnu relaciju

↳ postoje općenita pravila koja određuju je li neka virtualna relacija izmjenjiva (vidi prezentaciju i prouči pravila s razumijevanjem)

BITNO!

⇒ pravila je da kada kod radimo pogled koj će omogućit izmjene, stavljamo WITH CHECK OPTION

⇒ EKSTERNE SHEME:

↳ odredujeju tko vidi što

↳ time pomoću pogleda modificiramo pristup podacima na razini same baze podataka

↳ koristimo za neke tvrtke il organizacije za više razina pristupa

OPTIMIZACIJA UPITA

⇒ stvaran SQL-kod se prilikom prevođenja svodi na nižu razinu i tamo se optimizira
↳ ta niža razina je ustvari izraz relacijske algebre

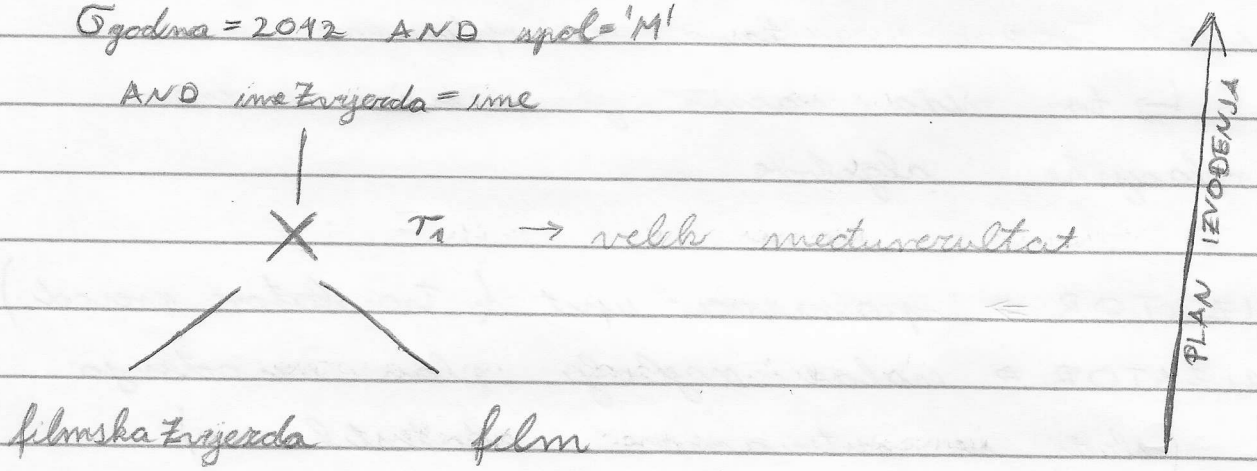
⇒ ANALIZATOR ⇒ parsira upit (↓ translator precd)
⇒ OPTIMIZATOR ⇒ nalazi najbolji plan izvođenja (služi se rutinama i statistikama)
⇒ IZVRŠITELJ ⇒ izvršava optimiziran upit

⇒ statistike za optimizaciju se ne ažuriraju prilikom izvršavanja svakog upita jer to može znatno usporiti rad baze
↳ stoga, baza je dosljedno pametna da shvati kada treba ažurirati statistiku
↳ eksplicitno ažuriranje statistike:
UPDATE STATISTICS

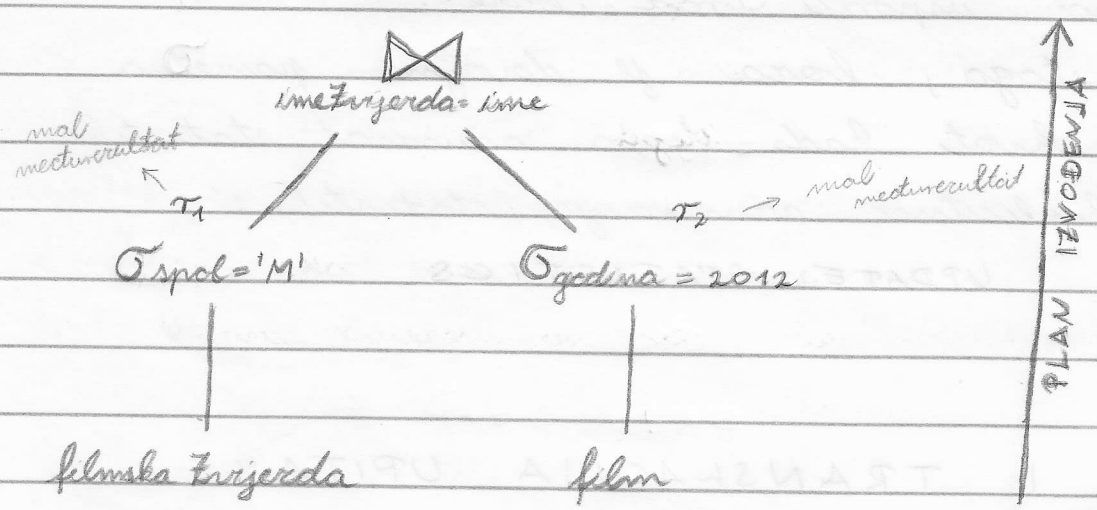
⇒ ANALIZA I TRANSLACIJA UPITA:
↳ prvi korak u procesu izvođenja
↳ analiza se još naziva i parsiranje
↳ stoga, translacija je prevođenje na razinu relacijske algebre

⇒ **Ideja optimizacije:**

↳ cilj je da međurezultati ne narastu na ekstremno velike tablice



↳ ekvivalentni upit sa drugačijim planom izvođenja:



ovaj upit je efikasniji jer se selekcija radi prije, odluka o spajanju se donosi odmah kod spajanja (bez međurezultata)

