SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Marko Banek

OBLIKOVANJE SKLADIŠTA PODATAKA IZ POLUSTRUKTURIRANIH IZVORA

MAGISTARSKI RAD

Zagreb, 2005.

Magistarski rad je izrađen na Zavodu za telekomunikacije Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Mentor: Prof. dr. sc. Zoran Skočir, dipl. ing.
FER, Sveučilište u Zagrebu

Magistarski rad ima 165 stranica.

Magistarski rad broj:

Povjerenstvo za ocjenu magistarskog rada:

1. Prof. dr. sc. Vjekoslav Sinković
   FER, Sveučilište u Zagrebu
2. Prof. dr. sc. Zoran Skočir
   FER, Sveučilište u Zagrebu
3. Prof. dr. sc. Katarina Ćurko
   Ekonomski fakultet, Sveučilište u Zagrebu

Povjerenstvo za obranu magistarskog rada:

1. Prof. dr. sc. Vjekoslav Sinković
   FER, Sveučilište u Zagrebu
2. Prof. dr. sc. Zoran Skočir
   FER, Sveučilište u Zagrebu
3. Prof. dr. sc. Katarina Ćurko
   Ekonomski fakultet, Sveučilište u Zagrebu

Datum obrane magistarskog rada: 1. travnja 2005.

Osobito sam zahvalan svojem mentoru, prof. dr. sc. Zoranu Skočiru, na znanju koje je podijelio sa mnom tijekom provođenja istraživanja i izrade rada te savjetima i prijedlozima. Zahvaljujem mu na potpori i poticanju mojeg znanstveno-istraživačkog rada tijekom cjelokupnog dodiplomskog i poslijediplomskog studija.

Za mnoge poticajne diskusije i sate provedene u našim zajedničkim istraživanjima najiskrenije zahvaljujem dr. sc. Borisu Vrdoljaku. Ovaj rad predstavlja izravni nastavak istraživanja opisanih u njegovoj doktorskoj disertaciji.

Zahvaljujem prof. dr. Stefanu Rizziju, Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna na brojnim savjetima.

Zahvaljujem svim kolegama sa Zavoda za osnove elektrotehnike i električka mjerenja i Zavoda za telekomunikacije Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu na podršci te stimulativnoj i ugodnoj radnoj okolini.

Naposljetku, zahvaljujem svojoj djevojci, obitelji i prijateljima na potpori i razumijevanju.

Sadržaj

Uvod

Uz ponudu kvalitetnih proizvoda ili usluga za uspješno poslovanje svakog poslovnog subjekta važan je i sustav poslovnog odlučivanja. Odluke se donose na temelju složenih analiza poslovanja koje se provode nad podacima pohranjenim u informacijskom sustavu poduzeća. Temelj sustava za poslovnu analizu čine skladišta podataka, baze podataka oblikovane upravo u svrhu potpore poslovnom odlučivanju. Skladišta podataka sadrže detaljne podatke o svim segmentima poslovanja, prikupljane kroz dugotrajno vremensko razdoblje koje se može protezati i na cijelo desetljeće. Svi relevantni podaci koji ulaze u informacijski sustav poduzeća pohranjuju se i u skladište podataka za potrebe poslovne analize.

Razvoj Interneta kao globalne komunikacijske mreže omogućio je da poduzeća međusobno posluju elektroničkim putem. Na taj način troškovi poslovanja su se smanjili, a sam proces postao jednostavniji, brži i efikasniji. Nakon što se elektronički oblik poslovanja među tvrtkama (*business-to-business*, B2B) pokazao vrlo uspješnim, započelo je njegovo uvođenje u državnu upravu, a istodobno su ga brojne tvrtke odlučile ponuditi svojim individualnim korisnicima odnosno potrošačima (*business-to-customer*, B2C). Na taj je način elektroničko poslovanje postalo uobičajeni dio svakodnevnog života.

Integracija podataka dobivenih elektroničkim poslovanjem u sustav skladištenja podataka ima odlučujuću ulogu u težnji da se donositeljima poslovnih odluka pravodobno podastru kvalitetne informacije o poslovanju. Za razmjenu podataka u elektroničkom poslovanju koriste se polustrukturirani podaci. XML (*eXtensible Markup Language*) je danas u praksi prihvaćen kao standardni format za zapisivanje polustrukturiranih podataka. Stoga integracija podataka o elektroničkom poslovanju zahtijeva pronalaženje načela za integraciju polustrukturiranih podataka, odnosno XML dokumenata u skladište.

Problemi pohrane polustrukturiranih podataka u skladište podataka predmet su zajedničkog znanstvenog istraživanja Zavoda za telekomunikacije Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu i zavoda DEIS (Dipartimento di Elettronica, Informatica e Sistemistica) Sveučilišta u Bologni od godine 2001. Rezultat istraživanja, u kojima je sudjelovao i autor ovog rada, je metodologija integracije podataka u formatu XML u skladište podataka. Skladište podataka oblikuje se na temelju strukture XML dokumenata za čiju je pohranu namijenjeno. Metodologija sadrži standardne korake konceptualnog i logičkog oblikovanja skladišta, oblikovanja procesa izvlačenja, transformacije i učitavanja podataka te fizičkog oblikovanja skladišta. Međutim, zbog izuzetno specifičnih svojstava XML-a, odnosno polustrukturiranih podataka općenito, svaki od opisanih koraka u potpunosti se razlikuje od analognih koraka u izgradnji skladišta za podatke iz transakcijskih baza.

Glavni zadatak ovog rada je izrada programskog sustava koji pruža potporu za izravno oblikovanje skladišta podataka iz polustrukturiranih izvora (što se u praksi odnosi na XML izvore). Programski sustav omogućuje verifikaciju iznesenih načela oblikovanja skladišta. Načela oblikovanja opisana su na jednostavnim primjerima. Prikazani konceptualni model prilagođen je relacijskim bazama podataka. Na kraju rada načela oblikovanja skladišta podataka iz polustrukturiranih izvora ispituju se na studijskom primjeru realnih XML dokumenata koje u elektroničkom poslovanju koristi tvrtka Agrokor.

U prvom poglavlju rada izneseni su razlozi koji su doveli do razvoja koncepta skladištenja podataka. Opisani su osnovni pojmovi vezani uz skladište podataka te navedeni podatkovni modeli skladišta na različitim razinama apstrakcije. Objašnjena su i osnovna načela izvedbe skladišta podataka u relacijskoj tehnologiji. Drugo poglavlje opisuje polustrukturirane podatke i format XML. Usporedno su prikazana dva tipa i gramatike za XML dokumente: DTD i XML Schema. U trećem poglavlju objašnjena je ideja o izravnoj pohrani polustrukturiranih podataka u skladište te prikazana metodologija za izravno oblikovanje skladišta podataka iz polustrukturiranih izvora. Četvrto poglavlje detaljno opisuje programski sustav u kojem su implementirana načela konceptualnog i logičkog oblikovanja skladišta iz polustrukturiranih izvora. Prikazane su dvije osnovne verzije sustava: aplikacija u *Javi* i *Java Applet* (*web*-aplikacija) te njihove komponente. U petom, šestom i sedmom poglavlju detaljno se opisuju pojedini koraci metodologije oblikovanja skladišta. Peto poglavlje opisuje uvodne korake u procesu oblikovanja vezane uz pohranu XML dokumenata koji će se koristiti u procesu. Šesto poglavlje objašnjava sve korake konceptualnog oblikovanja i specifičnosti njihove izvedbe u programskom sustavu. Konceptualni model najprije je definiran matematički, a potom naveden algoritam za njegovu izradu. Svi koraci algoritma su detaljno opisani. Sedmo poglavlje opisuje prilagodbu konceptualnog modela relacijskoj bazi podataka. Najprije se općenito iznose načela logičkog oblikovanja za relacijske baze podataka, a potom konkretni problemi pretvorbi tipova podataka. Za pokazane karakteristične primjere stvaraju se tablice u relacijskoj bazi. Posljednje, osmo poglavlje daje studijski primjer izravnog oblikovanja skladišta podataka na realnom dokumentu narudžbe koji koristi tvrtka Agrokor. Naglašene su razlike između jednostavnih dokumenata prikazanih u ranijim poglavljima i stvarnog dokumenta čija struktura je znatno složenija i redundantna, a oblikovanje skladišta kompliciranije. Studijski primjer, međutim, dokazuje da se ranije iskazana načela uspješno mogu primijeniti i na realne dokumente.

1. Skladištenje podataka

1.1. Osnovne značajke skladišta podataka

1.1.1. Uzroci razvoja skladišta podataka

Za uspješno poslovanje poduzeća nužan je dobar tržišni plasman roba ili usluga koje ono nudi, a istodobno i njegov kvalitetan unutarnji ustroj. Uprava poduzeća svaki dan donosi kratkoročne odluke vezane uz djelovanje poduzeća na tržištu: količinu i cijenu plasirane robe ili usluge te akcijske prodaje i popuste. Odluke na srednji odnosno dugi rok odnose se na izbor dobavljača pojedine robe uz ostvarivanje najpovoljnije cijene te na ustroj poduzeća po odjelima (ukoliko se radi o velikom poduzeću). Poslovne odluke imaju konačnu svrhu povećati dobit (profit) poduzeća odnosno ojačati njegovu ulogu na tržištu u odnosu na konkurenciju.

Menadžeri poduzeća donose odluke o cijenama i plasiranju robe na tržište temeljem svakodnevnih detaljnih analiza poslovanja i predviđanja kretanja na tržištu. Za odluke o izboru dobavljača i analizu uspješnosti unutarnje organizacije poduzeća provode se dugotrajna i detaljna ispitivanja i obrađuje izuzetno velika količina detaljnih podataka. Ispitivanja uključuju zbrajanje ili prebrojavanje vrlo velikog broja zapisa iz baza podataka uz računanje prosječnih, maksimalnih i minimalnih vrijednosti u skupu podataka. Za opisane analize redovito su potrebni podaci prikupljeni kroz vrlo dugo vremensko razdoblje, koje u najkraćem slučaju iznosi nekoliko mjeseci, a često i više godina. Njihov obujam iznosi više stotina gigabajta, nekoliko terabajta, pa čak i desetaka terabajta (za vrlo velika poduzeća kao što su davatelji telekomunikacijskih usluga).

Podaci o poslovanju (naručivanju i prodaji robe, financijskim transakcijama, stanju zaliha) svakodnevno se pohranjuju u baze podataka poduzeća. U takve baze podataka svakodnevno se dodaju novi zapisi, te brišu ili mijenjaju postojeći. Dnevne transakcije u načelu obuhvaćaju do nekoliko desetaka tisuća zapisa. Ovakve baze podataka, koje se nazivaju transakcijskima odnosno operacijskima, u pravilu su izvedene u relacijskom modelu podataka. U rjeđem broju slučajeva izvedene su u starijem hijerarhijskom ili mrežnom modelu. Transakcijske baze izvedene u relacijskom modelu sastoje se od velikog broja relacija čije su sheme u trećoj normalnoj formi, međusobno povezane mehanizmom primarnih i stranih ključeva [Cod70, SMV00]. Redovni korisnici transakcijske baze (službenici poduzeća) koriste samo nekoliko njenih relacija. Oni podatke iz baze dohvaćaju i u bazu spremaju pomoću standardnih sučelja koja su za njih napravili administratori baze.

Analiza poslovanja poduzeća zahtijeva da menadžer ili analitičar ima pregled nad svim podacima skupljenim u jednoj ili više transakcijskih baza poduzeća. Struktura transakcijskih baza izvedenih relacijski ne odgovara potrebama poslovne analize. Budući da se u transakcijske baze podataka pohranjuju svi podaci koji uđu u informacijski sustav poduzeća, znatni dio njih je beskoristan za poslovne analize, dok je dio podataka izražen u formatu neprikladnom za zbrajanje i druge matematičke operacije vezane uz poslovne analize.

Zbog toga su, počevši od osamdesetih godina dvadesetog stoljeća, u velikim poduzećima uz uobičajene transakcijske baze izgrađivane i posebne baze podataka namijenjene isključivo za potporu složenim poslovnim analizama. Takve baze, nazvane skladištima podataka (engl. *data warehouses*), zajedno sa svojom

specifičnom programskom podrškom čine sustav skladištenja podataka (engl. *data warehousing system*). Sustav skladištenja podataka zasnovan je na modelu podataka drukčijem od relacijskog modela: to je višedimenzijski model podataka koji će detaljno biti opisan u poglavlju 1.2. Višedimenzijski model analitičarima i rukovodstvu poduzeća daje intuitivan i jednostavan pregled nad podacima o poslovanju.

1.1.2. Definicija skladišta podataka

Najpoznatiju definiciju skladišta podataka dao je W. H. Inmon [Inm96]. Po toj definiciji, skladište podataka je:

- tematski orijentirani,
- integrirani,
- u vremenu različiti (*time-variant*),
- postojani

skup podataka koji služi kao potpora u procesu odlučivanja. U nastavku će biti detaljno objašnjena sva četiri ključna svojstva skladišta podataka.

Tematska orijentiranost. Skladište podataka je tematski orijentirano budući da je unaprijed, već prilikom projektiranja, vezano uz aktivnosti od ključnog značenja za poduzeće. Djelatnost poduzeća može biti nabava i prodaja robe, uslužna djelatnost, bankovna ili posrednička djelatnost, osiguranje, najam stanova ili poslovnih prostora itd. Dok transakcijska baza samo automatizira poslovne procese (nabava i isporuka robe, plaćanje kreditnom karticom, osiguranje vozila) skladište je orijentirano na one dijelove procesa koji su zanimljivi za analizu (troškovi nabave i isporuke robe, kašnjenje isporuke, zarada od prodaje robe). Oblikovanje skladišta ne obuhvaća procese nad podacima, važne u oblikovanju transakcijskih baza. U okruženju skladišta podataka bitno je isključivo oblikovanje podataka odnosno baze podataka.

Integriranost. Podaci u skladištu podataka najčešće se dobivaju iz više različitih izvora. Skladište se radi na razini cijelog poduzeća, dok svaki odjel poduzeća u pravilu posjeduje zasebnu transakcijsku bazu. Istovjetni podaci u svakoj od ovih baza mogu biti zapisani u drukčijem formatu. U skladištu moraju imati jedinstven format kako bi se omogućila njihova konzistentnost. Osobitu pažnju treba posvetiti na nazivima i ključevima podatkovnih elemenata, mjernim jedinicama za pojedine varijable i strukturi šifriranja (zanimljive primjere nudi [Vrd99]).

Vremenska različitost. Transakcijske baze sadrže podatke koji su točni u trenutku pristupa. Kasnije se podaci mijenjaju ili brišu, a na njihovo mjesto dolaze nove vrijednosti. Stari podaci arhiviraju se najčešće samo djelomično (nemoguće je izvršiti potpunu rekonstrukciju ranijeg stanja poslovanja), na rok ne dulji od nekoliko mjeseci, nakon čega se u potpunosti uklanjaju.

Skladište podataka po svojoj biti predstavlja skup vremenskih snimaka, od kojih je svaki bio aktualan u određenom trenutku. Za razliku od transakcijskih baza podataka, koje arhiviraju podatke najdulje nekoliko mjeseci, skladišta podataka sadrže cjelokupne podatke o poslovanju poduzeća stare 5, 10 pa i 20 godina.

Postojanost. Nakon što su podaci uneseni u skladište, oni se ne mijenjaju ako je unos napravljen ispravno. Nad podacima u transakcijskoj bazi, uz temeljnu operaciju odgovaranja na upit tj. čitanja podataka, vrše se i operacije unošenja, izmjene,

brisanja. Sve operacije dohvaćaju redom zapis po zapis, a ukupno se odnose najčešće na nekoliko desetaka ili stotina zapisa. U skladištu podataka uz operaciju čitanja postoji samo operacija inicijalnog učitavanja, koja se obavlja periodički. Budući da je podatke nemoguće promijeniti ili brisati, nestaje opasnost od nekonzistentnosti podataka odnosno anomalije izmjene ili brisanja podataka [SMV00]. Time se gubi potreba za pohranom podataka u vidu relacijskih shema visokog stupnja normaliziranosti i otvara mogućnost korištenja drukčijeg, višedimenzijskog, modela podataka.

1.1.3. Usporedba transakcijskih baza i skladišta podataka

Transakcijske baze podataka omogućuju vođenje svakodnevnih poslovnih aktivnosti pružajući uvid u trenutno stanje poslova. Sve dnevne operacije obrađuju se u transakcijskim bazama podataka. Skladište podataka orijentirano je na poslovne analize, izvještavanje o tijeku poslovanja kroz dugotrajno vremensko razdoblje. Transakcijsku bazu podataka, koja djeluje na dnevnoj, operativnoj razini vođenja poduzeća, koriste službenici dok skladište podataka služi taktičkom i strateškom vođenju poslovanja i koriste ga menadžeri poduzeća i analitičari. Skladište podataka koristi se kako bi se iz ogromne količine podataka kojom raspolaže informacijski sustav poduzeća dobile informacije: nove, dotad nepoznate (ili nepotvrđene) spoznaje koje opisuju poslovanje, a na temelju kojih se donose poslovne odluke [SMV00].

Službenik u radu s transakcijskom bazom podataka koristi aplikacije izrađene od strane administratora transakcijske baze postavljajući ustvari samo nekoliko vrsta upita nad bazom. Administrator unaprijed izrađuje aplikaciju budući da je struktura standardnog službenikovog upita nad transakcijskom bazom unaprijed poznata i fiksna te se tiče užeg područja poslovanja vezanog uz službenikovo radno mjesto. Kod skladišta podataka menadžer ili analitičar mora imati pregled nad cjelokupnim poslovnim procesom. Točno područje i struktura upita nikako ne mogu biti unaprijed poznati. Tijekom rada sa skladištem podataka upit se najčešće formira na osnovi rezultata prethodnog upita (tzv. *ad-hoc* upit). Upiti se međusobno značajno razlikuju po svojoj strukturi.

Transakcijska baza sadrži detaljne podatke o poslovanju. Budući da se prilikom poslovne analize podaci iz skladišta podataka uglavnom sumiraju, skladište podataka uz detaljne podatke sadrži i unaprijed sumirane podatke radi skraćivanja vremena obrade pojedinih zahtjeva. Transakcijska baza po svojoj biti sadrži aktualne podatke, dok obrađene sumarne ili arhivske podatke ne sadrži.

Kako transakcijska baza pohranjuje trenutne vrijednosti podataka podložne izmjeni vrlo je važno da se izmjena podataka izvrši bez odlaganja i u najkraćem mogućem vremenu. U suprotnom bi se iz baze čitale neodgovarajuće i pogrešne vrijednosti. Za transakcijske baze podataka ključna je njihova raspoloživost i brzina. Zbog toga se izmjene moraju vršiti polje po polje, a transakcija predstavlja osnovnu jedinicu obrade. Skladište podataka obavlja operacije nad nepromjenjivim podacima pa raspoloživost baze ne predstavlja prioritet za njegov rad.

Opisane razlike između transakcijskih baza podataka i skladišta podataka pregledno su navedene u tablici 1.1.

Transakcijske baze i skladišta podataka postavljaju se na odvojenim računalima. Ključni razlog odvajanja skladišta podataka na zasebno računalo u odnosu na sve transakcijske baze podataka je opasnost narušavanja raspoloživosti transakcijskih baza podataka. Ako bi se skladište podataka postavilo na isto računalo kao i

transakcijska baza, značajni dio procesorske moći računala trošio bi se na obradu upita nad skladištem podataka koji su po svojoj prirodi zahtijevaju znatno veći broj pristupa bazi i znatno su dugotrajniji.

Tablica 1.1. Razlike između transakcijske baze podataka i skladišta podataka

	Transakcijska baza podataka	**Skladište podataka**
Svrha	Vođenje poslovnih aktivnosti (dnevne operacije)	Izvještavanje o poslovanju, analiza poslovanja
Korisnici	Operativna razina (službenici)	Taktička i strateška razina (menadžeri, analitičari)
Sadržaj podataka	Trenutni podaci	Povijesni podaci
Detaljnost podataka	Detaljni podaci	Detaljni i sumarni podaci
Jedinica obrade	Transakcija	Upit
Interakcija korisnika i način obrade	Predodređena interakcija; strukturirani zahtjevi koji ponavljaju	Interakcija *ad-hoc*; nestrukturirani zahtjevi
Trajnost podataka	Vrlo promjenjivi podaci	Nepromjenjivi podaci
Način pristupa podacima	Čitanje i pisanje	Čitanje
Količina zapisa kojima se pristupa	Desecima ili stotinama	Stotinama tisuća, milijunima
Fokus	Pohrana podataka	Dobivanje informacija

1.1.4. Zrnatost u skladištu podataka

Zrnatost ili granularnost opisuje razinu detaljnosti odnosno sumiranosti podatkovnih zapisa u skladištu podataka. Najdetaljniji, najusitnjeniji podaci opisuju najnižu razinu zrnatosti. Podaci na najvišoj razini zrnatosti najviše su sumirani, okrupnjeni.

S obzirom na razinu zrnatosti podatke u skladištu može se podijeliti na tekuće i starije detaljne podatke, blago i jako sumirane podatke te meta-podatke [Vrd99].

Na najnižoj su razini zrnatosti *tekući detaljni podaci*, stari 2-5 godina. Dolaze iz operacijskog okruženja i njihova je zrnatost jednaka podacima u transakcijskom (operacijskom) okruženju. Oni se koriste za sumiranje podataka, a pohranjeni su na disku računala na kojem je skladište podataka.

Stariji detaljni podaci su raniji tekući detaljni podaci koji su zbog procesa starenja izbačeni s diska i pohranjeni na druge medije.

Blago sumirani podaci nastaju sumiranjem tekućih detaljnih podataka. Često je potrebno različitim korisnicima u poduzeću isporučiti blago sumirane podatke na različitim stupnjevima sumiranosti odnosno zrnatosti.

Jako sumirani podaci na najvišoj su razini zrnatosti i vrlo su kompaktni. Nalaze se pohranjeni na disku, jednako kao i blago sumirani podaci, kako bi im pristup bio čim brži.

Meta-podaci su podaci o drugim podacima. Oni opisuju način sumiranja podataka i transformiranje podataka pri njihovom unošenju u skladište iz operacijskog okruženja.

Jednostavnost pristupa podacima uvjetovana je njihovom zrnatošću. Podacima na visokoj razini zrnatosti (sumiranosti) lagano se pristupa jer njihova obrada troši malo

procesorskih resursa. Osobito je bitno dobro projektirati razinu blago sumiranih podataka. Njihova previsoka zrnatost zahtijeva previše obrade na razini detaljnih podataka što povlači velik potrošak resursa. Preniska zrnatost otežava obradu pri upitima jer je u tom slučaju potrebno zbrajati same blago sumirane podatke.

Zbrajanje se podataka iz tekućih detaljnih podataka izvodi redovito (periodički) ili dinamički, pri postavljenom upitu, a može se izvesti i na samom početku, pri unosu podataka iz operacijskog okruženja.

1.1.5. Učitavanje podataka u skladište i njihovo korištenje

Nakon što je skladište podataka uspješno oblikovano i pušteno u rad, aktivnosti vezane uz podatke u njemu mogu se svesti u tri različita procesa [Vrd99]:

1. izvlačenje, transformaciju i učitavanje podataka
2. spremanje podataka i upravljanje podacima
3. korištenje podataka u procesu poslovnog odlučivanja.

Izvlačenje, transformacija i učitavanje podataka. Izvlačenje, transformacija i učitavanje podataka (engl. *Extraction, Transformation and Loading*, ETL) predstavljaju aktivnosti vezane uz popunjavanje skladišta podacima iz transakcijskih baza podataka i drugih izvora. One se definiraju tijekom projektiranja skladišta podataka.

Izvlačenje podataka podrazumijeva detaljnu analizu svih izvora podataka. Skladište podataka u velikoj organizaciji najčešće se puni iz većeg broja transakcijskih baza podataka. Dodatni izvori podataka mogu biti starije baze podataka izvedene u modelu podataka koji nije relacijski, hijerarhijskom ili mrežnom modelu. To mogu biti i tekstualne datoteke, najčešće pisane u formatu koji odgovara prirodnom ljudskom jeziku pa su time razumljive čovjeku (engl. *human-readable*) ili pak datoteke za praćenje rada poslužitelja (tzv. *log files*, *log*-datoteke) u koje se zapisuju detaljni podaci o svakom zahtjevu koji dolazi na poslužitelj. Grupirajući podatke iz različitih izvora, izuzetno je bitno paziti da se odnose na isto vremensko razdoblje.

S obzirom da podaci u skladištu moraju imati jedinstven format, nad podacima izvučenim iz heterogenih izvora bit će potrebno izvršiti transformaciju. Transformacija uključuje primjenu matematičkih funkcija, funkcija za manipulaciju datumskim vrijednostima te manipulaciju nizovima znakova (tekstom).

Kod učitavanja podataka razlikuje se početno učitavanje podataka u prazno skladište od kasnijih periodičkih osvježavanja skladišta novonastalim podacima. Početnim učitavanjem obuhvaćena je znatno veća količina podataka. Periodičkim učitavanjem dodaju se novi podaci, aktualne vrijednosti parametara u razdoblju od posljednjeg učitavanja ili u isto vrijeme nastali posve novi podaci. Periodičko učitavanje obavlja se u trenutku dok su skladište podataka i transakcijske baze izvan funkcije, najčešće u noćnim ili ranojutarnjim satima. Skladište se osvježava dnevno ili tjedno. Ukoliko bi period osvježavanja bio dulji, korisnici skladišta podataka ne bi na vrijeme raspolagali upravo najsvježijim podacima. Kako bi se spriječilo da u skladište uđu nekvalitetni i nepotpuni podaci, već se u procesu izvlačenja, transformacije i učitavanja podataka vrši kontrola njihove kvalitete.

Spremanje podataka i upravljanje podacima. Pojam spremanja i upravljanja podacima opisuje aktivnosti administratora skladišta podataka kako bi se omogućio uspješan rad skladišta: brz i siguran korisnički pristup podacima. Povećanjem obujma

podataka u skladištu tijekom vremena ne smiju se narušiti performanse sustava. U slučaju da se u skladištu podataka pojave netočni ili neispravni podaci koje nije bilo moguće ranije otkriti kontrolom kvalitete (npr. zbog ranije pogreške u transakcijskoj bazi nalazi se neistinit, semantički neispravan podatak) mora postojati brz i efikasan način da se oni uklone iz skladišta i na njihovo mjesto postave ispravne vrijednosti.

Korištenje skladišta podataka. Rezultati upita nad podacima u skladištu predstavljaju značajan faktor u poslovnim odlukama poduzeća. S obzirom na kompleksnost analize podataka, vrijeme trajanja analize i znanje o skladištu podataka potrebno za analiziranje razlikuju se tri osnovna načina rada sa skladištem podataka: sumarni izvještaj, analitička obrada i dubinska analiza podataka.

Sumarni izvještaj (engl. *reporting*) najjednostavniji je način korištenja skladišta podataka i zahtijeva minimalno znanje o strukturi skladišta podataka. Postavljeni upiti su poznate, fiksne strukture. Stvaranje sumarnih izvještaja vrlo je slično radu s transakcijskom bazom podataka, s razlikom da skladište podataka sadrži mnogo veću količinu podataka i pruža kvalitetniji uvid u podatke.

Analitička obrada (engl. *Online Analytical Processing*, OLAP) je interaktivna dinamička analiza velike količine podataka. Upiti se postavljaju na temelju podataka dobivenih iz prethodnih upita tj. njihova struktura nije unaprijed poznata. Alati za analitičku obradu u potpunosti su prilagođeni višedimenzijskom modelu podataka. Zbog toga korisnik mora poznavati strukturu skladišta podataka i biti kvalitetno obučen za rad s alatima za analitičku obradu. Analitičkom obradom podataka bave se analitičari i menadžeri poduzeća. Znanje potrebno za rad sa skladištem stječe se pohađanjem tečajeva.

Dubinska analiza podataka je proces automatskog otkrivanja prethodno nepoznatih obrazaca i odnosa među podacima u bazama podataka [SMV00]. Alati za dubinsku analizu koriste vrlo složene, sofisticirane algoritme kako bi unutar velikog skupa podataka identificirali odnose i veze za koje se nije znalo da postoje. Osoba koja provodi dubinsku analizu podataka mora imati široko znanje o skladištima podataka, izuzetno dobro poznavati strukturu skladišta čiji sadržaj analizira i razumjeti sve algoritme koji joj stoje na raspolaganju kako bi ih mogla primijeniti na pravi način i u pravilnom redoslijedu. Za dubinsku analizu podataka potrebno je stručno obrazovanje i iskustvo pa se za takav rad otvaraju posebna radna mjesta namijenjena stručnjacima-inženjerima.

1.2. Višedimenzijski model podataka

1.2.1. Konceptualni, logički i fizički model podataka

Između bitova na disku računala koji predstavljaju stvarni zapis sadržaja baze podataka i načina na koji korisnik promatra podatke u bazi podataka postoji veći broj razina apstrakcije. Svaka razina apstrakcije predstavlja jednu vrstu modela tj. skup objekata i operacija nad tim objektima. [SMV00].

Na najnižoj se razini apstrakcije bazu podataka može promatrati kao skup ili datoteka na disku računala, koje za korisnika ne čine jedinstvenu cjelinu, već samo nepovezan skup podataka.

Višu razinu apstrakcije nude različiti modeli baze podataka. Model baze podataka sagledava bazu podataka kao jedinstvenu cjelinu. Autori s obzirom na razine apstrakcije najčešće razlikuju tri vrste modela baze podataka: konceptualni, logički i

fizički model podataka. Na najvišoj razni apstrakcije je konceptualni model podataka, slijedi logički model, dok je fizički model baze podataka na najnižoj razini apstrakcije.

Fizički model baze podataka definira parametre za pohranu podataka na fizički medij (format zapisa u memoriju, način pohranjivanja). Taj model se još uvijek nalazi na nižoj razini apstrakcije od uobičajenog korisničkog pogleda na podatke.

Korisnički pogled na podatke očituje se kroz konceptualni i logički model podataka. Konceptualni i logički model baze podataka u potpunosti su neovisni o načinu pohrane podataka koje definira fizički model.

Konceptualni model baze podataka u potpunosti je nezavisan o sustavu za upravljanje bazom podataka i operacijskom sustavu računala na kojem se baza podataka nalazi. On ostaje isti bez obzira upotrijebe li se hijerarhijski, mrežni ili relacijski sustav za upravljanje bazom podataka. U sklopu konceptualnog modela podataka isključivo se definiraju atributi za željene entitete i veze između entiteta. Primjer konceptualnog modela za baze podataka je model entitet-veza [SMV00].

Logički model podataka je korisnikovo viđenje podataka u modelu koji podržava odgovarajući sustav za upravljanje bazom podataka. Ovaj model definira način na koji su entiteti međusobno povezani, a time i mogućnosti odnosno načine pretraživanja baze podataka. Hijerarhijski, mrežni i relacijski model podataka su logički modeli baze podataka.

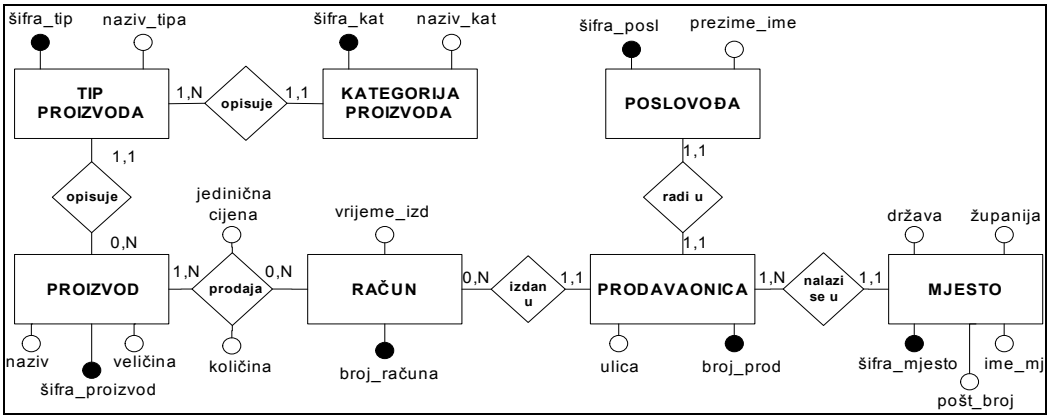
1.2.2. Model entitet-veza i njegova neprikladnost za oblikovanje skladišta podataka

Model entitet-veza (engl. *entity-relationship model*, ER) je konceptualni model podataka koji je prvi opisao Chen 1976. Model entitet-veza čine entiteti, njihovi atributi i veze između entiteta. Konceptualna shema predočava se grafički dijagramom entiteti-veze. Uz najstariji, Chenov, postoji veći broj drukčijih grafičkih prikaza, od kojih je najpoznatiji Martinov. Na slici 1.1 pojednostavnjeno su prikazani entiteti koji opisuju maloprodajno poduzeće (grafički prikaz prema Chenu).

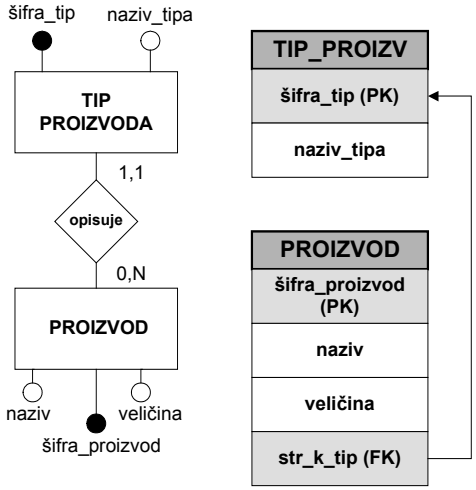
U modelu entitet-veza svakom se entitetu definiraju atributi, koji iskazuju njegova svojstva. Entitet uspostavlja veze s jednim ili nekoliko drugih entiteta. Ovakav konceptualni model izuzetno se lako preslikava u logički model za relacijsku bazu podataka. Entitet postaje relacijska shema, a njegovi atributi postaju atributi relacijske sheme. Binarne veze (tj. one koje uključuju samo dva entiteta) tipa 1:1 i 1:N izražene su stranim ključem na drugu relacijsku shemu (slika 1.2). Veza M:N se može pretvoriti u dodatni entitet, koji s početnim entitetima ostvaruje odnos 1:M odnosno 1:N.

