2023.

Baze podataka - beleške

NOVI SAD

JELENA ADAMOVIĆ

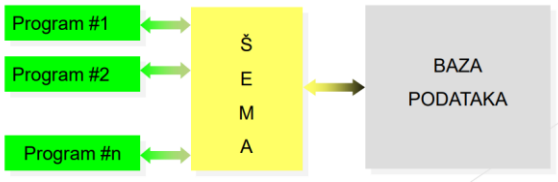
Sadržaj

Uvod

- **Cilj** je da definišemo željena stanja, da specificiramo šta je to što želimo da postignemo. Zatim treba da vidimo način na koji ćemo doći do tog cilja.
- **Resursi** su: novac, razna oprema, mašine, ljudski resursi, energenti, vreme... Obično kada računamo koliko košta neki projekat, računaćemo koliko časova smo proveli radeći na njemu. Još jedan bitan resurs su i **podaci**, što je ono što nas zanima na ovom predmetu.
- Za jedan realan sistem, podaci su jako bitni. Neki realni sistemi ne mogu čak ni da funkcionišu ukoliko im se nešto desi sa podacima. Npr. u banci ne možemo završiti neki posao, pa makar to bila i najobičnija uplata ili prijavu ispita u studentskoj službi, ako ne bismo imali pristup podacima. **U podacima nam je zapravo vrednost nekog sistema.**
- Podaci kao resurs su jako bitni, a sa druge strane, imamo i dosta podataka koji su van sistema, a koji mogu da nam budu jako bitni i čijom analizom bismo dobili možda mnogo potrebnih informacija.
- **Podatak i informacija su dve različite stvari!**
- Na osnovu podataka bismo dobili neke informacije, na osnovu informacija mi dobijamo neko znanje, a to znanje bi nam koristilo pre svega da postignemo one ciljeve koje smo na početku postavili.
- **Informacija** je **koristan podatak**. Sve su podaci, naravno ako imaju kontekst. Vrednost sa kontekstom je podatak, ali nije svaki podatak informacija. Informacija jeste podatak i informacije su nešto korisno, to je neko znanje koje nam je bitno iz nekog razloga, u ovom slučaju za ostvarenje ciljeva koje smo naveli.
- **Procesi** su neke svrsishodne delatnosti nad resursima koje vršimo da bismo postigli ove postavljene ciljeve**.**
- Struktura nam ovde nije bitna, a ni procesi nisu preterano bitni.
- **Okruženje** našeg sistema je nešto što utiče na naš sistem.
- Svaki program koji napišemo je apstrakcija nekog procesa koju pravimo pomoću nekog programskog okruženja.
- Nas interesuje apstrakcija putem matematičkih struktura, tj. interesuje nas apstrakcija na nivou matematičke rigoroznosti jer je mnogo lakše raditi kada imamo neku formulu.
- Mi želimo da izmodelujemo realan sistem jer nam za sve treba softver koji zapravo modelira procese koji se dešavaju u realnom sistemu i mi taj model zovemo **informacioni sistem**.
- Informacioni sistem nije isključivo softver, ali informacioni sistem jeste dominantno softver i bavimo se njime kao takvim.
- Dve bitne faze u razvoju informacionog sistema:
    - **Projektovanje (modeliranje)** – prva stvar koju treba da uradimo jeste da specificiramo cilj, odnosno domen koji želimo da radimo i šta je cilj koji želimo da postignemo.
    - **Izgradnja modela** – ne celog informacionog sistema, nego dela informacionog sistema.
    - **Realizacija** – ne treba da pravimo celu aplikaciju koja će da radi, nego implementaciju baze podataka i primenu nad tom bazom podataka.
- Ono što je važno da zapamtimo je da je organizacija datoteka i dalje aktuelna, samo je mi ne vidimo. Sistem datoteka je u jednom trenutku bio revolucija, iako je imao mane – redundantnost, nepovezanost aplikacija i čvrsta povezanost programa i podataka (tj. čvrsta povezanost sa načinom smeštanja datoteka na nekom medijumu).

- Sistem za upravljanje bazama podataka (SUBP), odnosno pojavom relacionih baza podataka prevazišli smo te probleme i dobili smo nešto novo, što je prilično olakšalo rad, da možemo podatke da gledamo na nekom višem apstraktnom nivou i da nas više ne zanima to kako idu oni pokazivači. SUBP može biti i prepreka, sada niko ne može ni da priđe podacima, a da se prethodno ne obrati SUBP-u za sve (da li hoće da kreira, da li hoće da unosi, da li hoće nešto da menja, da čita, višekorisnički režim rada, bezbednost, bekapi i ostalo je sve SUBP).
- Koliko je „moćan" SUBP – toliko su podaci zaštićeni. To uglavnom zavisi od toga koliko je plaćeno za SUBP, mada postoje i open source SUBP-ovi koji nisu toliko loši (npr. Postgre, MySQL, Oracle...).
- Oracle i Microsoft SQL Server rade sa standardnim SQL-om.
- Neki upit koji je napisan na standardnom SQL-u bi trebao da radi i na Oracle, i na Postgre itd. Mada postoje neke minorne razlike.
- PL/SQL je Oracle-ovo proceduralno proširenje standardnog SQL-a.
- **LSO** je **logička struktura obeležja**.
- **LSP** je **logička struktura podataka. LSP** nije kontekst. LSO daje kontekst podacima. Imamo podatke koji su povezani na neki način, a LSO odnosno šema daje kontekst tim podacima.
- Sa jedne strane imamo programe. Programi više ne vide da li ima pokazivača, da li ima transformacije, ne zna da li ima zone prekoračenja, da li je sve serijsko ili sekvencijalno, da li je spregnuta, da li ima indekse itd. Šema je logička struktura.
- U relacionim bazama podataka mi vidimo podatke kao skup tabela,li oni naravno nisu tako smešteni na uređaju, mi ih samo vidimo kao tabele, a to nam obezbeđuje SUBP.
- Preslikavanje LSP u fizičku strukturu podataka radi SUBP.

- Šema baze podataka – koji su to podaci koji su potrebni u sistemu koji želimo da napravimo i koje treba da mu pružimo. Ako šemu podataka uradimo dobro, softverska nadgradnja će biti vrlo laka.



- SUBP je aplikacija za rad sa bazom podataka (npr. SQL Developer) i tu radimo direktan pristup bazi podataka.

- U svakoj tabeli koju pravimo imamo neka obeležja, tipove podataka i to su tzv. Metapodaci. SUBP ima svoju bazu podataka u kojoj je smešteno sve što smo definisali.
- **Rečnik baze podataka** je baza podataka (ili parče baze podataka) SUBP-a gde on smešta sve što rade korisnici SUBP-a. Sve što mi kreiramo (a kreiramo u okviru šeme) stoji tu; svi atributi, tipovi podataka, ograničenja, serverske procedure, trigeri, serverske funkcije, paketi (oracle ima i pakete) i taj deo ne bismo trebali da diramo.
- **Bez šeme relacione baze podataka ne može ni šestica!**
- **Ključ** je minimalni skup obeležja (često i samo jedno obeležje) koja jedinstveno identifikuju svaku torku. Torka je ovde jedan red u tabeli.
- Podaci se povezuju tako što imamo propagaciju ključa iz jedne tabele u drugu. Taj propagirani ključ je strani ključ.
- Dva načina povezivanja preko propagacije ključa:
    - Propagacija ključa iz jedne u drugu tabelu
    - Propagacija ključa u jednu posebnu tabelu iz druge dve koje ta tabela povezuje

- Npr. ako imamo primer sa poslovnicom i direktorima. Ako u realnom svetu (realnom sistemu) važi da jedna poslovnica ima samo jednog direktora, onda možemo koristiti prvi način. Ali, ako poslovnica ima više direktora (npr. finansijski, generalni...) koji su predstavljeni ključem stranim MBR, onda prvi način više ne važi i tada moramo uvesti posebnu tabelu koja će povezivati ove dve tabele – direktora i poslovnicu i gde bismo imali kombinaciju radnik (direktor) i poslovnica, a u svakoj od tih poslovnica imamo nekog direktora, koliko god ih bilo. Ako u nekoj poslovnici imamo 5 direktora, biće 5 torki sa tom poslovnicom i različitim direktorima (radnicima). Te tabele služe kao vezne tabele i predstavljaju kombinacije ključeva, nije bitno da li ključ ima jedno ili više obeležja. **Vrednosti stranog ključa mogu biti samo iz skupa vrednosti primarnog ključa u tabeli odakle je taj ključ došao**.
- Ograničenje referencijalnog integriteta čuva podatke da budu u ispravnom stanju, da se ne bi desilo da unosimo podatak koji ne postoji.
- Kod tabela koje povezuju dve tabele strani ključevi ulaze u primarni ključ, to je specifičnost samo za ovakvu vrstu tabela.
- Strani ključ po difoltu može da ima null vrednost i nije unique. Strani ključ može biti unique ako realan sistem to zahteva i može biti da ne sme da ima null vrednost ako to realni sistem zahteva.
- Kada je strani ključ deo primarnog ključa, on ne sme da ima null vrednost zbog toga.
- U tabeli koja povezuje dve tabele ne može se pojaviti torka koja nema torku za spajanje sa bilo kojom od ovih tabela koje povezuje, jer je ovde svaka vrednost iz skupa vrednosti povezanih tabela.
- **Ograničenje ključa i ograničenje referencijalnog integriteta – pitanje za šesticu.**
- Relacione baze podataka su poprilično stare, ali se i dalje koriste. Jedno vreme se mislilo da će ih zameniti objektne baze podataka, ali to nije zaživelo. Ali, uvedena su neka proširenja u relacionim bazama kako bi mogle da podrže neke specifičnosti sistema (npr. za geoinformacione sisteme).
- Realni sistemi, organizacioni sistemi tipa banaka i svih ovih poslovnih i dalje su na relacionim bazama podataka i poslovanje im je na relacionim bazama podataka.
- Sada imamo društvene mreže, IoT uređaje, pametne fabrike gde se stvara velika količina podataka.
- Sa pojavom multimedijalnih podataka, pojavio se problem jer relacione baze podataka 70-ih godina nisu zamišljene za te namene i onda se tražio način kako da se izađe na kraj sa brzinom, a da opet podržimo sve te sadržaje koji se pojavljuju, pa smo došli do NoSQL baza podataka, gde spadaju sve baze podataka koje nisu relacione (ima ih više vrsta), mada ni one nisu uspele da zamene relacione baze podataka.
- Došlo se do toga da relacione baze imaju svoje mesto, a NoSQL baze svoje mesto, u zavisnosti od tipa sadržaja koji imamo biramo i vrstu NoSQL baze podataka. Za sada sve ima svoje mesto i međusobno se nadopunjuju.
- Prvi problem koji se javio kod relacionih baza je brzina, a da imamo ogromnu količinu podataka (brzina odziva). Da bi se rešila brzina, prvo smo se oslobađali ograničenja o kojima smo pričali, međutim sada se kod NoSQL baza uvode ograničenja jer tu nemamo konzistentnost podataka, pa ćemo videti šta će biti kada se tu uvedu ograničenja, da li će to možda da uspori brzinu.
- **Strukturirani podaci** - Imamo šemu koja definiše format podataka i ta šema mora da se strogo ispoštuje, u suprotnom imamo grešku – ne može bez šeme.

- **Polustrukturirani podaci** – kao što su JSON-i, YAML-ovi, HTML-ovi u vidu grafova ili stabla gde takođe imamo šemu. Imamo šemu, ali ona nije baš toliko striktna, imamo i delove koji mogu da se izostave.
- Krenulo se i sa tim šema-less, međutim to nije baš lako.
- **Nestrukturirani podaci** – može da bude bilo šta; mejl, dokument, neki multimedijalni sadržaj... naravno, sa stanovišta smeštanja podataka. Nema nekog strogog formata koji mora da se poštuje, ovo je već više za neku NoSQL bazu podataka.

Model podataka

- **Model podataka (MP)** je matematička apstrakcija (više apstrakcija na nivou matematičke rigoroznosti, nego da je sve matematička apstrakcija).
- **Ovde se na model podataka odnosi da je to zapravo jezik za modelovanje.**
- Zaostavština relacionih baza podataka je da mi u stvari model zovemo šemom baze podataka. Šema baze podataka je isto što i model podataka, ali **ne ovaj model podataka – ovo je apstrakcija putem koje se pravi šema baze podataka, odnosno jezik za modelovanje**.
- Treba da predstavi šemu, ograničenja koja se odnose na same podatke (odnosno vrednosti podataka) i ograničenja u odnosima između podataka i treba da predstavi dinamiku izmene stanja. Ono što se promeni u realnom sistemu, naravno, treba da se promeni i u samim podacima koji se čuvaju, tako da model treba da obezbedi opracije na koji način će doći do te izmene da bismo pratili dinamiku izmene stanja.
- **Trojka modela podataka: *(S, I, O)*.**
- Komponente modela podataka:
    - ***S* – strukturalna komponenta**
        - Modeluje statički deo sistema, statičku strukturu, šemu baze podataka
    - ***I* – integritetna komponenta**
        - Modeluje ona ograničenja
    - ***O* – operacijska komponenta**
        - Modeluje dinamiku izmene stanja, pri čemu mogu da se menjaju podaci, ali može da se menja i sama šema
- Nivoi apstrakcije:
    - Apstrakcija nam je jako bitna i već ovde, u bazama podataka, imamo dva nivoa apstrakcije minimum, pri čemu ti nivoi apstrakcije zavise od samog modela podataka, zavisi šta držimo za određeni nivo apstrakcije, onda nam je sledeći nivo apstrakcije možda nešto drugo, u sledećem primeru ono što je bilo na višem nivou apstrakcije sada može biti na nižem nivou.
    - **Nivo intenzije** – nivo intenzije je nivo tipa, nivo opisa, nivo LSO-a
    - **Nivo ekstenzije** – nivo ekstenzije je nivo konkretizacije, nivo podataka, nivo pojave tipa, konkretizacija = LSP
    - **Prime 1**: imamo studenta kao tip entiteta, student ima broj indeksa, ime, prezime, imejl adresu, adresu... To su neke informacije o studentu koje su bitne i relevantne za fakultet. Nivo ekstenzije je neki konkretan student, a intenzija je student (tip entiteta STUDENT).
    - **Primer 2**: sad nam je nivo ekstenzije student, znači na nivou konkretizacije imamo tip entiteta student, a sledeći nivo apstrakcije (viši nivo) je uopšteno tip entiteta koji se

definiše tako što ima skup obeležja, skup ograničenja i naziv, to je definicija tog tipa entiteta. Konkretizacija toga je tip entiteta *Student*, tip entiteta *Nastavnik*, TE *Predmet*, TE *Radnik*... Onda možemo da se vratimo na još jedan nivo apstrakcije ispod, pa da kažemo pojava tipa entiteta student *Marija*, pojava TE *Nastavnik* – *Slavica*... Ako je pojava TE *Marija*, ali u smislu modela, imamo sledeći nivo apstrakcije *Marija koja sedi ovde* i dalje nema, ali imamo jedan nivo apstrakcije više – kako definišemo tip entiteta, a to je neki jezik koji je samodefinišući i može pomoću njega sam za svoje potrebe da definiše, ali i koncepte nekih jezika. Zapravo, imamo meta object facility i on ima koncepte pomoću kojih se definišu koncepti jezika za nešto.

