Predavanje 3

# Mehanizmi RSUBP

## Deklarativni mehanizmi

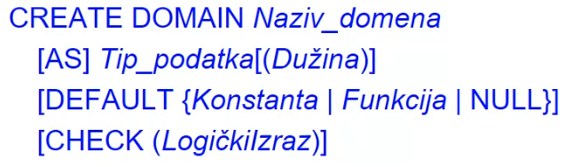
*… nastavak sa prethodnog predavanja…*

**Deklarativni mehanizam** – veci deo ponasanja mehanizma se podrazumeva u odnosu na deklaraciju koja je data

**Proceduralni mehanizmi** – mehanizam kod koga se kompletan ili vecina procesa programira

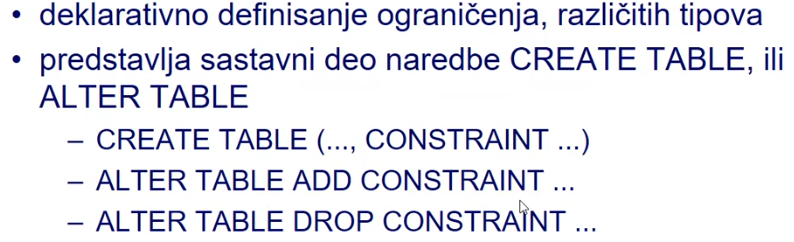
### CREATE DOMAIN

Nacin na koji mozemo kreirati domene SUBP-om onda kada je ova komanda podrzana u istom.



### SQL klauzula CONSTRAINT

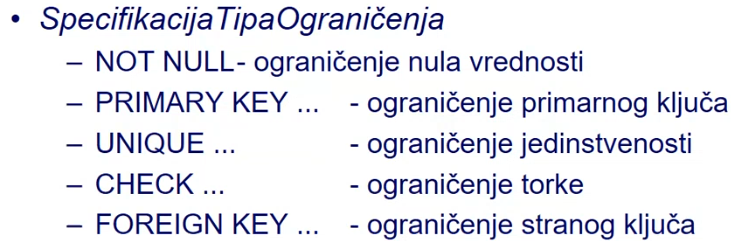
Mozemo dodavati ogranicenja koja sadrzaj tabele mora zadovoljavati.



Ove komande uspevaju onda kada trenutni sadrzaj tabele zadovoljava ta ogranicenja.



Sluzbena rec **CONSTRAINT** sa nazivom ogranicenja kao sto vidimo jeste opciona (kasnije ce biti objasnjeno kada se to ipak mora staviti)



#### Specifikacija trenutka provere ogranicenja

Kada koristimo mehanizam **constraint** klauzole u create ili alter table naredbi, odnosno kada smo deklarisali ogranicenje, treba znati da u zavisnosti od tipa ogranicenja koji je iskoriscen dalje, ogranicenje ce biti pokretano za kontrolu onoga sto mi zelimo da postignemo u smislu validnosti podataka na odredjene dogadjaje odnosno na odredjene operacije. Ono sto je pitanje jeste **u kom trenutku**  mi znamo da hocemo da se izvrsi provera vazenja naseg ogranicenja ?

Odnosno, pitanje je kada ce on(SUBP) odreagovati sa svojom proverom. Postoje dve opcije. **Momentalno[not deferrable]** okidanje provere vazenja ogranicenja i **odlozeno[deferrable]** okidanje provere vazenja ogranicenja. Klauzula **not deferrable[momentalno]** govori da je ne moguce odloziti kontrolu ogranicenja, tacnije *izvodi se u momentu izvodjenja* ***kriticne operacije***.

Recimo, ako je kriticna operacija insert a imamo **not deferrable**, to znaci da u paketu sa izvodjenjem operacije insert bice sprovedena i kontrola ovog ogranicenja. A u slucaju **deferrable** klauzuole, to znaci da je trenutak provere vazenja ogranicenja **odloziv**.

Klauzola **initally** oznacava kakvo je podrazumevano ponasanje. Ako je nas contstraint **not deferrable**, onda zadavanje klauzole  **initally**  nema nikakvu ulogu. Samo ako kazemo da je ogranicenje **defferable** onda sa sluzbenom recju **initally** mozemo da kazemo da li je po *defaultu* kontrola ogranicenja **immediate**(**momentalana**)ili **deferred** (**odlozena**). To znaci da je inicijalno tako, a kako ce biti kasnije, mi mozemo da menjamo na nivou transkacije odnosno na nivou sesije.

**Deklarativni** mehanizam radi tako da **initally immediate** kaze da se ogranicenje kontrolise i dalje **momentalno** na izvodjenje date operacije a **initally deferred** govori da je inicijalno i odlozeno izvodjenje kontrole vazenja ogranicenja i to za *commit.* (To je u deklarativnom mehanizmu tako, samo mozemo da kazemo I da se tako podrazumeva, tj. da se podrazumeva da je odlozeno za *kraj transakcije* odnosno za *komit* tj. za trenutak potvrdjenja transakcije).