Zbog opisanih kvaliteta model entitet-veza s vremenom je postao dominantni konceptualni model za oblikovanje transakcijskih baza podataka u informacijskim sustavima organizacija. On je iznimno pogodan u osmišljanju i projektiranju relacijskih baza podataka, ali i posve neprikladan za skladište podataka. Za korisnika skladišta podataka koji sagledava cjelokupno poslovanje svoje organizacije dijagram koji sadrži preko stotinu entiteta posve je nerazumljiv i beskoristan. Izabere li se jedan entitet za početni entitet u procesu poslovne analize i upravi li se pogled na ostale entitete, jasno u prvi plan izbijaju entiteti s kojima je taj entitet izravno povezan, potom preostali s njima povezani entiteti. Ostali entiteti se mogu dovesti u vezu s početnim entitetom isključivo preko već spomenutih, njemu susjednih entiteta. Time

se unaprijed implicira tijek poslovne analize. Kvalitetna analiza može se, naprotiv, obaviti samo u slučaju kad parametre možemo postavljati međusobno neovisno.



Slika 1.1. Pojednostavnjeni dijagram entitet-veza za maloprodajno poduzeće



Slika 1.2. Preslikavanje binarne veze iz modela entitet-veza u relacijsku shemu

Kod transakcijskih baza potrebno je postići minimalnu zalihost podataka pa se relacije normaliziraju do 3. normalne forme. Nastoji se stvoriti što više entiteta od kojih svaki ima razmjerno malo atributa. Za poslovnu je analizu, međutim, ključno zbrajati podatke na temelju postojanja funkcijskih ovisnosti. Primjerice, zaradu poduzeća zapisanu pojedinačno po prodavaonicama, može se zbrojiti po mjestima u kojima se prodavaonice nalaze te po županijama. Prodavaonica funkcijski određuje mjesto, a mjesto funkcijski određuje županiju. Za razliku od transakcijskih baza, gdje bi se podaci za prodavaonice, mjesta i županije zapisali u tri odvojene relacije, za poslovnu analizu najpogodnije je opisane parametre (međusobno ovisne funkcijski ili tranzitivno) skupiti na jednom mjestu.

1.2.3. Višedimenzijski konceptualni model podataka

Skladište podataka, kao bazu podataka za poslovne analize, potrebno je izraditi u konceptualnom modelu koji će jasno postaviti "početni" element za analizu, omogućiti istodobno postavljanje nekih parametara poslovanja kao međusobno posve nezavisnih, drugima (funkcijski i tranzitivno ovisnima) naglasiti zavisnost te biti

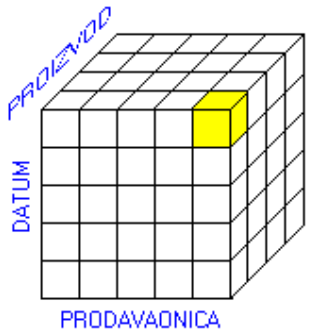
pogodan za sumiranje. Takve mogućnosti pruža višedimenzijski konceptualni model (engl. *multi-dimensional conceptual model*).

Osim pojma višedimenzijski konceptualni model [GMR98] u literaturi se koristi i naziv dimenzijski konceptualni model (engl. *dimensional conceptual model*), a pojam dimenzijski model (bez riječi "konceptualni", engl. *dimensional model*) se često odnosi na logički model koji opisuje realizaciju višedimenzijskog konceptualnog modela u sustavu za upravljanje relacijskom bazom podataka. Kimball [Kim96] opisuje dimenzijsko modeliranje kao "tehniku konceptualnog i implementacijskog oblikovanja kojom se putem standardiziranog intuitivnog koncepta omogućuje brz pristup podacima". Jednostavnim konceptualnim modelom opisuju se poslovni procesi preko dimenzija koje su često hijerarhijski organizirane. Višedimenzijski konceptualni model definira osnovne pojmove činjenice, mjere, dimenzije i hijerarhije.

Činjenica (engl. *fact*) predstavlja središnju točku interesa u procesu donošenja poslovne odluke [GMR98, GRV01]. To je polazna točka poslovne analize. **Mjere** (engl. *measures*) su atributi s kontinuiranim (neprekidnim) skupom vrijednosti (najčešće numerički) koji opisuju činjenicu. Budući da skladište podataka sadrži ogroman broj zapisa, upravo će se zbrajanjem numeričkih podataka dobiti vrijednosti zanimljive za analizu.

Za različite organizacije i oblike poslovanja razlikuju se primarne točke interesa za poslovnu analizu. Za lanac maloprodajnih trgovina središte interesa predstavlja prodaja proizvoda i njihova nabava od dobavljača. Prodaja i nabava bit će dvije različite činjenice. Ključni podaci o prodaji vezani su uz prodajne cijene proizvoda, prodane količine tih proizvoda te prihod ostvaren od njihove prodaje. Slično, za nabavu proizvoda bitne su nabavna cijena, količina nabavljene robe i troškovi nabave. Za poslovanje banke od ključnog će značenja biti računi i transakcije nad računima. Za činjenicu račun mjera je stanje računa, pri čemu se pretpostavlja da se stanje pohranjuje na kraju dana. Kako se u jednom danu može izvršiti više transakcija dobro je koristiti i mjeru broja transakcija toga dana. Činjenica račun nudi opći pregled nad poslovanje banke. Za detaljniji pregled koristi se činjenica transakcija koju opisuje mjera količine novca obuhvaćene transakcijom.

Dimenzije čine skup međusobno nezavisnih parametara koji opisuju činjenicu. Svaka dimenzija sastoji se od atributa diskretnog (konačnog) skupa vrijednosti. Najbolji opisni atributi su tekstualni. Dimenzije određuju razinu zrnatosti u skladištu podataka [GMR98]. Tipične dimenzije za činjenicu prodaje u skladištu podataka maloprodajnog lanca trgovina bit će vrijeme, trgovina (poslovnica) i prodani proizvod [Kim96]. Vrijednosti vremena, trgovine i prodanog proizvoda u analizi prodaje mogu se postavljati međusobno nezavisno. Budući da su sve dimenzije međusobno nezavisne, one se mogu predstaviti kao n-dimenzijski koordinatni sustav (u opisanom slučaju prodaje radi se o trodimenzijskom sustavu, slika 1.3), pri čemu su vrijednosti na osi diskretne (možemo ih poistovjetiti sa skupom prirodnih brojeva). Svaka n-torka koordinatnog sustava u primjeru prodaje opisuje prodaju određenog proizvoda za određeni dan u određenoj trgovini i predstavlja točku (odnosno kockicu, budući da se radi o diskretnom skupu vrijednosti) u koordinatnom sustavu. Ovakva n-torka naziva se **primarnim događajem** (engl. *primary event*). Unutar svake kockice pohranjene su vrijednosti mjera za tu n-torku.



Slika 1.3. Trodimenzijski prikaz prodaje maloprodajnog lanca trgovina

Dimenzijsku hijerarhiju čini niz međusobno funkcijski ovisnih **dimenzijskih atributa** unutar jedne dimenzije koji činjenicu opisuju na različitoj razini zrnatosti (detaljnosti). Primjerice, vremenska dimenzija koja opisuje prodaju sadrži na najvišoj razini detaljnosti atribut datum, a uz njega atribut mjesec (koji funkcijski ovisi o atributu datuma), atribut tromjesečja (koji funkcijski ovisi o atributu mjeseca, a tranzitivno o atributu datuma) i atribut godina (funkcijski ovisan o atributu tromjesečja i tranzitivno o atributima mjeseca i datuma). Najdetaljniji atribut, datum, određuje razinu zrnatosti za činjenicu prodaja. Analogno, u dimenziji trgovina na najvišoj razini detaljnosti (koja određuje ukupnu maksimalnu zrnatost za činjenicu) je pojedina trgovina-poslovnica, a potom slijede mjesto i veće zemljopisne cjeline (prodajno područje unutar županije, pa županija). U dimenziji proizvod najdetaljniju razinu određuje upravo proizvod, dok se zbrajanje može izvršiti s obzirom na tip proizvoda (tj. potkategoriju) i kategoriju proizvoda. Zbrajanje vrijednosti mjera s obzirom na vrijednost nekog atributa u hijerarhiji na višoj razini od osnovne naziva se **agregacijom** po tom atributu. Agregacijom prodaje po mjesecima međusobno se zbrajaju sve dnevne vrijednosti prihoda i količine prodanih proizvoda u nekoj trgovini koje pripadaju istom mjesecu u godini. Kockice osnovne veličine, primarni događaji, stapaju se u veću kocku, **sekundarni događaj**.

Opisni atributi unutar dimenzije su atributi pomoću kojih nije moguće izvršiti bilo kakvu agregaciju. Omogućuju kvalitetnije ograničenje (restrikciju) upita. Funkcijski su ovisni o najdetaljnijem atributu dimenzije. Takav je atribut telefonski broj poslovnice u dimenziji trgovine-poslovnice. Svaki od atributa koji čine hijerarhiju temelj je jedne **razine hijerarhije** (engl. *hierarchy level*). U istoj razini hijerarhije nalaze se oni opisni atributi koji opisuju temeljni atribut te razine hijerarhije. Tako atribut dan u tjednu ili atribut-zastavica (tj. atribut koji poprima samo vrijednost istinito ili neistinito) državni blagdan opisuju atribut datuma, dok atributi naziv mjeseca i broj dana u mjesecu opisuju atribut mjeseca.

1.2.4. Potpuno zbrojive, poluzbrojive i nezbrojive mjere

Mjere mogu biti potpuno zbrojive, poluzbrojive i nezbrojive. Potpuno zbrojive mjere mogu se zbrajati po svim dimenzijama koje opisuju činjenicu. Poluzbrojive mjere mogu se zbrajati samo po nekim dimenzijama. Za nezbrojive mjere ne može se izvršiti agregacija niti po jednoj dimenziji. Najčešće se radi o atributima koji bi se trebali naći u dimenzijama, ali zbog načela logičkog oblikovanja ostaju kao opis činjenice.

Prihod od prodaje je potpuno zbrojiva mjera jer se može sumirati po bilo kojoj od tri postojeće dimenzije. Stanje bankovnog računa nije zbrojivo po vremenskoj dimenziji.

Nema smisla zbrajati stanje računa kroz mjesec dana jer dobiveni broj nema nikakvo stvarno značenje. Parametar zanimljiv za poslovne analize je prosječno stanje računa (kroz mjesec dana ili neki drugi vremenski period). Za njegovo računanje potrebno je zbrojiti dnevne vrijednosti i podijeliti dobiveni rezultat brojem dana koji čine traženi vremenski period. S obzirom da se vremenska dimenzija nalazi u svakom skladištu podataka, mjera stanja računa će ostati poluzbrojiva, nezavisno o izboru ostalih dimenzija u skladištu.

Primjeri zbrojivih, poluzbrojivih i nezbrojivih mjera bit će navedeni u poglavlju 7.1.4.

1.2.5. Izvedba višedimenzijskog modela u sustavima za upravljanje relacijskim bazama podataka

Svaki dobro definirani konceptualni model podataka moguće je pretvoriti u bilo koji logički model za realizaciju u sustavu za upravljanje bazom podataka. Skladišta podataka u pravilu se izvode u relacijskim bazama podataka, ali su se u novije vrijeme pojavili i sustavi za upravljanje bazama podataka koji prvenstveno podržavaju višedimenzijski model podataka. Analitička obrada nad tako realiziranim skladištima podataka popularno se naziva MOLAP (engl. *Multi-dimensional OLAP*), a takve baze podataka nazivaju se višedimenzijskim bazama. Za analitičku obradu nad skladištima izvedenim u relacijskim bazama podataka koristi se analogni pojam ROLAP (engl. *Relational OLAP*) [Vrd99].

Izvedbe u relacijskoj tehnologiji danas su daleko najviše zastupljene budući da je ona dominantna na tržištu još od vremena kad je razvoj koncepta skladištenja podataka tek započeo. Relacijska tehnologija već se do tog trenutka pokazala iznimno kvalitetnom pa je izvedba skladišta podataka upravo u toj tehnologiji predstavljala financijsku uštedu u odnosu na razvoj posve novih, višedimenzijskih baza, a za relacijske tehnologije već su postojali i brojni obrazovani iskusni stručnjaci.

U početku razvoja skladišta podataka njihovo je oblikovanje počinjalo izravno logičkim modelom što se može vidjeti u starijoj literaturi, izdanoj polovicom devedesetih godina dvadesetog stoljeća (npr. [Kim96]). Prva skladišta podataka razvila su se još krajem osamdesetih godina dvadesetog stoljeća na poticaj uprava i analitičkih odjela tvrtki, koji su od svojih informatičkih odjela zahtijevali poboljšanja u mogućnostima poslovne analize. Tek je kasnije, u drugoj polovici devedesetih godina 20. stoljeća, počelo šire znanstveno proučavanje skladišta podataka pri čemu je na osnovi postojeće implementacijske prakse retrogradno razvijen višedimenzijski konceptualni model.

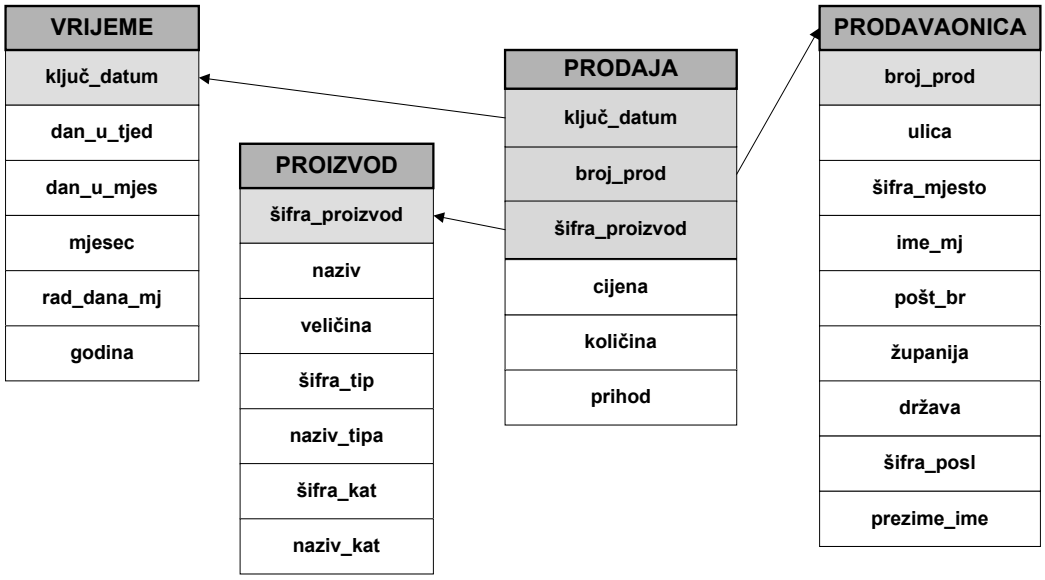
Najjednostavnija izvedba višedimenzijskog konceptualnog modela u relacijskoj tehnologiji jest **spoj zvijezda**. On se sastoji od središnje tablice činjenica i više dimenzijskih tablica.

Svakoj dimenziji u konceptualnom modelu odgovara jedna dimenzijska tablica (engl. *dimensional table*) logičkog modela. Svi dimenzijski atributi nalaze se u dimenzijskoj tablici, a za njen primarni ključ se uzima temeljni atribut najdetaljnije razine hijerarhije. Ključ dimenzijske tablice uvijek se sastoji od samo jednog atributa. Dimenzijska tablica je denormalizirana i u njoj postoji tranzitivna funkcijska ovisnost. Veza entiteta opisanog činjeničnom tablicom i bilo kojeg entiteta opisanog nekom od dimenzijskih tablica je uvijek jedan-prema-više. Za svaki zapis u činjeničnoj tablici postoji samo jedan zapis u svakoj dimenziji. Za bilo koji zapis u bilo kojoj dimenzijskoj tablici može postojati više odgovarajućih zapisa u činjeničnoj tablici.

Opisani međusobni odnos zapisa omogućuje sumiranje (agregaciju) zapisa u činjeničnoj tablici.

U tablici činjenica (engl. *fact table*) nalaze se sve mjere dane činjenice. Uz mjere, činjenična tablica sadrži po jedan strani ključ na svaku od dimenzija koje opisuju činjenicu. Svi strani ključevi na dimenzijske tablice zajedno čine složeni primarni ključ tablice činjenica. To znači da je redak činjenične tablice određen kombinacijom svih primarnih ključeva dimenzijskih tablica. To odgovara predodžbi o n-dimenzionalnom prostoru unutar kojeg se nalaze kockice kao dijelovi prostora (slika 1.3). Svaki redak činjenične tablice upravo je jedna kockica. Vrijednosti upisane unutar kockice (određene koordinatama n-dimenzionalnog prostora) su vrijednosti atributa mjera u činjeničnoj tablici (reci su određeni primarnim ključem činjenične tablice tj. kombinacijom svih dimenzija). Kako su za ključeve svih dimenzijskih tablica uzeti temeljni atributi najdetaljnije hijerarhijske razine, ta će razina zrnatosti biti ostvarena i u tablici činjenica. Upravo zrnatost u tablici činjenica odgovara pojmu zrnatosti skladišta.

Spoj zvijezda za višedimenzijski model prodaje u maloprodajnom lancu trgovina prikazan je na slici 1.4.



Slika 1.4. Shema spoja zvijezda za skladište podataka maloprodajnog lanca trgovina

Ključni atribut u tablici VRIJEME je ključ_datum. Vremensku hijerarhiju ostvaruju atributi datum (s opisnim atributima dan_u_tjed, dan_u_mjes), mjesec (s opisnim atributom rad_dana_mj) i godina. Izvor podataka je transakcijska baza podataka opisana dijagramom sa slike 1.1. U transakcijskoj bazi postoji samo atribut vremena izdavanja računa koji u sebi sadrži i datum. Svi ostali atributi u vremenskoj dimenziji dobivaju se iz datuma u procesu izvlačenja, transformacije i učitavanja podataka primjenom odgovarajućih funkcija. Vremenska dimenzija ispunjena podacima prikazana je u tablici 1.2, a vremenska hijerarhija na slici 1.5.

Ključni atribut u tablici PROIZVOD je šifra_proizvod, koji odgovara istoimenom atributu u transakcijskoj bazi. Ključni atribut tablice PRODAVAONICA je broj_prod. Primjer sadržaja činjenične tablice daje tablica 1.3.

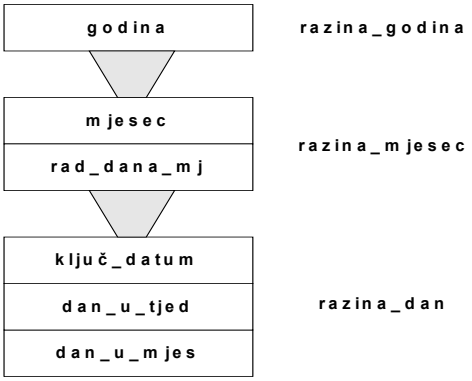
Tablica 1.2. Isječak iz dimenzijske tablice VRIJEME skladišta podataka maloprodajnog lanca trgovina

ključ_datum	dan_u_tjed	dan_u_mjes	mjesec	rad_dana_mj	godina
23.01.2004.	PETAK	23	1	21	2004
25.02.2004.	SRIJEDA	25	2	20	2004
26.02.2004.	ĆETVRTAK	26	2	20	2004

Tablica 1.3. Isječak iz činjenične tablice skladišta podataka maloprodajnog lanca trgovina

ključ_datum	broj_prod	šifra_proizvod	cijena	količina	prihod
23.01.2004.	002	267623	45.86	10	458.60
25.02.2004.	007	267623	46.87	7	328.09
02.03.2004.	023	035737	404.68	1	404.68

U transakcijskim bazama podataka nalaze se tablice u trećoj ili višoj normalnoj formi. Normaliziranost je nužna kako bi se zalihost, a time i mogućnost nekonzistentnosti podataka nakon promjene vrijednosti nekih redaka ili unosa novih redaka onemogućila ili svela na najmanju moguću mjeru. Kako skladište podataka sadrži nepromjenjive podatke, u njemu ne postoji opasnost od nekonzistentnosti uslijed ažuriranja. Time se gubi glavni razlog izgradnje velikog broja tablica od kojih svaka sadrži mali broj atributa.



Slika 1.5. Hijerarhija u dimenziji VRIJEME skladišta podataka maloprodajnog lanca trgovina

Dimenzija treba sadržavati što veći broj atributa kako bi se upiti mogli ograničavati na što više načina, a time i vršiti kvalitetnija analiza. Ukoliko se svi dimenzijski atributi smjeste u jednu tablicu hijerarhija se automatski implicira vrijednostima stupaca u tablici i omogućuje jednostavne metapodatke za programski alat koji služi za pripremu upita (tj. ograničenja kod upita). Kad bi se svaka razina hijerarhije izvodila u posebnoj tablici bili bi potrebni složeniji metapodaci, a izrada programskog alata bila bi kompliciranija. Drugi razlog denormaliziranosti tablice je način izvođenja upita. Upit se izvodi tako da se najprije u dimenzijskim tablicama pronađu sve vrijednosti ključeva koje zadovoljavaju postavljena ograničenja. Potom se u činjeničnoj tablici pronađu sve postojeće kombinacije tih ključeva i ti se reci dalje obrađuju. Ako bi se dimenzija izvela u većem broju tablica, kod svakog bi ograničenja atributom koji nije u istoj tablici kao i primarni ključ dimenzije bilo potrebno vršiti operacije prirodnog spajanja. To znatno usporava rad sustava. Razvojem brzih računala velike procesorske moći problem izrazitog usporavanja izvođenja upita zbog operacija prirodnog spajanja znatno je ublažen. Usprkos tome,

načelo denormaliziranosti dimenzijskih tablica i dalje se održalo, prije svega zbog jednostavnosti izvedbe (minimalni broj tablica) i jednostavnih metapodataka (laganog zadavanja ograničenja kod upita).

1.2.6. Područno skladište podataka, dijeljene dimenzije i model sabirnice

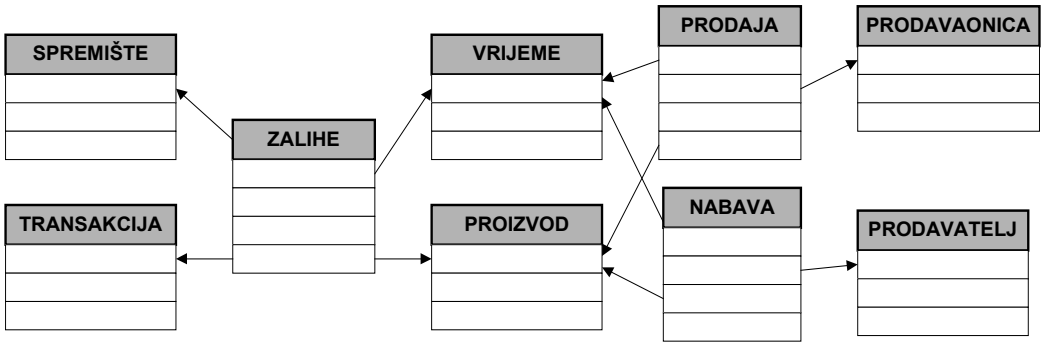
Prema Hackneyju, **područno skladište podataka** (engl. ***data mart***) je skup podataka oblikovan i konstruiran radi potpore odlučivanju, pri čijem se oblikovanju slijede načela oblikovanja skladišta podataka s tim da taj skup podataka poslužuje potrebe *homogene grupe korisnika*.

Pod pojmom homogene skupine korisnika podrazumijevaju se odjeli poduzeća ili manje radne grupe unutar odjela. Radna organizacija najčešće posjeduje jedno središnje skladište podataka i više područnih skladišta podataka. Odjel poduzeća bavi se jednim segmentom njegovog poslovanja pa njegovo područno skladišta najčešće sadrži jednu činjenicu i u tom se slučaju svodi na jednu činjeničnu tablicu izvedenu spojem zvijezda. Zbog toga je pojam *data mart* često sinonim za jedan spoj zvijezda.

U velikoj poslovnoj organizaciji moguće je fizički zasebno izvesti veliko središnje skladište podataka i područna skladišta, koja podatke izvlače iz središnjeg. Druga, znatno jeftinija mogućnost je načiniti samo područna skladišta podataka kao zasebne module te ih potom logički integrirati u skladište koje će sadržavati podatke za cijelu organizaciju. To znači da se područna skladišta pojedinih odjela ne smiju planirati odvojeno i nezavisno, već je potrebno razviti zajedničku razvojnu koncepciju na razini cjelokupne organizacije (moduli moraju biti kompatibilni). Za cijeli sustav skladištenja vrijedi zajednički model podataka (primjerice, sva se područna skladišta izvode u relacijskom modelu), a zbog konzistentnosti i izbjegavanja zalihosti koriste se zajednički sustavi za izvlačenje, transformaciju i učitavanje podataka. Metapodaci su zajednički i pohranjuju se na razini cjelokupnog sustava skladišta. Nakon definiranja zajedničke koncepcije moduli (područna skladišta) se mogu razvijati zasebno i paralelno.

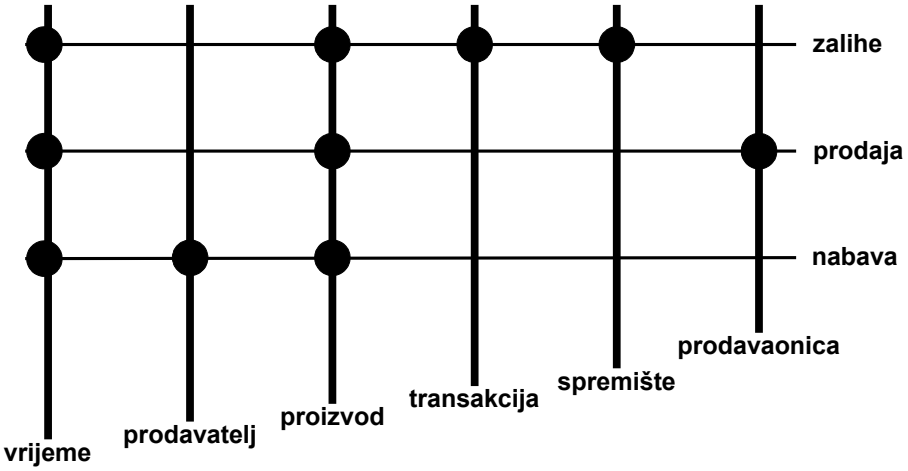
Izgradnja sustava na opisanim načelima u relacijskoj tehnologiji temelji se na tzv. **modelu sabirnice** pri čemu ključnu ulogu igraju dijeljene dimenzije. Svaki odjel dobiva vlastito područno skladište, ali se prethodno izrađuje zajednički model za sve odjele.

Za primjer se može uzeti ranije spomenuti lanac maloprodajnih trgovina koji kupuje gotove proizvode, eventualno ih prije prodaje neko vrijeme drži pohranjene u skladištu te ih naposljetku prodaje. U tom poduzeću postoje odjeli NABAVA, ZALIHE i PRODAJA. Svaki odjel ima zasebnu činjeničnu tablicu, kao osnovu spoja zvijezda, koja sadrži mjere specifične za taj odjel. Svaki odjel koristi vremensku dimenziju i dimenziju proizvod. Kako se u procesu nabave, skladištenja i prodaje radi s istim proizvodima, moguće je da ista dimenzija proizvoda posluži za sve tri činjenične tablice (činjenične tablice dijele istu dimenziju). Time će se izbjeći moguća zalihost i nekonzistentnost do koje bi došlo kad bi svaki odjel izrađivao svoju zasebnu dimenziju proizvoda. Ista vremenska dimenzija također se može upotrijebiti za sve tri činjenične tablice jer svaka od njih sadrži dnevne zapise na osnovnoj razini zrnatosti. Tri opisane činjenične tablice i pripadne im dimenzijske tablice prikazuje slika 1.6.



Slika 1.6. Dijeljene dimenzije

Globalni plan skladišta jednostavno se i slikovito prikazuje modelom sabirnice (slika 1.7). Sabirnički vodovi (stupci) predstavljaju sve dimenzije u skladištu, a postranični vodovi (reci) činjenične tablice. Ukoliko se postranični vod priključuje na sabirnicu, to znači da je dana dimenzijska tablica jedna od dimenzija te činjenične tablice.



Slika 1.7. Model sabirnice za prikaz globalnog plana skladišta.

Model sabirnice omogućuje brzu paralelnu izgradnju područnih skladišta. Nije potrebno formirati posebno središnje skladište podataka, nego je ono ostvareno skupom svih područnih skladišta. Istodobno, otklonjena je mogućnost postojanja zalihosti i nekonzistentnosti.

2. XML

2.1. Polustrukturirani podaci i komuniciranje mrežom

U današnje vrijeme na računalima je u elektroničkom obliku pohranjena izuzetno velika količina podataka. Podaci pohranjeni u baze podataka odlikuju se jasno zadanom strukturom, dok u datotečnim sustavima (engl. *file systems*) postoji znatna količina nestrukturiranih podataka. Tipičan slučaj nestrukturiranih podataka su datoteke koje predstavljaju slike i zvukove. Struktura podataka opisuje tip (vrstu) i redoslijed manjih jedinica podataka unutar većih podatkovnih jedinica, a zadana je shemom.

Kod strukturiranih podataka shema je apriorna (najprije se definira shema, a potom se počinju unositi podaci koji odgovaraju shemi) i odvojena od podataka. Shemu se može nazvati određenom vrstom metapodataka (podataka o podacima). Za primjer se mogu uzeti relacijske baze podataka gdje se najprije zadaje relacijska shema, a zatim stvara relacija koja se može puniti podacima.

Polustrukturirane podatke karakterizira barem jedno od sljedećih dvaju osnovnih obilježja [Abi97]:

- **Nepravilna struktura**. Veliki skupovi podataka često se sastoje od manjih podatkovnih jedinica čija struktura je međusobno slična, ali ne i posve jednaka. U primjeru na slici 2.1 može se uočiti da američki književnik uz ime i prezime ima i srednje ime, francuski autor nema srednje ime, a ruski uz ime i prezime u sredini ima očevo ime.

  Iz činjenice da struktura smije biti nepravilna, znači da shema nije strogo ograničavajuća i bez iznimke kao u slučaju potpuno strukturiranih podataka u bazi. Na nekom se mjestu unutar veće podatkovne jedinice može, ali i ne mora nalaziti manja podatkovna jedinica (npr. srednje ime ili očevo ime unutar podataka o književniku).

- **Implicitna struktura**. Shema podatkovne strukture ne mora biti sadržana u posebnom dokumentu, nego integrirana zajedno s njihovim sadržajem. Iz podataka na slici 2.2 vidljivo je da je `Verdi` prezime, a `Giuseppe` ime određenog kompozitora, da su njegovo djela `Traviata` i `Don Carlos` te da se u oba slučaja radi o operama. Značenje riječi `Verdi` nije napisano u odvojenom vanjskom dokumentu nego je upravo uz sadržaj podataka (`Verdi`) navedeno i što oni predstavljaju (prezime).

Budući da polustrukturirani podaci mogu unutar sebe sadržavati opis svoje strukture (tj. metapodatke) za njih se može reći da "opisuju sami sebe" (engl. *self-describing*) [Bun97]. Za njih tada nije nužna shema.

```
{autor: {prezime: "Poe"}, {ime: "Edgar"}, {srednje: "Allan"}}
{autor: {prezime: "Gide"}, {ime: "Andre"}}
{autor: {prezime: "Jesenjin"}, {ime: "Sergej"}, {očevo:
"Aleksandrovič"}}
```

Slika 2.1. Primjer nepravilne strukture kod polustrukturiranih podataka

```
{kompozitor:
      {prezime:"Verdi"}{ime="Giuseppe"}
      {djelo: {opera:"Traviata"}}
      {djelo: {opera:"Don Carlos"} } }
```

Slika 2.2. Primjer integriranosti sadržaja podataka i podatkovne strukture

Potreba za polustrukturiranim podacima nastaje kao rezultat težnje da se različiti podatkovni sadržaji (osobito *World Wide Web* kao dio Interneta) pretražuju i sistematiziraju na način što sličniji bazama podataka. Struktura www-dokumenata je izrazito heterogena. Jedini način da se zadrži strukturiranost, a ipak dozvoli raznovrsnost sadržaja jest upotreba polustrukturiranih podataka. Vrlo bitan poticaj proučavanju polustrukturiranih podataka dala je činjenica da HTML, dominantni jezik za izradu *web*-stranica podržava polustrukturiranost i da mnoge datoteke pisane u HTML-u u nekim svojim dijelovima imaju polustrukturirani sadržaj. Drugo područje u kojem se nameće polustrukturirani format je razmjena podataka između različitih baza podataka. Iako su podaci unutar svake baze u potpunosti strukturirani, njihove se strukture od baze do baze razlikuju. Ranije su navedene slične teškoće s kojima se susreću projektanti skladišta podataka prilikom složenog procesa integracije heterogenih transakcijskih baza podataka u skladište podataka.

Budući da su namijenjeni integraciji podataka iz raznorodnih izvora, polustrukturirani podaci smiju unutar sebe sadržavati potpuno nestrukturirane cjeline. Prilikom kasnije obrade podataka takve se cjeline izdvajaju i dalje ne obrađuju. Takvi nestrukturirani podaci mogu biti binarne datoteke iz baza podataka ili tekst koji se sprema kao jedna cjelina.

Za analizu polustrukturiranih podataka nije nužna posebna shema. Programu za obradu takvih podataka dovoljno je analizirati metapodatke unutar samog polustrukturiranog dokumenta. Zbog toga se često događa da se neki polustrukturirani format uvriježi mrežnom razmjenom, a tek potom se definira shema. Takva je shema aposteriorna, za razliku od strukturiranih podataka gdje je nužno apriorna. Dinamičnost sadržaja u nekim slučajevima uvjetuje promjene postojeće sheme.

Raznovrsnost podataka koji se opisuju polustrukturiranim formatom dovodi do nepravilne strukture, a time i kompliciranih shema. Kod polustrukturiranih podataka sheme su po količini memorije koju zauzima njihov zapis jednako velike, a najčešće i višestruko veće od samih podataka koje opisuju.

Korištenje polustrukturiranog podatkovnog oblika naraslo je do ogromnih razmjera primjenom formata XML. XML je nastao zbog potrebe da se sadržaj na Internetu stvara dinamički, na temelju razmjene podataka putem mreže.