- **Model podataka** je grafički ili tekstualni jezik kojim ćemo napraviti šemu naze podataka.
- **Strukturalna komponenta** služi za modelovanje statičke strukture sistema (šemu baze podataka), pri čemu ovo statička ne znači da je doslovno nepromenljiva, ali u odnosu na praćenje dinamike izmene stanja i modeliranje izmene dinamike stanja, ovo je statička struktura. Dinamiku izmene stanja modelujemo operacijskom komponentom. Integritetna komponenta je komponenta kojom treba da modelujemo ograničenja na podatke i ograničenja koja postoje u odnosima između podataka.
- Krajnji korisnik (program) vidi samo šemu baze podataka, on ne vidi kako zaista kako su podaci fizički smešteni. Sad mi treba nekako da predstavimo tu šemu tako da bude razumljiva krajnjem korisniku (programu, a programe pišemo mi).
- **Koncept** je apstraktna (i formalna) predstava jedne klase pojmova kojima se modeluju delovi realnog sveta. Npr. osnovni koncept u objektno orijentisanim jezicima je klasa. Znamo kada treba da upotrebimo klasu tako što znamo da ona treba da ima ime, atribute, da atributi imaju tipove podataka, da eventualno ima metode... Jednom klasom opisujemo skup objekata koji imaju zajedničke osobine. Tip entiteta je skup svih entiteta, tj. isto to što smo rekli za klasu, samo što nema metoda.
- Kad mapiramo iz relacionog u objektni, tip entiteta ćemo mapirati na klasu.
- Ono što nama treba i u ovom modelu podataka jesu koncepti pomoću kojih ćemo mi što vernije da preslikamo ono što je u realnom sistemu, jer nam je poenta da taj model koji pravimo bude što vernija slika realnog sistema jer mi hoćemo zapravo, uslovno rečeno, da simuliramo realni sistem.
- Radićemo ER model i relacioni model podataka, tu takođe imamo koncepte koji opisuju iste pojmove iz realnog sistema, ali imamo semantički bogatije i one koji su manje semantički bogati, a to može da se objasni kao da imamo azbuku sa 30 slova kojom možemo nešto da opišemo i onda npr. u sledećem slučaju nemamo 30, već 15 slova i treba da uradimo isto.
- Relacioni model je implementacioni, a ER model nije.
- **Primitivni koncept** se čak ni ne definiše, uvodi se tako, objašnjava, ima svoju semantiku, savladamo za šta i na koji način se koristi i nemamo definiciju. U zavisnosti od modela podataka imamo različite koncepte koje taj model podataka nudi (slično kao i kod programskih jezika).
- **Strukturalna komponenta** je ta komponenta koja ima taj neki skup pre svega primitivnih koncepata, pa onda bi trebalo da ima i pravila kako od primenjenih koncepata ili prethodno formiranih složenih koncepata da se formiraju izvedeni složeni koncepti.
- Neki semantički bogat model podataka može da ima i skup složenih koncepata koji su nam već predefinisani i dati da ne moramo da ih kreiramo. Tip entiteta i tip poveznika nisu primitivni

koncepti, to su već izvedeni koncepti, odnosno oni su već takvi dati na korišćenje, spadaju u grupu ovih složenih koncepata.

- Za svaki od tih koncepata, bilo da su primitivni ili složeni, pre svega treba da znamo osobine svakog od tih koncepata i kada treba da ih upotrebimo kako bismo znali kako da ih koristimo i imamo pravila za korišćenje za svaki koncept. Svaki koncept ima svoju semantiku.
- Kada naučimo koncepte koje neki model podataka nudi, tada možemo da se bavimo projektovanjem baza podataka.
- Integritetna komponenta služi za modelovanje ograničenja. Imamo ograničenja na podatke i ograničenja u odnosima između podataka-
- Strukturalnu komponentu čini skup koncepata, a kod integritetne komponente imamo skup tipova ograničenja. Svako ograničenje je nekog tipa. Integritetna komponenta definiše tipove ograničenja koje taj model podataka može da prosledi.
- Svaki od tih tipova ograničenja ima neki skup osobina, ima skup pravila kako da koristimo taj tip ograničenja, kao i opisanu smenatiku.
- Kod skupa osobina imamo dodatak kako da specificiramo odgovarajući tip ograničenja i uglavnom nam konkretan jezik nudi sintaksu kako da specificiramo dato ograničenje i kako da ga validiramo, odnosno kako da ga interpretiramo.
- Validacija ograničenja – provera da li naša implementacija zadovoljava ograničenje.
- Ograničenja nam omogućavaju da do greške zapravo ne dođe, odnosno da ne dođe do narušavanja konzistentnosti baze podataka.
- Imamo i pravila za izvođenje zaključaka o važenju ograničenja, kao i pravila za kreiranje novih tipova. U zavisnosti od modela podataka, neki to nude, neki ne.
- **Ključ** je minimalni skup obeležja koji jednoznačno određuje (identifikuje) neku instancu/pojavu.
- **Ograničenje ključa** – ne smeju postojati dve instance sa istom vrednošću ključa (ograničenje na vrednost podataka). Ograničenje u odnosima bi bilo npr. da jedan student može imati samo jednu zaključnu ocenu iz nekog predmeta.
- Sva ograničenja koja smo napravili, SUBP je upisao u neku svoju bazu podataka (rečnik baze podataka) i kada god hoćemo daa uradimo nešto nad podacima iz ovih tabela, SUBP će reagovati i proveriti koja sve ograničenja postoje nad tom tabelom, ako nađe da postoji neko ograničenje, proveriće da li je svako od tih ograničenja zadovoljeno, ako neko nije, dobićemo grešku, a ako je ispunjeno, nećemo primetiti da je SUBP išta radio. Ovo nam je zapravo **validacija**.
- SUBP vrši te validacije. Ta ograničenja koja se postavljaju na vrednosti podataka bi trebala da se implementiraju tamo gde su podaci, odnosno na nivou šeme baze podataka, jer ćemo onda imati situaciju da SUBP zapravo automatski proverava ograničenja. Mi ćemo ih projektovati i poslaćemo i onaj krajnji korisnik apsolutno neće biti svestan toga, a biće zaštićen, sve dok radi kako treba, on se neće ni setiti tih ograničenja sve dok ne uradi nešto što ne treba.
- Postoji veliki broj ograničenja koja ne mogu na takav način da se implementiraju, to su tzv. **Deklarativna ograničenja**, ona su prilično složenijeg oblika, te onda moramo da programiramo.
- Da li ćemo da programiramo opet na serveru, pa da imamo serverske procedure, trigere koji su vezani za tabele i koji će da reaguju kada se nešto desi ili ćemo to da uradimo na bekendu, na srednjem sloju, na aplikativnom nivou ili čak na nivou korisničkog interfejsa. Ako znamo da nešto ne sme da bude null (npr. neka polja moraju biti popunjena, to možemo proveriti već na nivou frontenda, bez obraćanja bazi podataka). Obraćanje bazi košta, ta baza je uglavnom dislocirana, treba nam SUBP i to troši neko vreme, ako znamo da neko polje mora biti popunjeno, onda

ćemo automaski tu validaciju da uradimo na frontendu. Bez obzira na to što na frontendu kontrolišemo da npr. nešto ne sme da bude null, ne smemo ni slučajno da izostavimo da se ta provera vrši i na nivou baze podataka, jer će se desiti da se pojavi druga aplikacija gde će se ad hoc pristupati bazi podataka, iako nema te provere, onda smo bazu podataka ostavili u vrlo problematičnoj poziciji da eventualno u nekom momentu dođe do nekonzistentnosti podataka.

- **Operacijska komponenta** modelira dinamiku izmene stanja, ona poseban značaj ima u sistemima gde se radi sa velikim količinama podataka (banke, kompanije za telekomunikacije...). Sve što se promeni u realnom sistemu, treba da se reflektuje promenama u bazi podataka. Promena ne mora da se odnosi samo na podatke u bazi, već može da se desi da imamo promenu i u šemi baze podataka. Mada je u odnosu na promenu u podacima to više statički i odnosi se na proširivanje sistema najčešće.
- **Operacijsku komponentu čini skup tipova operacija**.
- Upitni jezik (Query Language – QL) služi za iskazivanje upita (selekcije podataka) nad bazom podataka
- Jezik za manipulisanje podacima (Data Manipulation Langauge – DML) – tipovi operacija za izmenu stanja baze podataka (ažuriranje) u cilju praćenja izmena stanja podataka u realnom sistemu.
- Jezik za definisanje podataka (Data Definition Language – DDL) – tipovi operacija za kreiranje i modifikaciju specifikacija šeme baze podataka, fizičke strukture baze podataka, prava pristupa i zaštite baze podataka, novih tipova operacija (programa) za upravljanje podacima.
- SQL jezik nam čini operacijsku komponentu relacionog modela.
- Specifikacija operacije sadrži komponente:
    - **Aktivnost** – specifikacija akcije nad podacima u bazi podataka.
    - **Selekcija** – specifikacija dela baze podataka (u DML i QL) ili dela šeme baze podataka (u DDL) nad kojim se sprovodi specificirana aktivnost.
- Operacijska komponenta može biti:
    - **Proceduralna (navigaciona)**
        - Selekcija vrši izbor jednog objekta iz baze podataka
        - Selekcija se vrši putem indikatora aktuelnosti, ili putem odnosa između podataka
        - Proceduralnost sa programskim petljama i uslovnim grananjima
        - Definiše se ŠTA i KAKO
    - **Specifikaciona (deklarativna)**
        - Selekcija vrši izbor skupa objekata iz baze podataka
        - Selekcija se vrši na osnovu vrednosti obeležja
        - Neproceduralnost
        - Definiše se samo ŠTA
- **Modeli podataka**:
    - Model tipova entiteta i poveznika (ER) – jako star, ali i dalje je intuitivan i koristi se
    - Mrežni model
    - Hijerarhiski model – mrežni i hijerarhiski modeli su stariji od ER-a i više se toliko ne koriste, ali neke njihove komponente su ugrađene u ove novije
    - Relcioni model
    - Logički i verovatnostni (fuzzy) logički modeli – NoSQL model

- Objektno orijentisani model
- Objektno relacioni model – mislili da će zameniti relacioni, ali nije zaživeo. SUBP Oracle je objektno relacioni, ali mi koristimo samo relacioni deo.
- XML model

ER model

- Hoćemo da predstavimo realan sistem pomoću ovog modela podataka. Posmatrajući realan sistem, uočavamo **entitete**. Više entiteta čini **klasu**.
- Ovaj model je i dalje nezaobilazan zato što ima **dijagramsku reprezentaciju**. Mi možemo koristiti UML dijagrame klasa i iz njih takođe dobiti relacioni model. Dijagram klasa je vrlo sličan ovom modelu, mada ima neke razlike. Ovaj model je orijentisan samo na podatke.
- Klasa realnih entiteta.
- Osobina – misli se na realni sistem, atribut (obeležje) – u modelu podataka
- Obeležje može biti:
  - **Elementarno** – ne dekomponuje se, reprezentuje atomičnu (elementarnu vrednost). Primer: *Grad, Ulica, Broj, Stan...*
  - **Složeno** – može se dekomponovati na druga obeležja, reprezentuje složenu vrednost. Primer: *ADRESA = (Grad, Ulica, Broj, Stan)*
  - **Skupovno** – reprezentuje skup vrednosti istog tipa
- Strukturalnu komponentu čine koncepti. To su oni koncepti koji nam trebaju da opisujemo realni sistem.
- Primitivni koncepti strukturalne komponente ER modela podataka:
  - Vrednost – bilo koja konstanta
  - (predefinisani) domen – nije samo skup mogućih vrednosti, nego specifikacija mogućeg skupa vrednosti. Mi možemo zadati domen samo kao skup mogućih vrednosti, npr. {5, 6, 7, 8, 9, 10}, a možemo i kao specifikaciju tako što ćemo reći da je ceo broj veći ili jednak od 5, a manji ili jednak od 10 (ili veći od 4, a manji od 11). Domen može biti **predefinisan** i **korisnički definisan**. Predefinisani su oni ugrađeni tipovi koje imamo dostupne na raspolaganju u alatu koji nam služi za modelovanje. Različiti alati mogu imati dostupne različite tipove podataka. Dobro je da „strpamo“ model i zadamo tipove podataka i ako znamo već ciljnu platformu, da koristimo tipove podataka koje ta ciljna platforma podržava. Ciljna platforma je konkretan SUBP koji ćemo koristiti, npr. ako je to Oracle, kod njega nemamo string, imamo *varchar* i *varchar2*, ali možemo da napišemo string pomoću ove dve vrste. Kod Oracle-a imamo *number*, a možemo da koristimo integer, decimal, on konvertuje u taj svoj number. Korisnički izveden (korisnički definisan) – on je samo na nivou teorije, u praksi nemamo da nasleđujemo već prethodno kreirani domen. Eventualno imamo da definišemo korisnički definisane tipove podataka na osnovu postojećih tipova podataka. To je jedan nivo, a inače mi praktično nemamo pravo nasleđivanje.
  - Obeležje
- **Vrednost sa kontekstom je podatak**. Npr. ako kažemo 10, mi ne znamo šta je to – ocena, datum, broj nečega... sve dok nemamo kontekst, mi ne možemo biti sigurni šta je to.
- **Podatak** je uređena četvorka oblika ***(Entitet, Obeležje, Vreme, Vrednost)*.**
  - *Entitet* – identifikator (oznaka) entiteta

    - *Obeležje* – oznaka (mnemonik) obeležja
    - *Vreme* – vremenska odrednica – obično je izostavimo, uglavnom za nju imamo poseban atribut npr. ako hoćemo da kažemo kada je student položio ispit – tog i tog datuma, to je već drugo obeležje – datum, a ako izostavimo vremensku odrednicu (ako je nemamo eksplicitno zadatu), podrazumeva se da podatak važi u trenutku manipulisanja.
    - *Vrednost* – jedna vrednost iz *dom(A)*
    - . Naravno, moramo da znamo kontekst. Npr. kontekst je da je 10 ocena studenta, idemo jedan nivo apstrakcije više, nismo kod neke instance. A ako pričamo o konkretnoj vrednosti 10, onda je to ocena neke instance studenta.
- Naravno, moramo da znamo kontekst. Npr. kontekst je da je 10 ocena studenta, idemo jedan nivo apstrakcije više, nismo kod neke instance. A ako pričamo o konkretnoj vrednosti 10, onda je to ocena neke instance studenta.
- Informacija je neki koristan podatak, ali nisu svima isti podaci korisni.
- **Tip entiteta (TE)** je model klase realnih entiteta u informacionom sistemu. Nastaje od obeležja klase realnih entiteta, bitnih za realizaciju ciljeva informacionog sistema. Predstavlja uređenu strukturu: ***N(Q, C)***.
    - N – naziv tipa entiteta
    - $Q = \{A_1, ..., A_n\}$ – skup obeležja tipa entiteta
    - C – skup oraničenja tipa entiteta
    - $K = \{K_1, ..., K_m\} \subseteq C$ – skup ključeva tipa entiteta ($K \neq \emptyset$)
- Ovo nije primitivni koncept ER modela, ali jeste osnovni. Šema će nam se sastojati od najvećeg broja tipova entiteta i tipova poveznika i tu imamo naziv tipa entiteta, skup obelžja koji čine taj TE. Zapravo, TE je određen skupom obeležja koji sadrži. Eventualno, sadrži i skup ograničenja. Taj skup ograničenja nikada nije prazan skup, imamo makar ključeve (ograničenje ključa), a možemo imati i druge.
- Tip entiteta je naziv koncepta kojim ćemo da modelujemo klase realnih sistema. Ono što uočimo u realnom sistemu kao klasu realnih entiteta, u ovom jeziku (modelu podataka) ćemo modelovati pomoću koncepta koji se zove tip entiteta (entity type), ponekad samo entity. Mi ovde entity-jem smatramo instancu TE, a TE je viši nivo apstakcije, predstavlja osobine te instance. Ispravno je i jedno i drugo, samo se moramo držati jedne notacije.
- Pojava tipa entiteta modeluje jednu instancu entiteta. Moramo razlikovati konkretizaciju i opis studenta. Konkretizacija je samo one osobine koje su bitne za neki sistem.
- Pojava TE je skup podataka zato što svakom obeležju dodelimo konkretnu vrednost, pri čemu ta vrednost koja se dodeljuje atributu mora biti iz skupa, odnosno mora zadovoljavati ono što domen nalaže šta može da se pridruži tom obeležju.
- Svakom obeležju moramo da pridružimo domen.
- Ovde imamo samo interni identifikator TE.
- **DEFINICIJA KLJUČA I OGRANIČENJE KLJUČA – OBAVEZNO!!!**
- **Ključ** kod ER modela je **minimalni skup obeležja** koji jednoznačno identifikuje svaku pojavu TE (defincija ključa se prilagođava po modelu). Ne postoje dve pojave sa istom vrednošću ključa, iz ovoga sledi da vrednost ključa ne sme biti null, tj. ne sme biti nedostajuća, mora biti zadata inače narušavamo ovu osobinu jedinstvenosti i osobinu minimalnsoti.
- **Minimalnost** ne znači da je to samo jedno obeležje, već da je to neki skup obeležja bez kojeg ne može da dođe do toga da imamo jedinstvenu identifikaciju.