⇒ **ekvivalentni izrazi relacijske algebre:**

↳ koriste se svojstva relacijskih operacija

↳ komutativno, asocijativno:

- prirodno spajanje
- kartezijev produkt
- unija
- presjek

↳ theta-spajanje nije uvijek asocijativno (vidi primjer sa slajdova)

↳ theta-spajanje je asocijativno, ako koristimo samo atribute iz relacija koje spajamo

↳ **selekcija se može:**

a) podijeliti

$$\sigma_{C1 \text{ AND } C2}(r) = \sigma_{C1}(\sigma_{C2}(r)) = \sigma_{C2}(\sigma_{C1}(r))$$

$$\sigma_{C1 \text{ OR } C2}(r) = \sigma_{C1}(r) \cup \sigma_{C2}(r)$$

b) potisnuti:

- ideja je da se selekcija izvrši prije spajanja i tako se smanji broj podataka koji trebamo spojiti (to ne mijenja uvijek ⚠ Vidi slajdove ⚠)

⇒ **HEURISTIČKA PRAVILA:**

↳ na temelju iskustva se donosi odluka o optimizaciji plana izvođenja (iskustvo nije 100% učinkovito)

⇒ IZRAČUNAVANJE TROŠKOVA OPERACIJA:

↳ najveći troškovi dolaze od veličina međurezultata, stoga nam je bitna procjena veličine međurezultata selekcije:

$V(A, r) = 5$ → ovo znači da atribut „A" relacije „r" može poprimiti 5 različitih vrijednosti

• procjenjujemo da $\frac{1}{5}$ tablice „r" ima vrijednost atributa A jednaku $c \in values(A)$

1) $\sigma_{A=c}$

$$N(b) = \frac{N(r)}{V(A,r)}$$

• ako imamo nejednakost, statistički je pokazano da u prosjeku ostaje trećina „n-torki":

2) $\sigma_{A<c}$

$$N(b) = \frac{N(r)}{3}$$

• ako imamo kompliciranju nejednakost, veličina međurezultata je:

3) $\sigma_{A=c \wedge B>3}$

$$N(b) = \frac{N(r)}{3 \cdot V(A,r)}$$

↓

međurezultat

↳ također, bitno je pogledati veličine međurezultata spajanja:

↳ promatramo prirodno spajanje relacija „r" i „s": $N(r)$, $N(s)$

• relacije nemaju zajedničkih atributa:

1) $X \cap Y = \phi$

$$N(t) = N(r) \cdot N(s)$$

• relacije imaju zajednički atribut, i to je ključ jedne relacije:

2) $X \cap Y = K(r)$

$$N(t) = N(s)$$

} ako formula nije jasna, onda skraćuj relaciju

• relacije imaju zajedničke atribute, ali to nisu ključevi:

3) $X \cap Y = \phi$

↳ pretpostavka: zajednički atributi A je jednako u obje relacije (jednako V):

$$N(t) = \gamma \cdot \frac{N(r)}{V} \cdot \frac{N(s)}{V} = \frac{N(r) \cdot N(s)}{V}$$

} nacrtaj i objasni

↳ ako gornja pretpostavka ne vrijedi, uzimamo veći od mogućih nazivnika:

$$N(t) = \frac{N(r) \cdot N(s)}{\max[V(r), V(s)]}$$

⇒ METODE PRISTUPA PODACIMA U RELACIJI:

a) SLIJEDNO ČITANJE n-TORKI:

↳ ako nemamo upotrebljiv indeks il se čitaju sve il gotovo sve n-torke iz relacije

b) KEY-ONLY INDEX SCAN:

↳ ako su svi podaci koji se čitaju dijelovi jednog indeksa, sve potrebne vrijednosti se mogu pronaći u indeksnim blokovima

c) INDEX SCAN:

↳ klasično čitanje blokova podataka preko indeksa - tj. treba se čitati i blokovi s podacima, a ne samo blokovi s indeksima

⇒ METODE SPAJANJA RELACIJA:

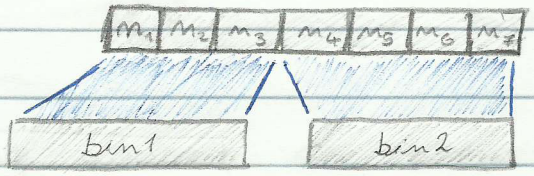
a) SPAJANJE UGNIJEŽĐENJIM PETLJAMA

↳ idemo preko svih n-torki tablica pomoću vanjske i unutarnje petlje i za svaku n-torku odlučujemo hoće li se ona uspješno spojiti

↳ NAPOMENA: indeks unutarnje petlje (relacije) uvelike pomaže kod brzine izvođenja upita - stoga, ako optimizator zaključi da se isplati napraviti privremeni indeks samo za spajanje

b) RASPRŠENO SPAJANJE:

↳ ideja je n-torke podijeliti u podskupove pomoću hash-funkcije i tako smanjiti kolčinu redaka koje slijedno pretražujemo



↳ ova operacija „hashiranja" koštа manje od izgradnje indeksa, a može uvelike pomoći

↳ hash-funkcija se također primjenjuje na unutarnju relaciju i to na atribute po kojima se spaja (kako bi kasnije mogli locirati „ii-torke koje zadovoljavaju uvjet spajanja)

Pr: Zadane su relacije:

MJESTO = { plbr, nazMjesto }

N(mjesto) = 500 V(nazMjesto, mjesto) = 500

STUD = { mbrStud, prezStud, plbr }

indeks

N(stud) = 10000

V(prezStud, stud) = 1000

ISPIT = { mbrStud, nfPred, datRok, ocjena }

N(ispit) = 100 000

V(ocjena, ispit) = 5

Optimizator ima na raspolaganju sljedeće metode

- sljedno pretraživanje
- indeksirano pretraživanje (ber autoindeksa)

Prikaži optimizaciju upita za oraj upit:

```
SELECT * FROM mjesto, stud, ispit
WHERE mjesto.plbr = stud.plbr
  AND stud.mbrStud = ispit.mbrStud
  AND stud.prezStud = 'Horvat'
  AND ispit.ocjena = 5;
```

a) Prikaži stablo upita za početni plan izvođenja!

b) Heuristički optimiziraj upit primjenom selekcija na odsječcima mjesima

c) Na stablu prikaži način pristupa podacima

d) Izračunaj procjenu međurezultata

a)

$\sigma_{ocjena = 5}$

|

$\sigma_{prez.Stud = 'Horvat'}$

|

$\sigma_{stud.mbr\ Stud = ispit.mbr\ Stud}$

|

$\sigma_{mjesto.pbr = stud.pbr}$

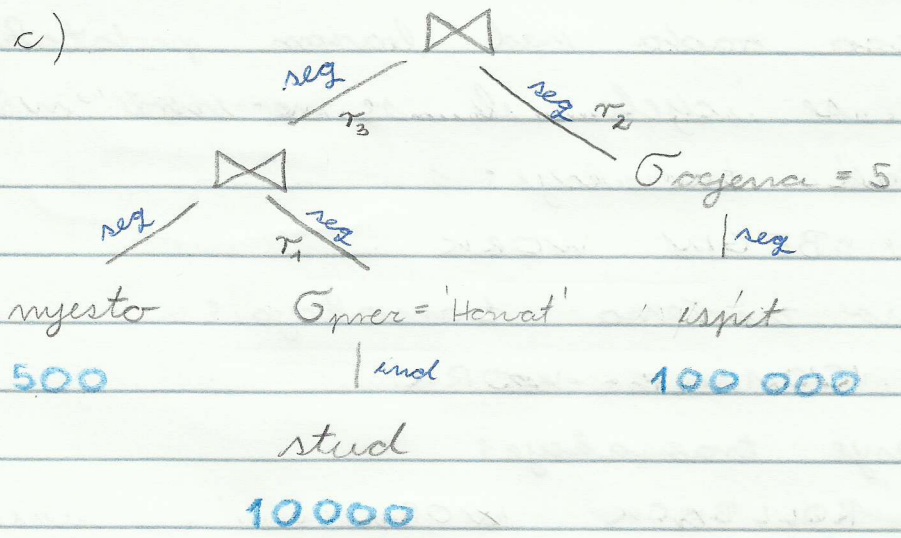
|

$\times$

$\times$ — $ispit$

$mjesta$ — $stud$

b) c)



d) $N(\tau_1) = 10$

$N(\tau_2) = 20\,000$

$N(\tau_3) = 10$

$\Rightarrow$ pravyet do broj „n-torki" u konačnom rezultatu nije bitan jer na njega izvana ne možemo utjecati

TRANSAKCIJE I OBNOVA BAZE PODATAKA U SLUČAJU RAZRUŠENJA

⇒ Sustav Upravljanja Bazom Podataka:
- „Database Management System" (DMS)
  - ↳ sakriva od korisnika detalje fizičke pohrane
  - ↳ optimira upite
  - ↳ štiti podatke:
    - a) integritet podataka
    - b) kontrola pristupa
  - ↳ upravlja transakcijama

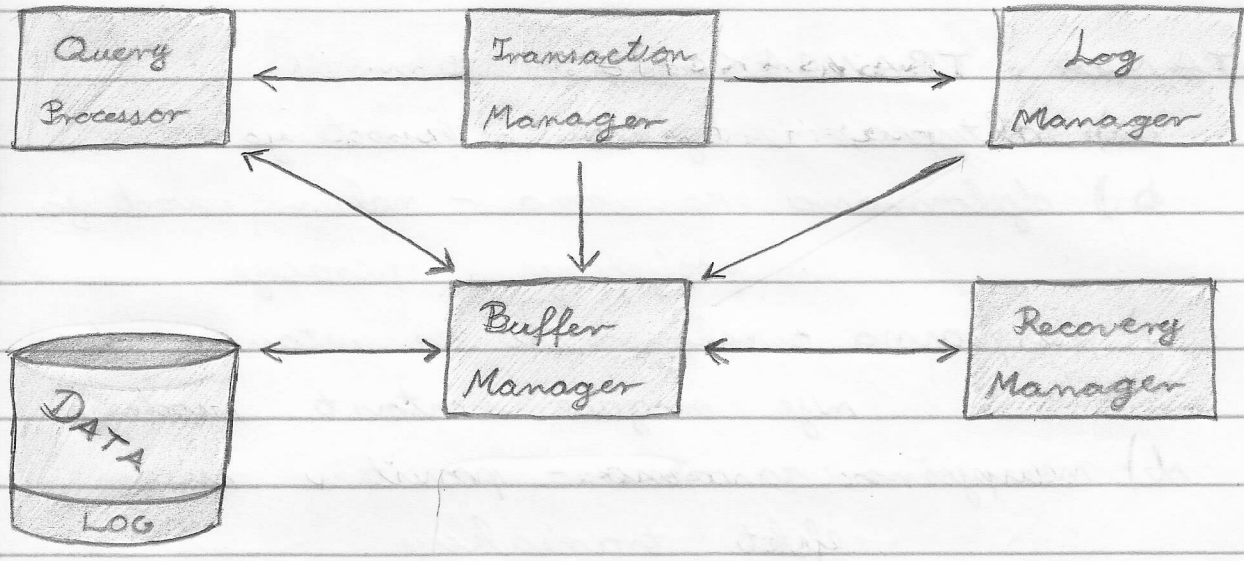
⇒ TRANSAKCIJA:
- ↳ jedinica rada nad bazom podataka
- ↳ izvrši se cijela il se ne izvrši ništa
- ↳ početak transakcije:

  BEGIN WORK
- ↳ uspješno završena transakcija:

  COMMIT WORK
- ↳ poništenje transakcije:

  ROLLBACK WORK

⇒ baza najčešće ima posebnu jedinicu za upravljanje transakcijama:
- ↳ ona osigurava željeno ponašanje u radu s transakcijama



⇒ POHRANJENA PROCEDURA:

↳ enkapsulacija više SQL upita u jedan okvir koj mogu sadržavat neke lokalne varijable

↳ ti upiti se izvršavaju proceduralno

↳ procedure su korisne u radu s transakcijama

⇒ NAPOMENA: UPDATE, DELETE i INSERT rade sa svi-transakcijama kada rade nad više podataka kako bi se osigurala konzistentnost

↳ primjer nekonzistentnosti:

• prvih 100 n-torki se je uspješno ažuriralo, a ostale se nisu zbog nestanka struje

⇒ STANJA TRANSAKCIJE:

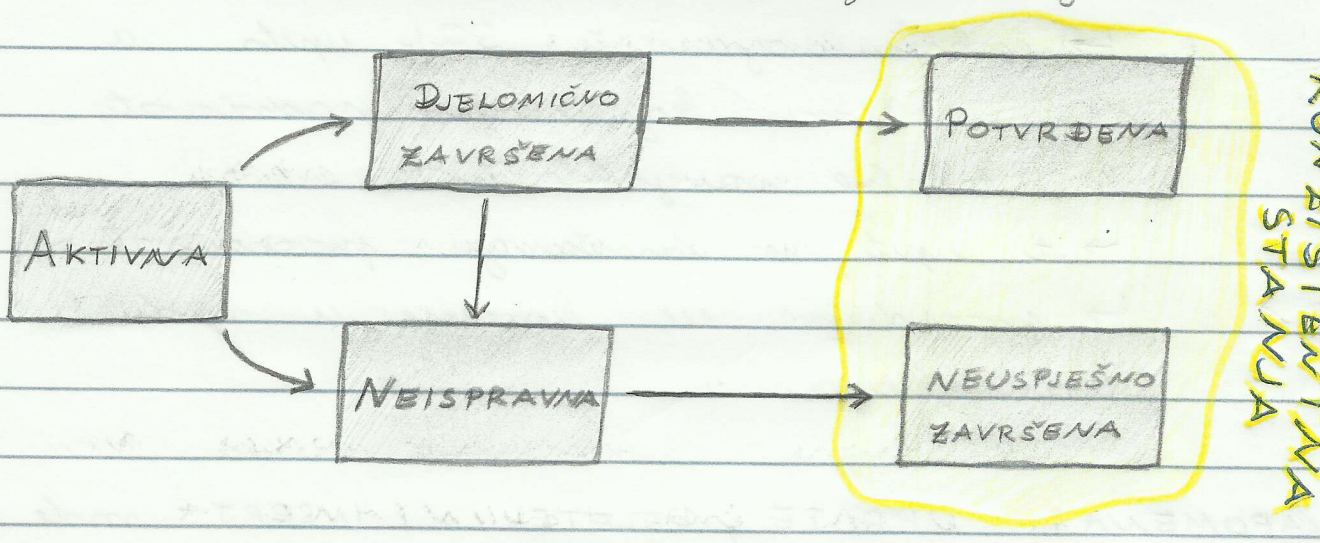
a) aktivna - tijekom izvođenja

b) djelomično završena - nakon izvođenja posljednje operacije

c) neispravna - nakon što se ustanovi da nije moguće nastaviti izvođenje

d) neuspješno završena - poništen su efekt transakcije

e) potvrđena - uspješno izvedena transakcija te su sve promjene trajne



⇒ SVOJSTVA TRANSAKCIJE:

↳ skraćeno ACID

a) ATOMICITY - nedjeljivost transakcije

b) CONSISTENCY - transakcijom bazu prelazi iz jednog konzistentnog stanja u drugo

c) ISOLATION - ako se izvršavaju dvije transakcije paralelno, njihov učinak mora biti isti kao da su se izvršavale jedna za drugu

d) DURABILITY - ako je transakcija obavila svoj posao, izmjene moraju ostati iako se je kvar dogodio neposredno nakon obavljanja transakcije

⇒ OBNOVA BAZE PODATAKA:

↳ mehanizam koji osigurava konzistentnost u slučaju kvara ili pogreške

↳ uzroci pogrešaka:

- pogreške opreme
- kolebanje izvora energije
- greška operacijskog sustava
- pogreške operatera
- ⋮

↳ za obnovu od uništenja medija nam je nužno potrebna REDUNDANCIJA

↳ redundancija se postiže:

a) zrcaljenjem podataka (mirroring)

b) sigurnosna kopija (backup)

c) dnevnikom izmjena

↳ postupak koji omogućuje obnovu:

1) periodičke sigurnosne kopije

2) svaka izmjena se zapisuje u dnevniku izmjena

⇒ što radimo kada nastane kvar?