2.2. Format XML

2.2.1. Standard Generalized Markup Language - SGML

U engleskom jeziku pojam *markup* označava zabilješke i napomene uz rukopis koje opisuju kako ga treba oblikovati u tiskari (vrsta i veličina slova, veličina razmaka između redaka, broj stupaca). U suvremenom nazivlju vezanom uz računala taj pojam označava oznake u tekstu koje nisu dio njegovog sadržaja, ali opisuju neke njegove značajke (oblik, strukturu, kontekst). U početku se pojam *markup* (koji bi se najbliže

mogao prevesti kao "označavanje") odnosio upravo na oznake o vizualnom prikazu i formatu teksta, pa je tako, zbog sličnosti s napomenama tiskarima, i dobio ime. Jezici "za označavanje" su 1986. međunarodno standardizirani kad je *Standard Generalized Markup Language* (SGML, opći standardni jezik za označavanje) postao standard organizacije ISO (ISO 8879:1986). SGML je metajezik koji propisuje kojih se pravila trebaju držati autori "jezika za označavanje" prilikom definiranja svojih oznaka za svoj vlastiti jezik. Po standardu SGML svako označavanje (*markup*) započinje oznakom početka (engl. *start-tag*). Prvi znak oznake početka je "<", slijedi ime označavanja, a posljednji znak je ">", kao u sljedećem primjeru:

```
<recenica><jezik>SGML</jezik>je metajezik</recenica>
```

Ukoliko unutar označenog područja postoji tekst, potrebno je na kraju označavanja (tj. označenog područja) navesti oznaku završetka (engl. *end-tag*). Oznaka završetka ima sljedeći izgled: prvo se navode znakovi "</", nakon toga slijedi ime označavanja (isto je kao i u oznaci početka), a na kraju dolazi znak ">". Označi li se dio dokumenta kao *jezik*, taj dio dokumenta počinje oznakom *<jezik>*, a završava oznakom *</jezik>*. SGML razlikuje velika i mala slova prilikom davanja imena različitim označavanjima.

2.2.2. Jezik HTML i razvoj formata XML

HyperText Markup Language (HTML) [Html97] je prvi masovno korišteni "jezik za označavanje". HTML slijedi standard SGML i definira konačan skup oznaka (engl. *tags*) za opis *web*-dokumenata. HTML je prvenstveno je orijentiran na vizualni prikaz sadržaja, pa se kao karakteristične oznake izdvajaju novi red, tabela, ili poveznica (engl. *link*) s drugim *web*-dokumentom. Tijekom devedesetih godina dvadesetog stoljeća pojavili su se milijuni *web*-stranica realiziranih u HTML-u. One su sadržavale izuzetno veliku količinu podataka pa se pojavila potreba za sistematizacijom i organiziranom pohranom takvih podataka. Mana HTML-a da ni na koji način ne opisuje semantičku strukturu svog sadržaja znatno ograničava mogućnost sistematizacije sadržaja u tom formatu i njegovu pohranu u baze podataka. Sistematizaciju sadržaja dodatno otežava specifičnost HTML-a kojom je dozvoljeno da se neke otvorene oznake unutar teksta nije nužno u tekstu formalno zatvoriti, već postoje pravila po kojima procesor za HTML to obavlja sam.

Velika mana HTML-a je i nemogućnost odvajanja sadržaja (tekst unutar oznaka) od njegovog prikaza (definiranog oznakama) odnosno promjene prikaza bez opasnosti od promjene sadržaja i obrnuto. Time se gubi mogućnost jednostavnog stvaranja stranica dinamičkog sadržaja uz stalni, nepromijenjen prikaz. Isto tako, kod velikih i složenih HTML-dokumenata promjenom se prikaza vrlo lako može nehotice izbrisati i dio samog sadržaja stranice.

Istodobno s problemima vezanim uz HTML javila se i potreba za razmjenom različitih vrsta sadržaja (teksta, slike, zvuka, videa) preko Interneta u standardiziranom obliku, uz mogućnost da korisnici sami opisuju sadržaj koji razmjenjuju. Već je ranije spomenuto da su za tu svrhu izuzetno pogodni polustrukturirani podaci. Novi standardni format za mrežnu razmjenu podataka morao je omogućiti polustrukturiranost podataka, omogućiti korisnicima da sami specificiraju metapodatke te zadržati dobra svojstva "jezika za označavanje". Tako je nastao format podataka XML.

XML (*eXtensible Markup Language*) zapravo je format podataka (tj. metajezik). To je znatno pojednostavnjena vrsta SGML-a izrađena za *Web* i Internet. Zadržava strogost

pravopisa koju definira SGML pa tako zahtijeva da se jednom upotrijebljena oznaka u dokumentu obavezno zatvori. XML predstavlja nadopunu HTML-a s obzirom na sljedeće aspekte:

- XML prvenstveno opisuje sadržaj podataka (semantiku), a ne prezentaciju,
- korisnik može definirati vlastite oznake i atribute te ih nazvati po želji,
- strukture se mogu ugnježđivati jedne unutar drugih do bilo koje željene razine,
- dokument u XML smije uz sebe imati dodatni dokument koji ga opisuje (shemu, gramatiku).

Nakon nekoliko godina razmatranja vodeća organizacija za standarde na području *Weba* i Interneta, *World Wide Web Consortium* (W3C) proglasila je 1998. standard XML 1.0. Ova organizacija svoje standarde službeno naziva preporukama (engl. *recommendation*). Standard XML 1.0 je višestruko dorađivan, a posljednja verzija je objavljena 2004. [Xml04]. Treba istaknuti razliku između HTML-a koji je jezik (za njega postoji točno definiran konačan skup oznaka) i XML-a koji je format podataka, metajezik (propisuje način na koji se definiraju oznake). Jezik čiji skup oznaka odgovara onom HTML-a, a format poštuje sva načela XML-a je XHTML. Također je standard organizacije W3C.

2.3. Model podataka za XML

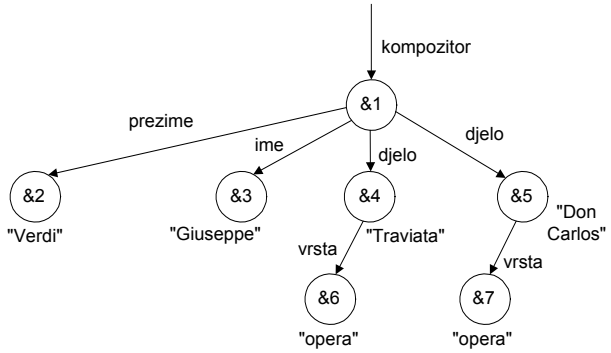
2.3.1. Matematički model za opis polustrukturiranih podataka

Polustrukturirani podaci mogu se promatrati kao veće podatkovne cjeline koje unutar sebe sadrže druge, manje cjeline koje su opet polustrukturirani podaci. Zbog toga se mogu predočavati matematičkim modelom stabla odnosno grafom.

Prvi matematički model polustrukturiranih podataka, Object Exchange Model (OEM) definiran je 1995. [PGMW95] na sveučilištu Stanford, te se profilirao tijekom daljnjih istraživanja na istom sveučilištu kroz projekt Lore (Lightweight Object REpository), namijenjen pohrani polustrukturiranog podatkovnog sadržaja, izvršavanju upita nad njim te njegovoj eventualnoj izmjeni. OEM se može predstaviti usmjerenim grafom koji ima korijenski element. Graf po modelu OEM za podatke o kompozitoru prikazuje slika 2.3. Bridovima grafa pridjeljena su imena podatkovnih struktura (strukturnih elemenata). Vrhovi predstavljaju spojnice ulaznih i izlaznih bridova i omogućuju grananje. Element strukture koja ne sadrži druge strukturne elemente, već samo podatkovni sadržaj (tj. vrijednosti) je brid usmjeren u vrh iz kojeg nema izlaznih bridova (tzv. vrh-list, engl. *leaf vertex*). Tom vrhu pridijeljena je vrijednost sadržaja strukture opisane bridom koji u njega ulazi. Ostali vrhovi nemaju niti ime niti vrijednost, već samo redni broj kako bi se međusobno razlikovali.

2.3.2. Tipovi čvorova u XML dokumentu

Prema specifikaciji XML 1.0 [Xml00], svaki XML dokument sadrži jedan ili više elemenata čije su granice početna i završna oznaka, ili, za prazne elemente oznaka praznog elementa. Između početne i završne oznake svakog elementa mogu se nalaziti niti jedan, jedan ili više podelemenata te tekstualni sadržaj. Na taj se način podaci ugnježđuju jedni unutar drugih, što je karakteristično svojstvo polustrukturiranih podataka. Kod davanja naziva elementima i atributima XML (kao i SGML) pravi razliku između velikih i malih slova (engl. *case sensitive*).



Slika 2.3. Prikaz polustrukturiranih podataka sa slike 2.2 u modelu OEM.

Podaci o kompozitoru Giuseppeu Verdiju ranije navedeni kao primjer polustrukturiranih podataka (slika 2.2) mogli bi u formatu XML imati izgled kao na slici 2.4. Podatke je moguće zapisati i na različite druge, međusobno slične načine (primjerice, bez uvođenja atributa `vrsta` koji ne postoji u zapisu na slici 2.2).

```
<?xml version="1.0">
<kompozitor>
     <prezime>Verdi</prezime>
     <ime>Giuseppe</ime>
     <djelo vrsta="opera">Traviata</djelo>
     <djelo vrsta="opera">Don Carlos</djelo>
     <!--Verdi je veliki operni skladatelj -->
</kompozitor>
```

Slika 2.4. Polustrukturirani podaci sa slike 2.2 napisani u formatu XML

Podaci u formatu XML mogu se, s obzirom na svojstvo polustrukturiranosti, prikazati modelom stabla. Takav model koriste DOM i XPath.

World Wide Web Consortium (W3C) preporučio je za opis podataka u XML-u *Document Object Model* (DOM) [Dom98]. DOM zapravo predstavlja aplikacijsko programsko sučelje (engl. *Application Programming Interface,* API) koje programskim aplikacijama omogućuje pristup podacima u XML-u i HTML-u, njihovu obradu, pretvaranje različitog sadržaja u XML (HTML) te stvaranje novih XML (HTML) dokumenata. Sučelje je u potpunosti nezavisno o implementacijskom programskom jeziku.

DOM je zasnovan na matematičkom modelu koji XML ili HTML dokument predočava u obliku stabla. U teoriji grafova stablo (engl. *tree*) je neciklički graf u kojemu između bilo koja dva vrha postoji samo jedan put (engl. *path*, skup bridova kojima se iz jednog vrha može dospjeti u drugi).

XPath (*XML Path Language*) [XPath99] je jezik koji se koristi za dohvat dijelova XML dokumenta. XML dokument promatra se kao logička struktura, koja je već pročitana i raščlanjena (engl. *parsed*) nekim programskim alatom (vrlo često radi se o aplikaciji koja koristi sučelje DOM). XPath je standard W3C. Model podataka organiziran je na načelu stabla i općenito se naslanja na model koji propisuje DOM.

U modelu OEM elementi podatkovne strukture opisani su bridovima, a sadržaj podataka vrhovima-listovima. Za programsku realizaciju znatno je jednostavnije ili obje komponente, strukturne elemente i podatkovni sadržaj, smjestiti u vrhove ili obje smjestiti u bridove. DOM i XPath sve strukturne elemente i podatkovni sadržaj

smještaju u vrhove, dok bridovi samo povezuju vrhove. Umjesto strogo matematičkog pojma "vrh" (engl. *vertex*) W3C u specifikaciji [Sch1-04] koristi pojam "čvor" (engl. *node*). U literaturi [GRV01, VBR03a, VBR03b] se često umjesto matematičkog pojma "brid" (engl. *edge*) koristi pojam "grana" (engl. *arc*, doslovno prevedeno na hrvatski, luk). U ovom radu koristit će se pojam "čvor" i "grana" osim kod izravnog pozivanja na literaturu u kojoj se rabi pojam "vrh".

XPath definira sedam osnovnih tipova čvorova u XML dokumentima:

- korijenski čvor,
- element,
- atribut,
- uputa za obradu,
- komentar,
- tekstualni čvor,
- imenički čvor.

Korijenski čvor (engl. *root node*) predstavlja korijen stabla. Svi čvorovi u stablu su potomci (engl. *descendants*) tj. podređeni čvorovi korijenskog čvora. U sebi ne sadrži nikakvu semantiku. Korijenski čvor obavezno sadrži barem jedno dijete: osnovni element XML dokumenta koji se često naziva i korijenski element (engl. *root element*). Pojmove korijenskog čvora i korijenskog elementa treba razlikovati. Druga djeca korijenskog čvora mogu biti komentari i upute za obradu koji su navedeni prije početka ili nakon završetka osnovnog elementa XML dokumenta.

Element (engl. *element*) je, uz korijenski čvor, jedini čvor koji može unutar sebe sadržavati druge čvorove. Pojmu elementa u XML-u ekvivalentan je pojam označavanja (*markup*) u SGML-u. Element je sadržan između početne i završne oznake. Početna i završna oznaka, odnosno oznaka praznog elementa nose ime elementa (u primjeru na slici 2.4 elementi su `kompozitor`, `prezime`, `ime` i `djelo`). Potčvorovi elementa mogu biti drugi elementi, tekstualni čvorovi, upute za obradu i komentari. U primjeru element `kompozitor` ima pet potčvorova: jedan podelement `prezime`, jedan podelement `ime`, dva podelementa `djelo` te jedan komentar. Element `prezime` ima jedan potčvor, tekstualni čvor `Verdi`. Element `ime` i oba elementa `djelo` imaju također svaki po jedan tekstualni potčvor. Poredak potčvorova je bitan, a neki element smije imati dva ili više potpuno istih potčvorova (npr. podelemenata). Svrha elemenata je definiranje strukture dokumenta. Ime elementa može sadržavati slovo engleske abecede, brojku i znakove '_', '-', '.' te neke posebne znakove. Početni znak u imenu elementa može biti samo slovo i znak '_'.

Atribut (engl. *attribute*) predstavlja par ime-vrijednost koji su pridruženi elementu. Svakom elementu pridružen je niz od nula ili više atributa čiji poredak nije bitan. Atributi nisu potčvorovi elementa pa se i ne navode između početne i završne oznake elementa, već unutar njegove početne oznake. Zapis atributa ima sljedeći oblik: najprije se navodi ime atributa, zatim slijedi znak jednakosti, a na kraju vrijednost atributa unutar navodnika. U primjeru podataka o kompozitoru bio je naveden atribut pridružen elementu `djelo` čije ime je "vrsta", a vrijednost "opera":

```
<djelo vrsta="opera">Traviata</djelo>
```

Imena atributa, koja smiju sadržavati iste znakove kao i imena elemenata, opisuju strukturu dokumenata, a vrijednosti njihov sadržaj. Vrijednost atributa može biti i prazan niz znakova (niz znakova duljine nula tj. `""`)

Upute za obradu (engl. *processing instructions*) sadrže instrukcije za programske aplikacije koje obrađuju dokumente nakon što su učitani i raščlanjeni (engl. *parsed*). Uputa za obradu počinje znakovima "<?", a završava znakovima "?>". Između ovih znakova nalazi se sadržaj upute. Prva riječ unutar instrukcije opisuje ciljnu aplikaciju (primjerice, uputa `<?perl lower-to-upper-case ?>` namijenjena je aplikaciji *perl*). Riječ *xml* je rezervirana za aplikaciju koja raščlanjuje XML dokument (npr. `<?xml version="1.0"?>`).

Komentar (engl. *comment*) sadrži neformalne primjedbe autora dokumenta njegovim korisnicima. Komentar počinje znakovima "<!--", a završava znakovima "-->". Između njih se nalazi sadržaj komentara. Komentar je zaseban čvor i nije ga moguće ubacivati bilo gdje unutar dokumenta (npr. unutar oznaka), već samo kao potčvor elementa ili korijenskog čvora.

Tekstualni čvor (engl. *textual node*) čine svi znakovi između dviju oznaka (početnih ili završnih oznaka elementa, oznaka praznog elementa, oznaka komentara ili uputa za obradu) koji sami nisu oznake. Tekstualni čvor je uvijek potčvor elementa. Ako taj element ima više potčvorova, neposredni prethodnik i sljedbenik tekstualnog čvora ne mogu biti tekstualni čvorovi, već samo elementi, komentari ili upute za obradu: upravo one vrste potčvorova čiji početak i kraj su definirani oznakom. Tekstualni čvorovi (uz atribute) nose sadržaj dokumenta. U danom primjeru XML dokumenta nalaze se četiri tekstualna čvora: "`Verdi`" kao potčvor elementa `prezime`, "`Giuseppe`" kao potčvor elementa `ime`, "`Traviata`" kao potčvor elementa `djelo` i "`Don Carlos`" kao potčvor elementa `djelo`. Posljednji tekstualni čvor sastoji se od dvije riječi, ali je riječ o jednom čvoru čije su granice definirane početnom oznakom elementa `djelo` i završnom oznakom elementa `djelo`. Razmak između riječi samo je jedan od ukupno 10 znakova koji čine čvor.

Primjer elementa koji sadrži više od jednog tekstualnog potčvora:

```
<fact><name>Wilde</name>, born in <city>Dublin</city>
<!--should I put Dublin, Ireland--> wrote the famous novel
<novel>Picture of Dorian Gray</novel></fact>
```

pokazuje kako su tekstualni potčvorovi razdvojeni drugim tipovima čvorova. Element `fact` sadrži šest potčvorova:

- podelement `name`,
- tekstualni čvor "`, born in `",
- podelement `city`,
- komentar "`should I put Dublin, Ireland`",
- tekstualni čvor "` wrote the famous novel `",
- podelement `novel`.

Kad element ima dva ili više tipova potčvorova njegov je sadržaj mješovit (engl. *mixed content*) [Xml04]. U XML dokumentima orijentiranim na prijenos sadržaja elementi u pravilu sadrže ili tekstualni potčvor ili podelemente. Naprotiv, kod dokumenata namijenjenih vizualnoj prezentaciji mješoviti sadržaj je vrlo čest.

Imenički čvorovi (engl. *namespace nodes*) rješavaju problem konflikta u imenovanju elemenata i atributa. Kako XML dozvoljava autorima dokumenata da nazivaju elemente i atribute prema vlastitoj želji, velika je vjerojatnost da će dokumenti različitih autora sadržavati elemente (i atribute) jednakih imena, a različitog značenja i strukture (tj. drukčijih podelemenata) ukoliko se radi o dokumentima slične tematike ili o imenima vrlo općenitog značenja (npr. engleske riječi *item* i *record*). Želi li korisnik nadograditi XML dokumente drugih korisnika, može nastati konflikt u imenovanju elemenata i atributa. Primjer ovakvog konflikta je ubacivanje vlastitog elementa `kompozitor` koji u XML dokument u kojem već postoji oznaka `kompozitor` definirana od drugog korisnika. Ukoliko novi korisnik definira svoj **imenik** (engl. *namespace*) [Sch0-04, Sch1-04, Sch2-04], konflikt će biti izbjegnut. Bilo bi vrlo dobro da i autor izvornog dokumenta definira svoj vlastiti imenik. Jedinstvenost i trajnost imenika jamči njegov URI (engl. *Uniform Source Identifier*), skup znakova koji jednoznačno identificira informacijski resurs, u ovom slučaju imenik. Svaki se imenik (tj. URI) u dokumentu pojavljuje s različitim prefiksom. Jedan imenik u dokumentu smije biti bez prefiksa. U primjeru na slici 2.5 prefiksom *mb* referencira se imenik jedinstvenog identifikatora *`http://marko.banek.net/namespaces/mb`*. Pripadnost elementa `prezime` imeniku navodi se tako da se u početnoj i završnoj oznaci ispred imena bez razmaka navode prefiks i dvotočka (`<mb:prezime>`).

```
<kompozitor xmlns:mb="http://marko.banek.net/namespaces/mb">
     <mb:prezime>Verdi</mb:prezime>
     <!--ostatak izostavljen-->
</kompozitor>
```

Slika 2.5. Uvođenje imenika u XML dokument

Slično nizu atributa, svakom je elementu pridružen niz od nula ili više imeničkih čvorova (čiji poredak je nebitan). Svaki takav čvor pojedinom prefiksu pridružuje URI jednog imenika. Imenički čvor se, poput atributa, navodi unutar početne oznake elementa kojem pripada. Imenički čvor počinje znakovima "xmlns" nakon kojih slijede dvotočka, prefiks pa znak jednakosti. Nakon znaka jednakosti pod navodnicima se navodi URI imenika: `xmlns:mb="http://marko.banek.net/namespaces/mb"`. Ako imenik nema prefiks, poslije znakova "xmlns" odmah slijedi znak jednakosti: `xmlns="http://noprefix.com/bestnamespace"`. Pridruživanje prefiksa imenicima ostvareno imeničkim čvorom u sklopu nekog elementa odnosi se na taj element i sve njegove podelemente. U prethodnom primjeru dodjeljivanje imenika prefiksu `mb` vrijedi za čvorove `kompozitor` i `prezime`. Unutar podelemenata moguće je novim imeničkim čvorom istom prefiksu dodijeliti drugi imenik. Nova dodjela odnosi se na taj podelement i na sve njegove podelemente.

2.3.3. Usporedba modela OEM i modela za XML (XPath)

Polustrukturiranost podataka može se postići izrazi li ih se u formatu XML. Međutim, opisani podatkovni model koji nudi XPath ne odgovara u potpunosti modelu OEM kao polaznom modelu za polustrukturirane podatke. Osnovna razlika je postojanje poretka među potčvorovima nekog elementa u XML-u dok je u modelu OEM poredak bridova koji izlaze iz istog čvora nebitan. Dvije strukture u formatu XML prikazane na slici 2.6 zbog različitog poretka elemenata `prezime` i `ime`, koji su potčvorovi elementa `kompozitor`, nisu međusobno ekvivalentne.

```
<kompozitor>
     <prezime>Verdi</prezime>
     <ime>Giuseppe</ime>
</kompozitor>
```

```
<kompozitor>
     <ime>Giuseppe</ime>
      <prezime>Verdi</prezime>
</kompozitor>
```

Slika 2.6. Međusobno neekvivalentne strukture u XML-u zbog različitog poretka među elementima

Uvođenje atributa u XML (koje je bilo nužno zbog kompatibilnosti s HTML-om) predstavlja proširenje modela OEM. Doduše, atributi nekog elementa se mogu pretvoriti u podelemente tog istog elementa. Izraz na slici 2.7 bi sa stajališta modela OEM odgovarao izrazu na slici 2.8.

```
<kompozitor prezime="Verdi" ime="Giuseppe"/>
```

Slika 2.7. Korištenje atributa u XML-u

```
<kompozitor>
     <prezime>Verdi</prezime>
     <ime>Giuseppe</ime>
</kompozitor>
```

Slika 2.8. Pretvaranje atributa nekog XML elementa u podelemente

2.4. Gramatika za XML

Struktura XML dokumenta može biti unaprijed zadana u posebnoj datoteci koja se naziva predloškom ili gramatikom dokumenta. Gramatika navodi:

- koji elementi postoje u XML dokumentu
- način ugnježđivanja elemenata tj. njihovog smještanja jednog unutar drugog
- poredak podelemenata nekog elementa te koliko puta se pojedini podelement pojavljuje unutar tog elementa
- atribute za pojedine elemente
- eventualno, kakav smije biti sadržaj elemenata (tj. tekstualni čvorovi) i vrijednosti atributa (tj. kakve nizove znakova smiju sadržavati)
- eventualno, koji imenici se pojavljuju u dokumentu.

Od opisanih sedam tipova čvorova u modelu za XML dokument predložak definira četiri: elemente (opisuju strukturu), atribute (imena atributa opisuju strukturu, a njihove vrijednosti su podatkovni sadržaj), tekstualne čvorove (sadržaj) i imeničke čvorove (sudjeluju u opisu strukture, ali i dozvoljenih vrijednosti sadržaja).

Primjer predloška je gramatika jezika XHTML. Ona definira sve elemente i atribute koji postoje u tom jeziku: elemente koji opisuju zaglavlje i tijelo dokumenta, različite tipove naslova, podebljani i kurzivni tekst, novi redak itd.

Dobro oblikovani XML dokument (engl. *well-formed document*) je XML dokument koji slijedi sva pravopisna pravila (pravila formata) XML-a. To znači da broj početnih i konačnih oznaka u svakom dokumentu mora biti isti. Svaki započeti element mora biti završen (za razliku od HTML-a gdje to nije bilo potrebno).

XML dokument je **valjan** (engl. *valid*) ukoliko:

- je dobro oblikovan
- za njega postoji definiran predložak (shema) u obliku posebne datoteke
- struktura dokumenta poštuje sva ograničenja zadana predloškom.

Postoje dva moguća zapisa predloška za XML dokument:

- Document Type Definition
- XML Schema.

Dva načina zapisivanja rezultat su povijesnog razvoja XML-a. DTD je stariji i jednostavniji, ali definira svoj posebnu notaciju. XML Schema je složenija, ali je i sama XML dokument. U nastavku će format zapisa i razlike s obzirom na mogućnost preciznog definiranja podatkovnih struktura biti prikazani na konkretnom primjeru narudžbe robe (DTD na slici 2.9, XML Schema u dodatku A1). XML dokument koji opisuje narudžbu robe sastavni je dio specifikacije XML Scheme [Sch0-04] i nalazi se u dodatku A2.

2.4.1. Document Type Definition - DTD

Document Type Definition (DTD) je starija varijanta predloška za XML dokumente razvijana paralelno sa samim formatom, bazirana na načelima preuzetim iz SGML-a. DTD ima notaciju u tzv. Backus-Naurovoj proširenoj formi (engl. *Extended Backus-Naur Form*, EBNF). DTD za svaki element može definirati listu podelemenata i/ili atributa. Može ga definirati i kao element s tekstualnim čvorom ili mješovitim sadržajem ili pak kao prazni element. Za model narudžbe robe opisan XML dokumentom u dodatku A2 pripadajući je DTD dan na slici 2.9.

Iz linije

```
<!ELEMENT purchaseOrder (shipTo, billTo, comment?, items)>
```

se može zaključiti da element `purchaseOrder` redom sadrži podelemente `shipTo`, `billTo`, `comment` i `items`. Redoslijed podelemenata upravo je jednak redoslijedu njihovog navođenja u ovoj deklaraciji: najprije dolazi element `shipTo`, potom `billTo` i `comment`, a na kraju `items`. Pod kardinalnošću nekog elementa podrazumijeva se broj pojavljivanja tog elementa unutar njemu nadređenog elementa. DTD razlikuje sljedeće kardinalnosti pojavljivanja podelementa unutar njemu nadređenog elementa:

- točno jednom,
- niti jednom ili jednom,
- jednom ili više puta,
- niti jednom, jednom ili više puta.

```
<!DOCTYPE purchaseOrder [
  <!ELEMENT purchaseOrder (shipTo, billTo, comment?, items)>
  <!ATTLIST purchaseOrder orderDate CDATA #IMPLIED>
  <!ELEMENT shipTo (name, street, city, state, zip)>
  <!ATTLIST shipTo country NMTOKEN #FIXED "US">
  <!ELEMENT billTo (name, street, city, state, zip)>
  <!ATTLIST billTo country NMTOKEN #FIXED "US">
  <!ELEMENT items (item*)>
  <!ELEMENT item (productName, quantity, USPrice, comment?,
shipDate?)>
  <!ATTLIST item partNum NMTOKEN #REQUIRED>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT street (#PCDATA)>
  <!ELEMENT city (#PCDATA)>
  <!ELEMENT state (#PCDATA)>
  <!ELEMENT zip (#PCDATA)>
  <!ELEMENT productName (#PCDATA)>
  <!ELEMENT quantity (#PCDATA)>
  <!ELEMENT USPrice (#PCDATA)>
  <!ELEMENT comment (#PCDATA)>
  <!ELEMENT shipDate (#PCDATA)>
]>
```

Slika 2.9. DTD za dokument koji opisuje narudžbu robe

Notacija operatora za označavanje kardinalnosti preuzeta je iz proširene Backus-Naurove forme (tablica 2.1).

Tablica 2.1. Operatori za označavanje kardinalnosti podelemenata kod DTD.

Kardinalnost	Operator
točno jednom	[bez oznake operatora]
niti jednom ili jednom	?
jednom ili više puta	+
niti jednom, jednom ili više puta	*

Tako se `comment` kao podelement od `purchaseOrder` odnosno podelement od `item` može pojaviti jednom, ali se ne mora pojaviti uopće. Element `items` može sadržavati jedan ili više podelemenata `item`, ali ne mora uopće sadržavati niti jedan.

Za razliku od podelemenata, atribut nekog elementa ne može se pojaviti više od jedanput, no moguće je da se uopće ne mora pojaviti.

Linija

```
<!ATTLIST item partNum NMTOKEN #REQUIRED>
```

navodi da lista atributa elementa `item` sadrži atribut `partNum`. Ključna riječ `#REQUIRED` označava da uz svaku pojavu elementa `item` u XML dokumentu koji odgovara ovom predlošku nužno mora pojaviti atribut `partNum`. Suprotno od toga, linijom

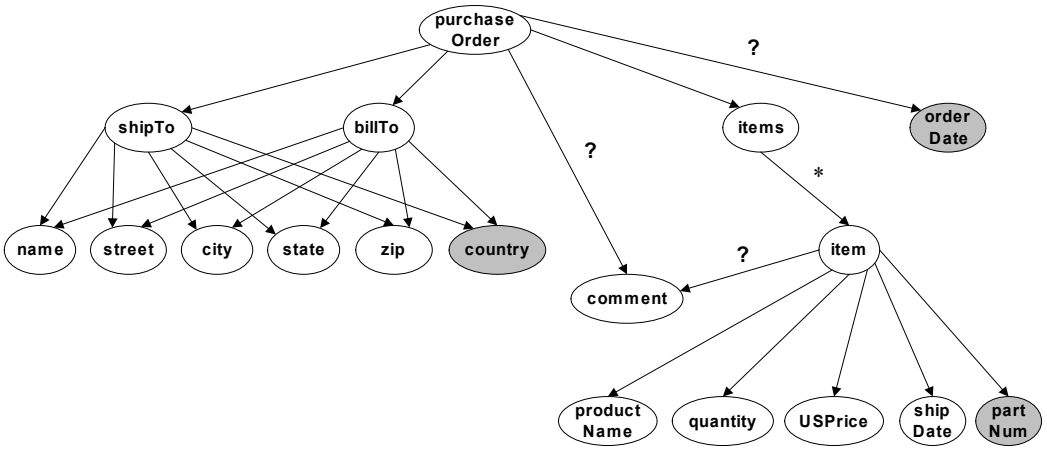
```
<!ATTLIST purchaseOrder orderDate CDATA #IMPLIED>
```

označava se da element `puchaseOrder` može, ali i ne mora imati pridružen atribut `orderDate`. Stoga se može govoriti i o kardinalnosti atributa. Kardinalnost atributa vezana je uz pitanje je li pojava atributa propisana (analogno elementu koji se

pojavljuje točno jednom), ili je moguća, no ne i obavezna (analogno elementu koji se pojavljuje niti jednom ili jednom. Na ovaj način objedinjuje se pojam kardinalnosti za elemente i atribute, pa se svaki atribut (poredak za atribute nije bitan) može promatrati kao element.

Tekstualni sadržaj opisuju ključne riječi `#PCDATA`, `CDATA` za osnovni, tekstualni tip podataka (engl. *string type*). Pobrojani tipovi (engl. *enumerated types*) opisuju listu koja se sastoji od osnovnog tekstualnog tipa. Dodatno proširenje nude znakovni tipovi (engl. *tokenized types*): `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`, `NMTOKEN`, `NMTOKENS` dobivene od osnovnog znakovnog odnosno pobrojanog tipa različitim ograničenjima u dozvoljenom pravopisu i semantici. `CDATA` (*Character DATA*) označava bilo kakav niz znakova (slova), uključujući i znakove koje bi programski alat za raščlambu XML dokumenta smatrao označavanjem (*markup*). Prema tome, tekst opisan s `CDATA` može sadržavati čitav XML dokument. `#PCDATA` znači *Parsed Character DATA* (pročitani i raščlanjen tekstualni podaci). Takav sadržaj ne smije u sebi imati znakove koji opisuju označavanje, već samo znakove koje će program za raščlambu uspješno interpretirati.

Struktura XML dokumenta koju definira pripadajući DTD može se prikazati u obliku grafa. Čvorovi grafa su elementi i atributi. Svakoj je grani pridružen operator kardinalnosti podelementa odnosno atributa. Između elemenata i atributa nema razlike, osim što su, zbog lakšeg uočavanja, atributi prikazani sivo. Grafički prikaz čvorova elemenata i atributa koji definira DTD sa slike 2.9 dan je na slici 2.10. Elementi su poredani slijeva udesno, a atributi su sasvim na desnoj strani, na kraju poretka. DTD dozvoljava da bilo koji deklarirani element bude korijenski element XML dokumenta.



Slika 2.10. Prikaz strukture XML dokumenta dobiven analizom DTD-a sa slike 2.9

Načelo primarnog i stranog ključa ostvaruje se pridjeljivanjem identifikatora i reference na identifikator. Identifikator, `ID`, ima jedinstvenu vrijednost unutar dokumenta (tj. tu vrijednost smije imati samo jedan element ili atribut danog imena u dokumentu) i opisuje primarni ključ. Referenca na identifikator, `IDREF`, je strani ključ i odnosi se na jedan i samo jedan identifikator. Loša strana ovog mehanizma jest povezivanje identifikatora i reference isključivo na temelju vrijednosti identifikatora. Nije moguće ograničiti ključ na element točno određenog tipa. Zamijene li se zabunom za dva para `ID-IDREF` vrijednosti pridružene `IDREF`, nije moguće otkriti

grešku. Dapače, svaka referenca `IDREF` smije poprimiti vrijednost bilo kojeg identifikatora `ID` u dokumentu.

DTD ne definira nikakav brojčani (numerički) tip podataka. Kod pridruživanja vrijednosti pojedinom elementu ili atributu ne nudi se mogućnost razlikovanja brojeva od običnog teksta. Osim toga, korisnik ne može po želji ograničiti maksimalni broj znakova ili dozvoliti samo neke znakove u tekstu.