- **Pojam ekvivalentnih ključeva** – npr. na fakultetu se identifikujemo pomoću broja indeksa, a imamo i JMBG. Broj indeksa i JMBG su ekvivalentni ključevi. Da bi neka dva atributa bili ekvivalentni ključevi, prvo moraju zadovoljvati sve osobine ključeva. Od svih tih ekvivalentnih biramo samo jedan primarni koji se podvlači. Broj obeležja u ključu može biti 1 ili više.
- **Tip poveznika** je drugi koncept ER modela podataka kojim ćemo da modelujemo veze između TE. Imamo neki niz povezanih tipova koje TP povezuje. Broj povezanih tipova koje TP može da povezuje je proizvoljan. Broj povezanih tipova koji ćemo mi koristiti je 2, odnosno koristićemo binarne tipove poveznika, oni se u praksi i najčešće koriste. Koriste se i trinarni, već veća arnost je komplikovanija. Ono što važi za 2 povezana tipa, važi i za N povezanih tipova.
- **Tip poveznika (TP)** je model veza između pojava povezanih TE ili TP. Uređena struktura: ***N(N_1, N_2, ..., N_m, Q, C)*.**
    - *N* – naziv tipa poveznika
    - N_i *(i* $\in \{1, \dots, m\}$) – povezani tip
        - Tip entiteta
        - Prethodno definisani tip entiteta
    - $Q = \{B_1, ..., B_m\}$ – skup obeležja TP
    - *C* – skup ograničenja TP
    - $K = \{K_1, ..., K_k\} \subseteq C$ – skup ključeva TP $(K \neq \emptyset)$
- Možemo imati TP bez obeležja i vrlo često ih imamo. *C* nikada nije prazan skup, ima barem ključ.
- TP povezuje TE ili prethodno definisane TP.
- Tipovi entiteta koje TP povezuje se mogu ponavljati, TP može da povezuje i dva ista TE, pa i dva ista TP.
- Ako TE jedinstveno identifikuje skup obeležja koji je podskup skupa obeležja TE, ta obeležja ključa su sadržana u obeležjima TE.
- Ključ tipa poveznika zavisi od ključeva povezanih tipova. Vrlo često, ključ TP je unija ključeva povezanih tipova i podskup od te unije. Ključ tipa poveznika ne čine obeležja iz skupa obeležja TP, ali on uvek ima ključ.
- ER model podataka je na konceptualnom nivou, tj. ne zavisi od konkretne implementacije. Kada opišemo šemu baze podataka pomoću ovog modela, posle možemo da je implementiramo na različitim SUBP-ovima, odnosno nema nikakvih podataka o samoj implementaciji.
- ER model je grafički jezik za modelovanje šeme baze podataka (ima dijagramsku reprezentaciju).
- Na ovom nivou bi trebao da učestvuje i krajnji korisnik i da razume ono što mi modelujemo. Ovi dijagrami jako lepo i razumljivo opisuju sistem, te u modelovanju može učestvovati i neko ko nije stručan u oblasti IT-ja.
- ER model ima svoje koncepte. Ako je jezik grafički, onda njegovi koncepti imaju dijagramsku reprezentaciju.
- CASE alati su softveri koji nam pomažu u raznim fazama razvoja softvera, pre svega u ovim prvim fazama. Imamo CASE alate koji služe za modelovanje šeme baze podataka pomoću ER modela. CASE alati su softveri, to su proizvodi koje imamo i open source, a imamo i komercijalne. U zavisnosti od CASE alata, imamo razlike u načinu predstavljanja ovih simbola.
- To kako CASE alat predstavlja neki koncept nama nije ni toliko bitno. Bitnije je da naučimo koncepte modele podataka i kako se predstavlja šema baze podataka.
- Zgodno je da imamo šemu u CASE alatu iz dva razloga:

    - Dokumentacija – kada već imamo dokumentovano opis i strukturu podataka čitavog sistema, kada dođe do reinženjeringa ili kada ide prebacivanje na novije tehnologije, mi već imamo opis sistema, samo treba da ga prebacimo na neku novu tehnologiju.
    - Kada na ovom nivou napravimo šemu, a CASE alat je okej, na jedan klik možemo dobiti implementacioni model.
- Kada kažemo da je neki model podataka konceptualni, to znači da nemamo SUBP koji je zasnovan da radi na osnovu njega, odnosno nemamo implementaciju. Taj model mora da se transformiše u implementacioni model (za ER model ovde je to relacioni model), pa zatim u šemu baze podataka opisanu pomoću SQL jezika, gde možemo te naredbe da izvršimo i zaista da imamo kreiranu implementiranu bazu podataka.
- Kada koristimo binarne poveznike, imamo vezu ka oba povezana tipa.
- Ako obeležje ulazi u ključ, biće podvučeno. Ako na dijagramu vidimo više podvučenih obeležja, to nije više ekvivalentnih ključeva, nego jedan koji ima više obeležja i podvlači se na dijagramu primarni ključ.
- Imamo dva nivoa detaljnosti prikaza:
    - Globalni nivo prikaza – kada izostavimo atribute
    - Detaljni nivo prikaza
- Rekurzivni tip poveznika koristimo kada treba da povežemo tipove entiteta iste klase. Moramo da znamo koncept da bismo ga primenili.
- Pri modelovanju, mi posmatramo realan sistem, uočavamo klase i biramo koncepte pomoću kojih ćemo ih predstaviti u šemi, tj. na dijagramu. Treba nam model koji će što realnije da preslika sistem.
- Ukoliko je model podataka semantički bogatiji (ima više koncepata), to ćemo lakše i vernije preslikati sistem na model. Što je semantika siromašnija, imamo manje koncepata i imamo teži zadatak da opišemo ono što vidimo.
- ER model je semantički bogat. Jedan od koncepata jeste i ovaj rekurzivni tip poveznika. Za rekurzivni tip poveznika važe ista pravila kao i za bilo koji binarni tip poveznika, samo što na dve različite strane nemamo dva različita povezana tipa, nego jedan isti.
- Ovo je bio deo strukturalne komponente. Ona sadrži skup primitivnih koncepata i eventualno izvedenih koncepata koji služe za modelovanje šeme baze podataka.
- Model podataka sadrži i integritetnu komponentu koja sadrži skup tipova ograničenja i služi za modelovanje ograničenja.
- Tipovi ograničenja u ER modelu podataka:
    - Ograničenje domena
    - Ograničenje vrednosti obeležja
    - Ograničenje pojava tipa
    - Kardinalitet tipa poveznika
    - Ograničenje ključa (integritet tipa):
        - Za tip entiteta i
        - Tip poveznika
- Ograničenje ključa za TE i TP se razlikuju jer se različito računaju, iako im je definicija ista.
- **Specifikacija domena:**
    - Struktura: *D(id(D), Predef)*
        - *D* – naziv domena

    - *Id(D)* – ograničenje domena
    - *Predef* – predefinisana vrednost metoda
- Predefinisana vrednost je ona vrednost koju pri specifikaciji domena možemo zadati. Domen se pridružuje atributima. Ukoliko se nekom atributu izostavi da se eksplicitno zada vrednost, atribut će dobiti vrednost koju smo stavili za predefinisanu. Ukoliko u specifikaciji domena nema predefinisane vrednosti (ona nije obavezna), onda će vrednost obeležja biti null ukoliko je eksplicitno ne zadamo.
- Za ograničenje domena moramo definisati tip, dužinu i uslov, pri čemu je jedino tip obavezna komponenta. Dužina i uslov nisu obavezni. Dužina ima smisla za neke tipove podataka, kao što je npr. niz karaktera.
- Uslov ukoliko postoji, možemo da ga definišemo, npr. uslov za ocenu studenta je da je u rasponu od 5 do 10, a za učenika u osnovnoj i srednjoj školi je od 1 do 5.
- CREATE DOMEN naredba ne postoji u većini SUBP-ova. Uslov iz ograničenja domena i ograničenje pojave tipa možemo postaviti korišćenjem constraint-a CHECK u Oracle-u. Ukoliko u uslovu kao operand imamo samo jedno obeležje, onda je to najčešće ograničenje domena. Delte označavaju da nema vrednosti.
- Ako je nešto null, to je nedostajuća vrednost. Za proveravanje da li je nešto null ili ne ne koristimo = i !=, nego is NULL i is NOT NULL.
- Svako obeležje treba da **specificiramo**. Imamo tip entiteta N i skup obeležja Q koji zapravo čine taj TE. Obeležja mogu biti pridružena i TP. Svakom obeležju treba da se uradi specifikacija obeležja, pri čemu specifikaciju obeležja čini ograničenje vrednosti obeležja.
- **Ograničenje vrednosti obeležja *id(N, A)*:**
    - Definiše se za svako obeležje tipa
    - Struktura: *id(N, A) = (Domen, Null)*
    - *A – atribut ili obeležje, pripada nekom TE ili TP pod nazivom N i pri specifikaciji obeležja možemo da definišemo predefinisanu vrednost. U ograničenju vrednosti obeležja imamo pridruživanje vrednosti obeležju i specifikaciju da li to obeležje sme ili ne sme da ima null vrednost. Za razliku od domena gde je samo tip obavezan, ovde su obavezne obe komponente, tip obeležja i specifikacija za null vrednost.*
    - *Ova predefinisana vrednost ima prednost nad predefinisanom vrednošću iz domena, ukoliko se navede, uzeće se Predef iz ograničenja, ukoliko ga nema, uzeće se Predef iz domena, a ako u domenu nema Predef, biće null, ali ako je u ograničenju zadato da ne sme da ima null vrednost, doći će do greške.*
        - *Domen* – oznaka (naziv) pridruženog domena oebležja
        - *Null* $\in \{T, \perp\}$:
            - *T* – dozvola dodele nula vrednosti obeležju unutar *N*
            - $\perp$ -- zabrana dodele nula vrednosti obeležju unutar *N*

- Ograničenje pojave tipa
    - definiše ograničenja na moguće vrednosti podataka unutar iste pojave TE ili TP
    - predstavlja skup ograničenja vrednosti obeležja, kojem je pridodat logički uslov
    - formalno, za tip *N*:

$$id(N) = (\{id(N, A) \mid A \in Q'\}, Uslov)$$

    - Q' - prošireni skup obeležja tipa
        - za TE je $Q' = Q$
        - za TP je $Q' = Q \cup K_p$, gde je K_p skup obeležja primarnog ključa TP

- U ograničenje pojave tipa pre svega ulaze sva ograničenja vrednosti obeležja koja pripadaju TE (ako se radi o TE), a ako se radi o TP, onda sva obeležja koja pripadaju TP. Ta obeležja čine skup Q'. Ako se radi o TE, onda je Q' = Q (isti taj skup obeležja), a kod TP onda taj skup čine $Q' = Q \cup K_m$, gde je K_m skup obeležja

primarnog ključa TP (ukoliko TP ima svoja obeležja, ukoliko ih nema, taj skup će biti prazan). Znači, ovaj skup čine obeležja TP i obeležja povezanih tipova. Modeluje ograničenja na vrednosti neke pojave.

- U Uslovu mogu biti samo operandi iz koji važe za obeležja iz skupa Q'.
- Ograničenje se odnosi samo na jednu pojavu.
- Neophodno je i obavezno da zadajemo kardinalitete prema oba povezana tipa. Jedan i drugi povezani tip učestvuju u pojavama tipa poveznika. Kardinalitet pre svega spada u integritetnu komponentu, tj. on je jedan od tipova ograničenja i ograničava u koliko pojava TP može da učestvuje jedna (bilo koja) pojava povezanog tipa (može biti TE ili prethodno definisan TP).
- A – minimalni kardinalitet, B – maksimalni kardinalitet
- Kod ER modela minimalni kardinalitet može biti samo 0 ili 1, maksimalni kardinalitet može biti 1 ili više, čak i ako znamo da nečega može biti maksimalno 3 ili 5, 10, ER model ne prepoznaje to, nego je uvek više, a ako je to neko ograničenje da znamo koliko je to tačno više, to mora biti zapisano dodatno. **Maksimalni kardinalitet ne može biti 0!**
- Kardinalitet se obavezno zadaje prema oba povezana tipa (ako je binarni), ako je n-arni, onda se zadaju kardinaliteti prema svim povezanim tipovima.
- Tri opšte grupe maksimalnih kardinaliteta:
    - *M : N* – „više-više“
    - *N* : 1 – „više-1“
    - 1 : 1 – „jedan-jedan“
- Ako ima 0, ne mora da učestvuje ni u jednoj pojavi TP. Iz realnog sistema (od pravila koja važe u realnom sistemu) crpimo informaciju da li je 0, 1, više itd.
- Neka ograničenja ne možemo zapisati na ovom dijagramu, pa se to mora negde drugde zapisati i kasnije implementirati (npr. radnik ne može biti šef sam sebi).
- Integritet TE je zapravo ono ograničenje ključa koje smo već definisali.
- Integritet TP je ograničenje ključa TP koje se ne računa i nije jednako kao kod ograničenja ključa TE, a u integritet TP ulazi i niz povezanih tipova.
- U zavisnosti od grupe kardinaliteta maksimalnih prema oba povezana tipa, različito se definiše integritet TP, u integritet ulazi i neki niz poveznika ili neki njegov neprazan podniz.
- Tri opšte grupe maksimalnih kardinaliteta:
    - M : N
    - N : 1
    - 1 : 1
        - Uticaj na formiranje ključeva TP
- Vrednosti maksimalnih kardinaliteta učestvuju na formiranje ključeva.
- TP ima ključ, svaka pojava TP se identifikuje na osnovu ključa. U ključ TP ulaze ključevi povezanih tipova, a kako se formira ključ, zavisi od maksimalnih kardinaliteta.
- Kada imamo M : N (više : više), onda se uniraju ključevi povezanih tipova.
- Nikada ne možemo imati dva obeležja koja se isto zovu. Ono što je inače pravilo je da svako obeležje identifikuje naziv obeležja (mnemonik), ako imamo iste nazive, smatra se da je to jedno obeležje. Ne možemo imati 2 različita obeležja sa istim nazivom.
- Kada kreiramo tabele u SUBP-u, možemo svugde imati isti skup obeležja, SUBP nam to neće zabraniti, ali po principima modela se to ne radi. Svako obeležje ima svoju semantiku i svoj naziv, ne ponavljaju se. Ako je rekurzija, postoji mehanizam preimenovanje.