Takodje mozemo da definisemo ponasanje i to na *nivou sesije.* Tipa ALTER SESSION ali moze i ALTER SYSTEM pa:

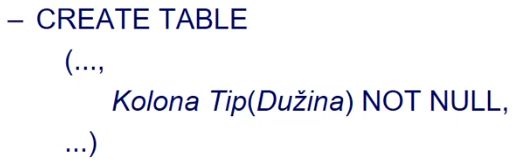


Bitno razumeti kod ovoga jeste da ne menjamo da li je **deferrable** ili **not deferrable** nego kakvo ce sad biti konkretno ogranicenje, kao sto smo pomocu klauzole **initially** to postavljali po defaultu, tako i sad definisemo za zeljena ogranicenja.

*U nastavku slede tipovi ogranicenja specifikacije,* ***not null, primary key, unique, check, foreign key.***

#### Not Null

Uvek se zadaje na *nivou obelezja* seme relacije (kolone tabele). Forma:



U prvoj formi vidimo *skracenu* sintaksu u kojoj je izostavljena sluzbena rec **constraint**. Problem sa prvom je sto se naziv sistemski generise, te posle ako zelimo da obrisemo neko ogranicenje po nazivu, imamo poteskoce dok saznamo koje je to ogranicenje.

Proverava se prilikom svakog pokusaja **insert**, **update** operacije (upisa nove vrednosti obelezja ili modifikacije postojece vrednosti obelezja). U slucaju pokusaja narusavanja ogranicenja, jedina moguca aktivnost sprecavanja operacije je **no action**.

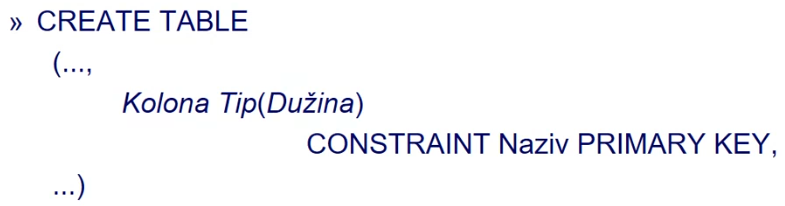
Bitno zapazanje vezano za *sprecavanje operacija* je da *ili celokupna naredba uspeva ili ne*. To znaci da ako 5000 torki zadovolji operaciju a 5001 ne, imamo situaciju da celokupna naredba nije uspesna! Odnosno, ponistavaju se izmene na svih 5001 torki a to se naziva **implicitni rollback**.

**Implicitni rollback** je rollback samo jedne naredbe(DLM naredbe). Nasa transakcija moze da ima puno drugih naredbi, i to onih kojih je vec realizovala, ako se desio implicitni rollback na neku od njih, mi mozemo obraditi taj *izuzetak* odnosno mozemo odluciti sta cemo uraditi ako se neka naredba nije uspesno izvrsila (rukovanje greskama). Podrazumevano ponasanje pada jedne naredbe implicira da **NECE** pasti transakcija! To znaci da je na programeru da rukuje izuzetcima odnosno greskama.

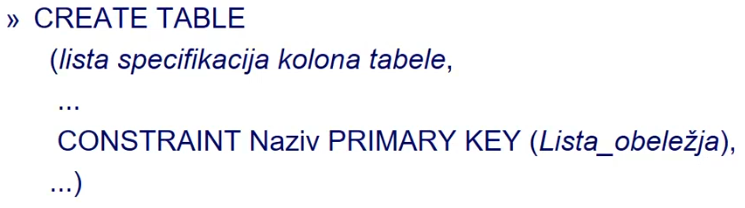
#### Primary Key



Zadaje se na *nivou obelezja* seme relacije koje jedino predstavlja *primarni kljuc*, bez navodjenja liste obelezja. Takodje se zadaje na *nivou celokupne seme relacije*(tabele), sa navodjenjem liste obelezja (uobicajeno i opstije resenje). Na *nivou obelezja(kolone)*:



Na *nivou seme relacije(tabele)*:



Znaci, prvo se nabroje sve *specifikacije svih kolona tabele*, a potom *specifikacije constraint klauzola*.

Podrazumeva se, bez posebnog deklarisanja, da je svako obelezje u *lista\_oblezja* deklarisano kao **not null** a proverava se prilikom svakog pokusaja upisa nove vrednosti obelezja kljuca (*insert)* ili modifikacije postojece vrednosti obelezja kljuca(*update)*. U slucaju pokusaja narusavanja ogranicenja, jedina moguca aktivnost je sprecavanje operacije (**no action**).

Samo jedan kljuc je moguce ovako definisati, sve ostale moramo na neki od sledecih mehanizma da definisemo.

Pokretanje ove klauzule automatski izaziva *kreiranje “unique” indeksa* (B+ stabla) nad *lista\_obelezja*.