2.4.2. XML Schema

Novija vrsta predloška za XML dokumente je XML Schema. Za razliku od DTD-a, koji svoju sintaksu i notaciju nasljeđuje iz SGML-a, XML Schema je definirana isključivo kao predložak za XML. Svaka XML Schema je i sama XML dokument, čime se omogućuje njena jednostavnija interpretacija: dovoljno je upotrijebiti standardni programski alat za čitanje i raščlambu (engl. *parsing*) XML dokumenata.

Nakon što je organizacija W3C 1998. službeno standardizirala format XML, pokazalo se da su u mnogim slučajevima njegove upotrebe potrebne znatno preciznije definicije dozvoljenih podatkovnih vrijednosti nego što to omogućuje DTD (kako je ranije napomenuto, jedan od osnovnih nedostataka DTD-a je nepostojanje bilo kakvog numeričkog tipa podataka). Isto tako, DTD je kod opisa ugnježđivanja elemenata dozvoljavao samo četiri tipa kardinalnosti pridruživanja podelementa elementu (tablica 2.1). U nekim slučajevima nije dovoljno da se specificira kardinalnost pridruživanja "jednom ili više puta", nego se traži točan broj. Osim što DTD nije omogućio definiranje tipova podatkovnog sadržaja nije dozvoljavao niti definiciju tipa strukture koja bi se poslije koristila više puta. Zbog toga je W3C potaknuo razvijanje XML Scheme kao novog tipa predloška. XML Schema je proglašena radnim nacrtom (engl. *working draft*) W3C početkom 2000. da bi u svibnju 2001. postala standard. Specifikacija XML Scheme [Sch0-04, Sch1-04, Sch2-04] u sebi uključuje i pojam imenika (*namespaces*) koji nisu dio u specifikacije XML 1.0 Second Edition [Xml00] već su definirani zasebno. Pojam XML Schema ne spominje se niti u posljednjoj verziji specifikacije iz veljače 2004, XML 1.0 Third edition [Xml04] i očito neće biti dio XML 1.0 u bilo kojoj reviziji.

U dodatku A1 nalazi se XML Schema koja definira ranije spomenuti XML dokument za narudžbu robe u dodatku A2. XML Schema je preuzeta iz [Sch0-04].

U nastavku će biti nabrojene dodatne mogućnosti u definiranju predloška za XML dokumente koje omogućuje XML Schema u odnosu na DTD.

- **Definiranje tipova podataka za tekstualni sadržaj**. XML Schema uvodi 44 različita tipa podataka za vrijednosti koje mogu poprimiti tekstualni čvorovi. Od toga je čak 19 primitivnih tipova, a preostalih 25 tipova izvedeno je iz primitivnih. Važno je da XML Schema uz osnovni znakovni primitivni tip `string` (i iz njega izvedene tipove) podržava i propisuje brojne numeričke tipove te tipove koji opisuju vrijeme, vremensko trajanje i datume. Uz to, XML Schema podržava i znakovne tipove koje definira DTD-a (*tokenized types*: ID, IDREF, NMTOKEN itd.). Od svakog osnovnog tipa korisnik može izvesti vlastite jednostavne ili pobrojane tipove. To se postiže ograničavanjem duljine riječi odnosno broja znamenki, postavljanjem maksimalnih i minimalnih vrijednosti, svođenjem domene na skup diskretnih vrijednosti itd. Ovakvi tipovi podataka u XML Schemi nazivaju se jednostavnim tipom (engl. *simple type*).

- **Definiranje složene strukture kao tipa podataka**. DTD je zahtijevao da se za svaki element struktura definira zasebno, premda više od jednog elementa u dokumentu može imati jednaku strukturu podelemenata i jednaku listu atributa. U XML dokumentu iz dodatka A2 identične podelemente (s identičnim poretkom) imaju elementi `shipTo` i `billTo`. To pokazuju i linije DTD-a:

  ```
  <!ELEMENT shipTo (name, street, city, state, zip)>
  <!ATTLIST shipTo country NMTOKEN #FIXED "US">
  ```

  odnosno

  ```
  <!ELEMENT billTo (name, street, city, state, zip)>
  <!ATTLIST billTo country NMTOKEN #FIXED "US">.
  ```

  Iako su strukture potpuno jednake, DTD ne omogućuje da se to definira formalno. XML Schema, naprotiv, omogućuje definiranje složene strukture podelementa (koji sami mogu sadržavati podelemente bilo koje razine ugniježđenosti) i liste atributa pod imenom složenog tipa (engl. *complex type*) čime svi elementi tog tipa postanu formalno jednaki po strukturi. U nekim primjenama XML dokumenata i njihovih predložaka podudaranje struktura može biti vrlo važno, kao što je slučaj s izravnim oblikovanjem skladišta podataka iz XML izvora koje se opisuje u ovom radu.

- **Mogućnost preciznog izražavanja kardinalnosti pridruživanja podelemenata nadređenim elementima**. DTD je definirao samo četiri osnovna tipa kardinalnosti pridruživanja (tablica 2.1), pri čemu se minimalni broj pridruženih podelemenata mogao postaviti na nulu, ali se niti minimalni broj niti maksimalni broj pridruženih podelemenata nije mogao točno zadati ukoliko je bio veći od jedan (DTD dozvoljava samo oblik "jedan ili više"). XML Schema omogućuje da se za minimalnu i maksimalnu kardinalnost pridruživanja postavi bilo koji prirodni broj (u slučaju minimalne vrijednosti i nula, tj. pridruživanje nije obavezno). Za maksimalnu kardinalnost pridruživanja dozvoljava se i vrijednost "neograničeno mnogo".

- **Podržavanje koncepta imenika**. Svi dozvoljeni elementi i atributi XML Scheme se nalaze u imenicima koje identificiraju identifikatori URI `http://www.w3.org/2001/XMLSchema` odnosno `http://www.w3.org/2001/XMLSchema-instance`. Korisnik može sve strukture koje definira XML Schemom pridružiti bilo kojem imeniku (osim dvama upravo navedenima koji opisuju dozvoljene elemente i atribute XML Scheme). XML Schema omogućuje korisniku preuzimanje elemenata, atributa i korisnički definiranih jednostavnih i složenih tipova iz drugih XML Schema. Ukoliko su oni definirani kao sastavni dio nekog imenika različitog od korisnikovog, korisnik u svojoj shemi može definirati strukture istog imena, ali kao dio svog imenika. Na taj se način omogućuje korisniku da samo nadogradi već postojeće strukture koje su mu stavljene na raspolaganje, a pritom mu se ostavlja potpuna sloboda davanja imena tim strukturama.

- **Jednoznačni mehanizam primarnog i stranog ključa**. XML Schema omogućuje da neki element jednostavnog tipa ili atribut poprimi jedinstvenu vrijednost unutar cijelog XML dokumenta ili dijela dokumenta. Time se na taj element ili atribut postavlja ograničenje primarnog ključa (ukoliko mu vrijednost ne smije biti nul-vrijednost) ili ograničenje jedinstvene vrijednosti

(engl. *unique constraint*, u slučaju da element može poprimiti nul-vrijednost). Primarni ključ može biti referenciran vrijednošću drugog elementa ili atributa, ali isključivo istog tipa. Ključevima se pridjeljuju imena, pa se jednoznačno može odrediti na koji se element ili atribut (koji nosi vrijednost primarnog ključa) odnosi strani ključ. To predstavlja značajno poboljšanje u odnosu na DTD gdje su primarni i strani ključevi bili zasebni tipovi podataka, ID i IDREF, a strani ključ, IDREF, se mogao referencirati na bilo koji ID unutar dokumenta.

DTD je znatno jednostavnija vrsta predloška za XML dokumente. Način koji DTD opisuje njihovu strukturu (pri definiciji elementa deklariraju se podelementi) je korisniku vrlo intuitivan. Analiza XML Scheme znatno je složenija i manje intuitivna jer je svaka definicija obično skup od barem 3 ugniježđena elementa, dok se u DTD-u sastoji od jednog retka. Analizu otežava i činjenica da se strukture mogu definirati na više načina. Usprkos tome, pokazalo se da su u ovom poglavlju opisane mogućnosti kvalitetnijeg i detaljnijeg definiranja predloška za XML dokumente često nužne, tj. da je njihovo nepostojanje kod DTD-a zapravo ozbiljan nedostatak. Zbog toga se XML Schema sve više koristi i sve više iz upotrebe potiskuje DTD.

2.5. Pohrana XML dokumenata

Nakon što je XML u praksi postao standard za razmjenu podataka među heterogenim informacijskim sustavima i mrežnu razmjenu općenito, pojavila se potreba za organiziranom pohranom dokumenata u tom formatu. Osnovni način pohrane XML dokumenta podrazumijeva pohranu u obliku datoteke. Ako se pohranjena datoteka želi analizirati ili se želi promijeniti njen sadržaj i struktura, potrebno ju je najprije pročitati i raščlaniti (engl. *parse*). Čitanje i raščlamba zahtijeva procesorske resurse i utrošak vremena, što kod mnogih velikih informacijskih sustava može značajno narušiti performanse. Dodatni utrošak procesorskih resursa i vremena zahtijevat će i usporedba s XML Schemom ili DTD-om, ukoliko nije obavljena ranije. Nakon svake promjene podaci se ponovno pohranjuju u obliku datoteke na disk računala (ili na vanjski medij) što iznova zahtijeva utrošak procesorskih resursa i vremena. Budući da se kod velikih XML dokumenata u praksi odjednom obrađuje samo njegov mali dio, vrlo je neefikasno svaki put čitati i raščlaniti cijeli dokument.

Mogućnost izravnog pristupa dijelu dokumenta (nekom elementu tj. njegovoj tekstualnoj vrijednosti ili atributima, neovisno o nadređenim i podređenim elementima) mogla bi znatno poboljšati performanse sustava koji obrađuju XML dokumente. Ovakav pristup mogao bi se postići ako bi se dokumenti pohranili u baze podataka specijalizirane za njihovu obradu. Vodeći proizvođači relacijskih baza podataka danas nude podršku za spremanje i obradu XML dokumenata. Istodobno, na tržištu su se pojavile i baze podataka specijalizirane isključivo za rad s XML dokumentima (tzv. izvorne baze za XML - *native XML databases*). Rezimirajući opisana načela i mogućnosti pohrane XML dokumenata, postoje četiri osnovna načina pohrane XML dokumenata, od kojih svaki nudi različite mogućnosti njihove obrade:

- pohrana u obliku datoteka,
- pohrana u objektno-relacijskoj bazi u vidu "velikih objekata",
- pohrana u objektno-relacijskoj bazi uz preslikavanje strukture XML dokumenta na relacije,
- pohrana u izvornoj bazi za XML.

Svakom od ova četiri načina pohrane navest će se osnovne prednosti i nedostaci.

2.5.1. Pohrana XML-a u obliku datoteka

Pohrana u obliku datoteka najstarija je i najjednostavnija te ne zahtijeva dodatne memorijske i programske resurse baze podataka. Kako je ranije naglašeno, glavni nedostatak ovakve pohrane, uz međusobnu nepovezanost sadržaja pohranjenog u različitim datotekama, jest nužnost čitanja i raščlambe dokumenta prilikom svake obrade. Pored toga, datotečni sustav ne nudi sigurnost i zaštitu podataka te istodobno paralelno korištenje od strane velikog broja korisnika što inače omogućuju baze.

2.5.2. Pohrana u objektno-relacijskoj bazi u vidu "velikih objekata"

Potreba spremanja multimedijskog sadržaja imala je za posljedicu dodavanje funkcionalnosti za objektne sadržaje relacijskim bazama podataka. Pritom je pristup podacima u bazi i dalje u potpunosti relacijski, a baza čuva integritet objekta kao jedinstvene cjeline (eventualno mogu postojati funkcije za njihovu dodatnu obradu). Svi vodeći proizvođači komercijalnih sustava za upravljanje bazom podataka definiraju jedan ili više tipova "velikih objekata" (engl. *large objects*, LOB). Oni omogućuju spremanje mnogo veće količine memorije nego klasični numerički, znakovni i datumski tipovi. Razlikuju se veliki objekti za pohranu netekstualnih datoteka (tipično slika, zvuk, PDF ili Microsoftovi formati) od onih za pohranu tekstualnog sadržaja (kodiranje prema standardima ASCII ili UTF što uključuje i XML dokumente standardno zapisane u formatima UTF-8 ili UTF-16).

Ovakav način pohrane pruža sve prednosti koje inače nudi pohrana u baze podataka u odnosu na obični sustav datoteka: veći stupanj sigurnosti i mogućnost kontroliranog paralelnog korištenja od stane većeg broja korisnika. Glavna mana ovakvog načina pohrane je neprepoznavanje ugniježđene strukture XML dokumenta: cijeli se dokument promatra kao nedjeljiv niz znakova. Nije moguće upitima u SQL-u dohvatiti ili promijeniti dio dokumenta. Za najmanju promjenu potrebno je izvući čitav tekst i obraditi ga, a potom prebrisati stari sadržaj novim. U slučaju da je riječ o XML dokumentu, to podrazumijeva čitanje i raščlambu dokumenta, a time i trošak vremena te procesorskih resursa.

2.5.3. Pohrana u objektno-relacijskoj bazi uz preslikavanje strukture XML dokumenta na relacije

Integracija XML-a u postojeće objektno-relacijske baze podrazumijeva proširenja koja će omogućiti da se svakom elementu ili atributu nekog XML dokumenta pohranjenom u bazu pristupi izravno pomoću upita u SQL-u te se po potrebi promijeni njihova vrijednost ili struktura podelemenata.

Brojni autori predložili su rješenja za pohranu XML strukture u relacijsku bazu. Treba spomenuti način pohrane za strukture bez predloška (XML Scheme ili DTD-a) predložen u [FK99], odnosno strukture s predloškom [STH+99].

Za dohvaćanje i izmjenu fragmenata XML dokumenata ili njegovih dijelova u izradi je standard SQL/XML [SX] kao dio šireg standarda SQL:2003. Vodeći svjetski proizvođači sustava za upravljanje objektno-relacijskih bazama podataka (IBM s bazom IBM DB2 XML Extender, Oracle s bazom Oracle 9i Release2 XML DB, Microsoft s bazom SQL Server 2000) nude mogućnost spremanja podataka u XML formatu u objektno-relacijsku bazu podataka te potom dohvaćanje cjelokupnih XML dokumenata ili njihovih dijelova te njihovu izmjenu pomoću upita u SQL-u.

Baza Oracle 9i Release2 XML DB definira tip podataka XMLType, čiji sadržaj može biti bilo kakav fragment XML dokumenta. XMLType se bazira na modelu DOM i dohvaća upitima u SQL temeljenim na standardu SQL/XML. XMLType omogućuje pohranu XML dokumenata bez predloška.

2.5.4. Pohrana u izvornu bazu za XML

Izvorne baze (sustavi za upravljanje bazom podataka) za XML oblikovane su upravo za pohranu XML dokumenata, za razliku od objektno-relacijskih baza koje za XML nude samo proširenja.

Temeljna jedinica pohrane u izvornim bazama za XML je XML dokument. Logički model baze mora razlikovati elemente, atribute i tekstualne čvorove te uzimati u obzir poredak elemenata. Fizički model može se temeljiti na hijerarhijskoj, relacijskoj, objektno-relacijskoj bazi podataka ili drugom načinu pohrane (posebni sustavi datoteka prilagođeni ovoj namjeni).

Izvorne baze za XML imaju sve odlike baza podataka: nude istodobni paralelni pristup za više korisnika, definiraju transakcijske operacije, štite podatke od nedozvoljenog pristupa.

Poznatije izvorne baze za XML su Tamino, Ipedo, eXcelon, dbXML te ranije spomenuti sustav Lore.

2.6. Upitni jezik XML Query

Upitni jezici za polustrukturirane podatke i podatke u formatu XML omogućuju njihov dohvat, ispis, obradu, izmjenu i pohranu bez poznavanja specifične programske podrške za čitanje, raščlambu i promjene podataka u XML-u. Dovoljno je znati sastaviti upit i imati program koji ga izvršava.

Poznati su jezici Lorel, UnQL (oba općenito za polustrukturirane podatke), YATL (najprije za polustrukturirane podatke, kasnije tijekom razvoja prilagođen XML-u), XML-QL, XML-GL, XQL, Quilt i XML Query.

Jezik XML Query (ili XQuery), [Xquery04], zasnovan je na jeziku Quilt, a razvijaju ga radne skupine *W3C XML Query Working Group* (koja se bavi upravo razvojem upitnih jezik za XML) i *W3C XSL Working Group* (koja se bavi razvojem XSLT, predloška za pretvaranje XML dokumenata u druge formate i jezike, primjerice HTML) u sklopu normativne organizacije W3C i trenutno ima status radnog nacrta (engl. *working draft*).

XQuery za imenovanje elemenata i atributa unutar dokumenata koristi jezik XPath [XPath99].

Povratni rezultat upita je XML element proizvoljno ugniježđene strukture koju korisnik prema svojim željama definira u samom upitu. Struktura upita temelji se na izrazu FLWOR (`FOR-LET-WHERE-ORDER BY-RETURN`). Za objašnjenje ovih ključnih riječi potrebno je promatrati XML dokument po.xml u dodatku A2.

Izraz `FOR` omogućuje iteraciju, a zasnovan je na standardnoj petlji kakve se koriste u mnogim proceduralnim programskim jezicima (Pascal, Basic, C, C++, Java). Izraz `LET` definira varijable unutar petlje `FOR`. `WHERE` predstavlja ograničenje nad čvorovima (elementima ili atributima) koje vraća petlja `FOR`. `ORDER BY` omogućuje da se čvorovi koje vraća petlja poredaju po određenom pravilu (uzlazno, `ascending`, ili silazno, `descending`, po brojčanom iznosu ili abecednom redu, a s obzirom na

jedan ili više elemenata i atributa). RETURN definira XML element (najčešće ugniježđenu strukturu) kojim se vrši ispis za svakog člana petlje. U upitu na slici 2.11, za svaki se element purchaseOrder/items/item u varijablu $p sprema njegova cijena (USPrice). Za one elemente purchaseOrder/items/item čija je cijena manja od 200 dolara ($p<200) ispisuje se XML element cheap koji sadrži atribut serialNumber čija vrijednost odgovara vrijednost atributa partNum (oznaka atributa u XPathu je @) danog elementa purchaseOrder/items/item.

```
<result>
 {
  for $i in document("po.xml")/purchaseOrder/items/item
  let $p=$i/USPrice
  where $p<200 order by USPrice descending
  return
    <cheap serialNumber="{$i/@partNum}">
      {$p}
    </cheap>
 }
</result>
```

Slika 2.11. Standardni upit u jeziku XQuery oblika FLWOR

Odgovor na upit prikazuje slika 2.12.

U izrazu FLWOR obavezni su samo FOR i RETURN., dok je postojanje LET, WHERE i ORDER BY uvjetno.

```
<result>
    <cheap serialNumber="872-AA">
        <USPrice>148.95</USPrice>
    </cheap>
    <cheap serialNumber="926-AA">
        <USPrice>39.98</USPrice>
    </cheap>
</result>
```

Slika 2.12. Odgovor na standardni upit oblika FLWOR sa slike 2.11

3. Oblikovanje područnog skladišta podataka iz XML izvora

3.1. Primjena polustrukturiranih podataka u razmjeni informacija

Prikupljanje podataka putem Interneta i njihova pohrana u skladište mogu se svesti na dva osnovna koncepta: pohranu internetskog sadržaja i pohranu sadržaja koji ne postoji kao javno dostupan na Internetu nego je rezultat dogovorene mrežne razmjene s drugom stranom. Metapodaci u oba slučaja igraju ključnu ulogu.

Pohrana ogromne količine dostupnog sadržaja *web*-stranica u skladišta podataka opisuje se engleskim pojmom ***web warehousing***. Analiza metapodataka *web*-stranice omogućuje da se izdvoji odgovarajući sadržaj. Budući da se u novije vrijeme izgled stranice (engl. *presentation*) definira u posebnom dokumentu, a sadržaj zadaje u formatu XML, pohrana sadržaja često se svodi na pohranu podataka u XML-u.

Drugi oblik mrežne razmjene obuhvaća organizacije (poduzeća) koje razmjenjuju podatke između svojih zatvorenih informacijskih sustava koji se oslanjaju na baze podataka. Izbor sustava za upravljanje bazom podataka, format podataka pohranjenih u bazi te format namijenjen internoj razmjeni dokumenata unutar organizacije odabrani su tijekom izgradnje informacijskog sustava i razlikuju se u odnosu na druge organizacije. Za razmjenu podataka između opisanih heterogenih podatkovnih sustava potreban je oblik podataka koji će omogućiti efikasan prijenos sadržaja uz veliku prilagodljivost (fleksibilnost) i što jednostavnije metapodatke. To su odlike polustrukturiranih podataka. XML, danas najrašireniji polustrukturirani podatkovni format, dominantan je u razmjeni podataka između različitih informacijskih sustava, a osobito u razmjeni putem mreže. Njegova pogodnost je i mogućnost definiranja predložaka u vidu posebnog, izdvojenog dokumenta (DTD-a ili XML Scheme). Za razliku od baza podataka, čiji sadržaj je potrebno pregledavati posebnim sučeljem, XML se može bez ikakvih transformacija ispisati na papir ili zaslon i biti na raspolaganju korisniku za čitanje. Naime, imena elemenata i atributa obično jesu ili (uz korištenje interpunkcijskih znakova) ukazuju na pojmove prirodnog ljudskog jezika, čime su razumljivi čovjeku (*human-readable*). Sadržaj pristigao u sustav mrežnom razmjenom u obliku XML dokumenta može se po dolasku u sustav pohraniti u bazu podataka ili kao obična datoteka.

Primjer razmjene podataka u formatu XML između različitih informacijskih sustava je sustav **ebXML** (*e-business XML* – XML za elektroničko poslovanje) [Ebxml01]. Velikim poslovnim organizacijama ovaj sustav služi kao potpora elektroničkom poslovanju. Omogućuje uspostavu poslovanja između pojedinih tvrtki (*Business to Business*, B2B) koje do tada nisu imale uspostavljenu suradnju, te dalje proširivanje suradnje. Svaka tvrtka izrađuje profil svog poslovanja koji se u obliku XML dokumenta, koji odgovara određenom unaprijed definiranom predlošku (XML Schemi), sprema u središnji repozitorij podataka odakle je dostupan svim zainteresiranim stranama. Nakon ostvarenog kontakta dvije ili više tvrtki na temelju sadržaja u profilima poslovanja definiraju opis ugovorenog posla. To je ponovno XML dokument.

3.2. Izravna pohrana sadržaja u formatu XML u skladište podataka

3.2.1. Paralelan razvoj sustava za elektroničko poslovanje i popratnog sustava skladištenja podataka

Skladište podataka oblikuje se na temelju saznanja o izvornim podacima. Podaci u velikoj mjeri ulaze u informacijski sustav upisivanjem u transakcijsku bazu pomoću različitih aplikacija. Upravo unosom u transakcijsku bazu oni dobivaju strukturirani oblik. Manji dio podataka prilikom ulaska u informacijski sustav već posjeduje definiranu strukturu: riječ je o dokumentima u XML-u ili nekom drugom polustrukturiranom formatu.

Izvuče li se potonji podatkovni sadržaj iz ulaznog strukturiranog dokumenta i pohrani u transakcijsku bazu podataka (bez očuvanja strukture samog dokumenta) postoji mogućnost da se dio ulaznih podataka kasnije ne može spremiti u skladište podataka.

Dio podataka zanimljivih za skladištenje mogao se nepovratno izgubiti višestrukom promjenom formata (najprije spremanje u transakcijsku bazu, a potom u skladište), a dio, koji je mogao biti zanimljiv za skladištenje, ali ne i za transakcijske baze podataka svjesno se nije pohranio u transakcijsku bazu.

Ovakve gubitke moguće je izbjeći ako se polustrukturirani ulazni podaci izravno pohrane u skladište podataka. Skladište se oblikuje prema izvornim podacima. Dijelovi skladišta namijenjeni podacima koji u sustav ulaze kroz transakcijsku bazu oblikuju se prema transakcijskoj bazi, a dijelovi su čiji ulazni podaci imali određenu strukturu oblikuju se izravno prema tim strukturama. Sustav za skladištenje podataka dobivenih elektroničkim poslovanjem u obliku XML dokumenata treba oblikovati prema samim XML dokumentima odnosno njihovim predlošcima

3.2.2. Oblikovanje skladišta na temelju strukture XML dokumenata

Oblikovanje konceptualnog i logičkog modela skladišta za strukturirane podatke iz transakcijskih baza u literaturi je detaljno opisano koncem devedesetih godina 20. stoljeća, npr. u [GMR98]. Konačni cilj pomnih istraživanja jest mogućnost djelomične ili potpune automatizacije nekih koraka u oblikovanju skladišta.

Mogućnosti pohrane polustrukturiranih podataka u skladište podataka predmet su istraživanja Zavoda za telekomunikacije Fakulteta elektrotehnike i računarstva od godine 2001. Istraživanje se provodi suradnji sa Sveučilištem u Bologni.

Početna istraživanja uključivala su definiranje osnovnog algoritma za poluautomatsko konceptualno oblikovanje skladišta podataka iz XML izvora [GRV01], budući da je XML daleko najšire zastupljeni oblik polustrukturiranih podataka. Oblikovanje se bazira na XML Schemi kao predlošku za XML dokumente. Programski sustav koji verificira ovaj algoritam izrađen je u sklopu diplomskog rada [Ban03] autora ovog magistarskog rada i predstavlja jezgru sadašnje programske podrške za konceptualno i logičko oblikovanje. U nastavku istraživanja početni je algoritam detaljno razrađen i značajno proširen. Detaljno su opisani upiti nad sadržajem dokumenata koje programski sustav izvodi u interakciji s projektantom skladišta [VBR03a, VBR03b].

Potom je specificirana opća metodologija za oblikovanje skladišta podataka iz XML izvora, koja definira sve korake procesa izgradnje skladišta od početne analize korisničkih zahtjeva i izvora podataka do punjenja podacima i konačnog korištenja [VBS04, Vrd04]. Cilj metodologije u konačnici je integracija polustrukturiranih

podataka u skladište, zajedno s podacima dobivenim iz drugih izvora. Načela cjelokupnog procesa konceptualnog i logičkog oblikovanja verificirana su programskim sustavom koji je tema ovog magistarskog rada.

Algoritam koji izvodi navedeni programski sustav automatizira najveći dio procesa oblikovanja i time ga bitno ubrzava. Otklanja se i mogućnost neželjenih pogrešaka u svim dijelovima procesa koji su automatizirani. Istodobno, projektant skladišta rasterećen je obavljanja znatnog dijela sada automatiziranog procesa te se može usredotočiti na njegove dijelove koje računalo ne može izvesti bez njegovog sudjelovanja.

U poglavlju 2.4 opisane su dvije vrste predloška za XML dokumente: DTD i XML Schema te iznesene njihove prednosti i nedostaci. U poglavlju 2.4.2 detaljno su objašnjena sva proširenja i dodatne mogućnosti koje XML Schema unosi u odnosu na stariji koncept DTD. U ranijim istraživanjima vezanim uz konceptualni model područnog skladišta podataka iz XML izvora predloženo je korištenje XML Scheme kao predloška [GRV01, VBR03a, VBR03b]. Iako se oblikovanje konceptualnog modela podataka može se koristiti i DTD, XML Schema podržava koncept imenika i omogućava definiranje složene strukture kao tipa podataka što u nekim slučajevima može biti od izuzetne koristi. Glavni argument za korištenje XML Scheme je dobro riješen mehanizam primarnog i stranog ključa. Način na koji primarni i strani ključ definira DTD ne omogućuje njihovo jednoznačno povezivanje na osnovi samog predloška, već je potrebno analizirati sadržaj XML dokumenata odnosno ispitivati semantiku (poglavlje 2.4.1). DTD omogućuje konceptualno oblikovanje samo na temelju ugniježđene strukture elemenata, čime se dio značajnih veza među podacima gubi. Odabirom XML Scheme za predložak dokumenta isti se podaci mogu bez gubitka unijeti u konceptualni model.

Nemogućnost korištenja DTD-a kao temelja za oblikovanje područnog skladišta podataka dolazi do izražaja kod logičkog oblikovanja. Budući da u ovom radu konceptualni model ostvaruje svoju izvedbu u sustavu za upravljanje relacijskom bazom podataka, potrebno je definirati atribute u relacijskim shemama. Prilikom definicije atributa nužno je navesti i tip podatka. Dok XML Schema razlikuje mnogo primitivnih tipova podataka, DTD definira samo tekstualni tip podataka. Zbog nepostojanja brojčanog i datumskog tipa podataka u predlošku, o tipu podataka može se zaključivati samo na osnovi postojećih dokumenata i poznavanja semantike, odnosno čitanja popratnih uputa za formiranje dokumenata. Tekstualna vrijednost svakog elementa odnosno vrijednost svakog atributa zadana DTD-om može biti tekst bez posebnog ograničenja (izuzevši ona za znakovne tipove, *tokenized types*). Ovaj ključni nedostatak u potpunosti onemogućuje uspješno oblikovanje područnog skladišta podataka na temelju DTD-a kao predloška.

U nastavku će biti opisani pojedini koraci oblikovanja skladišta podataka. Oni će u konačnici (poglavlje 8) biti verificirani u studijskom primjeru realnog dokumenta koji poduzeća koriste u elektroničkom poslovanju. Oblikovanje skladišta uvijek započinje proučavanjem izvora podataka za skladište. Potom se redom izgrađuju konceptualni, logički i fizički model skladišta. Nakon definiranja procesa izvlačenja, transformacije i učitavanja podataka, izrađeni fizički model se izvodi u bazi i vrši se početno popunjavanje skladišta podacima. Time skladište podataka počinje s radom. U ovom radu detaljno će se opisati konceptualno i logičko oblikovanje područnog skladišta podataka te aplikacija koja te procese automatizira. Logičko oblikovanje izvodi se za relacijske baze podataka.

Zatim će biti prikazana arhitektura cjelokupne programske podrške za oblikovanje skladišta iz XML izvora. Programska podrška napisana je u potpunosti u jeziku Java. Korištena je platforma *Java 2 Enterprise Edition* (J2EE). Ta platforma sadrži dodatna programska sučelja u odnosu na osnovnu platformu *Java 2 Standard Edition* (J2SE). Za čitanje i raščlambu XML dokumenta (*parsing*) autor je koristio sustav JDOM Beta 9 [Jdom03] izrađen u sklopu otvorenog projekta JDOM [Jdom04] te program za čitanje Apache Xerces [Xer]. Za izvršavanje upita u jeziku XML Query korištena je javno dostupna programska podrška IPSI [Fra] instituta Fraunhofer u Darmstadtu u Njemačkoj.

3.3. Metodologija oblikovanja skladišta podataka iz XML izvora

U ovom poglavlju prikazana je metodologija oblikovanja područnog skladišta podataka iz XML izvora. Temelj metodologije, iznesen u [GR99], odnosio se na dijagrame entitet-veza kao predloške izvora podataka, no njene osnovne faze općenito vrijede za proces oblikovanja područnog skladišta, bez obzira na podatkovni izvor. Kako su, međutim, izvorni podaci u polustrukturiranom obliku, neki se koraci izgradnje skladišta moraju dodatno proširiti. Metodologija prilagođena izvoru podataka u formatu XML dana je u [VBS04, Vrd04]. Razvijena programska podrška omogućuje da se na temelju XML Scheme i pripadajućih dokumenata izradi konceptualni i logički model područnog skladišta podataka. Za predloženi logički model program omogućuje stvaranje tablica u relacijskoj bazi podatka.

Prije početka procesa oblikovanja skladišta podataka potrebno je izvršiti neizbježne uvodne korake. Potom slijedi proces oblikovanja područnog skladišta podataka čije su glavne faze konceptualni, logički i fizički dizajn te projektiranje procesa izvlačenja, transformacije i učitavanja podataka.

Uvodni koraci se s obzirom na sudjelovanje projektanta skladišta dijele na:

- analizu korisničkih zahtjeva i izvora podataka (u kojem projektant aktivno sudjeluje). Prvi, najvažniji i neizbježni uvodni korak je prikupljanje zahtjeva budućih korisnika skladišta. Istodobno, projektant skladišta mora proučiti sve XML Scheme na temelju kojih će se oblikovati skladište.
- pohranu XML Schema i XML dokumenata (i drugi koraci koji ne zahtijevaju angažman projektanta, već se obavljaju automatski, uz pomoć programske podrške). Potrebno je ispitati ispravnost XML dokumenata s obzirom na pripadajuću im XML Schemu (engl. *validation*). Ispravni dokumenti i XML Schema učitavaju se u programski sustav i pohranjuju na način koji će omogućiti njihovu stalnu raspoloživost tijekom procesa oblikovanja skladišta.

Proces oblikovanja skladišta sastoji se, prema [GR99, VBS04, Vrd04] od pet različitih koraka:

- konceptualno oblikovanje (engl. *conceptual design*),
- ispitivanje i optimizacija konceptualnog modela (približan prijevod engleske sintagme *workload refinement*),
- logičko oblikovanje (engl. *logical design*),
- oblikovanje procesa izvlačenja, transformacije i učitavanja podataka (engl. *ETL design*),

- fizičko oblikovanje (engl. *physical design*).

U ovom radu naglasak se stavlja na procese konceptualnog i logičkog oblikovanja. Izrađeni programski sustav obuhvaća uvodne korake pohrane XML dokumenata i provjere njihove ispravnosti (*validation*) te konceptualno oblikovanje, djelomičnu optimizaciju konceptualnog modela te logičko oblikovanje. U konačnici se stvaraju naredbe u jeziku SQL (*Structured Query Language*) za izradu tablica u relacijskoj bazi podataka.

U tablici 3.1 su prikazani svi koraci oblikovanja područnog skladišta podataka. Za svaki korak navedena je početna, ulazna struktura, izlazna struktura kao konačni rezultat tog koraka oblikovanja te sudionici u tom koraku procesa.