- Kada imamo N : 1 (više : 1) u identifikator TP ulazi samo podskup niza povezanih tipova. Ulazi onaj povezani tip koji se nalazi na strani 1, a ključ TP je ključ poveezanog tipa koji je na strani 1. Svaku pojavu ovakvog TP identifikuje ključ povezanog tipa sa strane 1. TP od atributa ima samo ono što smo mu dodelili, a i ne mora da ima atribute.
- Kada imamo 1 : 1 – sada svaku pojavu TP može da identifikuje ključ sa strane 1, pošto su sada obe strane 1, može i jedno i drugo, pa su ovo ekvivalentni ključevi i mogu identifikovati svaku pojavu TP.
- Rekruzivni TP – imamo vezu između pojava iste klase. Pošto u ovom primeru sa obe strane imamo više-više (M : N), primenjuje se isto pravilo kao i kod običnog TP, odnosno ključ se dobija uniranjem ključeva povezanih tipova. Međutim, ispašće da 2 puta imamo isti id, ali nikako ne smemo da odbacimo 1 jer imamo deo kao komponentu i deo kao deo te komponente. Komponenta može biti složena i sastojati se iz više delova. Ovde kod rekurzije treba da imamo vezu komponente sa drugom komponentom koja je njen deo, tako da se 1 id odnosi na komponentu, a drugi deo se odnosi na njenu podkomponentu. Pošto ne smemo da imamo dva ključa istog naziva, radimo preimenovanje da bi se razlikovali i da znamo koja je koja uloga. Ne smemo nijedan od ova dva da izgubimo jer onda gubimo poentu rekurzije, svaki deo treba da ima svoju ulogu. Ako imamo rekurziju sa vezom N : 1, onda je ključ sa strane 1.
- Rekurzija kad je 1 : 1 nema baš smisla. Jako je bitno da imenujemo grane kod rekurzije, gde je 1, a gde je više.
- Moramo navoditi TP jer oni mogu da budu i poveznici i povezani tipovi. Ako strelica ide ka njima, onda su povezani tipovi, a ako ide od njih, onda su poveznici.
- Egzistencijalna zavisnost je minimalni kardinalitet 1. Minimalni kardinalitet 1 se ne implementira deklarativno, u CASE alatu kada nacrtamo, napraviće tabele, biće propagacija ključa, foreign key gde treba, ali ovu 1 će da ignoriše i zato moramo da je implementiramo dodatno.
- Identifikaciona zavisnost podrazumeva egzistencijalnu, ali je još stroža. To što podrazumeva egzistencijalnu zavisnost znači da je i kod identifikacione zavisnosti minimalni kardinalitet 1, ali kod identifikacione je i maksimalni kardinalitet 1. Kada imamo identifikacionu zavisnost, onda je uvek (1, 1), ali ako je (1, 1), to ne znači da imamo identifikacionu zavisnost, čak se i izostavlja sa dijagrama kardinalitet jer je identifikaciona zavisnost poseban koncept koji ima svoje simbole.
- Kada nacrtamo simbol za identifikacionu zavisnost, onda je jasno da je neki TE slab i da identifikaciono zavisi, tako da ne moramo da pišemo 1, 1, podrazumeva se.
- U skladu sa tim da li imamo identifikacionu zavisnost, možemo klasifikovati TP na identifikacione i neidentifikacione.
- Identifikacioni TP – ukazuje da se svaka pojava zavisnog TE ne može identifikovati samostalno tj. bez pomoći identifikatora nadređenog TE od kojeg zavisi.
- To znači da slabi tip nema ključ, slabi tip ima samo identifikaciona obeležja, a ključ identifikaciono zavisnog slabog TE se formira tako što uniramo identifikaciona obeležja sa obeležjima ključa regularnog tipa od kojeg zavisi.
- Identifikaciona zavisnost nije poželjna jer svaka identifikaciona zavisnost zahteva 1 JOIN, a to je skupa operacija, pa je treba koristiti samo tamo gde je zaista potrebna, a ne da je koristimo samo zbog propagacije ključa.
- Koncept IS-A hijerarhije spada u prošireni ER model. U nekom trenutku se tokom posmatranja realnih sistema uočilo da bi bilo korisno da imamo neki koncept koji će predstaviti određene probleme.

- Ovaj koncept koristimo kada u realnom sistemu uočimo neku klasu (klase uočavamo tako što imamo entitete koji imaju neke zajedničke osobine u okviru tog realnog sistema i formiramo klasu na osnovu tih zajedničkih osobina). Međutim, uočavamo još neke zakonitosti – npr. da u skupu radnika koji će nam biti pojave te klase radnika uočavamo neke podskupove koji naravno imaju sve te zajedničke osobine, ali imaju i neke specifične (nemaju svi radnici te specifične osobine, već samo neka grupa; svaka grupa radnika ima svoje specifične osobine).
- Klasa radnika je samo jedna klasa, ali je nećemo modelovati tipom entiteta, jer bi onda u toj klasi morale da se nađu sve osobine (i one koje su svima zajedničke i one koje su specifične, oni koji ne postoje u nekog grupi imaju null vrednost).
- Ovaj koncept nam omogućava da modelujemo klasu radnika posebnim TP. Imaćemo klasu, odnosno koncept koji se zove **superklasa** i on će nam zapravo čuvati sve zajedničke osobine, a imaćemo i posebne **podklase** koje će nam odražavati one grupe i modelovaće zapravo one grupe date klase sa specifičnim atributima.
- Ovde se uvodi pojam **specijalizacije**. Superklasa se specijalizuje u neku podklasu, odnosno pojava superklase specijalizuje se u neku pojavu podklasa.
- To je TP koji nam od jedne klase napravi superklasu i nekoliko podklasa u zavisnosti od toga koliko smo grupa prepoznali. U podklasama se nalaze samo specifične osobine, u superklasi se nalaze samo zajedničke osobine.
- Klasa podklasa je identifikaciono zavisna od superklase.
- IS-A hijerarhija modeluje jednu klasu, ne više klasa. Obično postoji obeležje po kom se vrši specijalizacija.
- IS-A hijerarhiju uvodimo kada uočimo da bismo imali mnogo pojava sa specifičnim osobinama jer bi u suprotnom to dovelo do toga da sve atribute (i zajedničke i specifične) stavljali u istu klasu i onda bismo imali dosta null vrednosti kod specifičnih atributa koji ne postoje u svim grupama.
- Podklase, osim specifičnih osobina, mogu da imaju i neke posebne veze koje nemaju sve pojave klase, već samo pojave neke grupe sa specifičnim osobinama.
- Primarni ključ superklase je i primarni ključ podklase, sa razlikom da su podklase identifikaciono zavisne od superklase. Slabi tip ne može da se identifikuje samo svojim identifikacionim obeležjima, već se svaka pojava slabog tipa koji je identifikaciono zavisan identifikuje na osnovu ključa nadređenog i njegovih identifikacionih obeležja. Ovde, kod IS-A hijerarhije, imamo identifikacionu zavisnost podklasa od superklase, ali nema identifikacionih obeležja u podklasama i isključivo je ključ podklase ključ superklase, nema uniranja drugih obeležja. Eventualno, postoji mogućnost da podklasa ima svoj sopstveni ključ – neki ključ koji je ekvivalentan sa ključem iz superklase, ali ne uniraju se, to su ekvivalentni ključevi. **Podklase nemaju svoja identifikaciona obeležja!**
- Za razliku od slabog tipa koji je bio samo 1 i gore je išao nadređeni, ovde imamo jednog nadređenog i više podklasa koje su na istom nivou. Pošto se radi o identifikacionoj zavisnosti, kardinalitet ka podklasi je (1, 1), pa se to može i izostaviti. Kardinaliteti ka superklasi se moraju definisati,kao i kod svakog TP. Minimalni kardinalitet može biti 0 ili 1, a maksimalni kardinalitet može biti 1 ili više.
- U zavisnosti od vrste kardinaliteta, imamo različite vrste IS-A hijerarhije.
- Ukoliko je minimalni kardinalitet 0, kaže se da je IS-A hijerarhija **parcijalna**.
- Ukoliko je minimalni kardinalitet 1, kaže se da je IS-A hijerarhija **totalna** (obavezna).

- Ako je maksimalni kardinalitet 1, onda je IS-A hijerarhija **nepresečna**.
- Ako je maksimalni kardinalitet N, onda je IS-A hijerarhija **presečna**.
- U zavisnosti od kombinacija, imamo:
    - Parcijalnu nepresečnu
    - Parcijalnu presečnu
    - Totalnu nepresečnu
    - Totalnu presečnu
- Ako je minimalni kardinalitet 0, možemo imati radnike koji nisu specijalizovani ni za jednu od oblasti, a ako je maksimalni kardinalitet N, radnik može biti specijalizovan za više oblasti.
- Ako je maksimalni kardinalitet 1, obeležje po kome se radi IS-A hijerarhija ostaje.
- Ukoliko je zanimanje više vrsta zanimanja, rešićemo to presečnom IS-A hijerarhijom, a obeležje ZANIMANJE možemo izostaviti.
- Ako klasa nema posebnih obeležja niti neku vezu prema nečemu, onda to najverovatnije nije podklasa.
- IS-A hijerarhija nije baš potpuno isti koncept kao nasleđivanje u objektno orijentisanom programiranju.
- Podklasa može biti superklasa za neku dalju specijalizaciju. Svaka superklasa i podklasa može dalje da učestvuje u vezi sa ostalim TE, TP, nema nikakvih ograničenja.
- **Kategorizacija nije isto što IS-A hijerarhija i ne mogu jedna drugu zameniti**.
- **Kategorizacija** je takođe poseban tip poveznika. Tu se uvodi pojam **klasifikacije**, pojave nekog TE se kategorizuju i isključivo mogu biti samo jedna kategorija. Tu imamo ekskluzivitet u tom TP, odnosno, maksimalni kardinalitet je uvek 1.
- Može biti više kategorija, nema smisla da bude manje od dve. Minimalno mora biti makar dve kategorije, a može biti i više, pri čemu svaka pojava TE može biti maksimalno jedna kategorija (imamo maksimalni kardinalitet 1).
- Minimalni kardinalitet može biti 0 ili 1. Kada je u pitanju kardinalitet prema kategorijama, on se takođe definiše po istim pravilima kao i do sada, minimalni može biti 0 ili 1, a maksimalni 1 ili više.
- U zavisnosti da li je minimalni kardinalitet ka regularnom TE 0 ili 1, imamo različite vrste kategorizacije:
    - Ako je 0, imamo **parcijalnu kategorizaciju.**
    - Ako je 1, imamo **totalnu kategorizaciju**.
- Nema identifikacione zavisnosti. Svaka kategorija je poseban TE, nezavisan od ostalih i ima svoj ključ ako je ovaj TE koji kategorišemo.
- Razlike u odnosu na IS-A hijerarhiju:
    - Tamo imamo superklasu i podklase, identifikacionu zavisnost. IS-A hijerarhija zapravo modeluje samo jednu klasu iz realnog sistema, a podklase su podgrupe te klase. Kod kategorizacije imamo potpuno nezavisne klase koje se modeluju ovim TE kao i kategorijama.
- Ako su tipovi podataka ključeva svih kategorija isti, onda možemo d eliminišemo strane ključeve iz tih kategorija i da ostavimo 1 strani ključ za kategoriju i onda koja god da je kategorija u pitanju, strani ključ će dobiti vrednost iz te kategorije.

- Ako su ključevi različitih tipova podataka kod svih kategorija, onda imamo više stranih ključeva za svaku kategoriju, to stoji, biće null-ovi i to je to. Ovo može biti problem kada imamo veći broj kategorija.
- Sve što važi za binarni tip poveznika, važi i za N-arni TP. Kardinaliteti moraju da se definišu ka svakom od povezanih tipova (minimalni i maksimalni). Možemo koristiti N-arni tip poveznika kada nam je potreban, ali je sa binarnim tipovima mnogo lakše raditi.
- ER model je koristan u prvim fazama razvoja kada dosta učestvuju domen-stručnjaci koji ne moraju biti iz oblasti IT-ja.
- Preko ER dijagrama lako možemo da razumemo kako stvari funkcionišu i to ih je održalo do danas.
- Bitno je da imenujemo grane kod rekurzivnih TP, naročito kada su u pitanju različiti kardinaliteti.
- Kada imamo pravilo da ne može biti povezana bilo koja pojava sa bilo kojom pojavom nekih TE nego imamo pravilo da samo **kombinacije** određenih pojava mogu da učestvuju u pojavama nekih drugih povezanih tipova, tada koristimo **gerund**, tj. tipove poveznika između dva, pa ćemo onda sa TP da vežemo taj treći, a neće biti situacija da imamo ono trinarno. Kod trinarnog bilo koje pojave mogu biti u vezi.
- **Gerund** je glagolska imenica.
- U ER modelu, gerund može biti:
    - Tip entiteta dobijen transformacijom tipa poveznika, tj.
    - Tip poveznika, koji predstavlja povezani tip nekom drugom tipu poveznika
- Dvojaka uloga gerunda, kao tipa – istovremeno i tip entiteta i tip poveznika
    - Tip poveznika, za neke druge povezane tipove
    - Tip entiteta u nekom drugim tipovima podataka
- Dat je TP $N(N_1, N_2, ..., N_m, \{B_1, ..., B_k\}, C)$
    - Neka je N_i, takođe, tip poveznika
    - N_i predstavlja gerund
    - N_i se ponaša kao TE u odnosu na N
- Upotreba gerunda:
    - Kada **ne mogu proizvoljne kombinacije** pojava nekih tipova biti sadržane u pojavi posmatranog tipa poveznika i
    - Postoji **pravilo koje kombinacije** pojava tih tipova mogu biti sadržane u pojavi posmatranog tipa poveznika
        - Tip poveznika – gerund uvodi se sa ciljem modeliranja tog pravila
- **Agregacija** obezbeđuje objedinjavanje složenijih ER struktura.
- Cela ER struktura se posmatra kao jedan tip entiteta
    - Predstavlja povezani tip za neki TP
    - Može predstavljati korisnički pogled na BP („virtuelni“ TE)
- Najjednostavniji primer agregacije – gerund

Relacioni model

- Strukturalna komponenta: I ovde, kao i kod ER modela, kao primitivne koncepte imamo atribut (obeležje) i domen (atribut i domenčine nivo intenzije – nivo LSO-a), a nivo ekstenzije nam je vrednost.
- LSO (nivo intenzije) je viši nivo apstrakcije u odnosu na LSP (nivo ekstenzije).
- Na nivou intenzije imamo domen, konkretizacija je vrednost koja zadovoljava specifikaciju domena.
- Na nivou intenzije imamo obeležje, a na nivou ekstenzije imamo podatak.
- Na nivou intenzije imamo skup obeležja, a na nivou ekstenzije imamo torku.
- Torka je neki skup podataka, gde svako obeležje dobija vrednost.
- Na nivou intenzije imamo šemu relacije, a na nivou ekstenzije imamo relaciju.
- Na nivou intenzije imamo šemu BP, a na nivou ekstenzije bazu podataka.
- **Torka** reprezentuje jednu pojavu entiteta ili poveznika. Pomoću torke se svakom obeležju, iz nekog skupa obeležja, dodeljuje konkretna vrednost iz skupa mogućih vrednosti definisanog domena.
- **Svako obeležje ima domen.**
- $U = \{A_1, ..., A_n\}$ – skup obeležja
- $DOM = \cup_{i=1}^{n} (dom(A_i))$ - skup domena
- Skup domena je unija domena svakog od ovih obeležja, ali broj domena ne mora da bude isti kao broj obeležja. Ako imamo N obeležja, broj domena može biti do N, a može biti i manji jer neka obeležja pripadaju istim domenima.
- Torka je preslikavanje skupa obeležja u skup vrednosti u odnosu na skup domena.
- Za svako obeležje iz skupa obeležja važi da je vrednost obeležja u okviru torke neka vrednost iz skupa mogućih vrednosti obeležja. Kada svakom obeležju iz skupa obeležja dodelimo vrednost, onda imamo torku.
- U ER modelu smo tip entiteta i tip poveznika modelovali kao pojavu TE ili pojavu TP. Za modelovanje neke realne veze (činioca realnog sistema) imamo samo torku. Ako nam je potrebno da razlikujemo TE i TP, moramo to uraditi na neki drugi način.
- Restrikcija (skraćenje) torke – imamo neki skup obeležja i vrednosti za ta obeležja i sada hoćemo da dobijemo restrikciju te torke na određeni skup obeležja koji je podskup polaznog skupa obeležja. Skup X je podskup skupa obeležja U. Restrikcija (skraćenje) torke sa skupa obeležja U na skup X se označava sa t[X].
- Vrednost obeležja iz X u restrikciji torke je ista ona koju ta obeležja imaju u polaznom skupu.