a) baza je uništena

↳ prvo, uzimamo posljednji „backup" i pomoću dnevnika izmjena dovedemo do trenutnog stanja

b) baza nije uništena

↳ pomoću dnevnika izmjena, poništavaju se one transakcije koje su bile prekinute dok je baza bila u nekonzistentnom stanju

⇒ KONTROLNA TOČKA:

↳ zapis svih aktivnih transakcija u dnevnik izmjena

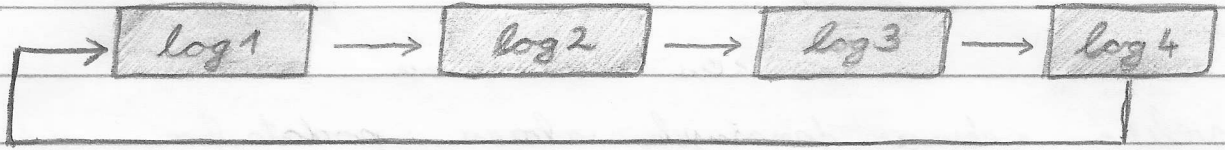
↳ služi kako bi sprječilo ponavljanje svih transakcija koje nisu završile od nastanka baze podataka

↳ kontrolna točka također „flush"-a buffere te se sve promjene spremaju u bazu

↳ kontrolna točka se radi otprilike svakih 5 minuta

↳ vidi sadržaj kontrolne točke na slajdu!

⇒ dnevnici se ciklički pišu jedan preko drugoga:



⇒ dnevnik se mora održati u memoriji sve dok je barem jedna transakcija iz dnevnika aktivna

↳ potencijalni problem su IDLE transakcije:
- njih se rješavamo pomoću „timeout" i „grace" perioda

KONTROLA ISTODOBNOG PRISTUPA

⇒ velika većina današnjih baza podataka jest višekorisnička
↳ stoga, treba uvesti kontrolu istodobnog pristupa

⇒ ne možemo jednostavno staviti u red sve transakcije dok god postoj aktivna transakcija
↳ to bi uzrokovalo da je baza podataka jako spora jer ne možemo računati na to da su sve transakcije kratke
↳ stoga, moramo uvesti paralelizam

⇒ što ako dvije transakcije rade izmjene nad istim resursom, a želimo da se odvijaju paralelno?
↳ ovo je temeljni problem i treba ga riješiti
↳ možemo uvesti PRIVIDNI PARALELIZAM
(malo vremena jednu, malo drugu transakciju - time duge transakcije neće jako dugo odgoditi izvođenje kraćih transakcija)

↳ no, previše paralelizma može dovesti do nekonzistentnog stanja, pa ćemo morati koristit ~~rollbackanje~~

⇒ PROBLEMI ISTODOBNOG ČITANJA:

a) PRLJAVO ČITANJE (dirty read)

↳ npr. transakcija 1 radi nešto nad podacima, transakcija 2 ih tada pročita te ratun transakcija 1 pozove ROLLBACK WORK - pročitali smo ~~prljave~~ podatke

b) NEPONOVLJIVO ČITANJE (nonrepeatable read)

↳ ista transakcija dvaput ~~dvaput~~ istog upita dobiva drugi rezultat

c) SABLASNE N-TORKE (phantom rows)

↳ ista transakcija dvaput istog upita dobiva više n-torki u rezultatu (il manje n-torki)

d) IZGUBLJENA IZMJENA (lost update)

↳ npr. prodamo dvije karte za isto mjesto u kinu zbog istodobnog pristupa i izmjena baze podataka

⇒ PROTOKOL ZA ZAKLJUČAVANJE:

↳ njime se bavi **locking manager**

↳ postoji više vrsta zaključavanja:

a) **ključ za izmjenu:**

WRITE LOCK, EXCLUSIVE LOCK

↳ nit jedna druga transakcija ne može zaključat resurs dok ga ne otključa transakcija koja ga je zaključala

↳ povezuju ga naredbe:

UPDATE, DELETE

b) **ključ za čitanje:**

READ LOCK, SHARED LOCK

↳ nit jedna transakcija ne može zaključat resurs za izmjenu, ali može za čitanje

↳ povezuju ga naredbe:

npr. SELECT

⇒ iz gornjih podataka o ključevima slijedi **matrica kompatibilnosti:**

T_1 \ T_2	WRITE	READ	NO LOCK
WRITE	X	X	✓
READ	X	✓	✓

⇒ no, samim protokolom za zaključavanje kojeg smo do sada otkrili podatke - još uvijek možemo doći u nekonzistentno stanje

↳ stoga, uvodi se TWO PHASE LOCKING PROTOCOL:

a) growing phase - u toj fazi se vrši zaključavanje resursa

b) shrinking phase - u toj fazi se vrši otključavanje resursa

↳ ovime smo spriječili nekonzistentnost, ali je moguć deadlock - SUBP to rješava tako da rollback-a jednu transakciju i otpusti ključeve te transakcije

⇒ GRANULACIJA PODATAKA:

n-torka — fina granulacija

fizička stranica

relacija

baza podataka — gruba granulacija

↳ odabrom fine granulacije uvećava se konkurentnost (simultanost) i troškovi portarjanja ključeva

↳ odabrom grublje granulacije smanjuje se konkurentnost i troškovi portarjanja ključeva

⇒ odabir granulacije mogućeg ovisi o podacima s kojim radimo, te ponekad o baz podataka s kojom radimo - svaka baza ima neki svoj standard

⇒ zaključavanje možemo definirat na razini tablice prilikom stvaranja same tablice:

```
CREATE TABLE (
    a   INTEGER
    b   CHAR(2)
) LOCK MODE ROW;
```

⇒ RAZINE IZOLACIJE:

KONZISTENTNOST ↓

a) READ UNCOMMITTED - podaci se čitaju bez provjere

b) READ COMMITTED - čitaju se isključivo potvrđene n-torke

c) REPEATABLE READ - osigurava ponovljivo čitanje podataka u okviru transakcije (ne sprječava pojavu zabranjenih n-torki)

d) SERIALIZABLE - najveća razina izolacije koja osigurava da neće doći do svjednog od problema vezanog čitanja

⇒ razumu vrlo je važno naglasiti odmah na početku same transakcije:

```
BEGIN WORK;
SET TRANSACTION ISOLATION LEVEL
    READ COMMITTED;
        .
        .
        .        } rad transakcije
COMMIT WORK;
```

SIGURNOST BAZE PODATAKA

⇒ ne smijemo miješat integritet i sigurnost:

↳ integritet - operacije nad podacima su ispravne

↳ sigurnost - korisnici koji obavljaju operacije nad podacima su ovlašteni za obavljanje tih operacija

⇒ narušavanje sigurnosti:

- neovlašteno čitanje, pisanje i uništavanje

⇒ ZAŠTITA:

a) na razini SUBP ⇒ spriječit pristup korisnicima koji nisu ovlašteni

b) na razini OS-a ⇒ spriječit pristup dijelovima diska gdje je baza

c) na razini mreže ⇒ spriječit presretanje poruka na mreži (sniffing)

d) fizička zaštita ⇒ zaštita od provale i krađe medija bare podataka

e) zaštita na razini korisnika ⇒ zaštita od ovlaštenih ljudi koji su, npr. primili mito

⇒ TRI RAZINE ZAŠTITE SIGURNOSTI:

1) IDENTIFIKACIJA – tko si ti?
2) AUTENTIFIKACIJA – dokaži da si to ti!
3) AUTORIZACIJA – dodjela ovlasti!

↳ za identifikaciju i autentifikaciju se najčešće koriste username i password

⇒ pravila autorizacije se pohranjuju u uređenim trojkama:

(tko, što, nad čim)

↑ koji korisnik ↑ koje operacije ↑ nad kojim podacima

⇒ MANDATNA KONTROLA PRISTUPA (MAC):

↳ koristi se klasifikacijska razina nad objektima (hijerarhija)

↳ svaki korisnik sa svojom dozvolom može pristupiti danom podatku

⇒ DISKRECIJSKA KONTROLA PRISTUPA (DAC):

↳ ovime sjema se u bazi

↳ sve dozvole se pohranjuju u uređene trojke opisane gore,

< korisnik, čitanje, ispit >

< korisnik, pisanje, ispit >

⇒ OBJEKT ⇒ sve čemu možemo definirat pristup
⇒ to su: relacija, atribut,
virtualna relacija, baza podataka

⇒ VLASNIK OBJEKTA ⇒ korisnik koji je kreirao objekt
⇒ on ima sve ovlasti nad objektom koji je stvorio, može ih dat drugima
↳ napomena: administrator je također vlasnik objekta baze podataka

⇒ VRSTE DOZVOLA U SQL-u:

DATABASE PRIVILEGE
1) CONNECT ⇒ uspostavljanje SQL-sjednice, davanje operacija za koje korisnik ima dozvolu
2) RESOURCE ⇒ „connect" + kreiranje novih relacija
3) DBA ⇒ „resource" + sve vrste operacija

TABLE PRIVILEGE
a) SELECT
b) UPDATE
c) INSERT
d) DELETE
e) REFERENCES
f) INDEX
g) ALTER
h) ALL PRIVILEGES

⇒ sintaksa za dodjeljivanje ovlasti:

GRANT dbPrivilege TO {PUBLIC | userList}
REVOKE dbPrivilege TO {PUBLIC | userList}

GRANT tablePrivilegeList ON {table | view}
TO ...
[WITH GRANT OPTION];

JEDAN POGLED ILI TABLICA

DAJE MOGUĆNOST DALJNJEG PROPAGIRANJA DOZVOLA

REVOKE tablePrivilegeList ON {table | view}
TO ...
[CASCADE | RESTRICT]

UKLONI PRIVILEGIJE KORISNIKU I SVIMA KOJIMA JE ON PROPAGIRAO (UKLANJA SAMO PROPAGIRANE OVLASTI)

UKLONI DOZVOLE SAMO AKO IH/JU KORISNIK NIJE DALJE PROPAGIRAO

⇒ SINONIMI:

↳ privatni sinonimi se specifični za naziv za neku tablicu od nekog korisnika

↳ ovo je lošo za vanjsku kompatibilnost baze podataka kako nebi nastali problemi povezani s vanjskom aplikacijom prilikom promjene naziva tablice il pogleda

⇒ DODJELJIVANJE ISTIH DOZVOLA VELIKOM BROJU KORISNIKA:

↳ definiraju se uloge:

CREATE ROLE

↳ tako preko uloga lakše i sigurnije definirama dozvole korisnika ovisno o njihovoj ulozi u danoj ustanovi ili sustavu

POHRANJENE PROCEDURE I FUNKCIJE

⇒ korisna SQL-a su raktjeval funkcije ra:
mpr. brojanje znakova, provjere je l nek string samo nr slova, ...
↳ stoga, uvedene su proceduralni dijelovi u SQL jezik

⇒ POHRANJENA PROCEDURA:
• potprogram koji je pohranjen u vječnu podataka i izvršava se u kontekstu same baze podataka

⇒ FUNKCIJA → nešto vraća
⇒ PROCEDURA → nita ne vraća

⇒ primjer izrade funkcije:

```
CREATE FUNCTION broj_znamenki (nr CHAR(255))
        RETURNING SMALLINT AS broj

DEFINE brojac, i SMALLINT;
LET brojac = 0;
  :          = 0;
RETURN brojac;
END FUNCTION;
```