Tablica 3.1. Metodologija oblikovanja područnog skladišta podataka iz XML izvora

	Korak	***Ulaz***	***Izlaz***	***Sudionici***
1	*Konceptualno oblikovanje*	*XML Schema i pripadajući XML dokumenti*	*Višedimenzijska činjenična shema (konceptualna shema)*	*Projektant skladišta, krajnji korisnici*
2	*Ispitivanje i optimizacija konceptualnog modela*	*Višedimenzijska činjenična shema*	*Provjerena višedimenzijska činjenična shema*	*Projektant skladišta, krajnji korisnici*
3	*Logičko oblikovanje*	*Provjerena višedimenzijska činjenična shema*	*Logička shema, naredbe u SQL-u*	*Projektant skladišta*
4	*Oblikovanje procesa izvlačenja transformacije i učitavanja pod.*	*XML Schema i pripadajući XML dokumenti, logička shema*	*Naredbe u XQueryju, naredbe u SQL-u*	*Projektant skladišta*
5	*Fizičko oblikovanje*	*Logička shema*	*Fizička shema*	*Projektant skladišta*

3.3.1. Konceptualno oblikovanje

Kvalitetno konceptualno oblikovanje određuje kvalitetu kasnijeg skladišta podataka. Uspješno zadovoljenje korisničkih zahtjeva izvedeno u vidu višedimenzijskog konceptualnog modela preduvjet je za kasnije korištenje skladišta. Višedimenzijski se model skladišta izrađuje na temelju podatkovnih izvora koji mogu biti heterogeni. Istraživanja su se najprije bavila konceptualnim modelom izrađenim na osnovi strukturiranih podatkovnih izvora iz transakcijskih baza podataka. Predložak po

kojem se oblikuju podatkovne strukture su dijagrami entitet-veza odnosno iz njih izvedene relacijske sheme (poglavlje 1.2.2). Općenito, odabir činjenica i dimenzija podrazumijeva pronalaženje odnosa jedan-prema-više među entitetima (poglavlje 1.2.5), a za oblikovanje hijerarhije potrebno je pronalaženje funkcijskih ovisnosti.

Podaci u XML-u zahtijevaju drukčiji način oblikovanja konceptualnog modela. U tom formatu veza između pojedinih entiteta (u XML-u predstavljenih elementima složenog tipa) ostvaruje se ugnježđivanjem elemenata. Struktura XML dokumenta po svojoj biti vrlo kvalitetno opisuje pridruživanje podelemenata nadređenim elementima, dok se suprotni odnos zanemaruje. Stoga je u XML Schemi kardinalnost veze kod ugnježđivanja opisana samo u smjeru od nadređenog elementa prema podelementu. Kardinalnost veze od podelementa k nadređenom elementu se ne definira, ali se u nekim slučajevima može utvrditi ispitivanjem sadržaja dokumenata. Ispitivanje sadržaja XML dokumenata zahtijeva razumijevanje njihove semantike pa je u tom slučaju potrebno sudjelovanje projektanta skladišta. Razumijevanje semantike nužno je i kod odabira činjenice, tako da proces oblikovanja konceptualnog modela nikad ne može u potpunosti biti automatiziran. Drugi način ostvarivanja veza u XML dokumentima je mehanizam primarnog i stranog ključa.

Rezultat konceptualnog oblikovanja je dimenzijski činjenični model [GMR98], opisan usmjerenim grafom ovisnosti. Konceptualno oblikovanje i njegova automatizacija razvojem programske podrške bit će detaljno opisani u poglavlju 6.

3.3.2. Ispitivanje i optimizacija konceptualnog modela

Početni konceptualni model izrađuje se uz pomoć programskog alata na osnovi korisničkih zahtjeva te analize XML Scheme i pripadajućih dokumenata. Valjanost konceptualnog modela ispituje se simulacijom njegovog stvarnog korištenja. Projektant skladišta, u interakciji s krajnjim korisnicima, definira skup najvažnijih i najčešćih upita nad budućim područnim skladištem namijenjen testiranju. Iz skupa upita, analizom ograničenja po različitim dimenzijama, otkrivaju se dimenzije i njihove hijerarhijske razine za koje se najčešće vrši agregacija (sumiranje zapisa). U svrhu ubrzavanja upita za često korištene agregacije podaci se zbrajaju unaprijed. Upravo na osnovi frekventnosti pojedinih upita odlučuje se hoće li se podaci zbrajati unaprijed. Konkretna izvedba agregacije je dio logičkog oblikovanja skladišta.

Na osnovi skupa najčešćih upita moguće su preinake konceptualnog modela. Za novi optimizirani model potrebno je ponovno odrediti skup najčešćih upita i napraviti testiranje. Preinake konceptualnog modela i njegovo testiranje obavljaju se sve dok se ne razvije konceptualni model koji u potpunosti zadovoljava korisnikove zahtjeve.

U ovom koraku oblikovanja skladišta potrebno je odrediti i veličinu svih tablica u relacijskoj bazi uključujući i sumarne agregacijske podatke. Treba odrediti tip podataka za sve buduće tablice područnog skladišta, te količinu memorije koja se osigurava za pohranu vrijednosti. Za određivanje količine memorije potrebne za spremanje podataka ispitivanje se vrši na konkretnom sadržaju XML dokumenata. Pritom se koristi upitni jezik XML Query.

3.3.3. Logičko oblikovanje za relacijske baze

Logičko oblikovanje je proces prilagođavanja optimiziranog konceptualnog modela sustavu za upravljanje bazom podataka. Pritom se primjenjuju podaci o obujmu podataka i količini memorije koja se rezervira za pojedini stupac dobiveni u prethodnom koraku oblikovanja skladišta (poglavlje 3.3.2).

U ovom radu opisuje se isključivo izvedba u relacijskoj bazi podataka. Temelj logičkog modela je spoj zvijezda. Postupci pretvaranja konceptualnog modela podataka u logički model za relacijske baze temelje se na načelnim pravilima opisanim u literaturi [GMR98]. Međutim, brojni detalji u postupku logičkog oblikovanja razlikuju se s obzirom na izvor podataka. XML Schema definira vlastite specifične tipove podataka pa je vrlo važno za svaki tip XML Scheme jasno navesti odgovarajući tip podataka u relacijskoj bazi u koji se on preslikava. Specifičnost XML-a kao izvornog podatkovnog formata uvjetuje da su u konceptualnom modelu u znatnoj mjeri zastupljene pojedine strukture koje su za druge izvore podataka razmjerno rijetke. Njihova realizacija u logičkom modelu za relacijske baze podataka sadrži brojne posebnosti, od kojih su neke poznate od ranije, ali su kod XML-a kao izvora podataka znatno više zastupljene (npr. dijeljena hijerarhija). Program oblikuje činjeničnu tablicu i dimenzijske tablice u spoju zvijezda, ali je moguće, za određene specijalne slučajeve koji će biti opisani u nastavku (poglavlje 7.2) realizirati i model pahuljice (engl. *snowflake*). U konačnici program stvara naredbe u jeziku SQL čijim izvođenjem u relacijskoj bazi nastaju dimenzijske tablice i činjenična tablica (u slučaju modela pahuljice i dodatne tablice).

3.3.4. Oblikovanje procesa izvlačenja, transformacije i učitavanja podataka

U ovom dijelu procesa oblikovanja područnog skladišta podataka određuje se način popunjavanja tablica stvorenih na kraju faze logičkog oblikovanja. Za vađenje podataka iz XML dokumenata može se koristiti upitni jezik XML Query ili programsko sučelje (*Application Programming Interface,* API) za rad s XML dokumentima (za programski jezik Java to može biti standardna izvedba sučelja DOM ili JDOM). Mora se osigurati promjena formata u slučajevima kad način zapisa u XML-u ne odgovara načinu zapisa u relacijskoj bazi podataka odnosno kad za tip podatka u XML-u ne postoji odgovarajući tip u relacijskoj bazi pa se pretvorba vrši u približni numerički, tekstualni ili datumski oblik zapisa.

3.3.5. Fizičko oblikovanje

Fizičko oblikovanje prvenstveno je vezano uz minimiziranje vremena potrebnog da se dobije odgovor na upit postavljen nad logički oblikovanim područnim skladištem (logičko oblikovanje također je usmjereno na skraćivanje vremena odgovora: upravo radi toga se u spoju zvijezda koriste denormalizirane tablice). Za izvedbu skladišta u relacijskoj bazi podataka minimiziranje vremena potrebnog za dobivanje odgovora postiže se optimalnim odabirom indeksa. Problem optimalnog odabira indeksa vrlo je složen pa se obično koriste heuristički algoritmi. Kod skladišta većina upita zahtjeva spajanje činjenične tablice s jednom ili više dimenzijskih tablica. Stoga pri odabiru indeksa pažnju treba posvetiti različitim algoritmima za spajanje.

Programski sustav oblikuje područno skladište podataka prema ovdje opisanoj metodologiji, zaključno s logičkim oblikovanjem. U budućnosti se dodatno može proširiti dodavanjem podrške za oblikovanje procesa izvlačenja, transformacije i učitavanja podataka.

4. Arhitektura programske podrške za oblikovanje skladišta

4.1. Programski jezik Java

Programski sustav napisan je u jeziku Java. Programska podrška za oblikovanje skladišta se jednako kvalitetno može pisati i u drugim programskim jezicima kao što su C++ ili Visual Basic. Izvođenje programskog koda u C++ je, dapače, brže nego izvođenje koda iste funkcije u Javi. Međutim, kvalitete jezika Java dolaze do izražaja u arhitekturi klijent-poslužitelj (engl. *client-server architecture*), odnosno u slučaju korištenja programskih komponenata distribuiranih unutar mreže koje međusobno komuniciraju.

Od drugih programskih platformi za distribuirano mrežno programiranje ističu se *CORBA* i *Microsoft .NET*. CORBA (*Common Object Request Broker Architecture*) je zamišljena krajem devedesetih godina 20. stoljeća kao koncept za distribuirano programiranje potpuno neovisan i o programskom jeziku i o operacijskom sustavu. Trebala je omogućiti da se detaljno specificirana programska sučelja mogu izvesti u bilo kojem programskom jeziku. Microsoft .NET je vlasnička tehnologija tvrtke Microsoft koja radi samo s operacijskim sustavom Windows i definira svoju inačicu programskih jezika (npr. C++.NET, VisualBasic.NET, JScript.NET te C#, verzija C++ koja omogućuje rad s XML-om i *web*-uslugama). Dok se koncept CORBE pokazao previše ambiciozno zamišljenim i vrlo kompliciranim za izvedbu, tehnologija .NET postala je, zbog opće rasprostranjenosti operacijskog sustava Windows i različitih Microsoftovih aplikacija (*web*-poslužitelj, baza podataka) vrlo popularna i korištena.

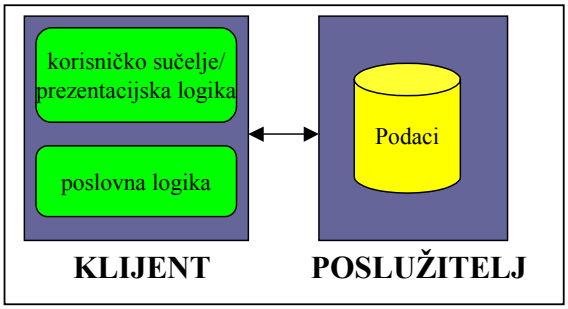
Programski sustavi pisani u Javi neovisni su o operacijskom ustavu računala na kojem se izvode. Njihova prednost u odnosu na Microsoft .NET je mogućnost izvođenja i na operacijskom sustavu Windows i na sustavima Unix i Linux. Načela programiranja u Javi, kao programskom jeziku namijenjenom izradi distribuiranih sustava, podrazumijevaju da se najprije izrade odgovarajuća sučelja, a razne strane kasnije mogu pisati konkretni kod za njihovu izvedbu. Zbog toga je Java prikladna za projekte u kojima surađuje više strana, što je vrlo čest slučaj u znanstvenim, eksperimentalnim te, općenito, otvorenim i nekomercijalnim projektima.

4.2. Dvoslojna i troslojna arhitektura distribuiranih informacijskih sustava

Svaki distribuirani sustav u načelu se sastoji od klijentskog (engl. *client*) i poslužiteljskog dijela (engl. *server*) koji se nalaze na međusobno različitim čvorovima u mreži (u načelu na različitim računalima, no mogu biti i na istom računalu, uz komunikaciju mrežnim protokolom kao da su na različitom). Korisnik ima pristup klijentskom dijelu sustava. Podaci koji će se obrađivati nalaze se na poslužiteljskom dijelu, a klijent ih dohvaća po potrebi.

Najstariji model distribuiranog sustava, dvoslojni (engl. *two-tier*) model sastoji se od klijentskog i poslužiteljskog sloja (slika 4.1). Podaci se nalaze na poslužitelju odakle ih dohvaća klijent. Sva obrada podataka, tzv. poslovna logika (engl. *business logic*) obavlja se na klijentu. Klijent se brine i za način na koji se obrađeni podaci prikazuju korisniku pomoću korisničkog sučelja. Priprema podataka za korisničko sučelje naziva se prezentacijska logika, engl. *presentation logic*. Budući da se cjelokupna

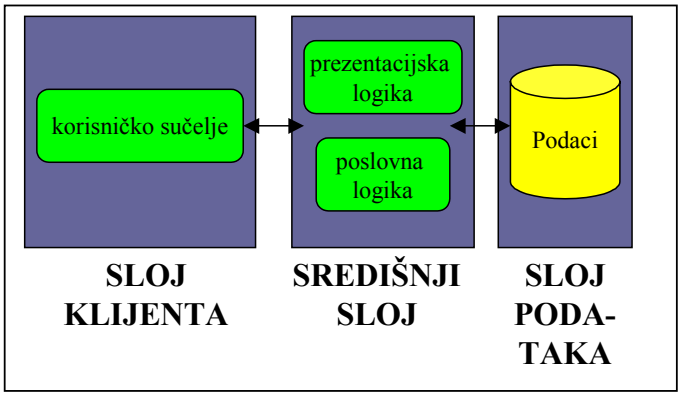
prezentacijska i poslovna logika obavlja na klijentu, takav se klijent naziva "debelim" (engl. *thick*). Model "debelog" klijenta omogućio je odvajanje baze podataka od programa za njihovu obradu. U osnovnom modelu klijent-poslužitelj na poslužiteljskoj strani nalazi se samo program koji dohvaća podatke iz baze.



Slika 4.1. Dvoslojni model distribuiranog sustava: arhitektura klijent-poslužitelj

Bilo kakva promjena poslovne logike do koje dolazi tijekom dugotrajnog rada sustava ima za posljedicu nužnost zamjene i nadogradnje svakog primjerka klijentskog programa koji postoji u sustavu.

Evolucijom modela klijent-poslužitelj klijent se nastojao rasteretiti bilo kakve logike i zadržati samo korisničko sučelje. Troslojna arhitektura (engl. *three-tier architecture*, slika 4.2) se sastoji od klijenta, srednjeg sloja i podatkovnog sloja. U odnosu na raniji model klijent-poslužitelj, "debeli" klijent je razdijeljen u sloj "tankog" klijenta i središnji sloj. Nekadašnji poslužiteljski sloj u troslojnoj arhitekturi postaje podatkovni sloj. Središnji sloj (engl. *middleware*) više nije dio klijenta već ga čine jedan ili više poslužitelja odvojenih od podatkovnog poslužitelja. Sadrži poslovnu i prezentacijski logiku. U troslojnoj se arhitekturi klijent naziva tankim (engl. *thin*).



Slika 4.2. Troslojni model distribuiranog sustava.

Najjednostavniji i ujedno ponajbolji mogući klijent je univerzalni *web*-pretraživač (engl. *web-browser*), npr. *Internet Explorer* ili *Netscape Navigator* u koji se s *web*-stranice učitava korisničko sučelje. *Web*-poslužitelj je dio srednjeg sloja. Na taj način klijentom u distribuiranom informacijskom sustavu može postati bilo koje računalo kojem administrator *web*-poslužitelja dozvoljava pristup *web*-stranici, uz uvjet da *web*-pretraživač sadrži sve potrebne dodatke za pokretanje klijentskog sučelja (npr. podrška za Javu ukoliko je sučelje napisano u Javi) i da dana verzija pretraživača omogućuje pregled sučelja (npr. verzija pretraživača Internet Explorer omogućuje verziju HTML-a u kojem je pisan kod *web*-stranice).

4.3. Tehnologije J2EE za distribuirane programske sustave

Tehnologije za izgradnju distribuiranih informacijskih sustava sastavni su dio platforme *Java 2 Enterprise Edition*. Tehnologije *Java Servlet*, *Java Server Pages* (JSP) i *Enterprise JavaBeans* (EJB) primarno služe modularnoj izgradnji distribuiranih sustava u skladu s troslojnim modelom. Na njih se veže i starija tehnologija *Java Applet*.

Java Applet je najstarija tehnologija u Javi koja je vezana za distribuirane sustave. Postoji i u osnovnoj platformi Jave, Java 2 Standard Edition (J2SE). Bazira se na dvoslojnom modelu distribuiranih informacijskih sustava: arhitekturi klijent-poslužitelj. *Applet* omogućuje da se aplikacija izvodi na klijentskom računalu, ali da se ne postavlja na to računalo već se dohvaća putem *web*-stranice. *Applet* je klasa grafičkog sučelja koja se ugrađuje u HTML-kod. Nedvojbena prednost *appleta* pred klasičnom klijentskom aplikacijom je činjenica da je njegov kod postavljen na poslužitelju, a klijentsko računalo mora samo raspolagati *web*-pretraživačem s ugrađenom podrškom za Javu kojim se dohvaća *applet*. Pogodan je u slučaju kad nema mnogo interakcije između programa (koji je dio *web*-stranice) i drugih dijelova stranice odnosno drugih stranica na poslužitelju, tj. kad je program po funkcionalnosti sličan pravoj aplikaciji. Negativna strana je nemogućnost interakcije s datotečnim sustavom klijentskog računala (interakcija je moguća indirektno, preko HTML-stranice) jer je to zbog sigurnosti klijentskog računala zabranjeno. U slučaju korištenja *appleta* klijent je "tanak" s obzirom na početne resurse (potreban samo *web*-pretraživač) i lagano unošenje promjena u logiku (promjena na jednom mjestu: poslužitelju). Međutim, da bi se *applet* mogao izvesti na klijentskom računalu potrebno je uz njegov HTML-kod dohvatiti i cjelokupni (komprimirani) oktetni kod programske logike srednjeg sloja (sloj prezentacije i sloj poslovne logike). Ako je kod veći od nekoliko stotina kilobajta, njegovo dohvaćanje preko modemske veze (engl. *dial-up*) trajat će nekoliko minuta, što djeluje vrlo loše na korisnikovu motiviranost za korištenje programskog sustava. Tako klijent za *Java Applet* ustvari postaje "debeli" klijent.

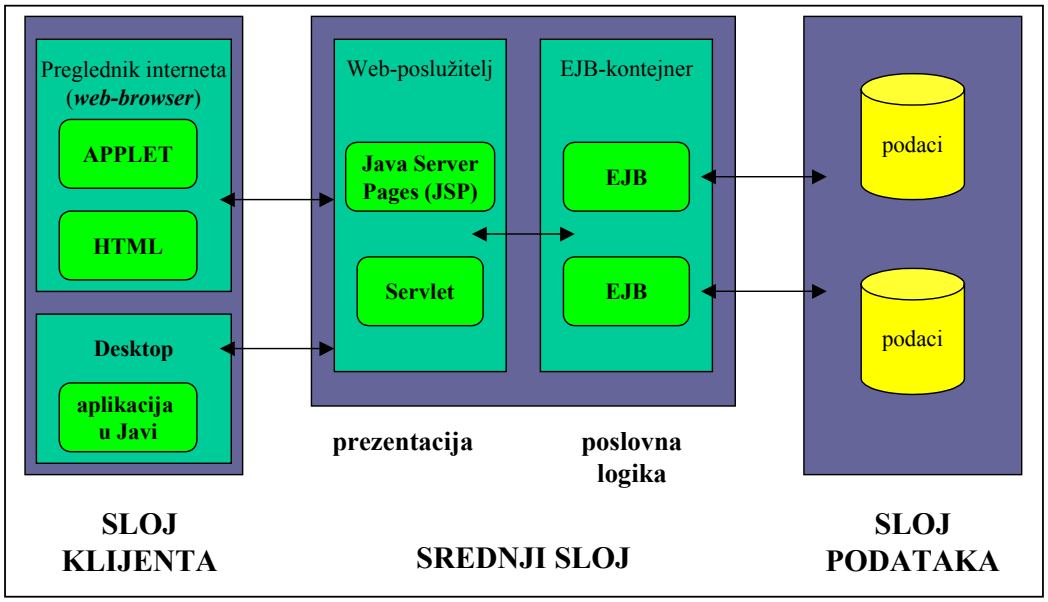
Servlet i **JSP** postoje samo u *Java 2 Enterprise Edition*. Služe za izvedbu prezentacijske logike srednjeg sloja kod distribuiranih sustava i upotpunjuju se s tehnologijom *Enterprise JavaBeans* koja služi za izvedbu poslovne logike i pojednostavnjeni pristup podatkovnom sloju. *Servlet* i JSP izgrađuju jednu *web*-stranicu koja se dohvaća protokolom HTTP.

Svaki *servlet* je istodobno *web*-stranica i klasa u Javi čija osnovna metoda vraća HTML-kod kao odgovor na HTTP-zahtjev GET za tom stranicom. *Servlet* stvara samo HTML-kod (uz multimedijalni sadržaj poput slike i zvuka) dok se u njemu sadržana logika (u praksi prezentacijska, a u jednostavnijem slučaju i poslovna logika) ne prenosi na klijentsko računalo, već ostaje na poslužitelju. Klijent je sada tanak u pravom smislu. Nedostatak *servleta* je nespretno definiranje interaktivnog sadržaja: *servlet* je klasa u Javi pa je primarno orijentiran na logiku, a minimalno na prezentaciju.

Java Server Pages (JSP) je nadogradnja na tehnologiju *servleta*. JSP-stranica se sastoji iz koda u HTML-u koji je na pojedinim mjestima dopunjen manjim odsječcima programa u Javi. Na poslužitelju se JSP automatski pretvara u *servlet*. JSP je prezentacijski orijentiran, pogodan za oblikovanje sadržajno (vizualno) bogatih i interaktivnih stranica. Zadržava prednosti *servleta* vezane uz tankog klijenta.

Enterprise JavaBeans je tehnologija za izradu distribuiranih objekata koja definira poslovnu logiku srednjeg sloja. EJB ostvaruje i kontakt s bazom podataka (tj. podatkovnim slojem) i poslovnu logiku. Zbog toga mora omogućiti istodobno upravljanje vrlo velikom količinom resursa (tj. velikim brojem zapisa u bazi) i vrlo velika brzina obrade (manipulacija nad sadržajem baze podataka ne smije kasniti). Za EJB se koristi poseban poslužitelj, tzv. EJB kontejner. Negativna strana ove tehnologije je velika složenost, koja usporava razvoj sustava pa se koristi samo za jako velike informacijske sustave.

Slika 4.3 prikazuje područja troslojne arhitekture koja pokrivaju pojedine tehnologije platforme J2EE. *Servlet* i JSP se koriste za izvedbu prezentacijske, a EJB poslovne logike. EJB izravno pristupa sustavu za upravljanje bazom podataka. Klijent je *web*-pretraživač koji dohvaća HTML-kod *web*-stranicu s *web*-poslužitelja. *Web*-stranica može biti *Java Servlet* ili JSP čiji se HTML-kod dobiva kao odgovor na zahtjev GET, ili obična statička stranica koja u sebi sadrži *Java Applet*. Klijentski program može biti i aplikacija u Javi.



Slika 4.3. Primjena J2EE u realizaciji troslojne arhitekture distribuiranih informacijskih sustava

4.4. Aplikacija i web-aplikacija

Zadatak ovog rada je izrada programskog sustava za konceptualno i logičko oblikovanje područnog skladišta podataka iz polustrukturiranih izvora (odnosno izvora u XML-u). Konceptualno i logičko oblikovanje temeljni su dio metodologije opisane u poglavlju 3.3. Programski sustav omogućuje verifikaciju teorijskih načela izgradnje skladišta podataka iznesenih u [GR99, GRV01, VBR03a, VBR03b, Vrd04, VBS04].

Programski sustav zamišljen je u dva oblika, različite razine funkcionalnosti:

- **klasična aplikacija**, nezavisna o mrežnim resursima, s većim skupom funkcija
- ***web*-aplikacija**, sa skupom funkcija ograničenim da omoguće pregled nekoliko studijskih primjera

Obje verzije sustava posjeduju isto grafičko korisničko sučelje (*Graphical User Interface*, GUI). Klasična aplikacija korisniku omogućuje testiranje metodologije izgradnje skladišta na vlastitim primjerima. Izvorne dokumente (XML Schemu i pripadajuće dokumente) uzima s diska računala. W*eb*-aplikacija je ograničena na nekoliko pokaznih primjera koji se preko posebnog poslužitelja (različitog od *web*-poslužitelja s kojeg se dohvaća *web*-aplikacija) s kojim *web*-aplikacija komunicira dohvaćaju iz baze podataka. Isti poslužitelj na zahtjev *web*-aplikacije izvodi upite u jeziku XML Query, dok ih aplikacija izvodi sama, na matičnom računalu. Aplikacija dokumente može pohraniti na disk računala. Za dohvat dokumenata može se koristiti i poslužiteljem namijenjenim *web*-aplikaciji.

Klasična aplikacija omogućuje sljedeće funkcije vezane uz pripremu podataka za proces oblikovanja skladišta, kao i za same procese konceptualnog i logičkog oblikovanja:

- učitavanje jedne ili više međusobno zavisnih XML Schema s diska matičnog računala i formiranje "radnog projekta" za oblikovanje skladišta,
- učitavanje XML dokumenata i njihovo dodavanje u radni projekt ukoliko se procesom provjere ispravnosti (*validation*) dokaže da po strukturi odgovaraju XML Schemama u projektu,
- dohvat primjera gotovih radnih projekata s poslužitelja te njihovo eventualno nadopunjavanje novim XML dokumentima,
- grafički prikaz strukture XML dokumenta koju definira XML Schema te pregled XML Schema i pripadajućih dokumenata koji spadaju u projekt,
- poluautomatsko konceptualno oblikovanje u kojem projektant skladišta odabire činjenicu na temelju razumijevanja semantike XML dokumenta; projektant kasnije po potrebi može preurediti početnu konceptualnu shemu dobivenu maksimalno automatiziranim procesom,
- pohrana početne ili preuređene konceptualne sheme,
- izvođenje upita nad XML dokumentima u jeziku XML Query na matičnom računalu ukoliko se tijekom konceptualnog oblikovanja ukaže potreba za upitima,
- logičko oblikovanje područnog skladišta podataka za izvedbu u relacijskoj bazi podataka; proces uključuje određivanje tipa podataka za svaki atribut u dimenzijskim tablicama i činjeničnoj tablici te odabir hijerarhija,
- specifikacija sustava za upravljanje bazom podataka (proizvođač i verzija) u kojem će se stvoriti dimenzijske tablice i činjenična tablica; to je nužno zbog postojanja razlika u tipovima podataka između pojedinih proizvođača odnosno verzija istog proizvođača,
- ispis naredbi u jeziku SQL za stvaranje tablica definiranih u procesu logičkog oblikovanja; izvođenje naredbi u relacijskoj bazi podataka koju odredi korisnik (mora znati adresu računala na kojem je baza, korisničko ime i lozinku).

Web-aplikacija mora omogućiti ove funkcije:

- dohvat primjera gotovih radnih projekata s mrežnog poslužitelja; za razliku od aplikacije, nema mogućnosti eventualnog nadopunjavanja novim dokumentima,
- grafički prikaz strukture XML dokumenta koju definira XML Schema te pregled XML Schema i pripadajućih dokumenata koji su dio projekta (jednako kao i aplikacija),
- poluautomatsko konceptualno oblikovanje u kojem projektant skladišta odabire činjenicu na temelju razumijevanja semantike XML dokumenta; projektant kasnije po potrebi može preurediti početnu konceptualnu shemu dobivenu maksimalno automatiziranim procesom (jednako kao i aplikacija),
- izvođenje upita nad XML dokumentima u jeziku XML Query putem mrežnog poslužitelja,
- logičko oblikovanje područnog skladišta podataka za izvedbu u relacijskoj bazi podataka, uključujući određivanje tipa podataka za svaki atribut u dimenzijskim tablicama i činjeničnoj tablici, odabir hijerarhija te specifikaciju proizvođača i verzije sustava za upravljanje bazom podataka (isto kao i aplikacija),
- ispis naredbi u jeziku SQL za stvaranje tablica definiranih u procesu logičkog oblikovanja; izvođenje naredbi u relacijskoj bazi podataka čiju adresu (uz znanje korisničkog imena i lozinke) odredi korisnik. Spoj se ne ostvaruje izravno od klijenta, već putem posebnog poslužitelja.

Funkcije klasične aplikacije koje su kod *web*-aplikacije ograničene ili uopće ne postoje su mogućnost stvaranja vlastitih korisničkih radnih projekata, neposredno izvođenje upita u jeziku XQuery i naredbi u jeziku SQL na matičnom računalu (bez poslužitelja). Stvaranje vlastitih korisničkih projekata na poslužitelju s kojeg se dohvaćaju gotovi radni projekti bilo bi moguće izvesti, ali je *web*-aplikacija zamišljena kao pokazna inačica klasične aplikacije. Stoga korisniku *web*-aplikacije nije omogućeno napraviti vlastite radne projekte.

Od opisanih tehnologija koje nudi J2EE za realizaciju *web*-aplikacija odabrana je tehnologija *Java Applet*. Ključni razlog odabira ove tehnologije je mogućnost razvoja zajedničkog grafičkog sučelja pomoću grafičkih komponenata iz paketa Swing (javax.swing). Sučelje koje se koristi za konceptualno modeliranje vrlo je složeno. Sastoji se od komponenata koje se mišem mogu pomicati po ekranu, a povezane su s logikom za oblikovanje skladišta. Kad bi se koristilo JSP ili *servlet*, bilo bi potrebno cijelo sučelje razviti u HTML-u, paralelno sa sučeljem za klasičnu aplikaciju koje koristi Swing-komponente. Ako se kao temelj *web*-aplikacije uzme *applet*, omogućuje se višestruko korištenje istog programskog koda (engl. *reusability*), a time je znatno manje programerskog posla. Višestruko korištenje istog koda jedna je od osnovnih značajki objektnog pristupa programiranju. Dobra osobina ovako zamišljenog sustava je relativno mala količina komunikacije *appleta* i poslužitelja tijekom procesa oblikovanja skladišta. Negativna strana ovakve *web*-aplikacije je veličina koda prezentacijske i poslovne logike. Opisana *web*-aplikacija postaje nezgodna za upotrebu ako se veza s poslužiteljem ostvaruje preko modema.

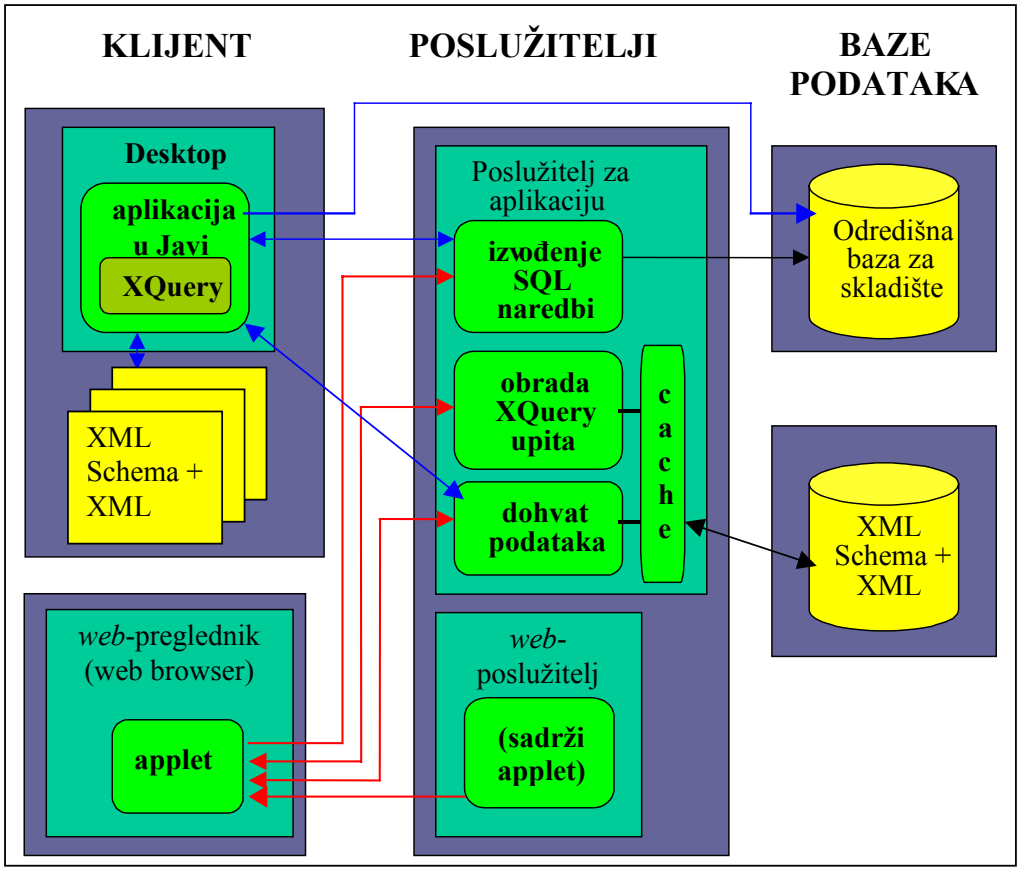
Funkcijske jedinice programskog sustava prikazuje slika 4.4.

Klasična aplikacija može dohvatiti XML Scheme i XML dokumente iz datotečnog sustava računala ili gotovi radni projekt s poslužitelja. Eventualni upiti nad XML

dokumentima u jeziku XML Query izvode se u sklopu aplikacije. Prilikom izvođenja upita na disku računala stvara se privremena datoteka koja se po izvođenju upita briše. SQL naredbe za stvaranje tablica skladišta podataka izvode se direktnim spajanjem na bazu podataka preko sučelja JDBC (*Java Database Connectivity*) [Jdbc].

Web-aplikacija (*applet*) upite u XML Queryju ne izvodi u sklopu klijenta. Budući da je za izvođenje upita potrebno stvoriti privremenu datoteku, upiti će se izvršavati na posebnom poslužitelju. Programski kod za izvršavanje upita je toliko opsežan (više megabajta) da se ne bi isplatilo izvršavati upite na klijentu i kad to zbog drugih razloga ne bi bilo onemogućeno. Isto vrijedi i za spoj na odredišnu bazu podataka u kojoj će biti smješteno skladište. Kod za JDBC zauzima previše veliku količinu memorije (npr. JDBC *driver* za Oracleovu bazu Oracle 9i sadrži 1.3 MB) da bi ga se isplatilo dohvaćati prilikom svake inicijalizacije *appleta*.