Formalni zapisi obavezni za 9 i 10!

- $X \subseteq U$, $t: U \rightarrow DOM$,
- $t[X]: X \rightarrow DOM$

$$(\forall A \in X)(t[X](A) = t(A))$$

- **Relacija** je konačan skup torki (konačan skup torki). Reprezentuje skup realnih entiteta ili poveznika. U ER modelu smo imali TE i TP da predstavljamo klase realnih entiteta i poveznika, a ovde imamo samo 1 koncept, a to je relacija.

- **Torka** reprezentuje pojavu TE i TP, relacija kao skup torki reprezentuje i skup entiteta i skup poveznika, odnosno klase realnih entiteta i realnih poveznika.
- **Relacija** se označava slovom *r*.

$$r(\boldsymbol{U}) \subseteq \{t \mid t: \boldsymbol{U} \rightarrow DOM\}, \mid r \mid \in \mathbb{N}_0$$

Skup svih mogućih torki nad skupom obeležja $\boldsymbol{U}$ - $Tuple(\boldsymbol{U})$

- Formalni zapis relacije
- r je skup torki koji preslikava U na skup domena
- Relaciju može da čini i samo jedna torka, ali može i da ih ima više.

- U relaciji se ne mogu pojaviti identične torke, onda je to ista torka, samo dva puta prikazana.
- Sadržaj relacije najčešće predstavljamo tabelarno. Relaciju predstavlja kompletan sadržaj tabele, pa zato često relaciju nazivamo tabelom.
- Redosled kolona u tabeli (relaciji) je apsolutno nebitan, svaka kolona se odnosi na odgovarajuće obeležje, a u toj koloni su vrednosti tog obeležja za različite torke.
- Redosled torki u tabeli je takođe nebitan jer mi do određene torke dolazimo pomoću nečega šro se zove identifikator (identifikacija).
- Ono što **ne sme** da se desi je da se u relaciji pojave dve potpuno identične torke, ukoliko imamo identične vrednosti za obeležja, onda se tu radi o istoj torki.
- LSO daje kontekst LSP, moramo imati kontekst da znamo kome koje vrednosti pripadaju.
- **Šema relacije** je LSO koja daje kontekst LSP koja se zove relacija.
- **Šema relacije** je imenovani par *N(R, O)* gde je:
  - *N* – naziv šeme relacije (može biti izostavljen)
  - *R* – skup obeležja šeme relacije
  - *O* – skup ograničenja šeme relacije → **skup ograničenja koji se odnosi na datu šemu relacije**
- Ako je neka relacija pojava nad tom šemom, onda taj skup torki koji čini relaciju mora da zadovoljava sva ograničenja zadata u okviru šeme relacije. Ako ne zadovoljava ta ograničenja, onda to nije pojava nad tom šemom. A nas zanimaju relacije koje su pojave nad odgovarajućom šemom. Ako je relacija pojava nad datom šemom, to znači da smo ispoštovali sve ono što je zadato na nivou šeme, tako funkcionišu stvari u relacionim BP.
- Ovim konceptom šeme relacija moramo da predstavimo i TE i TP koje smo imali u ER modelu, mada neki tipovi poveznika neće biti deo šeme relacija, već samo propagacijom ključa.
- Koncept šeme relacije se izjednačava pre svega sa tipom entiteta.
- **Relaciona šema baze podataka** predstavlja imenovani par *(S, I)* gde je:

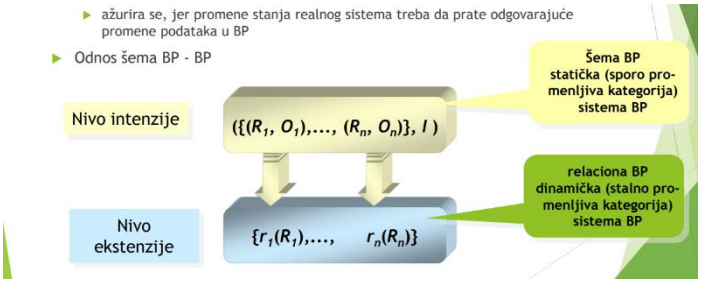
$$S = \{(R_i, O_i) \mid i \in \{1, \ldots, n\}\}$$

  - *S* – skup šema relacija
  - *I* – skup međurelacionih ograničenja
- I ovo je nivo intenzije, nivo LSO.
- Ograničenja O su važila samo nad tom šemom relacije i ta ograničenja moraju da zadovolje pojave nad tom šemom.
- Međurelaciona ograničenja moraju da zadovolje više od jedne šeme relacija.
- Šema relacije je zaglavlje tabele, a kada je popunimo podacima, dobijemo relaciju, odnosno sadržaj tabele i dobijemo čitavu tabelu.

- **Relacionu bazu podataka** čini skup pojava šeme relacija i skup međurelacionih ograničenja. Ne možemo uneti novu torku u neku tabelu (relaciju) ako nije ispunila ograničenja tabele (relacije) koja je povezana sa njom. Ono što se dešava u jednoj relaciji zavisi od onoga što se dešava u drugoj relaciji i zato se i zovu međurelaciona ograničenja.
- Svaka relacija je pojava nad zadatom šemom relacije iz skupa šema relacije koji čini relacionu šemu baze podataka.
- Skup relacija pri čemu svaka relacija je pojava nad zadatom šemom relacije i skupa šeme relacije koju čini.
- Šema relacije je zaglavlje tabele, relativnu bazu podataka čini skup tih zaglavlja (da znamo koje podatke imamo), a baza podataka je komplet s podacima.
- Baza podataka je skup tabela, a svaka tabela je skup torki, samo što te tabele ne zovemo više tabele nego relacije. Relacionu bazu podataka čini skup relacija, a svaku relaciju čini skup torki. Da bi to bila baza podataka nad onom šemom baze podataka, onda moramo da pričamo o zadovoljenju onih ograničenja.
- Svaka pojedinačna relacija iz tog skupa relacija koje čine bazu podataka zadovoljava ograničenja *O* jer relacija je pojava nad nekom šemom relacija. Svaka relacija mora da zadovolji skup ograničenja *O*, a sve relacije zajedno moraju da zadovolje i skup međurelacionih ograničenja *I*. Tek tada, kada sve te relacije (odnosno, pojave nad datim šemama) zadovoljavaju sva ova ograničenja, onda možemo da pričamo o tome da taj skup relacija čini relacionu bazu podataka koja je zapravo pojava nad šemom koja zadaje ta ograničenja.
- Kao što je relacija pojava nad zadatom šemom relacije, tako je i relaciona BP pojava nad zadatom relacionom šemom baze podataka.

$s\colon S \to \{r_i \mid i \in \{1,\ldots,n\}\},\ (\forall i) s(R_i, O_i) = r_i$

- Šema BP je nivo intenzije, odnosno daje kontekst (LSO). Šema BP je dosta manje promenljiva od onoga što imamo na nivou ekstenzije.
- Kada definišemo šemu BP (a to je upravo ono što radimo pomoću ER modela) na kraju će se implementirati kao relaciona šema BP, pa kada je popunimo podacima, dobićemo i relacionu BP.
- Kada napravimo šemu BP sa skupom šema relacija ili imamo šemu BP pomoću TE i TP predstavljenu, ne znači da u nekom trenutku nećemo trebati da nešto dodamo ili izmenimo u toj šemi (dodavanje obeležja, promena kardinaliteta, promena tipa podatka za neko obeležje... – to su sve koncepti šeme i mogu se menjati, međutim, najčešće se neće više menjati kada završimo sa šemom BP).



- **Konzistentno stanje baze podataka** se može podeliti na:
  - Formalno konzistentno stanje
  - Suštinski konzistentno stanje
- **Formalno konzistentno stanje** znači da svaka relacija nad datom šemom jeste pojava, odnosno zadovoljava sva ona ograničenja zadata šemom relacije i da sve relacije zadovoljavaju sva međurelaciona ograničenja. Formalna konzistentnost automatski se proverava kada definišemo ograničenja na nivou SUBP-a i eventualno se sprovode mere da baza ostane u konzistentnom stanju ukoliko dođe do nekih operacija koje bi mogle sve ovo da naruše.

- **Suštinsko konzistentno stanje** – pre svega, suštinsko konzistentno stanje podrazumeva ovo formalno konzistentno stanje – ovo formalno se uopšte ne stavlja pod znak pitanja, to ne sme da se naruši ni na koji način. Suštinski konzistentno stanje ne mora da se razlikuje od formalno konzistentnog stanja, sve što nam je u bazi je as is u realnom sistemu (nemamo u sistemu ništa što nemamo u bazi i obrnuto), ali se može desiti i najčešće se dešava situacija da se u realnom sistemu desi neka promena, ali postoji neki vremenski gap koji kao čini da baza nije u suštinski konzistentnom stanju zato što nismo još svaka promena iz realnog sistema reflektovana na sadržaj baze podataka. To je u praksi realna situacija i često se dešava. **SUBP ne može da kontroliše suštinski konzistentno stanje!** To je nešto čemu se teži, da se zadaju ograničenja na nivou šeme, na nivou SUBP-a da budu implementirana i da je onda zapravo SUBP taj koji će ta ograničenja da kontroliše i da onda više apsolutno nije bitno i da nemamo više nikakvu bojazan da li je neka od aplikacija koje pristupaju bazi podataka nešto od ograničenja uvažila ili ne. Kada imamo proveru konzistentnosti na nivou SUBP-a, onda bilo ko da pristupa bazi (preko aplikacije, neki korisnik, ili ad hoc da pristupa podacima kojima ima pravo pristupa) ne može da zaobiđe kontrolu ograničenja. Kada jednom zadamo ovu kontrolu na nivou SUBP-a, ne moramo više da mislimo da li je neko na aplikativnom nivou da li je neko ograničenje zadovoljeno ili ne, to mora da bude zadovoljeno. Kada imamo zadata ta ograničenja, SUBP to sve vreme u pozadini kontroliše i proverava, ukoliko je sve u redu, mi nismo ni svesni da on to proverava. Ukoliko nešto nije u redu, mi dobijemo poruku o greški. Naravno, kada ad hoc pristupamo bazi, mi dobijamo poruku dole, a kad zapravo aplikacija pristupa, onda se pojavi ta greška, moramo na aplikativnom nivou da hvatamo taj izuzetak i da hendlujemo tu grešku i da vidimo šta ćemo sa njom. U najgorem slučaju da kažemo „Neki je problem“, ali aplikacija ne sme da pukne, treba da hvatamo grešku i da znamo koja se greška desila, da li nešto nismo pronašli (kada ne nađemo grešku, to takođe ide kao izuzetak)
- **Šema relacije je izvedeni koncept (kao TE i TP)!**
- **Integritetna komponenta** – integritetnu komponentu čine tipovi ograničenja koje modelujemo tako da budu ograničenja na podatke, ali takođe o na odnose koji postoje između realnih entiteta (među podacima).
- **OVO DOSTA PITA NA ISPITU**!!!
- Za svaki tip ograničenja imamo odgovarajuće karakteristike. Pre svega, za svaki tip imamo **formalizam za zapisivanje** (to je kao sintaksa programskog jezika koja nam kaže na koji način nešto može da se uradi, kao CONSTRAINT, PRIMARY KEY,), ,...).
- **Pravila za interpretaciju (validaciju)** – ograničenja dolaze iz realnog sistema (constraint, primary key, check, a mogu postojati i neka složenija ograničenja. **Moramo znati na koji način se proverava da li je ograničenje narušeno ili nije.** SUBP na osnovu toga što smo mi zadali ograničenje treba da proverava da li je to ograničenje narupeno ili nije. Ako je sve u redu, ograničenje nije narušeno. Imamo ograničenje domena – npr. imamo neko obeležje tipa broj, ali tu pokušavamo da unesemo string, narušili smo ograničenje – to se validira tako što se proverava da li je tip podatka odgovarajući, ako tip podatka nije odgovarajući – imamo grešku (validirali smo i dobili smo da je narušeno ograničenje. **Pravilo za interpretaciju predstavlja način na koji se proverava da li je neko ograničenje narušeno ili nije**.
- Svaki tip ograničenja mora imati **oblast definisanosti** – nad kojom oblašću (nad kojom LSO) je definisan tip ograničenja.