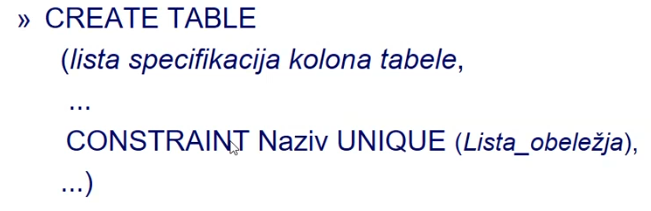
#### Unique



Zadaje se na *nivou obelezja* seme relacije koje jedino zadovoljava *ogranicenje jedinstvenosti* i to bez navodjenja liste obelezja. A takodje se zadaje i *na nivou celokupne seme relacije*(tabele) sa navodjenjem liste obelezja koja zadovoljava svojstvo jedinstvenosti (uobicajeno i opstije resenje). Zadavanje na *nivou obelezja:*



Zadavanje na *nivou celokupne seme relacije*:



Obelezja u *lista\_obelezja* mogu biti deklarisana kao **not null** a ne moraju, a kada se deklarisu, moraju se eksplicitno deklarisati. Provera se vrsi prilikom svakog pokusaja upisa nove vrednosti obelezja iz liste (*inserta)* ili modifikacije postojece vrednosti obelezja iz liste (*update*). U slucaju pokusaja narusavanja ogranicenja, jedina moguca aktivnost je sprecavanje operacije (**no action**).

Unique constraint nam takoreci sluzi za definisanje **ekvivalentnih kljuceva** a takodje cemo deklarisati I svaki zaisti unique constrain koji imamo u projektu seme baze podataka. Posto nemamo nista od mehanizama za ostale kljuceve a **primary key** mozemo iskljucivo za primarni kljuc, ostaje nam **unique** za ostale kljuceve.

Redosled obelezja u *lista\_obelezja* je bitan. Redosled obelezja u **primary key** klauzoli je vazan jer se nad primary key-em kreira B+ stablo. Mi obezbedjujemo redosled elemenata u svakom cvoru B stabla koji je rastuci ili opadajuci, u ovom slucaju defaultno rastuci, prvo po prvoj koloni, pa zatim po drugoj koloni, pa zatim po trecoj itd. Znaci, redosled je vrlo vazan! Jer ce to biti redosled kako ce se uredjivati sadrzaj svakog cvora, odnosno kako ce elementi biti poredjani u svakom cvoru. Znaci ako imam obelezja (A, B, C) I imamo upit tipa pretrazivanja nad (A, B) ili nad (A), znaci nad levim podnizovima ovog naseg niza (A, B, C) onda treba tako organizovati redosled da sto vise upita koji obuhvataju leve podnizove moze biti realizovan putem B+ stabla, znaci, ako imam upit koji ukljucuje B i C on nece moci najverovatnije da bude realizovan putem ovog B+ stabla (A, B, C) a ako imamo upit nad A,B ili nad A on hoce, a ako imamo ceste upite nad B, C onda bi trebalo stablo organizovati kao (B, C, A). Ukratko, *prvo stavljamo obelezja po kojima ce biti* ***frekventnije pretrage*** ili pak *zahtevniji upiti*. Ideja je da probamo da pokrijemo sto vise upita jednim indeksom, odnosno da smanjimo sto vise broj indeksa a da sa njima pokrijemo sto vise upita, stoga, redosled obelezja je od jako velikog znacaja

[Podsetnik] ***Trazenje*** *je algoritam koji iskljucivo na ulazu prihvata argument koji predstavlja vrednost iz domena kljuca. Rezultat trazenja je pre svega indikator uspesnosti (true,false). Odnosno, ima li torke sa vrednoscu kljuca ili nema, pa zatim mozda sadrzaj tog sloga a na kraju mozda i adresa na kom se trazenje zaustavilo.*

[Podsetnik] ***Pretrazivanje*** *je nalazenje torke koja zadovoljava odredjeni logicki uslov (koji nije povezan sa vrednostima kljuca).*

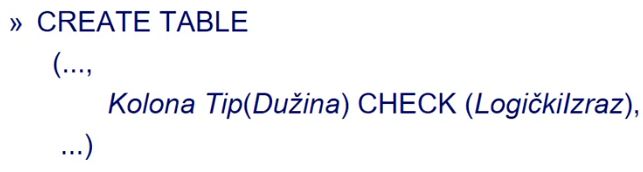
#### Check



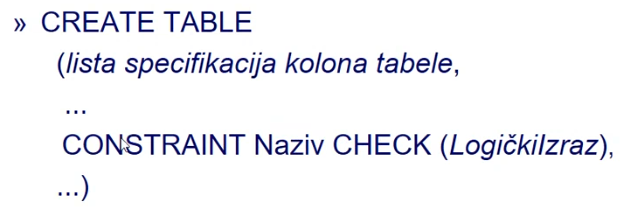
Zadaje se na nivou *obelezja* seme relacije koje je jedino upotrebljeno u *LogickomIzrazu*, takodje se zadaje i na nivou *celokupne seme* relacije(tabele) i to obavezno, kada *logicki izraz* obuhvata vise od jednog obelezja seme relacije (uobicajeno i opstije resennje).

Ovo proverava **ogranicenje torke** a moze i da sluzi za **ogranicenja domena** (kada nema **create domain** mehanizma, a prakticno ga cesto nema).

Zadavanje na nivou obelezja:



Zadavanje na *nivou seme relacije*:



1.50:40

#### Foreign Key