SINTAKSA PRIDRUŽIVANJA

⇒ kada smo načinil proceduru / funkciju, ona se nalazi u rječniku baze podataka, al ju samo vlasnik može izvršiti (il administrator)

↳ kako li dal dopuštenja, pišemo:

```
GRANT EXECUTE ON brojZnamenki
TO PUBLIC;
```

⇒ npr. ako netko u banci može samo prebacivati novac sa jednog računa na drug, nećemo mu dati UPDATE jer je to previše ovlasti

↳ izgradimo proceduru isključivo za prebacivanje novca i damo mu ovlast za izvršavanje te procedure

⇒ IZNIMKE:

↳ koristan mehanizam koj možemo koristit ako je, npr. radnik u banci poslao proceduru za prebacivanje novca s nepostojećim brojem računa - trebamo načinit svoju iznimku

```
IF ( SELECT COUNT(*) FROM racun
     WHERE brRacun = sRacunBr ) = 0 THEN
  RAISE EXCEPTION -746, 0, "Ne postoj račun";
END IF;
```

KOD ZA USER-DEFINED EXCEPTION

OKIDAČI

⇒ kao i procedure, okidači su alat koji su se naknadno pojavili u SQL

⇒ ideja je:

- kada se nešto dogodi negdje, to nešto mora imati utjecaja negdje drugdje
- to je EVENT-BASED PROGRAMMING, a u bazama podataka, to je zavrijedilo pod imenom EVENT - CONDITION - ACTION

on event
if condition then action

⇒ sintaksa:

CREATE TRIGGER ime Uplata Isplata
INSERT ON uplata Isplata EVENT
REFERENCING NEW AS nova Uplat Isplata
FOR EACH ROW
WHEN (...) CONDITION
(UPDATE ...
WHERE ...) ACTION

⇒ napomena: u tijelu okidača ne možemo izvršavati izvršnu, već za to radimo proceduru koju onda pozivamo u ACTION dijelu okidača

⇒ okidači su korisni i za praćenje rada nekog korisnika nad tablicama

↳ npr. prilikom izmjena, pohranjena te izmjene u neki „log" ili ih pošaljemo na e-mail

⇒ INSTEAD OF okidač:

- najčešće ih koristimo kada npr. ne možemo dodati „ni"-torku u neku virtualnu relaciju jer je nemoguća, onda umjesto te akcije dodamo „ni"-torku u neku drugu relaciju

ENTITY - RELATION MODEL BAZE PODATAKA

⇒ što ako nemamo bazu podataka - već ju treba dizajnirat

↳ to može bit teško - fraktetno i pametno načinit bazu podataka iz nule

↳ tu nam pomaže ER model

⇒ ENTITET ⇒ nešto što ima bit i posjeduje značajke pomoću kojih se može razlučit od okoline

⇒ SKUP VEZA ⇒ matematička relacija između „n"-entiteta

$$STUDENT = \{s_1, s_2, \dots, s_m\}$$

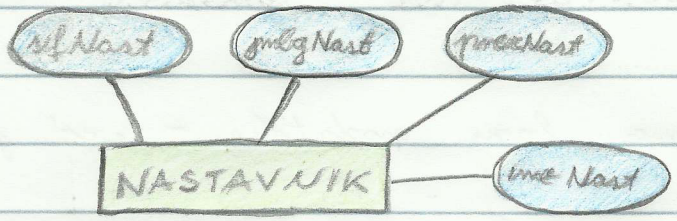
$$PREDMET = \{p_1, p_2, \dots, p_m\}$$

$$STUDENT \times PREDMET = \{(s_1, p_1), (s_1, p_2), \dots, (s_n, p_m)\}$$

$$SLUŠA \subseteq STUDENT \times PREDMET$$

SLUŠA ↓ SKUP VEZA

⇒ atribut entiteta upisuju se u oval, ali ako ih je previše, pišemo samo entitete:



NASTAVNIK = { sifNast, gmlgNast, prezNast, imeNast }

podcrtano u drugi razred jer to nije kompozitni ključ već je gmlgNast alternativni ključ

⇒ regularni entitet - može postojati sam od sebe

⇒ slabi entitet - ne postoji ukoliko ne postoji neki drugi entitet (označava se duplim pravokutnikom)

- to je egzistencijalno slabi entitet

⇒ identifikacijski slabi entitet - entitet se ne može odrediti bez atributa neke druge tablice

⇒ SPOJNOST VEZE: vrijednosti su jedan ili više

↳ oznaka od 0 do N : N

↳ oznaka od 1 do 1 : 1

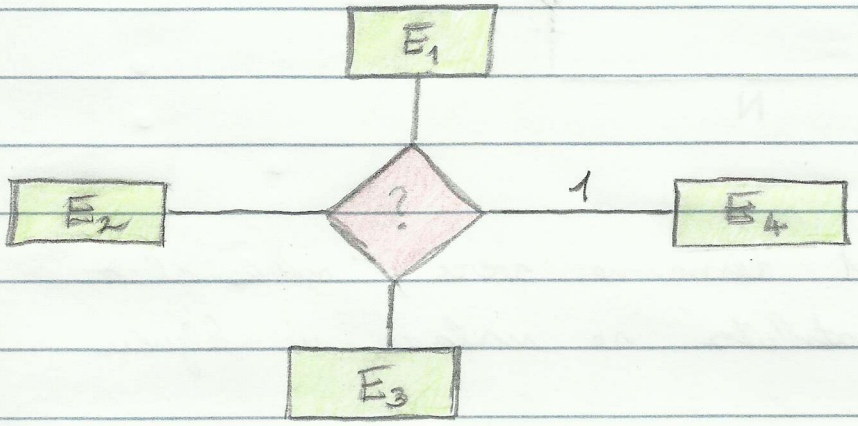
VEZA STUPNJA DVA

⇒ ATRIBUTI VEZA:

- veza mora sadržavati atribute koji su ključevi entiteta koje povezuje

↳ Teorijski teorem:

↳ u vezi N na 1, pornavanjem atribut na „N" strani, ga znam atribut na „1" strani



pornavanjem atributa u E_1, E_2, E_3 znam atribut u E_4

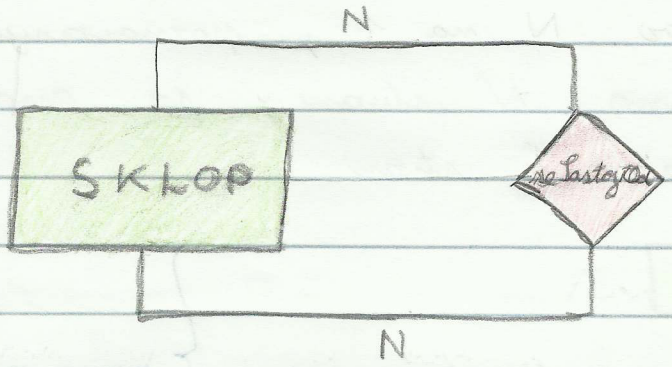
⇒ preslikavanje ER modela u RELACIJSKI MODEL:

- ovo nije teško, samo treba paziti o kakvim spojnostima se radi (VIDI SLAJD)

⇒ ALGORITAM OBLIKOVANJA ER MODELA:

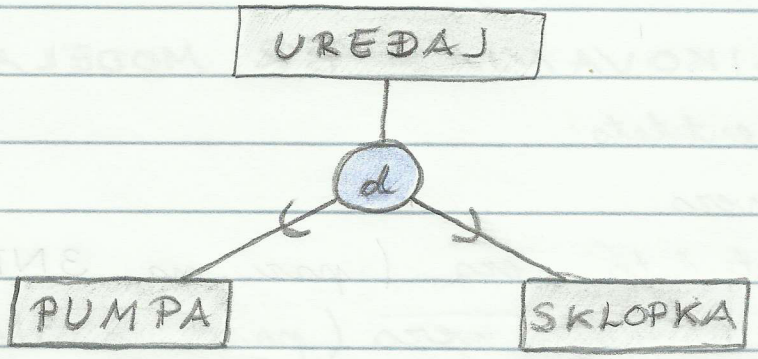
1) definiranje entiteta
2) definiranje veza
3) definiranje atributa veza (par na 3NF)
4) definiranje atributa veza (par na 3NF)

→ homogena mreža → svaki element može imat 0 il više podredenih i 0 il više nadredenih elemenata



⇒ NAPOMENA: kod ternarne veze, minimalno dva atributa se nalaze u ključu

⇒ SPECIJALIZACIJA: analogija naslijedivanja kod objektno orjentiranog programiranja
↳ iz nekog višeg objekta činimo podredeni koj naslijedi svojstva gornjeg



„d" → disjoint
(ekskluzivna specijalizacija)
↳ „pumpa" ne može biti „sklopka"

DISTRIBUIRANE BAZE PODATAKA

⇒ DISTRIBUIRANA BAZA:

- skup različitih baza koje su logički povezane
- ideja je da te baze funkcioniraju onako kako bi funkcionirala jedna centralizirana baza
- koriste se za:
  - ↳ baze s velikim brojem korisnika
  - ↳ baze koje primaju podatke sa različitih mjesta
  - ↳ ubrzanje baza podataka (u nekim slučajevima)
- distribuirane baze se teže dizajniraju i teško se održava integritet podataka (OVERHEAD)

⇒ oblikovanje distribuirane baze:

↳ podaci se smještaju u čvorove u kojima se najčešće koriste

OBLIKOVANJE DISTRIBUCIJE = FRAGMENTACIJA + ALOKACIJA

- podatke podijelimo na cjelne → podatke dodijelimo lokalnim bazama

↳ stupanj replikacije fragmenata:

- broj čvorova u kojima je fragment alociran

⇒ transakcije u distribuiranoj bazi podataka:

↳ velik problem se javlja kod sustava:

a) ATOMICITY i ISOLATION

↳ na nivou čvorova se treba uspješno izvršit pripadni dio transakcije - to se radi two-phase commit protokol

⇒ kvarovi su znatno češća pojava u distribuiranoj bazi podataka

↳ ako je baza relacijska, imamo jako puno scenarija upravljanja rizicima

NoSQL BAZE PODATAKA

⇒ BIG DATA: količina podataka prevelika za konvencionalne načine pohrane i obrade podataka

↳ pojam se pojavio 2012. godine, al sama ideja pohrane „svega" postoji znatno dulje vrijeme

⇒ relacijske baze podataka nisu se mogle nositi sa 3 glavna svojstva big data:

1) VELOCITY
2) VARIETY
3) VOLUME

} three Vs

↳ stoga, počinju se razvijati NoSQL baze podataka

⇒ Google je bio prvi koji se susreo s problemom „big data" kada je dizajnirao svoju tražilicu

↳ konvencionalna pohrana indeksa web stranica nije bila dobra, pa su krenuli dizajnirati vlastiti model baze podataka od nule (stvorili su datotečni sustav, sustav za paralelizam i vlastitu bazu podataka)