- Svaki tip ograničenja mora imati **oblast interpretacije** – tip logičke strukture podatka nad kojim se ograničenje interpretira. Ovo je konkretizacija, nivo ekstenzije, stvarni podaci nad kojima se proverava da li je ograničenje ispunjeno ili ne. Postoje 4 LSP-a kao varijanta za oblast interpretacije.
- CREATE TABLE je DDL naredba kojom kreiramo šemu, odnosno delove šeme relacije.
- Ograničenja mogu da naruše neke operacije nad podacima u BP, a to su: DML naredbe – INSERT, UPDATE i DELETE su operacije nad podacima u bazi koje mogu da budu kritične – to se zovu **kritčne operacije**. One se zovu tako što mogu da naruše ograničenje.
- Za različite tipove ograničenja, različite su tipične operacije. Nisu za svaki tip ograničenja sve operacije kritične. Negde je unos, negde brisanje i ažuriranje, ali uvek je neka operacija kritična (ne postoji ni jedan slučaj gde neka operacija nije kritična).
- **Svaki tip ograničenja kontroliše nešto drugo, jer u suprotnom ne bi postojao kao poseban tip**.
- **MORA SE ZNATI ŠTA OGRANIČENJE OGRANIČAVA!!!**
- Postoje aktivnosti koje se mogu primeniti da bi baza ostala u konzistentnom stanju, bez obzira na kritičnu operaciju.
- No action exception (restricted) – kada nam SUBP zabrani da unesemo torku sa identifikatorom koji već postoji, tzv. Pasivna aktivnost, nije ništa uradio, samo nam je zabranio tu aktivnost. To je difoltna operacija.
- Kada se desi kritična operacija kojom bi se narušilo ograničenje i konzistentnost BP, SUBP zabranjuje operaciju, odnosno nema efekta, ne dozvoljava da se operacija izvrši.
- Postoje i druge aktivnosti, kao što je set null, set default, cascade koje dozvole operaciju, ali pritom ta aktivnost još nešto uradi da bi baza ipak ostala u konzistentnom stanju.
- Ova difoltna aktivnost je najčešća, ove ostale se ne mogu primeniti u svim slučajevima, samo u pojedinim neke od tih aktivnosti mogu biti upotrebljene.
- Ove aktivnosti nisu omogućene za svaku operaciju i za svaki tip ograničenja, možemo ih primeniti samo u pojedinim tipovima ograničenja.
- Kada pokušamo da unesemo radnika sa istim mbr-om kao neki već postojeći, tu smo pokušali da narušimo ograničenje ključa.
- Oblasti definisanosti u relacionom modelu podataka:
    - **Vanrelaciono ograničenje** – definiše se izvan konteksta relacije (npr. ograničenje tipa, ograničenje domena)
    - **Jednorelaciono** (untarrelaciono, lokalno) ograničenje – definiše se nad tačno jednom šemom relacije. Ovo su tipovi ograničenja koji mogu da dođu u okviru skupa *O*.
    - **Višerelaciono ograničenje** – definiše se nad skupom ili nizom šema relacija koji sadrži bar 2 člana. Odnosi se na ograničenja unutar skupa *I* (skupa međurelacionih ograničenja).
- **SUBP** ne može na deklarativan način da implementira neka složenija ograničenja iz poslovne logike, ali može neka jednostavnija u okviru create table naredbe.
- **Oblasti interpretacije u relacionom modelu podataka**:
    - **Ograničenje vrednosti** – implementira senad tačno jednom vrednošću nekog obeležja. Nad vrednošću obeležja se proverava da li je neko ograničenje zadovoljeno ili ne.
    - **Ograničenje torki** – implementira se nad jednom torkom bilo koje relacije
    - **Relaciono ograničenje** – interpretira se nad skupom torki bilo koje relacije

    - **Međurelaciono ograničenje** – interpretira se nad barem dve, bilo koje relacije. Relacija može biti neka tabela u bazi ili ono što se dobije kada spojimo dve tabele ili uradimo select, pa filtriramo samo neke torke ili uradimo restrikciju, pa dobijemo skup torki, ali skup restrikcija odgovarajućih torki iz tabele je takođe relacija.
- *TIPOVI OGRANIČENJA OBAVEZNO MORAJU DA SE ZNAJU – POGOTOVO OGRANIČENJE KLJUČA I OGRANIČENJE REFERENCIJALNOG INTEGRITETA!!!*
- Tipovi ograničenja u relacionom modelu podataka:
    - Ograničenje domena
    - Ograničenje vrednosti obeležja
    - Ograničenje torke
    - Integritet entiteta (ograničenje ključa)
    - Ograničenje jedinstvenosti vrednosti obeležja
    - Zavisnost sadržavanja
    - Ograničenje referencijalnog integriteta
    - Funkcionalna zavisnost

Realizacija ograničenja šeme relacione baze podataka putem SUBP

- **Pasivne akcije**:
    - ***NoAction (Restrict)*** – zabrana sprovođenja operacije koja bi izazvala narušavanje kontrolisanog ograničenja
- **Aktivne akcije**:
    - ***Cascade*** – kaskadna propagacija operacije na podatke, povezane s podacima koji se ažuriraju i kontrolišu putem ograničenja; pri kontroli ograničenja dozvoljava se operacija koja bi narušila ograničenje, ali kaskadno bi se propagirala operacija na podatke koji su povezani sa podacima na koje bi se odnosila operacija
    - ***SetNull*** – svođenje na nula vrednosti podataka, povezanim s podacimak koji se ažuriraju i kontrolišu putem ograničenja; imamo kritičnu operaciju, u datom slučaju kritična operacija će biti pokušaj narušavanja ograničenja, dolazi do toga da SUBP dozvoljava da se izvrši operacija koja bi narušila konzistentnost baze, ali se baza ostavlja u konzistentnom stanju tako što se podaci koji su povezani sa podacima na koje se odnosi operacija postavljali na null vrednost
    - ***SetDefault*** – isto kao SetNull, propagira se operacija na podatke koji su povezani sa podacima na koje se odnosi ova kritična operacija tako što sada te podatke postavljamo na neku difoltnu vrednost
    - ***Cascade, SetNull i SetDefault podržavaju SUBP***-ovi
    - ***<<UserDef>>*** - načelno, teoretski pomoću ovoga možemo da isprogramiramo bilo šta za kontrolu ograničenja, odnosno za svođenje na konzistentno stanje BP, da baza i pri specifičnoj operaciji koja bi zapravo narušila ograničenje vratimo opet podatke u konzistentno stanje.
    - ***Zato su ovo aktivni mehanizmi, imamo dalje izvršavanje operacija***
- Primer: ako imamo poslovne partnere i fakture gde faktura referencira poslovnog partnera tako što je ID poslovnog partnera (što je primarni ključ kod poslovnog partnera) kome je ta faktura

izdata njen strani ključ. Kod ograničenja referencijalnog integriteta, kritična operacija je brisanje i to brisanje iz referencirane tabele. Ako imamo *NoAction (Restrict)* – aktivnost zabrane koja bi narušila ograničenje, tako da ako hoćemo da izbrišemo torku u referenciranoj tabeli i ukoliko nema torki koje referenciraju tu torku, brisanje će biti dozvoljeno. Ukoliko imamo drugačiju situaciju, tj. da imamo zapravo za tu torku koju želimo da obrišemo neke referencirane torke (želimo da obrišemo nekog poslovnog partnera koji već ima neke fakture), ova operacija (brisanje) ne može biti izvršeno, aktiviraće se *NoAction*, jer za poslovanje ne bi bilo dobro da obrišemo partnera i fakture kao da nikada nisu postojali. Ako neki poslodavac sa kojim smo pre sarađivali ima svoje fakture, on ne sme da se obriše ni nakon što smo prekinuli saradnju sa njim.

- Ako imamo nekog neaktivnog partnera sa kojim više ne poslujemo i želimo da obrišemo i njega i njegove fakture, u nekim slučajevima nam neće zabranjivati, nego će se pri brisanju sprovesti *kaskadna aktivnost* – dozvoliće se brisanje poslovnog partnera, ali da bi baza ostala konzistentna, onda se automatski brišu i sve fakture koje su povezane za tog poslovnog partnera. Pri projektovanju, moramo dobro da razmotrimo da li u odnosu na pravila poslovanja smmo to da dozvolimo ili ne. Npr. kod finansija ovo baš i ne bi smelo, ali ako imamo samo neku evidenciju aktivnih poslovnih partnera, onda bi moglo. Ovo ne bi moglo da se desi na osnovu fakture, ali može na osnovu poslovnog partnera.
- SetNull – situacija je ista, želimo da obrišemo nekog poslovnog partnera. Gledamo da li za njega imamo vezane neke fakture koje pripadaju ovom poslovnom partneru. Ukoliko postoje, u suštini ne bi trebalo da tek tako možemo da obrišemo torku poslovnog partnera koja ima referencirane neke torke (ima referencirajuće torke). Eventualno, možemo da orišemo poslovnog partnera sa kojim više ne poslujemo, ali da ostavimo fakture da bismo znali šta smo imali od naplaćenih stvari. Ako nam je podešeno da ID poslovnog partnera (stranog ključa u fakturi) ne sme da bude null, ne smemo ovo koristiti.
- SetDefault – radi po istom principu, brišemo poslovnog partnera koji ima fakture koji ga referenciraju, tj. vezane su za tog poslovnog partnera.
- Ukoliko imamo spajanje dva dosadašnja poslovna partnera u jednog novog, mi ove prošle fakture prevezujemo na ID tog novog partnera i strani ključ postavljamo na odgovarajuću difoltnu vrednost.
- Situacija može biti i sledeća, možemo imati neku difoltnu torku i da svako ovo brisanje vežemo za tu istu torku. Ograničenje referencijalnog integriteta je ispoštovano. Ovaj setDefault je nezgodan i nije čest u upotrebi.
- Mehanizmi RSUBP namenjeni za implementaciju:
    - Skupa šema relacije BP
    - Ograničenja (integritetne komponente) šeme BP
    - Ostalih pravila poslovanja, koja
        - Ne rezultuju u ograničenjima šeme BP
        - Odnose se obično na
            - Unapred definisani redosled, obaveze i uslovljenosti izvođenja operacija nad BP, ili
            - Obavezu uvođenja nekih obeležja nad BP, pod određenim uslovima definisanim putem odnosa vrednosti obeležja u BP
- Implementacija skupa šeme relacija:
    - Kreiranje, modifikovanje i brisanje korisnički definisanog domena

        - CREATE DOMAIN – SUBP-ovi ga uglavnom ne podržavaju, pa ga i ne radimo;
        - ALTER DOMAIN
        - DROP DOMAIN
    - Kreiranje, modifkovanje i brisanje složenog tipa podataka
        - CREATE TYPE – ovo SUBP-ovi eventualno podržavaju
        - DROP TYPE
    - Kreiranje, modfikovanje i brisanje tabele (šeme relacije)
        - CREATE TABLE
        - ALTER TABLE
        - DROP TABLE
    - Dodavanje, modifikovanje i brisanje kolone tabele (obeležja šeme relacije)
        - ALTER TABLE / ADD
        - MODIFY
        - DROP
- Implementacija ograničenja šeme BP:
    - **Deklarativni mehanizmi**
        - Aktivnosti provere važenja ograničenja i konzistentnosti se, većim delom, podrazumevaju
            - SQL klauzula CONSTRAINT i tu definišemo sva ograničenja, svi oni tipovi ograničenja mogu tu da se implementiraju pomoću deklarativnih mehanizama
            - CREATE DOMAIN, CREATE ASSERTION
        - Sve ono što ne može deklarativnim mehanizmima, npr. slučaj kada imamo ograničenje referencijalnog integriteta (minimalni kardinalitet 1), trebaju nam proceduralni mehanizmi
    - **Proceduralni mehanizmi**
        - Aktivnosti provere važenja ograničenja i očuvanja konzistentnosti se, većim delom, programiraju – trebaju nam proceduralna proširenja SQL-a
            - Putem proceduralnog jezika
            - CREATE TRIGGER
            - CREATE PROCEDURE
            - CREATE FUNCTION
            - CREATE PACKAGE
            - CREATE PACKAGE BODY
        - Različiti SUBP-ovi podržavaju na različite načine ove mehanizme.
        - Trigger je nešto što je standardno, oni SUBP-ovi koji imaju ta proširenja, svakako podržavaju trigger. Međutim, ono što trigeri mogu da urade (klauzule trigera) nisu u svim SUBP-ovima iste. Standard definiše deklaraciju trigera, procedure ili funkcija, ali ne obezbeđuje zadatke tj. aktivnost tih proceduralnih stvari (trigera, procedura, funkcija).
        - SUBP-ovi imaju svoja sopstvena proceduralna proširenja koja ne podležu standardima. Npr. Oracle ima PL/SQL, a Microsoft ima svoj standard sql. I jedan i drugi imaju i funkcije i procedure i trigere, ali se sintaksa solidno razlikuje. Ni

blizu ne možemo kod zapisan na jednom da koristimo na drugom SUBP-u. Čak i deklaracija promenljivih ide potpuno različito, trigeri idu različito...

- Proceduralnim mehanizmima, osim nekih ograničenja koja smo već uočili, možemo implementirati i neka pravila poslovanja koja imamo. Ne moraju sva pravila poslovanja da se implementiraju na srednjem sloju, bekendu ili u biznis logici, mi možemo imati pravila poslovanja implementirana na nivou SUBP-a (na nivou šeme baze podataka) koja su smeštena na serveru i mogu da se pozivaju iz ovog sloja ili iz trigera.
- Ta proceduralna proširenja imaju sve komponente bilo kog proceduralnog jezika, sve naredbe kontrole i toka programa, tako da apsolutno možemo da programiramo kako smo navikli, samo što je proceduralno, nije objektno orijentisano.
- CREATE DOMAIN postoji, ali ga SUBP-ovi ne podržavaju!
- Tipovi ograničenja koji se mogu deklarativno zadati:
    - NOT NULL – ograničenje nula vrednosti; na nivou samog obeležja, bez CONSTRAINT-a, uvek se zadaje na nivou svakog obeležja šeme relacije (kolone tabele), proverava se prilikom svakog pokušaja upisa nove vrednosti ili modifikacije postojeće vrednosti obeležja; u slučaju pokušaja narušavanja ograničenja, jedina moguća aktivnost je sprečavanje operacije (NO ACTION); svi savremeni SUBP-ovi podržavaju klauzulu NOT NULL
    - PRIMARY KEY – ograničenje primarnog ključa; ako nam je samo jedno obeležje PRIMARY KEY, možemo definisati ograničenje odmah na nivou tog obeležja ili na nivou cele šeme relacija, onako kako se to zapravo i navikli i najčešće se to i radi tako što idemo sa klauzulom CONSTRAINT *Naziv* PRIMARY KEY – ključna reč koja određuje tip ograničenja (ovo je zapravo definicija koja je zadata SQL jezikom) i ide lista obeležja koja predstavljaju ključ; pdrazumeva se da je svako PRIMARY KEY obeležje deklarisano kao NOT NULL obeležje i to nije potrebno posebno deklarisati; proverava se prilikom svakog pokušaja upisa nove vrednosti obeležja primarnog ključa ili modifikacije postojeće vrednosti obeležja ključa; u slučaju pokušaja narušavanja ograničenja, jedina moguća aktivnost je sprečavanje operacije (NO ACTION); svi savremeni SUBP podržavaju klauzulu PRIMARY KEY, ali kod nekih SUBP, pokretanje obe klauzule automatski izaziva kreiranje UNIQUE indeksa (B^+ stabla) nad listom obeležja
    - UNIQUE – ograničenje jedinstvenosti; može na nivou obeležja (kao PRIMARY KEY) ili na nivou čitave šeme relacije tako što zadamo dovoljno CONSTRAINT-a; proverava se prilikom svakog pokušaja (kritične operacije su): upis nove vrednosti obeležja iz liste, ili modifikacija postojeće vrednosti obeležja iz liste; u slučaju pokušaja narušavanja ograničenja, jedina moguća aktivnost je sprečavanje operacije (NO ACTION)
    - CHECK – ograničenje torke; možemo implementirati ograničenja torke kada u logičkom izrazu imamo više od jednog obeležja, a ako imamo samo jedno obeležje u logičkom izrazu, može da se definiše na nivou samo jednog tog obeležja, ne mora na nivou šeme relacija i najčešće zapravo to znači onaj uslov u ograničenju domena; mi nemamo eksplicitno CREATE DOMAIN mehanizam kod SUBP-a, ali komponente domena možemo da realizujemo tako što imamo tip podatka, dužinu u zavisnosti od tipa, a uslov da implementiramo sa ovim CHECK; to na obeležje domena se odnosi kada je samo jedno obeležje; u ovom logičkom izrazu mogu da stoje samo obeležja te jedne šeme relacija; to podržavaju SUBP-ovi; postoje ograničenja koja se zovu proširena i gde u logičkom izrazu