Dohvat XML Schema i XML dokumenata, izvođenje upita u XQueryju te izvođenje SQL naredbi izvode se unutar istog poslužitelja (na slici 4.4 vide se kao tri jarko zelena zaobljena pravokutnika unutar jednog tirkizno-zelenog pravokutnika, koji predstavlja zajednički poslužitelj) koji će se nazivati aplikacijskim poslužiteljem (poslužiteljem za aplikaciju). Aplikacijski poslužitelj se može, ali i ne mora nalaziti na istom računalu kao i *web*-poslužitelj na kojem se stranica koja sadrži *applet*. Između aplikacijskog poslužitelja i *web*-poslužitelja ne postoji nikakva izravna komunikacija.



Slika 4.4. Funkcijske jedinice programske podrške za oblikovanje područnog skladišta podataka: klasična aplikacija i *web*-aplikacija

Aplikacijski poslužitelj na zahtjev dohvaća iz baze podataka traženi radni projekt (XML Schemu ili više povezanih XML Schema s pripadajućim XML dokumentima) i stavlja ga u svoju privremenu memoriju (engl. *cache*). Dok se radni projekt nalazi se u privremenoj memoriji, za svaki sljedeći zahtjev nad njim nije potrebno dohvaćanje baze podataka čime se skraćuje vrijeme potrebno da se proslijedi odgovor na taj zahtjev. Prilikom dohvaćanja radnog projekta iz baze podataka automatski se stvara i privremena datoteka potrebna za upit u XQueryju i pohranjuje u privremenu memoriju. Vrijeme potrebno da poslužitelj proslijedi odgovor na upit u XQueryju tako se svodi samo na vrijeme potrebno za efektivno izvođenje upita, a štedi se i vrijeme potrebno za dohvaćanje podataka iz baze i vrijeme potrebno za stvaranje privremene datoteke. Ako poslužitelj u sat vremena ne zaprimi niti jedan zahtjev za dohvaćanjem sadržaja radnog projekta ili upitom nad njegovim dokumentima, on se izbacuje iz privremene memorije.

Svaki zahtjev nad aplikacijskim poslužiteljem dobiva vlastitu nit (engl. *thread*) koja provodi njegovu obradu. Na taj način poslužitelj paralelno može obraditi velik broj zahtjeva.

4.5. Aplikacijski poslužitelj

4.5.1. Baza aplikacijskog poslužitelja

Korištenje aplikacijskog poslužitelja moguće je samo registriranim korisnicima. Svaki korisnik ima svoje korisničko ime (engl. *user name*) i lozinku (engl. *password*) te pridruženu listu radnih projekata. Baza podataka aplikacijskog poslužitelja je relacijska baza koja sadrži pohranjene XML Scheme i pripadajuće dokumente. U njoj se također nalaze i tablice s korisničkim imenima i lozinkama i tablice radnih projekata. Shema baze podataka prikazana je na slici 4.5.

Aplikacijski poslužitelj pristupa svojoj bazi preko sučelja JDBC. Klasični način rada s bazom podataka preko sučelja JDBC podrazumijevao bi prilikom svakog zahtjeva stvaranje veze s bazom, izvršavanje naredbe u SQL-u te potom raskid veze. Trajanje uspostave veze ima isti red veličine kao i izvođenje upita, pa bi takav način rada bitno usporio rad poslužitelja na koji istodobno pristiže velik broj zahtjeva.

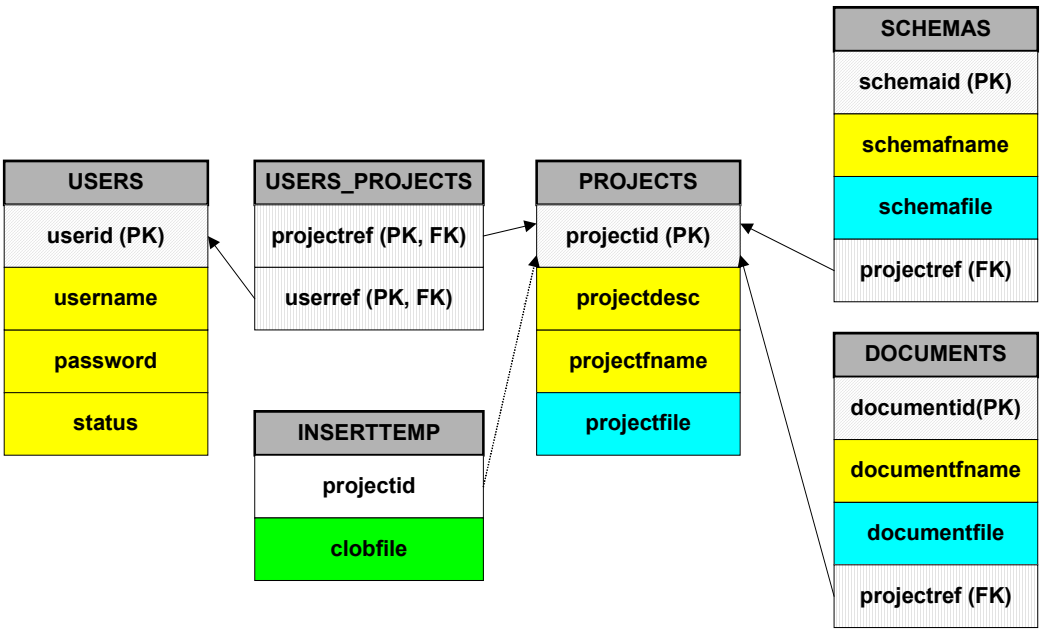
Rad poslužitelja može se ubrzati primjenom načela dijeljenja resursa iz zajedničkog fonda. Zajednički fond veza opisuje se poznatijim engleskim pojmom *connection pool*. Prilikom pokretanja poslužitelja odmah se otvara određen broj veza s bazom podataka koje poslužitelj trajno ima na raspolaganju. U trenutku kad nit poslužitelja prilikom obrade zahtjeva korisnika zatreba korištenje baze podataka, dobiva već uspostavljenu vezu s bazom iz zajedničkog fonda. Operacija se nad bazom izvrši, a potom se veza vraća u zajednički fond. Premali broj otvorenih veza s bazom imat će za posljedicu usporavanje rada poslužitelja. Ako u nekom trenutku broj paralelnih zahtjeva nad bazom podataka postane jednak broju veza u fondu, za svaki se dodatni istodobni zahtjev mora otvarati nova veza, što znatno produljuje obradu tog zahtjeva. Novostvorene veze se po završetku obrade zahtjeva ne raskidaju nego odlaze u zajednički fond. Istodobno, nepotrebno velik broj otvorenih veza narušava performanse baze podataka u cjelini. U slučaju korištenja zajedničkog fonda veza bitno je utvrditi realno opterećenje poslužitelja. Stoga se periodički provjerava ne postoji li u fondu kroz dugi vremenski period višak veza. Na taj se način broj veza u fondu dinamički prilagođuje opterećenju poslužitelja. Za poslužitelj izrađen u sklopu ovog rada teško je pretpostaviti više od pet paralelnih zahtjeva istodobno te će početni broj veza u fondu iznositi deset.

Logička shema baze podataka aplikacijskog poslužitelja prikazana je na slici 4.5. Baza je izvedena u sustavu Oracle 9i. XML dokumenti pohranjeni su na način koji omogućuje izravan pristup dijelovima dokumenta SQL-upitom, u obliku tipa XMLType. Pohrana se mogla izvesti i u vidu "velikog" tekstualnog objekta (CLOB), a razlog ovakve izvedbe je naveden u poglavlju 4.5.2. Atributi tipa XMLType na slici su označeni svjetloplavom bojom. Brojčani atributi označeni su bijelom, a nizovi znakova žutom bojom.

Tablica `USERS` sadrži korisnički identifikacijski broj (primarni ključ), korisničko ime (ograničenje jedinstvene vrijednosti, tj. ovaj je atribut važeći alternativni primarni ključ) i lozinku te status (znak T-*true* ili F-*false* koji opisuje je li korisniku dozvoljen pristup radnim projektima).

Radni projekti sadrže XML Scheme i pripadajuće dokumente. Za potrebe aplikacije i pohrane radnih projekata na disk korisničkog računala definiran je oblik zapisa radnog projekta u XML-u. Ime XML datoteke sa zapisom projekta sadržaj je atributa `projectfname` tablici `PROJECTS`, dok je sam zapis pohranjen kao vrijednost atributa `projectfile`. Odnos entiteta korisnik i entiteta projekt je više-prema-više: svakom korisniku može biti pridružen jedan ili više projekata. Istodobno, svaki projekt može biti pridružen većem broju korisnika.

Sve XML Scheme nalaze se u tablici `SCHEMAS`, dok su svi dokumenti u tablici `DOCUMENTS`. Zapisi shema i dokumenata sadrže identifikacijski broj kao primarni ključ, ime datoteke, njen sadržaj te strani ključ na tablicu projekata. Tablica `INSERTTEMP` je pomoćna tablica za unos XML dokumenata. Njena uloga bit će objašnjena u poglavlju 5.2. Zelena boja označava tip podatka CLOB (*Character Large OBject*) koji je namijenjen pohrani velikih tekstualnih datoteka veličine do 2GB (u bazi Oracle 9i definiran je i tip podataka BLOB, *Binary Large Object*, namijenjen pohrani velikih netekstualnih datoteka).



Slika 4.5. Shema baze podataka aplikacijskog poslužitelja

4.5.2. Protokol za komunikaciju aplikacije i *appleta* s aplikacijskim poslužiteljem

Aplikacija odnosno *web*-aplikacija (*applet*) komuniciraju s aplikacijskim poslužiteljem posebnim protokolom na aplikacijskoj razini. Protokol koristi transportni protokol TCP i mrežni protokol IP. Radi jednostavnosti izvedbe ne podržava model sjednice (engl. *session*), što znači da je svaki zahtjev neovisan o svim prethodnim zahtjevima koje je u bilo kojem vremenskom razdoblju uputio isti korisnik. Postoje četiri tipa zahtjeva nad aplikacijskim poslužiteljem:

- REGISTER,
- GETPROJECT,
- XQUERY,
- SQL.

Tablica 4.1 prikazuje ulazne i povratne parametre svih zahtjeva.

Korištenje aplikacijskog poslužitelja moguće je samo registriranim korisnicima. Budući da protokol ne radi na načelu sjednice, korisničko ime (*user name*) i lozinka (*password*) prenose se prilikom svakog zahtjeva.

Tablica 4.1. Tipovi zahtjeva nad aplikacijskim poslužiteljem s ulaznim i povratnim parametrima

Zahtjev	Ulazni parametri	Povratni parametri	Opis
REGISTER	userName (*string*) password (*string*)	projectNames (*niz parova string-integer*)	dohvaćanje liste korisniku dostupnih radnih projekata
GETPROJECT	userName (*string*) password (*string*) projectID (*integer*)	project (*objekt klase Project*)	dohvaćanje radnog projekta specificiranog identifikatorom *projectID*
XQUERY	userName (*string*) password (*string*) projectID (*integer*) query (*string*)	queryAnswer (*string*)	izvršavanje upita *query* u XQueryju nad XML dokumentima specificiranim identifikatorom *projectID*
SQL	userName (*string*) password (*string*) dbParameters(*lista stringova*) statements (*lista stringova*)	status (*integer*)	Izvršavanje niza naredbi u SQL-u (*statements*) u bazi podataka specificiranoj pomoću *dbParameters*

Zahtjev REGISTER dolazi na aplikacijski poslužitelj svaki put kad novi korisnik započne rad s *appletom* odnosno kad prvi put tijekom rada na aplikaciji želi dohvatiti

pokazne radne projekte. To se događa nakon što je *applet* tek dohvaćen s *web*-stranicom odnosno neposredno nakon što je aplikacija pokrenuta. Rjeđi, ali moguć slučaj je promjena korisnika na ranije korištenom *appletu* odnosno aplikaciji. U bazi podataka aplikacijskog poslužitelja dohvaća se tablica `USERS` i pronalazi redak za koji je vrijednost atributa `username` jednaka korisničkom imenu proslijeđenom kao parametar zahtjeva (`username` mora imati jedinstvenu vrijednost i ne smije poprimiti nul-vrijednost kako bi imao ulogu alternativnog primarnog ključa). Ukoliko postoji redak s takvim korisničkim imenom provjerava se je li vrijednost atributa `password` jednaka lozinci proslijeđenoj u zahtjevu. U slučaju da korisničko ime postoji i da je lozinka ispravna, korisniku se prosljeđuje lista s podacima o korisnikovim radnim projektima. Lista se sastoji od parova, a svaki par čine identifikacijski broj radnog projekta (prirodni broj ili nula) i njegov opis. Prolazom kroz sve retke tablice `USERS_PROJECTS` temeljem vrijednosti atributa `userref` izdvajaju se projekti (`projectref`) koji su pridruženi tom korisniku. Potom se mehanizmom stranog ključa prelazi na tablicu `PROJECTS` te se iz nje prosljeđuju vrijednosti atributa `projectid` i `projectdesc` u odabranim recima.

Zahtjev GETPROJECT služi za dohvat jednog radnog projekta. Korisnik prigodom prijave (zahtjev REGISTER) dobiva listu s opisom radnih projekata. Odlučivši se za dohvat nekog projekta on poslužitelju uz svoje korisničko ime i lozinku navodi i identifikacijski broj projekta, dobiven u paru s njegovim opisom. Poslužitelj u tablici XML Schema, `SCHEMAS` pronalazi vrijednost atributa `projectref` (strani ključ na tablicu radnih projekata `PROJECTS`) koja odgovara identifikacijskom broju projekta. Analogno se postupi s tablicom XML dokumenata, `DOCUMENTS`. Dokumenti su u tablici pohranjeni na način da baza "prepoznaje" njihovu strukturu. Međutim, aplikacijski poslužitelj učitava i obrađuje dokumente prema modelu JDOM. Sve XML Scheme i prateći XML dokumenti se pročitaju i raščlane (*parse*). Sa stajališta aplikacijskog poslužitelja, nema razlike između pohrane dokumenata na način da baza podataka raspoznaje njihovu strukturu ili jednostavnije pohrane u obliku "velikih" objekata tipa CLOB koji predstavljaju neraščlanjeni niz znakova. Međutim, pohrana s mogućnošću raspoznavanja strukture omogućuje administratoru baze podataka lakši rad s pohranjenim dokumentima i dijelovima dokumenata. Nakon što su XML Scheme i pripadajući XML dokumenti pročitani i raščlanjeni, na poslužitelju se formira objekt klase `Project`. Ova klasa je apstrakcija radnog projekta u aplikaciji odnosno *web*-aplikaciji. XML Scheme i XML dokumenti u toj su klasi sadržani kao JDOM-strukture. Ako je radni projekt nedavno već dohvaćen od strane istog ili bilo kojeg drugog korisnika, pripadajući objekt klase `Project` već postoji u privremenoj memoriji pa nije potrebno spajati se na bazu.

Zahtjev XQUERY služi za izvršavanje upita u jeziku XQuery. U velikoj većini slučajeva korisnik je učitao sadržaj radnog projekta pa nedugo potom započeo proces oblikovanja skladišta tijekom kojeg je potrebno vršiti upit u XQueryju. Stoga se u privremenoj memoriji najčešće nalazi instanca klase `Project` te privremena datoteka namijenjena XQuery-upitu. U slučaju da projekt ne postoji u privremenoj memoriji, on se dohvaća iz baze na način opisan u objašnjenju zahtjeva GETPROJECT. Upit se izvodi nad privremenom datotekom iz privremene memorije. Ta datoteka je poseban XML dokument koji u sebi sadrži sve XML dokumente radnog projekta. Razlozi korištenja ovakvog XML dokumenta za izvođenje upita objašnjeni su u poglavlju 6.8.

Zahtjev SQL služi za stvaranje dimenzijskih tablica i tablica činjenica u bazi podataka. Sustavi za upravljanje bazom podataka različitih proizvođača imaju

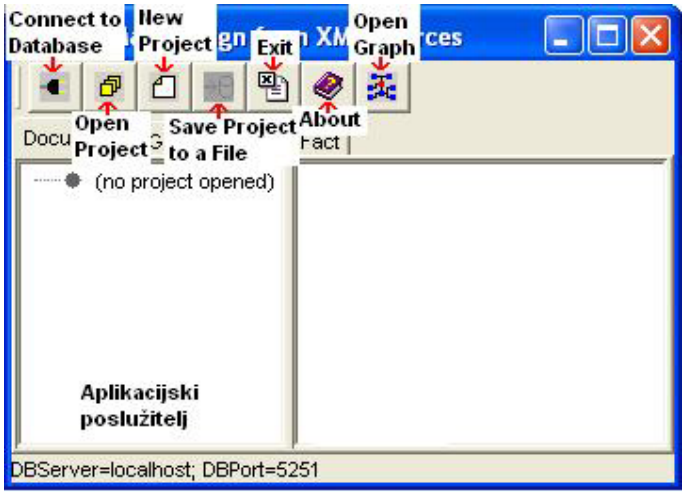
drukčije parametre za spajanje, no za sve je baze podataka nužno navesti adresu računala na kojem je odredišna baza, korisničko ime i lozinku. JDBC omogućuje da se iste naredbe u jeziku SQL izvedu na bilo kojem sustavu za upravljanje bazom podataka. Međutim, da bi se ostvario spoj s bazom, potrebno je imati ugrađenu podršku (engl. *driver*) za JDBC koji ovisi o sustavu za upravljanje bazom. Želi li se ostvariti mogućnost spajanja na različite sustave za upravljanje bazama podataka, poslužitelj mora imati ugrađen *JDBC driver* za svaki od tih sustava. U konkretnom primjeru moguće je spojiti se samo na bazu Oracle 9i. Isti sustav za upravljanje bazom podataka koristi se i kao baza poslužitelja pa na poslužitelju postoji samo *JDBC driver* za Oracle 9i [Orj].

4.6. Osnovno grafičko sučelje

Na slici 4.6 nalazi se osnovno grafičko sučelje aplikacije. Sučelje nudi sljedeće osnovne funkcije:

- ***Connect to Database***: spajanje na bazu aplikacijskog poslužitelja
- ***Open Project***: otvaranje postojećeg radnog projekta (odnosi se i na radne projekte iz datotečnog sustava, koje može otvoriti samo aplikacija i na radne projekte s poslužitelja)
- ***New Project***: stvaranje novog radnog projekta (postoji samo kod aplikacije)
- ***Save Project to a File***: pohrana radnog projekta u datotečni sustav (samo kod aplikacije)
- ***Exit***: zatvaranje programa (ikona se ne pojavljuje kod *appleta* jer se rad *appleta* prekida zatvaranjem prozora *web*-pretraživača)
- ***About***: podaci o programu (verzija, autor, godine u kojima je verzija izrađivana i završena)
- ***Open Graph***: otvaranje gotovih konceptualnih shema radi njihovog preuređivanje ili izvedbe logičkog oblikovanja (samo kod aplikacije)

Osnovno grafičko sučelje u varijanti za *applet* prikazuje slika 4.7.



Slika 4.6. Osnovno grafičko sučelje (aplikacija).

Slika 4.7. Osnovno grafičko sučelje (*applet*).

5. Izvedba uvodnih koraka u procesu oblikovanja područnog skladišta podataka iz XML izvora

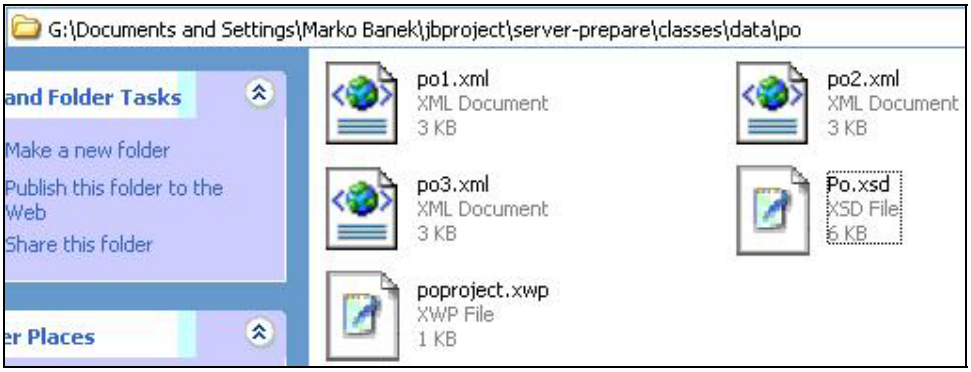
Kako je ranije istaknuto, programski sustav razvijen u sklopu ovog rada sastoji se od klasične aplikacije i *web*-aplikacije (u obliku *appleta*) reduciranih funkcija u odnosu na aplikaciju. Aplikacija svoje radne projekte (XML Scheme i pripadajuće dokumente) dohvaća iz datotečnog sustava ili s aplikacijskog poslužitelja (poglavlje 4.5), dok *applet* može koristiti samo aplikacijski poslužitelj.

5.1. Pohrana i provjera ispravnosti dokumenata kod aplikacije

5.1.1. Format za pohranu radnog projekta

Prilikom pohrane u datotečnom sustavu sve XML Scheme i svi pripadajući dokumenti nalaze se u istom direktoriju. Kako bi se naznačila njihova zajednička pripadnost radnom projektu, oblikuje se mala jednostavna XML datoteka u kojoj su nabrojene sve XML Scheme i svi dokumenti.

U primjeru (slika 5.1) u direktoriju `G:\Documents and Settings\Marko Banek\jbproject\server-prepare\classes\data\po` nalazi se jedna XML Schema (`po.xsd`) i tri pripadajuće datoteke (`po1.xml`, `po2.xml` i `po3.xml`). Datoteka `poproject.xwp` opisuje sadržaj radnog projekta (znakovi *xwp* su kratica za XML Warehousing Project). Sadržaj datoteke prikazuje slika 5.2.



Slika 5.1. Direktorij koji sadrži sve datoteke radnog projekta

```
<project>
   <schemas>
      <schema>po.xsd</schema>
   </schemas>
   <files>
      <file>po1.xml</file>
      <file>po2.xml</file>
      <file>po3.xml</file>
   </files>
</project>
```

Slika 5.2. XML datoteka za opis radnog projekta

U datoteci se posebno navode sve XML Scheme (jedna ili više njih), a posebno XML dokumenti (jedan ili više njih). Imena svih XML Schema i dokumenata navode se bez oznake direktorija jer je eksplicitno pretpostavljeno da su svi u istom direktoriju kao i datoteka *xwp* za opis radnog projekta.

5.1.2. Novi radni projekt i provjera ispravnosti dokumenata s obzirom na XML Schemu

Prilikom stvaranja novog radnog projekta najprije je potrebno odabrati sve XML Scheme koje ulaze u projekt. U projektu se može nalaziti više od jedne XML Scheme, ali tada sve one moraju biti međusobno povezane.

Kako je ranije naglašeno, XML Schema je XML dokument čija struktura se sastoji od elemenata i atributa zadanih imenicima `http://www.w3.org/2001/XMLSchema` i `http://www.w3.org/2001/XMLSchema-instance`. Osnovni (glavni, temeljni, korijenski) element je `schema`, a njegovi su podelementi definicije elemenata (`element`), atributa (`attribute`), jednostavnih (`simpleType`) i složenih tipova (`complexType`). Ako se definirani elementi, atributi i tipovi žele spremiti u određeni imenik kako bi se kasnije mogli upotrijebiti u drugim Schemama, odredišni se imenik navodi kao vrijednost atributa `targetNamespace` elementa `schema`. Svi imenici koji se spominju u dokumentu navode se u obliku xmlns:prefiks="URI". U dokumentu na slici 5.3, spominju se četiri imenika. Osnovni imenik XML Scheme ima prefiks `xs`. Imenik bez prefiksa odgovara odredišnom imeniku XML Scheme. (`http://www.openapplications.org/003_process_po_007`).

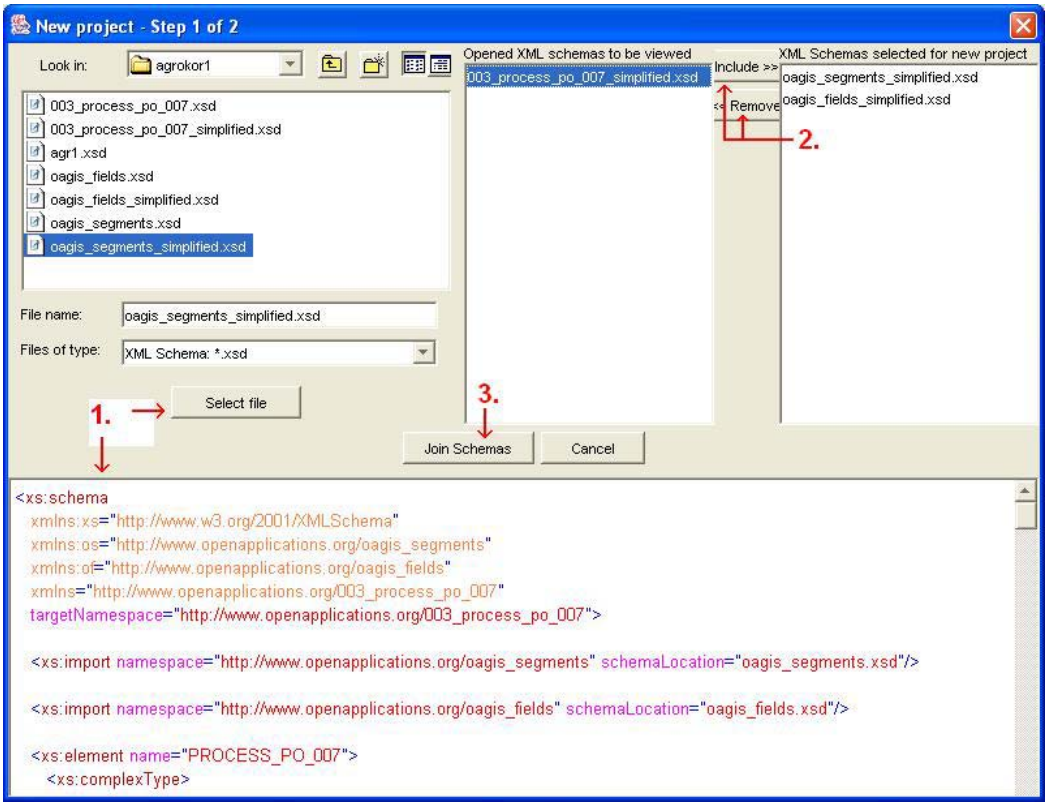
```
<xs:schema
 targetNamespace="http://www.openapplications.org/003_process_po_007"
          xmlns:os="http://www.openapplications.org/oagis_segments"
          xmlns:of="http://www.openapplications.org/oagis_fields"
          xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns="http://www.openapplications.org/003_process_po_007">
      <xs:import
          namespace="http://www.openapplications.org/oagis_segments"
          schemaLocation="oagis_segments.xsd"/>
      <xs:import
          namespace="http://www.openapplications.org/oagis_fields"
          schemaLocation="oagis_fields.xsd"/>
      <xs:element name="PROCESS_PO_007">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="os:CNTROLAREA"/>
                        <xs:element ref="DATAAREA"
                                          maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="DATAAREA">
            . . . <!-- ostatak definicije elementa--> . . .
      </xs:element>
      . . . <!-- ostatak scheme -->
</xs:schema>
```

Slika 5.3. Primjer XML Scheme s odredišnim imenikom i korištenjem definicija iz drugih XML Schema

Imenici označeni prefiksima `os` i `of` preneseni su iz drugih XML Schema, što je izravno navedeno podelementima `import` korijenskog elementa `schema`.

Odabir XML Schema za radni projekt u aplikaciji provodi se kroz tri koraka (slika 5.4):

1. Moguće je na zaslonu pogledati sve XML Scheme koje stoje na raspolaganju. U lijevom gornjem uglu prozora na slici odabere se datoteka XML Scheme. Vrši se njeno učitavanje i raščlamba. Nakon toga moguće ju je pročitati u donjem dijelu prozora.
2. Od otvorenih XML Schema odaberu se sheme koje se želi uključiti u projekt (u primjeru na slici radi se o tri međusobno povezane XML Scheme).
3. Za odabrane sheme potrebno je ispitati služe li jedna drugoj kao nadopuna. To znači da odredišni imenik jedne XML Scheme mora biti korišten u drugoj shemi. Aplikacija ispituje sve imenike u različitim XML Schemama i postojanje podelementa `import`. U slučaju da su bilo koja odabrana Schema nema veze s preostalima, aplikacija javlja grešku i proces se prekida.

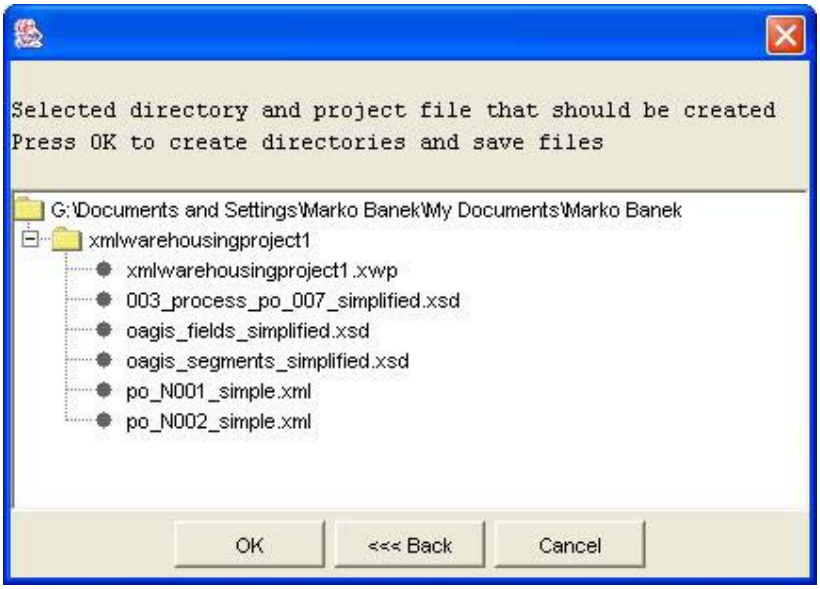


Slika 5.4. Aplikacijsko sučelje za stvaranje novog radnog projekta.

Nakon toga se prelazi na odabir XML dokumenata za radni projekt. Čim se dokument odabere, provjerava se odgovara li njihova struktura predlošku zadanom odabranim XML Schemama. U slučaju pogreške aplikacija javlja u kojim recima je došlo do pogreške. Za usporedbu strukture dokumenta s propisanim predloškom koristi se programski paket Oracle XDK (XML Developer's Kit) for Java [Orx]. Paket je razvio Oracle kao široku i obuhvatnu programsku podršku za obradu XML dokumenata. Njena je svrha potpora svekolikom korištenju XML dokumenata s ciljem njihove

integracije u bazu Oracle 9i Release2 XML DB. Paket nudi potporu za model DOM organizacije W3C (inače standardni dio J2EE 4.0), sučelje SAX te potporu za rad s XML Schemama i XSLT-om [Xslt99]. SAX (Simple API for XML) je programsko aplikacijsko sučelje za čitanje i raščlambu XML dokumenata koje omogućuje znatno brže čitanje i rad s dokumentima nego alati koji vrše raščlambu po modelu DOM. Model podataka koji će se dobiti raščlambom nije zadan, već ga definira korisnik pa se SAX koristi kad se XML dokument prebacuje u neki alternativni model podataka, primjerice JDOM. Podrška za rad s XML Schemama uključuje i provjeru ispravnosti dokumenta u donosu na XML Schemu. Standardni alat za raščlambu XML dokumenta u stablo prema modelu DOM (paket `javax.xml.parsers`) omogućuje provjeru ispravnosti spram DTD-a, ali ne i spram XML Scheme te se mora upotrijebiti spomenuti dodatni paket.

Među XML dokumentima koji su u skladu s predloškom korisnik odabire one koji postaju dio projekta. Time je proces stvaranja radnog projekta završen. Projekt je moguće pohraniti u datotečnom sustavu računala. Sve XML Scheme, pripadajući dokumenti i datoteka s opisom projekta nalaze se u jednom, izoliranom direktoriju (slika 5.5).



Slika 5.5. Spremanje novog ranog projekta u datotečni sustav.

Korisnik u projekt može naknadno dodati nove XML dokumente ili izbaciti postojeće (pritom u projektu mora ostati jedan dokument). Iz projekta nije moguće izbaciti XML Schemu jer se time mijenja i smisao svih dokumenata u projektu. U tom slučaju potrebno je stvoriti novi radni projekt.

Korisnicima nije omogućeno pohranjivati projekte u bazu podataka aplikacijskog poslužitelja. Projekti pohranjeni na aplikacijskom poslužitelju su pokazni. Naime, da bi korisnici mogli pohranjivati svoje projekte na poslužitelj, trebalo bi ugraditi sigurnosne mehanizme te izvesti zaštitu baze od preopterećenja. Rješavanje tih problema ne ulazi u područje interesa ovog rada.

5.2. Pohrana kod aplikacijskog poslužitelja

Kako je ranije naglašeno (poglavlje 4.5.1), XML dokumenti se u bazu aplikacijskog poslužitelja pohranjuju u obliku tipa XMLType. Promjenu podataka te dodavanje

novih podataka na aplikacijski poslužitelj smije obaviti samo administrator poslužitelja odnosno administrator baze.

Novi XML dokument u načelu se u tablicu ubacuje tako da se cijeli dokument pretvori u niz znakova, a potom pomoću SQL naredbe unese u tablicu. U sljedećem odsječku SQL-koda XML datoteka imena `filename` dodaje se u tablicu `DOCUMENTS` (slika 4.5):

```
insert into DOCUMENTS values (docid, filename,
XMLTYPE(content), projectref);
```

Tekst dokumenta sadržan je unutar niza znakova `content` tipa VARCHAR2. Redni broj dokumenta, `docid`, dodjeljuje se pomoću generatora slijednih vrijednosti (engl. *sequence generator*), koji se u bazi definira zajedno s tablicama. Dokument će se nalaziti u sklopu projekta `projectref`.