ima potrebe da stoje obeležja i neke druge šeme relacija, odnosno tada se ovo CHECK ograničenje odnosi na spoj dve ili više šema relacija; mehanizam SUBP-a ne dozvoljava deklarativni da se to reši.
    - FOREIGN KEY – ograničenje ključa
- **Naziv CONSTRAINT-a mora biti jedinstven na nivou čitave šeme baze podataka**!!! Obeležja mogu da se ponavljaju u različitim šemama relacije, nazivi constraint-a moraju biti jedinstveni na nivou čitave šeme relacija; ako zadamo naziv CONSTRAINT-a da bude isti u dve šeme relacija, onda SUBP neće to dozvoliti.
- **Na nivou firme/projekta treba da imamo pravilo (konvenciju) kako se formiraju nazivi constraint**-a. Najčešće idu: naziv šeme relacije + prefiks + sufiks (ako je primarni ključ – PK, unique – UK,...)
- **Na početku projekta treba da bude jasno specificirano kako će se šta imenovati**.
- **U ovim logičkim izrazima ne smeju da budu obeležja drugih šema relacija i ne smeju da se pojave podupiti** (ne možemo da imamo SELECT-e u kojima opet idemo ka nekim drugim šemama relacija).
- **Rezultat provere logičkog izraza može biti:**
    - **TRUE**
    - **FALSE**
    - **NULL**
- Ograničenje je narušeno samo kada je rezultat FALSE.
- CHECK se proverava prilikom svakog pokušaja upisa nove torke u relaciju ili modifikacije postojeće vrednosti obeležja, obuhvaćenog zadatim logičkim izrazom.
- Ograničenje je narušeno kada je rezultat logičkog izraza FALSE.
- Ograničenje nije narušeno kada je rezultat logičkog izraza TRUE, ili NULL.
- U slučaju pokušaja narušavanja ograničenja, jedina moguća aktivnost je sprečavanje operacije – NO ACTION.
- Mi u deklaraciji ograničenja ne definišemo nikakvu aktivnost, pa se u tim slučajevima podrazumeva da je NO ACTION i nije ni logično da bude neka druga.
- U FOREIGN KEY imamo listu obeležja koja je zapravo lista obeležja stranog ključa
- REFERENCES ka – šema relacije stranog ključa; kada se referencira, ide *ListaRefObeležja* što je zapravo lista obeležja u referenciranoj šemi relacije – kada tu ne stavimo ništa dalje, podrazumeva se da smo izabrali NO ACTION (zabranu kritične operacije).
- Pri deklarativnom zadavanju, tj. prilikom kreiranja tabele, možemo odmah da kažemo umesto NO ACTION koji se podrazumeva – ON DELETE i ON UPDATE i to se odnosi samo na operacije u referenciranoj.
- Kritične operacije u referencirajućoj relaciji su: unos nove torke i izmena vrednosti stranog ključa u referencirajućoj relaciji.
- Kada unosimo novu torku i stavljamo za vrednost stranog ključa nešto što ne sme da bude, što ne postoji u referenciranoj, ta operacija prosto mora da se zabrani, nema drugog načina, NO ACTION je jedino podrazumevano i moguće. Ali, kada radimo brisanje i referenciramo ili izmenu vrednosti obeležja u referenciranoj, onda možemo da biramo. I onda kada je kritična operacija DELETE idemo ON DELETE i postavljamo npr. CASCADE ili ON UPDATE kažemo SetDefault ili SetNull.

- Lista obeležja stranog ključa može da bude deklarisana. Znači, obeležja stranog ključa mogu eksplicitno biti zadata kao NOT NULL, ali i ne moraju.
- Što se tiče liste referenciranih obeležja koja su u referenciranoj, ako su obeležja ključa respektivno su NOT NULL, ali ne moraju da budu samo obeležja ključa. Može da bude alternativni (ekvivalentni) ključ. Moramo zadati tako da ima sve osobine primarnog ključa, ali da nije primarni. Osobine ključa su UNIQUE i NOT NULL – zadamo da je UNIQUE i NOT NULL i to nam je alternativni ključ.
- Osim ključeva, ova *ListaRefObeležja* mogu da budu obeležja nad kojim je definisana samo UNIQUE.
- Takođe, teoretski, po standardu ova *ListaRefObeležja* mogu da budu obeležja koja uopšte nisu ključ (čak ni alternativni) nego su samo domenski kompatibilna sa obeležjima stranog ključa. Ovo je teoretski, SUBP-ovi to baš i ne podržavaju.
- Kada je samo UNIQUE, to nije ograničenje referencijalnog integriteta.
- Ograničenje referencijalnog integriteta je samo specijalan slučaj zavisnosti održavanja gde je strani ključ zapravo primarni ključ u referenciranoj šemi relacije.
- Kritične operacije za FOREIGN KEY su:
    - Pokušaj upisa nove torke u referencirajuću relaciju
        - Jedina moguća aktivnost očuvanja konzistentnosti je NO ACTION
    - Pokušaj modifikacije vrednosti stranog ključa, datog putem *ListaObeležja*
        - Jedina moguća aktivnost očuvanja konzistentnosti je NO ACTION
    - Pokušaj brisanja postojeće torke iz referencirane relacije
        - Specifikacija aktivnosti očuvanja konzistentnosti baze podataka putem klauzule ON DELETE
        - ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}
        - Podrazumevana aktivnost je NO DELETE
    - Pokušaj modifikacije vrednosti obeležja, sadržanih u *ListaRefObeležja*
        - Specifikacija aktivnosti očuvanja konzistentnosti baze podataka putem klauzule ON UPDATE
        - ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}
        - Podrazumevana aktivnost je NO ACTION
    - **OVO DOSTA PITA!!!**
- Centralna provera konzistentnosti je na nivou šeme, na nivou SUBP-a, tu proverimo jednom i ne moramo posle na svim sledećim slojevima da radimo provere. Ako želimo, možemo da dupliramo provere. Ali, proveru tamo gde su nam podaci konzistentni (u bazi podataka) zaista niko ne može da naruši. Ako nam je provera na preduslove, sve aplikacije koje su napravljene tako da su se setili da provere sve što treba je okej, ali može nešto i da se zaboravi i da pravi problem u bazi. Primarni ključ i unique nećemo proveravati na srednjem sloju. Eventualno, možemo na poslednjem sloju – u user interfejsu da proverimo sa * da li je neko polje prazno ili ne. Kada zadamo prazno, onda se aplikacija obrati bazi, to neko vreme traje, baza tj. SUBP obradi taj naš zahtev. Kada za neko obeležje pošaljemo da je prazno (tj. da je NULL), onda SUBP proverava da li za to obeležje postoji ograničenje da je NOT NULL, ako postoji, a mi smo ostavili NULL, onda ide exception koji moramo da obradimo i da vidimo šta ćemo s njim, onda dobijemo grešku. Ali, krajni rezultat neće biti greška koju SUBP vrati kao svoj kod greške, već moramo mi svoj exception da prihvatimo i da u nekoj formi prijemčivoj za onog ko gleda, pročita šta se

desilo (npr. * koje podsećaju ljude da popune polja i tako se provera polja koja je obavezno popuniti vrši na user interfejsu. Svakako, to je duplirana provera, NOT NULL mora da se proverava i na nivou BP.

- Postoje neke stvari koje ne možemo uraditi deeklarativnim mehanizmima, pa tu moramo da koristimo proceduralne mehanizme.
- Proceduralni mehanizmi su:
    - Okidači (trigeri)
        - CREATE TRIGGER
    - Procedure i funkcije baze podataka
        - CREATE PROCEDURE
        - CREATE FUNCTION
    - Paketi baze podataka
        - CREATE PACKAGE
        - CREATE PACKAGE BODY
- Trigeri se mogu aktivirati na neki uslov. U okviru trigera imamo kod (aktivnost) koja se izvršava kada se nešto desi. Ono što može da se desi je neka od ovih DML operacija (unesi, izmeni, obriši). Zapravo, trigger se vezuje za tabelu, eventualno može da se veže i za pogled, aktivira se na neku od ovih operacija, pri čemu jedan triger može da se aktivira na IF ALL, npr. i kad uradiš unos i kad uradiš izmenu, aktiviraj triger.
- Jedan triger može biti vezan za više operacija, pri čemu jedan triger ne može biti vezan za više tabela – 1 triger = 1 tabela.
- Postoje i trigeri koji su na nivou čitave šeme baze. Može da se napravi triger kada radimo npr. SHUTDOWN baze i onda se izvrši neki kod kad god radi SHUTDOWN. Može se napraviti i triger na CONNECT, tako da se neki kod izvršava svaki put kada pokrenemo (kada se konektujemo na bazu).
- Jedino Oracle SUBP ima pakete.
- Specifikacija trigera:
    - Oblast aktiviranja
        - Tabela (ili pogled) nad kojim se definiše
    - Specifikacija operacija koje ga pokreću
    - Uslovi pod kojima se triger aktivira
    - Vreme aktiviranja
        - Neposredno pre ili posle same operacije
    - Frekvencija aktiviranja:
        - Jednom za celu operaciju, ili
        - Za svaku torku, koja je predmet operacije, pojedinačno
    - Aktivnost (procedura) koju triger treba da realizuje
- Operacija DELETE može da se odnosi na skup torki, brišemo pod nekim uslovom sve torke koje zadovoljavaju taj uslov se obrišu.
- Triger može da aktivira kod ako je tzv. *Statement level* – to znači da se aktivira samo jednom za sve torke koje su za brisanje. Imamo i drugu situaciju kada imamo za svaku torku koja podleže brisanju se po jednom izvrši ono što stoji u kodu trigera. To je *row level* triger (Microsoft nema *row level* triger, samo statement level, ali možemo da isimuliramo nekim mehanizmima – pomoćne tabele koje možemo da iskoristimo).

- Aktivnost – kod u telu trigera, ono čemu je triger namenjen i za šta služi.
- **Triger se aktivira automatski** – niti možemo da ga zaustavimo, niti možemo da ga pozovemo. On se aktivira na događaj za koji smo ga vezali. Postoji opcija da uradimo DISABLE TRIGGER ili da ga obrišemo, ali to nije to. Nema poziva trigera iz neke funkcije ili procedure, ili iz nekog drugog trigera. **Ne postoji naredba za pokretanje trigera!**
- Vreme okidanja:
    - BEFORE – neposredno pre akcije naredbe; ono što je u telu trigera se izvrši pre operacije koja izaziva triger
    - AFTER – neposredno nakon akcije naredbe; ono što je u telu trigera se izvršava nakon operacije koja ga je izazvala; Microsoft nema BEFORE, automatski je AFTER
    - INSTEAD OF – umesto same akcije naredbe (samo za poglede); ovo se izvršava umesto onoga što bi trebalo da se uradi i vezuje se za poglede.
- INSERT, UPDATE i DELETE su operacije koje izazivaju trigere, tj. trigeruju (okidaju) da se izvrši ovaj kod u telu trigera.
- Kada je u pitanju UPDATE, mi možemo da zadamo da se triger ne aktivira na bilo kakav UPDATE, nego na UPDATE samo određenih obeležja. Možemo da navedemo listu obeležja čiji bi pokušaj dodavanja izazvao triger.
- Row Level Trigger – FOR EACH ROW
- Triger se kreira sa CREATE [OR REPLACE] naredbom. OR REPLACE se dodaje da ne bismo morali da radimo DROP TRIGGER prilikom izmena definicije trigera. Naravno, to moramo da radimo pažljivo (kao i kod pogleda), da ne bismo pregazili nešto staro što se zove isto.
- Podrazumevani je statement level trigger, ako hoćemo row level trigger, moramo da dodamo FOR EACH ROW. FOR EACH ROW ima eventualno i uslov. To znači da može da se zada uslov pod kojim se triger eventualno aktivira, neće svaki put, nego zapravo će se aktivirati na odgovarajuću operaciju, ali će proveriti ovaj logički uslov pokretanja i ako je zadovoljen, onda će odraditi, a ako ne, onda će preskočiti – triger se aktivirao ali nije izvršio kod koji se nalazi u njemu.
- Imamo sekciju za deklaraciju i ide deo za izvršavanje i imamo exception deo.
- Kod Oracle-a imamo promenljive OLD i NEW, u tim promenljivama nam se nalaze stare i nove vrednosti. Ako hoćemo da obrišemo nešto, onda imamo staru vrednost, novu i nemamo. Ako hoćemo da insertujemo nešto, onda imamo novu, staru nemamo. Kada hoćemo da apdejtujemo, immao i staru i novu i možemo apsolutno odmah da se referenciramo, u tim promenljivama automatski imamo te vrednosti. Ovo se odnosi na FOR EACH ROW (Row Level Trigger) zato što ako imamo više torki koje podležu operaciji, ovo je promenljiva koja se odnosi i tipa je jedne torke.
- U zavisnosti od tabele za koju smo vezali trigger, tip podatka je ROW TYPE, ima sva obeležja, odnosno sve kolone odgovarajuće tabele, kao da u toj promenljivoj imamo čitavu jednu torku.
- Ako je skup torki rezultat nad kojima se vrši operacija, mi u NEW imamo novu vrednost.
- Nekada nam treba da se referenciramo, pa idemo OLD.NazivKolone. eventualno, u ovoj klauzuli možemo da promenimo, da damo sinonime za te promenljive.
- Microsoft nema ovo, on ima tabele INSERTED i DELETED i u tim tabelama drži torke koje su za brisanje ako je brisanje, torke koje su za unos ako je unos. Vevrovatno i Oracle ima te dve tabele, međutim mi im ne pristupamo kao tabelama, već koristimo ove promenljive. Mi ako bismo hteli da pristupimo svakoj torki iz tih tabela, mi onda moramo da imamo mehanizam koji se zove **kursor** i da mi ručno programiramo da prođemo svaku torku te tabele i da na taj način

simuliramo ovo FOR EACH ROW i da na taj način pristupimo starim vrednostima ili da pristupimo novim vrednostima onoga što se trenutno radi.

- Kod microsoft-a nemamo promenljivu ROW, nego svaki atribut (promenljivu) koji treba da dobije vrednost atributa iz baze moramo da deklarišemo tu promenljivu, svaku pojedinačno, ne čitavu torku. Ako tabela ima 10 kolona, 10 promenljivih koje moramo da deklarišemo tako da svaka od tih promenljivih dobije vrednost odgovarajuće kolone.
- Kod oracle-a stavimo jednu promenljivu tipa row i u jednom potezu pokupimo čitavu torku.
- SQL bez problema možemo da koristimo u okviru PL/SQL-a, s tim što bi SELECT naredba morala malo da doživi promenu, tamo je bilo SELECT iz nekih kolona iz neke tabele, pa ispis na ekran. Međutim, sada nam treba da rezultat SELECT-a smestimo u neku promenljivu. Jedino što mora da se promeni kod SELECT-a je da ide SELECT INTO promenljiva, pri čemu tu promenljivu deklarišemo u sekciji za deklaraciju i onda se nešto pokupi iz baze sa SELECT-om.
- Procedure i funkcije stoje na serveru, mogu da se pozivaju iz trigera, tu sa servera, iz drugih procedura i funkcija, a mogu da se pozivaju i iz srednjeg sloja (iz nekog progarmskog jezika možemo da pozivamo ove serverske procedure koje će samo da stoje na serveru i to i source i kompajlirani oblik).
- Neki SUBP-ovi imaju pakete, neki nemaju, Oracle ih ima. Paketom možemo da postignemo da imamo perzistentnost podataka na nivou jedne sesije.
- Globalne promenljive inače ne bi trebalo da se koriste, bolje je raditi prenos preko parametara. Međutim, nekada zaista ne može da se zaobiđe globalna promenljiva, odnosno promenljiva gde će više procesa da koristi tu promenljivu kao zajedničku, e ovde za to može da se iskoristi paket, pri čemu pri prvom startovanju (pokretanju) inicijalizuju se odgovarajuće promenljive i inicijalizuju se na odgovarajući način, kako nam treba, zavisi od situacije i onda u toku te sesije, koja traje dok ne uradimo SHUTDOWN (DISCONNECT) sa baze, ta promenljiva čuva ono što nam treba, a ti procesi koji figurišu i eventualno im treba vrednost te promenljive mogu da je menjaju u zavisnosti od toga šta dozvoljavamo i šta radimo i paket nam to dozvoljava.
- Paket je kontejner. Imamo procedure i funkcije koje javno mogu da se koriste, a imamo i procedure i funkcije koje mogu samo unutar paketa da se koriste i da ih koriste procedure koje su u tom paketu. Imamo i overloading unutar paketa, koji važi samo u tom paketu.
- Imamo i biznis logiku koja može da stoji na serveru.
- Kod FOR EACH ROW možemo postaviti uslov pod kojim će se eventualno aktivirati triger.

Serversko programiranje

- PL/SQL je proceduralno proširenje, proceduralni jezik. To je jezik za serversko programiranje i u SQL se ugrađuje bez ikakvih dodataka.
- := - naredba dodele vrednosti u trigeru.
- Kod PL/SQL-a imamo nešto što se zove anonimni (netipizovani) PL/SQL blok i imamo tipizovani PL/SQL blok (procedure, funkcije). Anonimni PL/SQL blok se ne čuva, dok procedure i funkcije imamo na serveru i source kod i kompajlirani kod, tako da mogu procedure i funkcije da se pozivaju i kasnije, dok anonimni PL/SQL kod koji kucamo se izvršava i u toku te sesije nam je tu, ali nije sačuvan. Pri završetku sesije prosto sve izgubimo.
- BEGIN i END su obavezni delovi, ako nemamo šta da deklarišemo, stavljamo {}

▶ Neimenovani (anonimni) blok

```
[DECLARE
   ...          -- Deklarativni deo bloka
]
BEGIN
   ...          -- Izvršni deo bloka
[EXCEPTION
   ...          -- Deo bloka za obradu izuzetaka
]
END;
```

- Deklarativni deo bloka služi za deklaraciju svega što nam je potrebno, ide BEGIN i END i imamo EXCEPTION sekciju ako obrađujemo izuzetke.

- DECLARE i EXCEPTION su opcioni delovi, ali bilo bi poželjno da imamo EXCEPTION deo u kodu, pošto izuzeci moraju da se obrađuju da ne bi došlo do nekog događaja koji bi srušio celu aplikaciju.

```
[DECLARE
    Deklarativni (neobavezni) deo programa:
      * deklaracija i inicijalizacija promenljivih
      * deklaracija i inicijalizacija konstanti
      * deklaracija tipova podataka
      * deklaracija kursora
      * deklaracija izuzetaka
      * deklaracija procedura i funkcija
]
BEGIN
    Izvršni (obavezni) deo programa:
      * Proceduralne naredbe
      * SQL naredbe
[EXCEPTION
    Deo za obradu izuzetaka (neobavezni):
      * WHEN <izuzetak> THEN <blok izvršnih naredbi>
]
END;
```

- Proceduralna proširenja koristimo kada nešto što pročitamo iz baze treba da smestimo u promenljivu. Da bismo smestili podatke u neku promenljivu, ta promenljiva treba da bude istog tipa podataka kao što je taj podatak ili neka kolona (obeležje) u BP. Tip podatka i dužinu možemo da proverimo u šemi.
- L_OznDeo Deo.OznDeo%TYPE; - L_OznDeo će biti istog tipa kao OznDeo iz tabele Deo. Na taj način uopšte ne moramo da idemo u šemu i gledamo koji je to tip podatka, već smo odmah deklarisali promenljivu koja je istog tipa podatka kao kolona u tabeli.
- Vrednost promenljive u koju smo smestili podatke koje smo pokupili iz baze možemo dalje proslediti korisniku ili nekoj drugoj proceduri.
- Klauzula INTO za smeštanje onog što smo pročitali u promenljivu.
- Ako želimo da se kod koji smo napisali sačuva, moramo da napravimo imenovani programski blok, pri čemu ovde i dalje imamo procedure i funkcije.

Zaglavlje_programskog_bloka

```
IS | AS
[  ...          -- Deklarativni deo bloka
]
BEGIN
   ...          -- Izvršni deo bloka
[EXCEPTION
   ...          -- Deo bloka za obradu izuzetaka
]
END;
```

- Vrste procedura i funkcija (imenovanih programskih blokova):

  - Serverska procedura ili funkcija
    - Služe da ih pozivamo iz drugih procedura ili funkcija, trigera, sa srednjeg sloja; to je i izvorni i kompajlirani kod
  - Lokalna procedura ili funkcija
    - One koje su pozivane unutar PL/SQL bloka (i deklaracija i poziv su unutar PL/SQL bloka, nisu smeštene na serveru tako da možemo posebno da ih pozivamo
  - Klijentska procedura ili funkcija
    - Ono što ide na srednji sloj

- Parametar procedure može da bude IN, OUT ili IN OUT. Imamo nešto što se zove default – kada zadamo da je neki parametar sa difoltnom vrednošću, onda pri pozivu procedure i funkcije možemo da izostavimo taj parametar.
- Kada je u pitanju IN parametar, to su samo ulazni parametri i njihove vrednosti ne mogu da se menjaju unutar procedure. Prenose se po referenci.
- OUT parametar se prenosi po vrednosti, to je samo izlazni parametar koji procedura vraća kao rezultat. Ako baš želimo da ga prenesemo po referenci, onda možemo da dodamo opciju NO COPY
- IN OUT – i ulazni i izlazni parametar, prenosi se kao samo izlazni – po vrednosti. Može da uđe kao ulazni parametar, da mu promenimo vrednost i da izađe kao rezultat. Dodavanje opcije NO COPY važi i za IN OUT parametre.
- Po vrednosti – moramo da kopiramo u objekat, po referenci – pokazivač.
- Kod funkcije imamo povratnu vrednost i ako imamo potrebu da imamo neke silne izlazne parametre, onda smo promašili, onda ne treba da biramo funkciju, nego proceduru. Ako imamo povratnu vrednost, onda je to funkcija i ovi in-ovi, out-ovi, ne savetuje se da imamo IN OUT parametre.
- Promenljiva može biti tipa neke kolone u tabeli, ali možemo i deklarisati neku promenljivu i onda deklarisati sledeću gde ćemo reći da će tip te promenljive biti isti kao tip promenljive koju smo već deklarisali gore.
- Ne smeju se promenljive nazivati nazivima kolona ili nazivima samih tabela!
- U direktnom načinu upotrebe select naredbe, ona mora da vrati samo 1 red, u suprotnom, treba da koristimo kursor.
- Kada imamo potrebu da vraćamo više od jednog reda, koristimo nešto što se zove kursor. Kursor može biti implicitni ili eksplicitni.
- Implicitni kursor – svaka DML naredba izazove taj implicitni kursor, time se bavi SUBP – on otvori kursorsko područje, DML naredba se izvrši, određeni broj torki može da bude rezultat te DML naredbe i SUBP sam i zatvori to kursorsko područje. Postoje metode koje možemo da koristimo da bismo videli šta se dešava u tom kursoru u toku neke DML naredbe.
- Taj implicitni kursor se zove SQL, pri čemu kada kažemo SQL, pri čemu kada kažemo SQL%FOUND, otvori se metoda FOUND koja će da vrati odgovarajuću vrednost TRUE ili FALSE, u zavisnosti od toga da li je rezultat te primenjene naredbe bio jedna ili više torki – ako je TRUE, onda je bar jedna torka bila rezultat te poslednje DML operacije, inače je FALSE, a imamo i obrnuti poziv.
- SQL%NOTFOUND – biće TRUE ako nije bilo torki koje su bile rezultat te poslednje DML naredbe.
- Možemo pozvati i metodu SQL%ROWCOUNT koja će nam dati broj torki koje jesu bile rezultat (koje su bile predmet te poslednje DML naredbe)
- SQL%ISOPEN – uvek će biti FALSE, jer kad dođe trenutak kada možemo da pitamo ovo, SUBP je već implicitno zatvorio kursor. Ova metoda ima smisla kod eksplicitnog kursora.
- DMBS_OUTPUT.PUT_LINE – naredba za ispitivanje rezultata
- Implicitni kursor koristimo kada želimo da znamo da li je neki update uradio nešto ili ne, ali najčešće ga koristimo da ne bismo imali one EXCEPTION-e za NO DATA FOUND. Ako nije našao podatke nad kojima će nešto da izvrši, onda imamo EXCEPTION, a sa ovim možemo da predupredimo tako što ćemo mi da pitamo da li je ROWCOUNT bio 0, ako je ROWCOUNT bio 0 onda radi jedno, ako nije onda radi nešto drugo i nemamo problema sa EXCEPTION-ima.

- Imamo i eksplicitne kursore, koje mi deklarišemo, koristimo u situacijama kada ona SELECT naredba može da radi kao rezultat više od jedne torke. U kursoru možemo imati više naredbi SQL-a i PL/SQL-a gde će se nešto raditi sa tim torkama koje budu rezultat nekog upita, rezultat, odnosno sve torke tog upita su u okviru kursorskog područja koje pre svega treba da deklarišemo i ide ključna reč CURSOR *NazivKursora*, kursor može imati parametre, što će značiti da ovaj SELECT tj. naredba koja ide u okviru definicije kursora će biti parametrizovana. Onda moramo eksplicitno da radimo otvaranje kursora. Kursor (naredbe u okviru kursora) će vratiti više torki i kursor je taj koji vodi računa o tim torkama, a mi ćemo sa FETCH da idemo kao da idemo sa FOR od 1 do broja torki koliko ih ima u kursoru i zapravo ćemo za svaku tu torku nešto da radimo (ako ništa, da je ispisujemo), tako da ovaj FETCH kreće od prve torke rezultata SELECT naredbe u kursoru i ide redom (imamo loop – petlju koja vrti 1 po 1) dok ne dođemo do kraja tj. do poslednje torke kursora gde imamo izlaz iz te petlje. Moramo eksplicitno da uradimo CLOSE (da zatvorimo kursor).
- U kursoru kao da imamo tabelicu koja čuva skup torki koje je našao.
- Kada god imamo SELECT naredbu gde znamo da rezultat neće biti jedna naredba, moramo da koristimo kursor.
- Zgodna stvar je i kursor sa parametrima – gde mi deklairišemo jedan kursor, ali možemo ga pozivati više puta i koristiti, čime ćemo mu prosleđivati drugačije vrednosti.
- Još jedna zgodna stvar je da možemo da preuzmemo i deklarišemo promenljivu koja će da preuzme čitavu torku iz tabele i mi sada preko naziva te promenljive možemo da pristupimo bilo kojoj koloni.
- Zapravo, parametrizovana je ova SELECT naredba gde možemo da zadajemo različite vrednosti.
- Možemo da imamo kursor u kursoru.
- Složeni tipovi podataka:
    - PL/SQL tip sloga
    - PL/SQL tip kolekcije
        - INDEX BY tables – indeksirane tabele
        - Nested tables – „ugnježdene“ tabele
        - VARRAY – nizovi ograničene maksimalne dužine
- Tipovi slogova liče na klasu bez metoda. Ključna reč TYPE. Prvo deklarišemo tip podatka, a onda deklarišemo promenljive koje će biti ovog tipa. Tip je slogovski, pri čemu mi onda zadajemo polja, slog može imati više polja. Svako polje ima naziv polja, tip polja – koji može biti eksplicitno zadat neki primitivni tip, kao što je INTEGER, STRING, BOOLEAN, možemo da zadamo %TYPE neke već zadate promenljive, pa %TYPE neke kolone iz tabele ili čak može biti ROWTYPE. Zapravo, ROWTYPE nam je već slogovska promenljiva, samo što je ista kao samo što je ista kao čitav red u tabeli. Svako to polje po specifikaciji moće biti zadato da bude NOT NULL (ne sme da ima NULL vrednost) i eventualno može da ima zadatu DEFAULT vrednost. Druga stavka je da deklarišemo promenljivu koja će biti ovog tipa.
- Kursorska FOR petlja – način da se oslobodimo OPEN/CLOSE, može biti parametrizovan. Moramo deklarisati kursor na početku, imamo automatsko otvaranje, zatvaranje i preuzimanje torki iz kursora. Ne moramo to eksplicitno da radimo – nema OPEN, CLOSE i FETCH. Tekući red nemamo u deklaraciji, kursorska FOR petlja nam i to skraćuje, da ne deklarišemo posebnu promenljivu u kojoj ćemo da radimo FETCH svake torke iz kursora. Ovo će da uradi open i fetch, na kraju će vratiti i CLOSE.

- Pri pozivu kursora prosleđujemo parametre. Ako nismo deklarisali kursor u sekciji za deklaraciju, onda on ne može biti parametrizovan. Promenljiva koja prihvata torku iz kursora ne mora da bude prethodno deklarisana.
- Trigerom možemo da zabranimo izmenu vrednosti ključa. SUBP nam automatski ne zabranjuje promenu ključa, ali ako želimo da zabranimo izmenu vrednosti ključa, možemo da realizujemo to trigerom.
- Žurnal tabela – posebna tabela u koju se beleže, neka vrsta log-tabele zapravo, beležimo sve operacije koje se vrše nad tabelom. Npr. aktivira se triger kad god uradimo insert, aktivira se triger i kada se aktivira triger, ta nova torka bude unešena u tabelu, ali zbog trigera ta torka takođe bude unešena i u log tabelu (žurnal tabelu). Ako uradimo brisanje nad tabelom, brisanje će se uraditi, ali se triger aktivira pri brisanju i ide u žurnal tabelu stara torka koju brišemo, tj. ne gubimo je, u žurnal tabeli će ostati ta torka koja nam je bila, brišemo je iz oroginalne tabele, ali u žurnal tabeli stoji torka za brisanje. Eventualno, žurnal tabela treba da sadrži dodatnu kolonu u kojoj će biti napisan unos. Žurnal tabela će imati isto torke sa vrednostima odgovarajućih kolona, ali treba da znamo je li to uneto ili je obrisano, tako da u žurnal tabelu ide i vrsta operacije koja je izvršena nad torkom. Bilo bi dobro da žurnal tabela ima datum i vreme kada je neka operacija izvršena, kao kod klasične log tabele. SUBP u pozadini loguje.