Prilikom izvršavanja ovog odsječka koda dolazi do problema zato što u bazi Oracle 9i Release2 XML DB funkcija za pretvaranje promjenjivog niza znakova (tip VARCHAR2) u XMLType ne dozvoljava da ulazni niz znakova bude dulji od 4KB. Funkcija, međutim, dozvoljava da ulazni parametar bude tipa CLOB. Zbog toga se niz znakova najprije pretvori u objekt tipa CLOB i pohrani. Potom se objekt tipa CLOB proslijedi kao argument pretvorbene funkcije `XMLTYPE()`. Zbog opisane višestruke promjene formata podataka potrebna je pomoćna tablica `INSERTTEMP` koja ima dva atributa: `projectid` (tipa NUMBER, koji služi kao kontrolno polje) i `clobfile` (tipa CLOB). Najprije se niz znakova `content` pretvori u CLOB i ubaci u privremenu tablicu naredbom:

```
insert into INSERTEMP values (projectref, content);
```

Potom se objekt tipa CLOB iz privremene tablice ubaci u bazu podataka kao XMLType naredbom:

```
insert into DOCUMENTS values (docid, filename, XMLTYPE(select
ct.clobfile from INSERTTEMP ct where ct.projectid=
projectref), projectref);
```

da bi se redak iz privremene tablice izbrisao odmah potom naredbom:

```
delete from INSERTTEMP where projectid=projectref;
```

6. Izvedba konceptualnog oblikovanja područnog skladišta podataka iz XML izvora

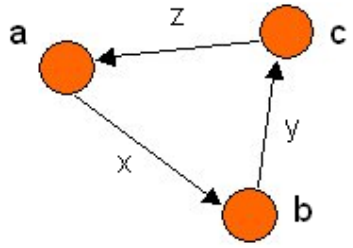
6.1. Dimenzijski činjenični model

Uz bilo koji konceptualni višedimenzijski model skladišta podataka vezani su osnovni pojmovi činjenice, mjere, dimenzije i dimenzijske hijerarhije. Svaki konceptualni model zasnovan na višedimenzijskom pristupu je matematički model podataka. Različiti konceptualni modeli razlikuju se u svojim strogim matematičkim definicijama konceptualne sheme i njenih dijelova, te njihovog pretvaranja u logički model za relacijsku bazu podataka. U ovom radu koristi se dimenzijski činjenični model (engl. *dimensional fact model*) [GMR98] na kojem se zasnivaju radovi [GRV01, VRB03a, VBR03b]. U nastavku poglavlja navest će se matematičke definicije vezane uz dimenzijski činjenični model. U definicijama vezanim uz grafove koristit će se matematički izrazi vrh i brid.

Nepravo stablo (engl. *quasi-tree*) [GMR98]. Zadan je graf g sa skupom vrhova V (od engl. *vertices*) i skupom bridova E (od engl. *edges*). Neka je graf g=(V,E) usmjereni (engl. *directed*), neciklički (engl. *acyclic*), slabo povezani (engl. *weakly connected*) graf. Za takav graf kažemo da je nepravo stablo s korijenom u $v_0 \in V$ ako se u svaki preostali vrh $v_j \in V$ može stići iz korijenskog vrha v_0 po barem jednom usmjerenom putu (engl. *directed path*).

Usmjereni graf je graf u kojem je svakom bridu definiran smjer [Gra99]. Ako su vrhovi v_i i v_j povezani usmjerenim bridom e koji izlazi iz v_i, to znači da se njime iz v_i može stići u v_j, ali se tim istim bridom ne može stići iz v_j u v_i. U grafu na slici 6.1 bridom x može se stići iz vrha a u vrh b. Iz vrha b u vrh a ne može se stići bridom x.

Put predstavlja prolazak kroz niz vrhova preko niza bridova. Ciklus je put u kojem je početni i završni vrh isti. Ciklički usmjereni graf prikazuje slika 6.1.



Slika 6.1. Ciklički usmjereni graf.

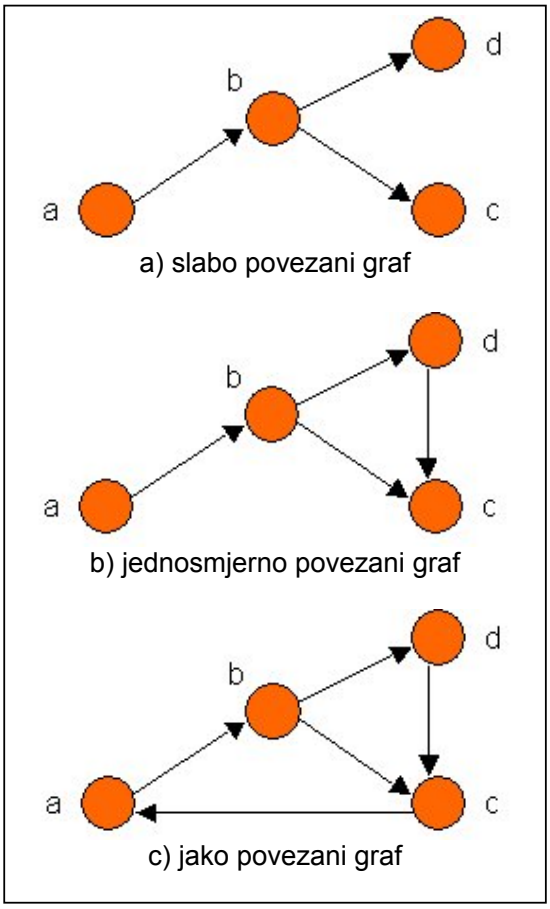
Pojam povezanosti (engl. *connectedness*) grafa, [Gra99], vezan je uz mogućnost da se iz polaznog vrha određenim putom stigne u odredišni vrh. Usmjereni graf može biti slabo povezan (engl. *weakly connected*), jednosmjerno povezan (engl. *unilaterally connected*) ili jako povezan (engl. *strongly connected*).

Graf je slabo povezan ako postoji par vrhova v_i i v_j takav da ne postoji niti put iz v_i u v_j, niti put iz v_j u v_i. Na slici 6.2 gornji je graf (graf a) slabo povezan jer se iz vrha c ne može stići u vrh d, niti iz vrha d u vrh c.

Graf je jednosmjerno povezan ako za svaki par vrhova vrijedi da se u barem jedan od njih može stići iz onog drugog. Graf u sredini slike (graf b) jednosmjerno je povezan. Iz vrha a u tom grafu može se stići u sve preostale, ali se niti iz jednog od njih ne

može stići u vrh a (jednosmjerna povezanost). Iz vrha b može se stići bilo u c bilo u d, ali ne i obrnuto. Iz vrha d može se stići u vrh c. Jednosmjerno povezani graf smije sadržavati parove vrhova povezane u oba smjera (tj. iz bilo kojeg od njih može stići u drugi), ali ta tvrdnja ne vrijedi za sve parove vrhova.

U jako povezanom grafu za svaki par vrhova vrijedi da se iz jednog vrha može stići u drugi, a iz drugog vrha u prvi. Takav je donji graf na slici 6.2 (graf c).



Slika 6.2. Slabo povezani, jednosmjerno povezani i jako povezani graf.

Nepravo podstablo osnovnog stabla s korijenom u v_0 jest podgraf $sub(g,v_i) \subset g$ čiji korijen je vrh $v_i \neq v_0$.

Iz opisane definicije nepravog stabla može se zaključiti da mogu postojati parovi vrhova koji su povezani više nego jednim putom. Ukoliko u nepravom stablu ne postoji niti jedan par vrhova povezanih više nego jednim putom, nepravo stablo postaje pravo, usmjereno stablo.

Dimenzijska shema (engl. *dimensional scheme*) predstavlja cjelokupni konceptualni pregled skladišta podataka, a sastoji se od činjeničnih shema (engl. *fact scheme*). Činjeničnu shemu tvore činjenica i njene mjere te dimenzije i njihove hijerarhije. Dimenzijski činjenični model definira pojam činjenice, dimenzije, hijerarhije te dimenzijskih i opisnih atributa na način opisan u poglavlju 1.2.3.

Formalna definicija hijerarhije [GR02] uključuje pojam djelomičnog uređaja: **hijerarhija** h je par

$h=(A, \leq_h)$

pri čemu su:

- A konačni skup dimenzijskih atributa,
- $\leq_h$ djelomični uređaj (engl. *partial ordering*) u skupu u kojem postoji jedan i samo jedan najveći element koji se naziva ključem ili korijenom dimenzije. To znači da je atribut a_i funkcijski određen atributom a_j od kojeg je "manji ili jednak": $a_i \leq_h a_j$ when $a_j \rightarrow a_i$.

Činjenična shema je trojka

$f=(H, M, S)$

[GR02] pri čemu je:

- H konačni skup disjunktnih hijerarhija. Niti jedna hijerarhija ne dijeli niti jedan dimenzijski atribut s bilo kojom drugom hijerarhijom.
- M skup mjera (koji može biti i prazan). Svaka mjera $m_i \in M$ poprima ili brojčanu vrijednost ili vrijednost dvočlanog Booleovog skupa {1, 0} (tzv. logički tip podataka).
- S skup agregacijskih tvrdnji (engl. *aggregation statement*). Agregacijska tvrdnja je trojka (h, m, Ω) $h \in H$, $m \in M$, Ω operator agregacije: $\Omega \in$ {'SUM', 'COUNT', 'AVG', 'MAX', 'MIN', 'AND', 'OR', …}. Tvrdnja (h, m, Ω)$\in$S označava da se mjera m može agregirati po dimenzijskoj hijerarhiji h s obzirom na agregacijski operator Ω.

Ako za par (h, m) ne postoji agregacijska tvrdnja, to znači da se u toj dimenziji uopće ne može raditi nikakva agregacija.

6.2. Algoritam za konceptualno oblikovanje područnog skladišta podataka izravno iz XML Schema

Konačni rezultat postupka konceptualnog oblikovanja je niz činjeničnih shema, od kojih je svaka temelj za područno skladište podataka.

Na samom početku procesa vrši se čitanje i raščlamba XML Scheme (ili više njih) i pretvorba u model zasnovan na stablu. Detaljno se ispituju definicije elemenata i atributa u XML dokumentima zadane njihovim predloškom, XML Schemom nakon čega se izgrađuje stablo čiji su čvorovi elementi i atributi iz XML Scheme, a grane veze između njih. Na temelju semantike odlučuje se koji element ili veza elemenata predstavlja činjenicu. Nakon toga se na temelju XML Scheme otkriva koje veze između elemenata i atributa predstavljaju funkcijske ovisnosti kako bi se moglo izgraditi dimenzijske hijerarhije. Također se pronalaze elementi ili atributi s brojčanog ili logičkog (Booleovog) tipa podataka koji postaju mjere. Ovim postupkom dobiva se osnovna činjenična shema. Projektant skladišta na temelju poznavanja semantičkog značenja elemenata i atributa preuređuje osnovnu shemu te analizira mogućnosti agregacije. Time je proces konceptualnog oblikovanja dovršen. Programska podrška obavlja opisani postupak poluautomatski (uz sudjelovanje projektanta skladišta) na temelju precizno definiranog algoritma [GRV01]. Na početku program raspolaže XML Schemom i pripadajućim dokumentima koji se nalaze unutar radnog projekta opisanog u poglavlju 4.4. Algoritam ima 4 osnovna koraka:

1. **Pojednostavnjenje XML Scheme**

   Odgovarajući čvorovi u Schemi koji nisu značajni za ovaj algoritam izbacuju se, dok se drugi čvorovi preoblikuju u oblik povoljniji za dalju obradu. Obavlja se u potpunosti automatski.

2. **Izrada grafa sheme**

   Graf sheme je nepravo stablo (po definiciji u 6.1) koje opisuje pojednostavnjenu strukturu XML Scheme na način prikladan za korisničko razumijevanje semantike i strukture. Graf sheme ima jedinstveni korijen koji se ne prikazuje, a njegovi su potčvorovi (tj. prva razina vidljivih čvorova) globalni elementi i atributi. Izrada grafa sheme obavlja se u potpunosti automatski. Pritom je nužno otkriti kardinalnost veza unutar XML Scheme.

3. **Odabir činjenice**

   Činjenica, kako je naglašeno u poglavlju 1.2.3, predstavlja središnju točku interesa korisnika skladišta podataka. Ona je polazna točka u procesu konceptualnog oblikovanja područnog skladišta podataka, a odabire je projektant skladišta na temelju korisničkih zahtjeva. Činjenica može biti neki entitet (element ili atribut koji definira XML Schema) ili veza. Kako je odabir činjenice semantičko pitanje, aplikacija, kao automat, korisniku mora ponuditi sve entitete i veze. Prije toga je veze potrebno pretvoriti u entitete. Projektant skladišta odabire čvor iz grafa sheme za činjenicu.

4. **za svaku činjenicu**:

   4.1. **Izgradnja grafa ovisnosti**

   Graf ovisnosti je nepravo stablo (prema 6.1) koje se izrađuje s obzirom na graf sheme, i predstavlja osnovu za činjeničnu shemu. Korijenski čvor je činjenica, a preostali čvorovi su kandidati za mjere te dimenzijske i opisne atribute. U grafu ovisnosti važna je kardinalnost veza u smjeru od korijena (činjenice) prema listovima. Tijekom postupka u nekim se slučajevima postavljaju se dodatni upiti projektantu s obzirom na uključivanje pojedinih čvorova u hijerarhije ili način njihovog prikaza. Ovaj korak algoritma odvija se u interakciji programa i korisnika.

   4.2. **Preuređivanje grafa ovisnosti**

   U grafu ovisnosti mogu se nalaziti čvorovi nezanimljivi ili previše detaljni za višedimenzijsku analizu. Projektant skladišta može ih ukloniti iz grafa ovisnosti. Neki čvorovi mogu promijeniti roditeljski čvor. Primjerice, čvor koji je kandidat za mjeru u budućoj činjeničnoj shemi u XML dokumentu najčešće nije izravno povezan s čvorom činjenice, nego se nalazi u potencijalnoj hijerarhijskoj strukturi, kao potčvor kandidata za dimenzijski ključ. U graf ovisnosti mogu biti dodani i potpuno novi čvorovi: u XML dokumentu mogu biti navedeni prodajna cijena proizvoda i prodana količina tog proizvoda, no potrebno je dodati čvor koji opisuje ukupni prihod od njegove prodaje (umnožak cijene i količine).

   4.3. **Određivanje dimenzija i mjera**

   Nakon što je projektant skladišta preuredio graf ovisnosti, određuje se koji čvorovi povezani s korijenskim čvorom postaju mjere, a koji ključevi dimenzije. U dimenzijama se definiraju hijerarhijske razine i njihovi ključevi,

te određuje koji su atributi u dimenzijama opisni. Na osnovi definiranih mjera i hijerarhija u dimenzijama definiraju se agregacije (agregacijske tvrdnje).

4.4. **Konačno definiranje činjenične sheme**

Tijekom određivanja mjera i dimenzija te postavljanja agregacijskih tvrdnji u prethodnom koraku moguće je da projektant utvrdi potrebu za dodatnim preuređivanjem grafa ovisnosti. Koraci algoritma 4.2 i 4.3 mogu se slijedno ponavljati nekoliko puta. Naposljetku se graf ovisnosti, u kojem su zadani svi članovi trojke f=(H, M, S): hijerarhije (koje čine dimenzijski i opisni atributi u dimenzijama povezani funkcijskim ovisnostima), mjere te agregacijske tvrdnje, proglašava činjeničnom shemom.

Nakon što je posljednji korak algoritma izvršen za sve odabrane činjenice, dobiva se konačna dimenzijska shema. Dimenzijska shema zasnovana je modelu dijeljenih dimenzija, tj. na arhitekturi sabirnice.

6.3. Veze u XML Schemi

Buduće mjere u činjeničnim tablicama odnosno atributi dimenzijskih tablica u osnovi se temelje na elementima i atributima XML dokumenata koji su izvor podataka. Čvorovi grafa ovisnosti proizlaze iz definicija elemenata i atributa u XML Schemi. Atributi u dimenzijskim tablicama funkcijski ovise o dimenzijskom ključu, a na funkcijskoj se ovisnosti temelji i agregacija po dimenzijama. Stoga ključan korak u konceptualnom modeliranju skladišta iz XML izvora (tj. izradi činjenične sheme) predstavlja identifikacija funkcijskih ovisnosti koje postoje u XML Schemi između dvaju elemenata odnosno elementa i atributa.

6.3.1. Struktura XML Scheme

Osnovni čvor XML Scheme je element `schema`. Njegovi najvažniji podelementi su `element`, `attribute`, `simpleType` i `complexType`. U XML Schemi koja opisuje narudžbu (dodatak A1) `schema` ima šest podelementa: dva podelementa `element`, tri podelementa `complexType` i jedan podelement `simpleType`.

`simpleType` i `complexType` predstavljaju definiciju jednostavnog odnosno složenog tipa. Definicija jednostavnog tipa zapravo je sužavanje domene jednog od 44 osnovna tipa podataka. Za razliku od jednostavnog tipa, koji opisuje vrijednost atributa ili vrijednost tekstualnog sadržaj elementa, složeni tip definira strukturu elemenata i atributa. Svaki element tog složenog tipa ima upravo one atribute i onakvu strukturu podređenih elemenata kako ih određuje složeni tip. Budući da predstavlja definiciju složenog tipa, element `complexType` u svojoj strukturi sadrži podelemente `element` i `attribute`.

Deklaracija elementa i atributa navodi ime elementa i pridjeljuje mu jednostavni ili složeni tip definiran na drugom mjestu unutar XML Scheme (slika 6.3).

```
<xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
<xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
```

Slika 6.3. Deklaracija atributa i deklaracija elementa

Ime elementa ili atributa je vrijednost atributa `name`, dok je tip naveden kao vrijednost atributa `type`.

Ukoliko su `element` ili `attribute` izravni podelementi osnovnog elementa `schema`, oni se nazivaju **globalnim elementom** odnosno **globalnim atributom**. Ako su dio podstrukture elementa `complexType`, tj. dio definicije složenog tipa, radi se o lokalnoj deklaraciji. U shemi u dodatku A1 postoje globalni elementi `purchaseOrder` i `comment` (slika 6.4), a ne postoji niti jedan globalni atribut. Globalna se deklaracija odnosi na cijelu XML Schemu. Lokalna deklaracija elementa i atributa odnosi se samo na složeni tip unutar kojeg se nalazi. To znači da se unutar XML Scheme može zadati više istoimenih elemenata ili atributa ako ih se deklarira lokalno.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
   <xsd:element name="comment" type="xsd:string"/>
           <!--definicije tipova -->
</xsd:schema>
```

Slika 6.4. Globalni elementi u XML Schemi.

Analogno elementu i atributu, i tipovi podataka mogu se definirati globalno i lokalno. Globalna definicija tipa podataka odnosi se na cijelu XML Schemu, a takav tip ima i svoje ime. Tip podataka se može definirati i lokalno, u sklopu podstrukture elementa. U tom slučaju u deklaraciji elementa ne navodi se tip elementa, već samo njegovo ime. Lokalno definirani složeni tip se naziva bezimenim (engl. *anonymous*). Element `item` u XML Schemi iz dodatka A1 je bezimenog složenog tipa. Njegov podelement `quantity` je bezimenog jednostavnog tipa (slika 6.5).

```
<xsd:element name="quantity">
   <xsd:simpleType>
      <xsd:restriction base="xsd:positiveInteger">
         <xsd:maxExclusive value="100"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
```

Slika 6.5. Bezimeni jednostavni tip

Prilikom deklaracije elementa nije uvijek nužno navesti njegov tip. Definirajući globalne ili bezimene složene tipove kao podelement se može deklarirati i globalni element. U XML Schemi iz dodatka A1 globalni element `comment` pojavljuje se unutar definicije složenog tipa `PurchaseOrderType`. Taj isječak XML Scheme prikazuje slika 6.6. Umjesto imena i tipa elementa (atributi `name` i `type`) globalni element se navodi kao vrijednost atributa `ref`.

```
<xsd:complexType name="PurchaseOrderType">
   <xsd:sequence>
      <xsd:element name="shipTo" type="USAddress"/>
      <xsd:element name="billTo" type="USAddress"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="items"  type="Items"/>
   </xsd:sequence>
   <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

Slika 6.6. Globalni element ugniježđen kao podelement u definiciji složenog tipa

Dva su osnovna načela pomoću kojih se u XML Schemi ostvaruju veze između pojedinih dijelova složene strukture XML dokumenata:

1. ugnježđivanje elemenata i pridruživanje atributa pomoću složenih tipova,
2. tehnika primarnog i stranog ključa

6.3.2. Ugnježđivanje elemenata i pridruživanje atributa pomoću složenih tipova

Element jednostavnog tipa poprima tekstualnu vrijednost, a nema podelemenata, niti atributa. Element složenog tipa može imati podelemente, atribute i tekstualnu vrijednost. Višestrukim ugnježđivanjem složenih elemenata (slučaj kad je složeni element podelement složenog elementa) dobivaju se za XML uobičajene složene strukture s većim brojem razina.

6.3.2.1. Kardinalnost pridruživanja podelementa

U definiciji složenog tipa navodi se ime i tip podelementa te najveći i najmanji broj pojavljivanja podelementa unutar nadređenog elementa. Najveći broj pojavljivanja navodi se u deklaraciji elementa kao vrijednost atributa `maxOccurs`. Analogno, minimalni broj pojavljivanja je vrijednost atributa `minOccurs`. Atribut `maxOccurs` poprima vrijednosti iz skupa prirodnih brojeva, ali se gornju granicu pojavljivanja podelementa može ukinuti postavljajući vrijednost `maxOccurs` na neograničenu vrijednost, `"unbounded"`. Atribut `minOccurs` poprima vrijednosti iz skupa prirodnih brojeva ili vrijednost nula. Pretpostavljene (engl. *default*) vrijednosti obaju atributa iznose `"1"`. To znači da je nadređenom elementu pridružen točno jedan podelement.

Iz isječka sheme na slici 6.6 vidi se da će svaki element tipa `PurchaseOrderType` imati točno po jedan podelement `shipTo`, `billTo` i `items`, dok se `comment` može pojaviti niti jednom ili jednom. Na slici 6.7 element `PROCESS_PO` ima dva podelementa: `POORDERHDR` koji se pojavljuje točno jednom, i POORDERLIN za koji nije određeno koliko puta se pojavljuje, ali se mora pojaviti bar jednom.

```
<xs:element name="PROCESS_PO">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="POORDERHDR"/>
         <xs:element ref="POORDERLIN" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Slika 6.7. Specificiranje najvećeg dozvoljenog broja pojavljivanja podelementa u nadređenom elementu

Proučavanje veza među elementima u XML dokumentima vezano je uz pronalaženje funkcijskih ovisnosti. Funkcijske ovisnosti karakterizira veza prema-jedan. Stoga sa stajališta konceptualnog modeliranja sve veze možemo podijeliti na veze prema-jedan i veze prema-više. Doda li se u obzir da XML Schema definira i minimalni broj pojava nekog elementa, mogu se razlikovati obavezne i uvjetne veze. Naposljetku se mogu izdvojiti četiri osnovna tipa veza koji u potpunosti odgovaraju četirima tipovima veza kod DTD-a:

- **obavezna veza prema-jedan (1,1)** (engl. *-to-one relationship*); svakom roditeljskom elementu pridružen je točno jedan podelement
- **uvjetna veza prema-jedan (0,1)** (engl. *optional -to-one relationship*); svakom roditeljskom elementu pridružen je jedan ili niti jedan podelement
- **obavezna veza prema-više (1,M)** (engl. *-to-many relationship*); svakom roditeljskom elementu pridružen je jedan ili više podelemenata
- **uvjetna veza prema-više (0,M)** (engl. *optional -to-many relationship*); svakom roditeljskom pridružen je niti jedan, jedan ili više podelemenata

Budući da XML dokument ima oblik stabla koje se širi od korijenskog elementa, veze se opisuju isključivo u smjeru od nadređenog elementa prema njegovim podelementima. Zbog tog inherentnog svojstva XML-a kao načina zapisivanja podataka niti XML Schema niti DTD ne definiraju kardinalnost veze od podelementa k nadređenom elementu. Za ispitivanje kardinalnosti veze u tom smjeru potrebna je analiza sadržaja XML dokumenata. Budući da se u velikom broju slučajeva u XML dokumentu pojavljuju samo po jedan nadređeni element i podelement, za eventualno utvrđivanje da je kardinalnost veze prema-više potrebno je ispitati sadržaj barem dvaju, a u praksi što je moguće više dokumenata.

6.3.2.2. Poredak podelemenata u definiciji složenog tipa

U XML-u je poredak podelemenata unutar nadređenog elementa od bitnog značenja (slika 2.6). Podelementi se unutar definicije složenog tipa navode kao niz. Oznaka niza, `sequence`, je podelement elementa `complexType`. Svaka deklaracija elementa, `element`, je podelement elementa `sequence`. U XML dokumentu deklarirani će podelementi unutar nadređenog elementa biti poredani upravo onim redom kojim su njihove deklaracije navedene kao podelementi elementa `sequence`. Svi podelementi, osim onih čija vrijednost atributa `minOccurs` je nula, moraju se pojaviti u XML dokumentu. Odsječak XML Scheme iz dodatka A1, prikazan na slici 6.6, navodi poredak podelemenata za svaki element tipa `PurchaseOrderType`: redom slijede element `shipTo`, potom `billTo`, zatim je moguća, no ne i obavezna pojava jednog elementa `comment`, nakon čega na kraju dolazi `items`.

XML Schema omogućuje da se iz skupa od više elemenata navede točno jedan. Tome služi element XML Scheme `choice`, koji se, poput `sequence`, navodi kao podelement elementa `complexType`, a njegovi podelementi su deklaracije elemenata. Dozvoljeno je međusobno ulančavanje elemenata `choice` i `sequence` u XML Schemi. Na slici 6.8 definiran je proizvod tvrtke, `item`, čija je prodaja predviđena u Hrvatskoj, Europskoj uniji i Sjedinjenim Američkim Državama. Element `item` redom sadrži podelemente `productID`, `itemName` i jednu od tri cijene: u dolarima (`priceUSD`), eurima (`priceEUR`) i kunama (`priceHRK`).

Na slici 6.9 nalaze se proizvodi koji odgovaraju ovoj Schemi. Tuna u maslinovom ulju prodaje se u Hrvatskoj gdje je cijena u kunama, te u Njemačkoj, gdje je cijena u eurima.

Treći način definiranja poretka podelemenata je struktura `all`. Poput `sequence` i `choice`, i `all` je izravni podelement od `complexType`. Za podelemente unutar `all` propisan je najveći mogući broj pojavljivanja 1, a elementi se smiju navoditi u bilo kojem poretku.

```
<xsd:element name="item">
   <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="productID" type="xsd:positiveInteger"/>
        <xsd:element name="itemName" type="xsd:string"/>
        <xsd:choice>
           <xsd:element name="priceUSD" type="priceType"/>
           <xsd:element name="priceEUR" type="priceType"/>
           <xsd:element name="priceHRK" type="priceType"/>
        </xsd:choice>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Slika 6.8. Korištenje strukture *choice* u XML Schemi

```
<item>
       <productID>374</productID>
       <itemName>Tuna u maslinovom ulju</itemName>
       <priceHRK>21.37</priceHRK>
</item>
.....................................................
<item>
       <productID>374</productID>
       <itemName>Thunfisch im Olivenoel</itemName>
       <priceEUR>1.29</priceEUR>
</item>
```

Slika 6.9. Primjer dokumenta u čijoj shemi je korišten element *choice*.

Unutar elementa `all` smiju se navoditi `sequence` i `choice` kao podelementi, ali ne vrijedi obratno: element `all` nužno mora biti izravni podelement od `complexType`. Isječak XML Scheme u kojem se koristi struktura `all` (slika 6.10), vrlo je sličan isječku sa slike 6.6 gdje se koristi `sequence`. Međutim, sequence zahtijeva strogi poredak podelemenata, dok ga `all` ne zadaje. XML struktura koja je ispravna za definiciju sa `sequence` ispravna je i za definiciju s `all`, dok obratno ne vrijedi.

```
<xsd:complexType name="PurchaseOrderType">
   <xsd:all>
      <xsd:element name="shipTo" type="USAddress"/>
      <xsd:element name="billTo" type="USAddress"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="items"  type="Items"/>
   </xsd:all>
   <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

Slika 6.10. Korištenje strukture *all* u XML Schemi

6.3.2.3. Grupe elemenata i grupe atributa

Ako se nekoliko elemenata ili nekoliko atributa više puta zajedno navodi u definicijama složenih tipova moguće je načiniti grupu elemenata ili grupu atributa. Grupa elemenata navodi se unutar elementa `group`, a grupa atributa unutar elementa `attributeGroup`. Grupe elemenata i grupe atributa unutar sebe mogu sadržavati i druge grupe elemenata odnosno atributa. Definiraju se globalno pa svaka ima svoje

ime. Primjer korištenja grupe elemenata pokazuje slika 6.11: ako bi se deklaracije elemenata `shipTo` i `billTo` pojavljivale zajedno u većem broju složenih tipova mogla bi se definirati grupa elemenata `shipAndBill`. Ona će se pojaviti u definiciji složenog tipa `PurchaseOrderType`. Isječak XML Scheme na slici 6.11 i onaj na slici 6.6 definiraju u potpunosti jednaku strukturu.

```
<xsd:group name="shipAndBill">
   <xsd:sequence>
      <xsd:element name="shipTo" type="USAddress"/>
      <xsd:element name="billTo" type="USAddress"/>
   </xsd:sequence>
</xsd:group>

<xsd:complexType name="PurchaseOrderType">
   <xsd:sequence>
      <xsd:group ref="shipAndBill"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="items"  type="Items"/>
   </xsd:sequence>
   <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

Slika 6.11. Korištenje grupe elemenata u XML Schemi

6.3.2.4. Mješoviti sadržaj i prazni sadržaj

Potčvorovi elementa mogu biti podelementi i tekstualni čvorovi. Sadržaj elementa čine svi njegovi potčvorovi elementa zajedno. S obzirom na potčvorove moguće je definirati ukupno četiri različita tipa sadržaja:

- jednostavni sadržaj (samo tekstualni sadržaj),
- sadržaj koji čine smo podelementi,
- mješoviti sadržaj (kombinacija podelemenata i tekstualnih čvorova),
- prazni sadržaj (bez podelemenata i bez tekstualnih čvorova).

U najvećem dijelu XML dokumenata koristi se kombinacija prva dva tipa sadržaja: samo najdublje ugniježđeni ("najniži") elementi, bez vlastitih podelemenata, sadrže tekst. Naprotiv, elementi koji sadrže podelemente ne sadrže tekst.

Element jednostavnog sadržaja (bez podelemenata) koji također ima i atribute zadaje se složenim tipom. Primjerice, element cijene, `price`, jednostavnog sadržaja osnovnog tipa `decimal`, uz sebe sadrži i atribut valute u kojoj je cijena izražena, `currency`. U ovakvoj definiciji koristi se element `simpleContent` kao podelement od `complexType`: to znači da je tip složen, ali je sam sadržaj jednostavan (slika 6.12).

Mješoviti se sadržaj zadaje složenim tipom uz napomenu da je riječ o mješovitom sadržaju. Ako neki element ima mješoviti sadržaj, tekst se može nalaziti između bilo koja dva podelementa, ispred prvog te iza zadnjeg podelementa. Za razliku od tekstualnog sadržaja elemenata jednostavnog tipa čija se domena definira jednim od 44 osnovna tipa ili njihovom izvedenicom, tekstualni sadržaj između podelemenata nema ograničenje domene. Tekst ustvari odgovara najopćenitijem tipu podataka, `string`, koji prihvaća svaki znak koji je uopće dozvoljen u XML-u. Primjer mješovitog sadržaja u XML dokumentu i njegovog predloška XML Schemom daje

slika 6.13. Oznaka mješovitog sadržaja je navođenje atributa `mixed` s vrijednošću `"true"` u elementu `complexType`. Ostatak definicije složenog tipa isti je kao u slučaju sadržaja koji čine samo podelementi.

```
<price currency="EUR">423.46</price>

<xsd:element name="price">
   <xsd:complexType>
      <xsd:simpleContent>
         <xsd:extension base="xsd:decimal">
            <xsd:attribute name="currency" type="xsd:string"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
</xsd:element>
```

Slika 6.12. Element jednostavnog sadržaja s atributom

```
<salutation>Dear Mr.<name>Robert Smith</name>.</salutation>

<xsd:element name="salutation">
    <xsd:complexType mixed="true">
     <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Slika 6.13. Mješoviti sadržaj

Element `salutation` ima samo jedan podelement, `name`. Tekstualni sadržaj može se nalaziti ispred elementa `name` (između početne oznake elementa `salutation` i početne oznake elementa `name`): `"Dear Mr."` te iza `name` (između završne oznake elementa `name` i završne oznake elementa `salutation`): `"."`. Tekstualni dio sadržaja može se pojaviti između podelemenata, ali i ne mora, a nije određeno na kojem će se mjestu (između kojih elemenata) pojaviti, a na kojem neće. Zato se tekstualni dio mješovitog sadržaja neće skladištiti pa će element mješovitog sadržaja u procesu oblikovanja skladišta podataka biti smatran elementom koji ima samo podelemente. Dodatni je problem nemogućnost uže specifikacije tipova: cjelokupni sadržaj se smatra tekstualnim.

Sadržaj elementa može biti i prazan. To znači da element nema niti podelemente niti tekstualni sadržaj, ali smije imati atribute (slika 6.14) budući da atributi ne spadaju u sadržaj elementa kojem su pridruženi).

```
<price currency="EUR" value="423.46"/>

<xsd:element name="price">
   <xsd:complexType>
      <xsd:attribute name="currency" type="xsd:string"/>
      <xsd:attribute name="value"    type="xsd:decimal"/>
   </xsd:complexType>
</xsd:element>
```

Slika 6.14. Element bez sadržaja, s atributima

Cijenu proizvoda, umjesto na način prikazan na slici 6.12, moguće je definirati elementom bez sadržaja s dvama atributima, iznosom cijene i valutom. Definicija takvog složenog tipa u potpunosti je jednaka ranije prikazanim definicijama složenih tipova, osim što `complexElement` ima samo podelemente `attribute`, a nema podelemenata `sequence`, `choice` ili `all`.

6.3.2.5. Pridruživanje atributa u definiciji složenog tipa

Atribut uz sebe veže samo jednu vrijednost, a ne cjelokupnu strukturu. Stoga neki element može imati samo jedan atribut određenog imena, a redoslijed je navođenja različitih atributa unutar elementa kojem pripadaju nebitan. Kod pridruživanja atributa nekom elementu u deklaraciji je dovoljno navesti ime i tip atributa te kardinalnost pridruživanja. Veza od elementa prema njegovom atributu uvijek je prema-jedan.

Za atribut je potrebno navesti radi li se o obaveznom ili uvjetnom pridruživanju. Obvezu pridruživanja navodi se u deklaraciji atributa (element `attribute`) atributom `use`. Obavezno pridruživanje opisuje riječ `"required"`, a uvjetno riječ `"optional"`. Ne navede li se u deklaraciji atributa u XML Schemi atribut `use`, predefinirana vrijednost je `"optional"`, tj. pridruživanje nije obavezno.

Obavezno pridruživanje atributa elementu odgovara obaveznoj vezi prema-jedan, opisanoj kod pridruživanja podelemenata. Uvjetno pridruživanje odgovara uvjetnoj vezi prema-jedan. Na taj način izjednačuju se veze prema atributima i veze prema podelementima pa se i atributi mogu predstaviti kao potčvorovi nadređenog elementa kojeg opisuju.

Kako za atribute nije definiran redoslijed, njihove se deklaracije navode kao izravni podelementi od `complexType`. U istoj su razini s elementom koji opisuje način izbora i poretka podelemenata (`sequence`, `choice`, `all`) i slijede nakon njega. U XML Schemi u dodatku A1 element složenog tipa `PurchaseOrderType` ima jedan atribut, `orderDate` osnovnog tipa `date`. Atribut je pridružen uvjetno. U istoj shemi u definiciji bezimenog tipa za element `item` postoji atribut `partNum` koji je obavezno pridružen elementu.

6.3.2.6. Stalne i predefinirane vrijednosti

Dodatni atributi u deklaraciji elemenata ili atributa mogu biti `fixed` i `default`. Postoji li u deklaraciji dodatni atribut `fixed`, deklarirani element ili atribut će u svakom dokumentu imati vrijednost navedenu s `fixed`. Ukoliko je pridruživanje uvjetno te se element ili atribut u dokumentu ne navede, alat za obradu XML Scheme mora automatski dodati element ili atribut s vrijednošću propisanom s `fixed`. U XML Schemi iz dodatka A1 u deklaraciji tipa `USAddress` postoji atribut `country` čija je fiksirana vrijednost `"US"`. Prisutnost atributa `default` u deklaraciji dozvoljava korisniku postavljanje vlastite vrijednosti za deklarirani element ili atribut, no ako je pridruživanje uvjetno pa element ili atribut u dokumentu ne postoji, automatski se postavlja vrijednost propisana preko `default`.

6.3.3. Tehnika primarnog i stranog ključa

XML Schema omogućuje povezivanje dijelova XML dokumenta načelom koje se temelji na mehanizmu primarnog i stranog ključa u relacijskim bazama podataka. Neki element jednostavnog tipa ili atribut je ključ u XML dokumentu ako je njegova vrijednost jedinstvena u cijelom dokumentu ili dijelu dokumenta. U dokumentu

odnosno dijelu dokumenta ne smiju postojati dvije instance tog elementa i atributa čije bi vrijednosti bile jednake.

Element ili atribut koji ima svojstvo primarnog odnosno stranog ključa navodi se izrazom u jeziku XPath [XPath99]. Kod ključa je bitno definirati:

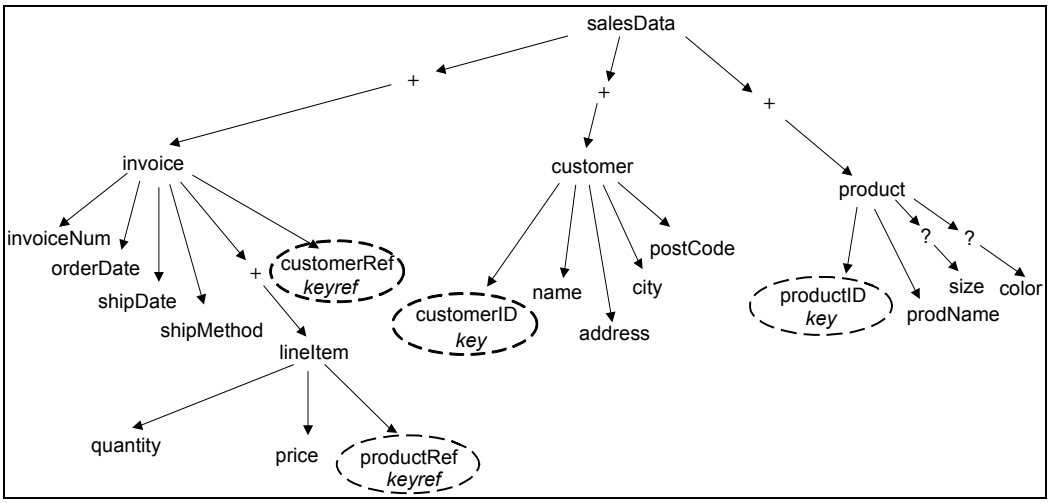
- element ili atribut koji će imati svojstvo ključa
- područje unutar dokumenta za koje vrijedi jedinstvena vrijednost primarnog ključa odnosno ograničenje stranog ključa

Definicija primarnog odnosno stranog ključa navodi se unutar deklaracije elementa unutar čije strukture su vrijednosti primarnog ključa jedinstvene ili postoje ograničenja stranog ključa.

6.3.3.1. Primarni ključ od jednog elementa

Primarni ključ se opisuje elementom `key` koji je izravni potčvor od `element` (tj. deklaracije elementa unutar kojeg vrijedi jedinstvenost ključa). Unutar elementa `key` su redom elementi `selector` i `field`. Oba elementa su bez sadržaja, ali svaki od njih ima atribut `xpath`. U atributu `xpath` elementa `field` navodi se izraz u XPath-u za sam element ili atribut koji poprima vrijednost ključa, dok se u istoimenom atributu elementa `selector` navodi struktura ugniježđenih elemenata od onog u čijoj se deklaraciji definira ključ do roditeljskog elementa ključa, opisanog elementom `field`.

U dodatku B1 nalazi se XML Schema `salesdata.xsd` s opisom podataka o poslovanju neke trgovine. U njoj su definirana dva primarna i dva strana ključa. Primjer dokumenta koji odgovara Schemi je `sale1.xml` u dodatku B2. Skica strukture koju definira XML Schema prikazana je na slici 6.15.



Slika 6.15. Graf sheme za podatke o trgovini

Svaki element ili atribut je čvor grafa, pri čemu nema razlike između elemenata i atributa. Veza od elementa prema atributu prikazana je strelicom. Kardinalnost veze opisana je operatorima koji se koriste kod DTD-a (?, *, +) budući da je za potrebe pronalaženja funkcijskih ovisnosti veze moguće svesti na četiri tipa kardinalnosti koje odgovaraju kardinalnostima pridruživanja kod DTD-a (poglavlje 6.3.2.1). Ovakva

struktura zapravo je graf sheme, nepravo stablo koje će detaljno biti opisano u poglavlju 6.6. Iz skice se vidi da se podaci o trgovini sastoje od tri cjeline: podataka o kupcima, podataka o ponuđenim proizvodima i podataka o izdanim računima. Na računu nisu detaljni podaci o kupcu i prodanom proizvodu već samo šifra kupca i šifra proizvoda o kojima se detaljni podaci nalaze u drugom dijelu dokumenta. Ovakav način bilježenja zapravo je karakterističan za relacijske baze podataka. Proizvod (element `product`) je opisan podelementima `prodName`, `size` i `color` te atributom `productID` koji predstavlja šifru proizvoda, njegov primarni ključ. Element `key` u XML Schemi (slika 6.16) postavlja `product` kao primarni ključ.

```
<xsd:element name="salesData">
   <xsd:complexType>
      <!-- definicija složenog tipa -->
   </xsd:complexType>

   <xsd:key name="prodPKey">
      <xsd:selector xpath="product"/>
      <xsd:field xpath="@productID"/>
   </xsd:key>
   <xsd:keyref name="prodFKey" refer="prodPKey">
      <xsd:selector xpath="invoice/lineItem"/>
      <xsd:field xpath="productRef"/>
   </xsd:keyref>

   <!-- drugi ključevi -->

</xsd:element>
```

Slika 6.16. Definicija primarnog ključa za proizvod i stranog ključa koji se na njega odnosi

Vrijednost atributa je jedinstvena (ne smije se ponavljati) unutar svakog elementa `salesData` jer se definicija ključa nalazi unutar deklaracije elementa `salesData`. Kako je riječ o jedinom globalnom elementu, početnom elementu svakog dokumenta, ograničenje se ustvari odnosi na cijeli dokument. Atribut `productID` specificiran je kao ključ preko vrijednosti atributa `xpath` od `field`. Vrijednost atributa `xpath` od `selector` opisuje put od `salesData` do `productID`: u ovom slučaju poprima vrijednost `product` budući da je potpuni izraz za `productID` u XPathu `salesData/product/@productID`.

U elementu `selector` nije potrebno navoditi element po element. Moguće je upotrijebiti skraćeni izraz da se ključ odnosi na sve podelemente ili atribute bilo kojeg stupnja ugniježđenosti u odnosu na element u sklopu kojeg je definiran ključ: **".//*"**. Takav izraz koristi se u definiciji primarnog ključa za kupca (slika 6.17).

6.3.3.2. Strani ključ od jednog elementa

Strani ključ se uvijek odnosi na neki primarni ključ (tzv. referencijski integritet). Definira li se element ili atribut kao strani ključ, njegova vrijednost mora odgovarati jednoj od postojećih vrijednosti primarnog ključa u dokumentu odnosno dijelu dokumenta. Strani ključ mora se po tipu podudarati s primarnim ključem, a dozvoljeno je i da primarni ključ bude element, a strani ključ atribut ili obrnuto. Strani ključ ima svoje područje vrijednosti. Za istoimeni element ili atribut izvan područja vrijednosti određenog izrazom u XPathu ne vrijedi ograničenje stranog ključa.

Strani ključ definira se, analogno primarnom, elementom `keyref` koji unutar sebe ima elemente `selector` i `field`, s jednakim značenjem kao u definiciji primarnog ključa. Za razliku od elementa `key`, koji ima samo atribut `name` (ime primarnog ključa), `keyref` uz `name` ima i atribut `refer`, čija je vrijednost ime (`name`) primarnog ključa na koji se odnosi strani ključ. U primjeru na slici definira se strani ključ `prodFKey`, koji se odnosi na primarni ključ `prodPKey`. Područje vrijednosti je element `salesData`, a time zapravo i cijeli XML dokument definiran XML Schemom. Referenca na primarni ključ je element `salesData/invoice/lineItem/productRef`.

```
<xsd:element name="salesData">
   <xsd:complexType>
      <!-- definicija složenog tipa -->
   </xsd:complexType>

   <xsd:key name="custPKey">
      <xsd:selector xpath=".//*"/>
      <xsd:field xpath="@customerID"/>
   </xsd:key>
   <xsd:keyref name="custFKey" refer="custPKey">
      <xsd:selector xpath=".//*"/>
      <xsd:field xpath="customerRef"/>
   </xsd:keyref>
   <!-- drugi ključevi -->

</xsd:element>
```

```
<xsd:element name="salesData">
      <xsd:complexType>
         <!—definicija složenog tipa -->
      </xsd:complexType>

      <xsd:key name="prodPKey">
         <xsd:selector xpath="product"/>
         <xsd:field xpath="@prodID"/>
      </xsd:key>
      <xsd:keyref name="prodFKey" refer="prodPKey">
         <xsd:selector xpath="invoice/lineItem"/>
         <xsd:field xpath="productRef"/>
      </xsd:keyref>

      <xsd:keyref name="custFKey" refer="custPKey">
         <xsd:selector xpath=".//*"/>
         <xsd:field xpath="customerRef"/>
      </xsd:keyref>
      <xsd:key name="custPKey">
         <xsd:selector xpath=".//*"/>
         <xsd:field xpath="@customerID"/>
      </xsd:key>

</xsd:element>
```

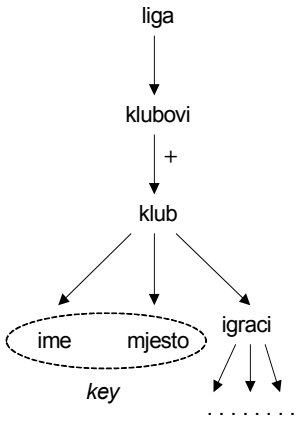
Slika 6.17. Definicija primarnog ključa za kupca i stranog ključa koji se na njega odnosi

6.3.3.3. Ključ kao kombinacija više elemenata

U relacijskim bazama podataka moguće je da više atributa neke relacije zajedno čini njezin složeni ključ. XML Schema analogno omogućuje da dva ili više elemenata i

atributa unutar nekog područja vrijednosti zajedno budu definirani kao složeni primarni ključ. U definiciji takvog ključa je jedan element `selector` te po jedan element `field` za svaki ključ. Nogometni klub je, primjerice, određen svojim imenom i mjestom (ime kluba nije dovoljno jer postoje istoimeni klubovi u mnogim gradovima: Borussia Dortmund i Borussia Mönchengladbach). Isječak XML Scheme je na slici 6.18. U općenitom slučaju nije potrebno da svi dijelovi složenog ključa imaju zajednički roditeljski element kao u primjeru. Na složeni primarni ključ može se odnositi složeni strani ključ. Broj komponenata ključa u oba slučaja mora biti isti, a svaka komponenta primarnog ključa ima par među komponentama stranog ključa kojim se podudara po tipu.

```
<xsd:element name="klubovi">

    <xsd:complexType>
         <!-- definicija podelemenata -->
    </xsd:complexType>

    <xsd:key name="klubPKey">
         <xsd:selector xpath="klub"/>
         <xsd:field xpath="ime"/>
         <xsd:field xpath="mjesto"/>
    </xsd:key>

</xsd:element>
```



Slika 6.18. Složeni primarni ključ

6.4. Graf sheme

XML Schema se može predočiti kao skup šest različitih vrsta čvorova koje mogu biti međusobno ugniježđene. To su deklaracija elementa i deklaracija atributa, definicija složenog tipa i definicija jednostavnog tipa, bez obzira radi li se o globalnom ili bezimenom (tj. lokalno definiranom) tipu te definicija grupe elemenata i grupe atributa. XML Schema se formalno može opisati kao desetorka

sch=(E, A, e, a, C, S, c, s, G, H)

pri čemu su:

- E skup svih deklariranih elemenata,
- A skup svih deklariranih atributa,
- e skup svih globalnih elemenata, podskup skupa svih deklariranih elemenata, $e \subseteq E$,
- a skup svih globalnih atributa, podskup skupa svih deklariranih atributa $a \subseteq A$,
- C skup svih definicija složenih tipova,
- S skup svih definicija jednostavnih tipova,
- c skup svih definicija globalnih složenih tipova,
- s skup svih definicija globalnih jednostavnih tipova,
- G skup svih grupa elemenata,
- H skup svih grupa atributa.

Jednostavni i složeni tipovi te grupe elemenata i atributa u funkciji su definiranja ugniježđene strukture elemenata, koja još sadrži i atribute. Ugniježđenu strukturu tj. strukturu XML dokumenta koji se ravna po XML Schemi prikazuje graf XML Scheme. Kako se neki složeni tip ili globalni element može više puta u shemi koristiti u deklaraciji podelementa, struktura definirana složenim tipom odnosno struktura

globalnog elementa sa svim podelementima može se u XML dokumentu koji odgovara XML Schemi pojaviti mnogo puta. U grafu sheme prikaz se može napraviti ili samo jednom jedinstvenom strukturom ili zasebnim strukturama.

Graf sheme (engl. *schema graph*) je nepravo stablo (po definiciji u poglavlju 6.1) koje opisuje XML strukturu definiranu XML Schemom na način prikladan za korisničko razumijevanje semantike i strukture. Graf sheme ima tri tipa čvorova: elemente, atribute i operatore kardinalnosti pridruživanja te posebni bezimeni korijenski čvor v_0. Graf sheme je osmorka:

$gs=(E, A, O, G, X, R, P, F)$

gdje je:

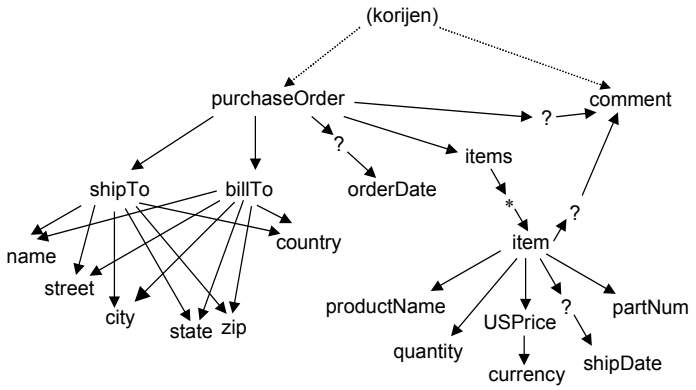
- E skup elemenata. Svaki element $e_i \in E$ je čvor u grafu sheme, a može se poistovjetiti s parom $e_i(n_i, t_i)$, pri čemu n_i ime elementa, a t_i njegov tip. Tip elementa $t_i \in C \cup S \cup T$ poprima vrijednost iz skupa složenih tipova C ili jednostavnih tipova S definiranih XML Schemom, ili iz skupa 44 osnovna jednostavna tipa T.
- A skup atributa. Svaki atribut $a_j \in A$ je čvor u grafu sheme te se može poistovjetiti s parom $e_j(n_j, t_j)$ koji čine ime atributa n_j i tip atributa t_j. Tip atributa $t_j \in S \cup T$ poprima vrijednosti iz skupa jednostavnih tipova S definiranih XML Schemom, ili iz skupa 44 osnovna jednostavna tipa T.
- O skup operatora kardinalnosti pridruživanja. Za pridruživanje različito od obavezne veze prema-jedan u grafu sheme se između nadređenog elementa te podređenog elementa ili atributa stavlja poseban čvor. Tipovi kardinalnosti veza predstavljeni su operatorima preuzetima iz DTD-a: $O=\{?, +, *\}$.
- G skup globalnih elemenata $G \subseteq E$. Svaki globalni element $g_i \in G$ je potčvor korijenskog čvora v_0.
- X skup uređenih parova oblika (v_0,g_i) pri čemu je $g_i \in G$, i predstavlja grane koje korijenski element grafa sheme vežu sva svakim od globalnih elemenata.
- R skup uređenih parova oblika (v_i,v_j) pri čemu je $v_i \in E \cup O$, $v_j \in E \cup A \cup O$, $v_i \neq v_j$, $v_i \in O \rightarrow v_j \notin O$, $v_j \in O \rightarrow v_i \notin O$ takvih da nepravo stablo sa skupom vrhova V, $V \subseteq (E \cup A \cup O)$ i skupom grana R, $qt(g_k)=(V, R)$ ima korijen u globalnom elementu g_k. Grana je usmjerena iz čvora v_i u čvor v_j. Barem jedan od čvorova povezanih granom nije operator kardinalnosti.
- P skup primarnih ključeva u grafu sheme. Svaki ključ p_i obuhvaća jedan ili više čvorova $v \in (E_S \cup A)$ grafa sheme. E_S su elementi jednostavnog tipa $E_S=\{e_i \in E \mid t(e_i) \in S\}$.
- F skup stranih ključeva u grafu sheme. Svaki ključ f_i također obuhvaća jedan ili više čvorova $v \in (E_S \cup A)$.

Čvorovi grafa sheme predstavljaju elemente i atribute, a svaki element i atribut imaju ime i tip. Čvor u grafu sheme bit će grafički predstavljen imenom. Graf sheme ima jedinstveni korijen koji nije niti element niti atribut, a njegovi su potčvorovi globalni elementi i atributi. Budući da graf sheme služi za prikaz strukture elemenata u svrhu odabira činjenice i kasnijeg prepoznavanja funkcijskih ovisnosti među njegovim čvorovima, u njemu nije bitan poredak elemenata.

Svaki globalni element u XML Schemi je kandidat za osnovni čvor XML dokumenta. Tako osim dokumenta narudžbe robe (*purchase order*) iz dodatka A2, XML Schemi za narudžbu robe iz dodatka A1 u potpunosti odgovara i sljedeći dokument koji se sastoji samo od jednog jedinog elementa, globalnog elementa `comment`.

```
<?xml version="1.0"?>
<comment>I ovo je ispravno za purchaseOrder.xsd</comment>
```

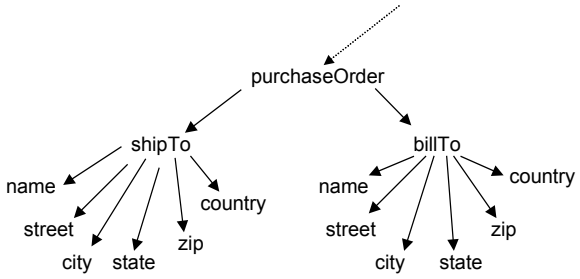
Slika 6.19 prikazuje graf XML Scheme za narudžbu robe iz dodatka A1. Oba globalna elementa, `purchaseOrder` i `comment`, su isprekidanom crtom povezana s korijenskim elementom. Element `purchaseOrder` ima podelemente `shipTo`, `billTo`, `comment` i `items` te atribut `orderDate`. Budući da su elementi `shipTo` i `billTo` istog složenog tipa, `USAddress`, njihovi podelementi su zajednički: u svaki od čvorova `name`, `street`, `city`, `state`, `zip`, `country` ulazi po jedna grana iz `shipTo` i jedna grana iz `billTo`. Globalni se element `comment` dvaput referencira od strane elemenata iz podstabla strukture globalnog elementa `purchaseOrder`. U grafu sheme na četiri su čvora operatora uvjetnog pridruživanja prema-jedan (?), a jedan je čvor operator uvjetnog pridruživanja prema-više (*).



Slika 6.19. Graf sheme za narudžbu robe

Prikazivanje svih referenci na globalni element istim čvorom te formiranje zajedničke podstrukture za elemente istog složenog tipa omogućuje bolje razumijevanje strukture za kasniji proces skladištenja. Međutim, takav je grafički prikaz nezgodan za izvedbu u programskim sustavima jer za njega ne postoje uobičajene gotove komponente.

Za pravo stablo postoje gotove komponente: u paketu Java Swing to je klasa `javax.swing.JTree` uz koju je vezan i paket klasa za grafičku potporu i obradu čvorova stabla, `javax.swing.tree`. Prikaz nepravim stablom nudi potpuniji, a pravo stablo jasniji pregled nad strukturom elemenata i atributa jer u svaki čvor takvog grafa ulazi samo jedna grana (slika 6.20).



Slika 6.20. Dio grafa sheme prikazan u obliku pravog stabla

Kod izbora činjenice (3. korak algoritma u poglavlju 6.2) projektantu skladišta mora jasno biti prikazan svaki čvor zasebno. U sučelju aplikacije i *appleta* će stoga graf sheme biti prikazan kao pravo stablo.

6.5. Pojednostavnjenje XML Scheme

Pojednostavnjenje XML Scheme prvi je korak algoritma za konceptualno oblikovanje područnog skladišta podataka iz XML izvora opisanog u poglavlju 6.2. Ono obuhvaća različite operacije nad XML Schemama u radnom projektu prije nego se napravi graf sheme. Pojednostavnjenje uključuje:

- uklanjanje elemenata i atributa u XML Schemama koji ne opisuju strukturu XML dokumenta i nisu upotrebljivi za dalji postupak oblikovanja skladišta,
- pojednostavnjenje složenih ugniježđenih struktura (uključujući dodavanje novih elemenata i atributa ili izbacivanje postojećih),
- prilagođavanje elemenata i atributa XML Scheme s ciljem omogućavanja njihove dalje nedvosmislene obrade.

Pojednostavnjena XML Schema više nije valjana s obzirom na predložak za XML Schemu i u nastavku služi samo za izradu grafa sheme. Slijedi opis svih postupaka pojednostavnjenja.

Izbacivanje elementa `annotation` i njegovih podelemenata. U svrhu boljeg razumijevanja od strane korisnika te za dodatne opaske aplikacijama koje s njom rade, XML Schema definira element `annotation` s mogućim podelementima `documentation` i `appInfo`. Podelement `documentation` sadrži proizvoljan tekst namijenjen korisnicima uz navođenje jezika kojim je pisan tekst u svom atributu `xml:lang`. Podelement `appInfo` sadrži upute aplikacijama, obrascima (engl. *stylesheets*) i drugim programskim alatima. Element `annotation` može se pojaviti kao potčvor elemenata `schema`, `element`, `attribute`, `complexType` ili `simpleType`. Kako je njegovo pojavljivanje beskorisno za izgradnju grafa sheme i kasnije oblikovanje skladišta, svaki se takav čvor u zadanoj XML Schemi uklanja.

Izbacivanje opisa ograničenja kod korisnički definiranih jednostavnih tipova. Za potrebe oblikovanja skladišta dovoljno je znati osnovni tip iz kojeg je korisnički tip izveden, budući da se prilikom logičkog oblikovanja 44 osnovna odnosno 19 primitivnih tipova svode na 3 ili 4 tipa u bazi podataka. Primjer pojednostavnjenja opisan je na tipu SKU u XML Schemi za narudžbu robe (isječak iz XML Scheme na slici 6.21).

```
<xsd:simpleType name="SKU">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}"/>
   </xsd:restriction>
</xsd:simpleType>
```

Slika 6.21. Definicija jednostavnog tipa

Valjanu vrijednost za tip određuje atribut `value` elementa `pattern`: SKU je niz od pet znakova, od čega su prva tri znamenke (`d{3}`), a posljednja dva velika slova (`[A-Z]{2}`). U pojednostavnjenoj verziji (slika 6.22) element `pattern` je izbačen.

```
<xsd:simpleType name="SKU">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
```

Slika 6.22. Pojednostavnjena definicija jednostavnog tipa

Pojednostavnjenje ugniježđenih struktura `sequence`, `choice` i `all`. Umjesto međusobno ugniježđenih struktura `sequence`, `choice` i `all` želi se dobiti sve deklaracije podelemenata na istoj razini kao podelemente od `complexType`, opisane odgovarajućim oznakama kardinalnosti pridruživanja. Kako u procesu konceptualnog oblikovanja skladišta poredak elemenata nije važan, razlike između `sequence` i `all` postaju nebitne.

Složenost procesa izbacivanja `sequence`, `choice` i `all` te potrebu njegovog provođenja u nekoliko koraka ilustrirat će se usporedbom dvaju primjera.

U prvom primjeru (slika 6.23) dani su identifikacijski podaci studenta. Uobičajeno je da se student identificira imenom i prezimenom, u kombinaciji s jedinstvenim matičnim brojem akademskog građanina JMBAG: to je zapravo broj indeksa studenta).

```
<xsd:element name="student">
  <xsd:complexType>
    <xsd:choice>
      <xsd:sequence>
        <xsd:element name="jmbag" type="xsd:string"/>
        <xsd:element name="ime" type="xsd:string">
        <xsd:element name="prezime" type="xsd:string">
      </xsd:sequence>
      <xsd:sequence>
         <xsd:element name="ime" type="xsd:string"/>
         <xsd:element name="prezime" type="xsd:string"/>
         <xsd:element name="datRodj" type="xsd:date"/>
      </xsd:sequence>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Slika 6.23. Višestruka deklaracija istog elementa unutar strukture *choice*

Ukoliko nije poznat JMBAG (npr. student je zaboravio indeks) navodi se ime, prezime te datum rođenja (vjerojatnost da na fakultetu studiraju dvije osobe istog imena i prezimena rođene istog datuma je minimalna).

U obje opcije koje nudi element `choice` student bit će točno jednom pridruženi podelementi ime i prezime. Elementi `jmbag` i `datRodj` pojavljuju se samo u jednoj od opcija pa je njihovo postojanje u dokumentu uvjetno. Uklonivši elemente `sequence` i `choice`, element `complexType` imat će za podelemente četiri deklaracije elemenata. Budući da je pojavljivanje `jmbag` i `datRodj` uvjetno, u njihovu se deklaraciju postavlja atribut `minOccurs` s vrijednošću `"0"`. Pojednostavljeni isječak XML Scheme prikazuje slika 6.24.

Drugi primjer opisuje mrežu linija gradskog tramvaja navodeći stanice i udaljenosti između njih u kilometrima. Pritom se posebno ističe zadnja stanica. Isječak XML Scheme dan je na slici 6.25. XML Schemi odgovaraju fragmenti na slici 6.26.

```
<xsd:element name="student">
  <xsd:complexType>
     <xsd:element name="jmbag" type="xsd:string" minOccurs="0"/>
     <xsd:element name="ime" type="xsd:string"/>
     <xsd:element name="prezime" type="xsd:string"/>
     <xsd:element name="datRodj" type="xsd:date" minOccurs="0"/>
  </xsd:complexType>
</xsd:element>
```

Slika 6.24. Deklaracije elemenata nakon izbacivanja *sequence* i *choice*

```
<xsd:element name="dionica">
   <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="stanica" type="xsd:string"/>
        <xsd:choice>
           <xsd:element name="stanica" type=" xsd:string "/>
           <xsd:element name="zadnja" type=" xsd:string "/>
        </xsd:choice>
        <xsd:element name="km" type="xsd:decimal">
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Slika 6.25. Više deklaracija istog elementa od kojih neke nisu unutar strukture *choice*

```
<dionica>
    <stanica>Jelacicev Trg</stanica>
    <stanica>Frankopanska</stanica>
    <km>0.5</km>
</dionica>

<dionica>
    <stanica>Budakova</stanica>
    <zadnja>Borongaj</zadnja>
    <km>0.4</km>
</dionica>
```

Slika 6.26. Primjer XML fragmenta prema isječku sheme sa slike 6.25

U svakom elementu dionica, podelement `stanica` pojavljuje se barem jednom, a ponekad i više puta, pa će veza s nadređenim elementom biti obavezna prema-više. U deklaraciju elementa `stanica` dodaje se atribut `maxOccurs` postavljen na `"unbounded"` (prava vrijednost formalno bi bila `"2"`).
Kardinalnost podelementa `zadnja` može se opisati operatorom uvjetne veze prema-jedan. Konačni pojednostavnjeni prikaz deklaracija podelemenata (analogno prikazu na slici 6.24) daje slika 6.27.

```
<xsd:element name="dionica">
   <xsd:complexType>
      <xsd:element name="stanica" type="xsd:string"
                                          maxOccurs="unbounded"/>
      <xsd:element name="zadnja" type="xsd:string"/ maxOccurs="0">
      <xsd:element name="km" type="xsd:decimal">
  </xsd:complexType>
</xsd:element>
```

Slika 6.27. Deklaracije elemenata nakon izbacivanja *sequence* i *choice*

Uspoređujući ova dva primjera vidimo da je u prvom slučaju višestruka deklaracija pretvorena u jednostruku uz zadržavanje kardinalnosti prema-jedan, dok je u drugom primjeru kardinalnost konačnog elementa postavljena kao prema-više. U prvom slučaju vršila se obrada strukture `choice` koja je u sebi sadržavala strukture `sequence`. U drugom slučaju naglasak je bio na strukturi `sequence` koja je u sebi je sadržavala i strukturu `choice`.

Može se zaključiti da se eliminacija struktura `sequence`, `choice` i `all` vrši prema sljedećem algoritmu:

- vrši se prolazak kroz ugniježđene strukture `sequence`, `choice` i `all` do zadnje (najdublje) razine ugniježđenosti. Uklanjanje se provodi korak po korak od najdublje razine pa sve do strukture koja je direktni podelement od `complexType`
- kod strukture `sequence` (i `all`) višestruka pojava istog elementa (istog imena i istog tipa) rezultira jedinstvenim elementom s kardinalnošću prema-više. U deklaraciji elementa atribut `maxOccurs` se postavi na vrijednost `"unbounded"`. Ako je bilo koja instanca elementa uvjetno prisutna, stopljeni element također će biti uvjetno pridružen nadređenom elementu. Zbog jednostavnosti, element se zove a. Opisan je operatorima kardinalnosti pridruživanja, (za obavezno pridruživanje prema-jedan nema operatora)

a? , a? → a*
a? , a → a+
a? , a+ → a+
a? , a* → a*
a , a → a+
a , a+ → a+
a , a* → a+
a+ , a+ → a+
a+ , a* → a+
a* , a* → a*

- kod strukture `choice` treba razlikovati element koji se pojavljuje unutar svake iznesene opcije (u primjeru sa slike 6.23 to su bili `ime` i `prezime`) od ostalih elemenata. Element koji se unutar svake opcije pojavljuje bar jednom, nužno se mora pojaviti u cjelini. Za opis transformacija koristi se oznaka | iz DTD-a koja označava isključivo ILI (tj. upravo strukturu ekvivalentnu `choice` u XML Schemi). Transformacije su sljedeće:

(a? | a?) → a?
(a? | a) → a?
(a? | a+) → a*
(a? | a*) → a*
(a | a) → a

(a | a+) → a+

(a | a*) → a*

(a+ | a+) → a+

(a+ | a*) → a*

(a* | a*) → a*

Ostali elementi dobivaju oznaku uvjetnog pridruživanja. U deklaraciju elemenata atribut `minOccurs` postavlja se na vrijednost `"0"`. Maksimalni broj pojavljivanja elementa (tj. odluka o vezi prema-jedan ili prema-više) se ne mijenja. Taj slučaj odgovara transformacijama opisanim u [STH+99] koje se u načelu svode na (a | b) → a?, b?:

a? → a?

a → a?

a+ → a*

a* → a*

Postavljanje oznake obaveznog pridruživanja kod elementa ili atributa s propisanom vrijednošću. Kako je istaknuto u poglavlju 6.3.2.6, elementima i atributima se u deklaraciji može postaviti stalna vrijednost (`fixed`) ili zadati predefinirana vrijednost (`default`). Ako je pojavljivanje u dokumentu uvjetno, program za kontrolu ispravnosti dokumenta u odnosu na XML Schemu kao predložak proglasit će valjanim dokument u kojem dani element ili atribut nije naveden, ali će ga potom dodati i postaviti na vrijednost propisanu XML Schemom. Kako je vrijednost elementa odnosno atributa poznata makar ga se i ne navede, može se pretpostaviti da se on u dokumentu pojavljuje uvijek. Prilikom oblikovanja procesa izvlačenja, transformacije i učitavanja podataka vodit će se računa o predefiniranoj vrijednosti, za slučaj da se element odnosno atribut ne pojavi u XML dokumentu iz kojeg se podaci spremaju. U slučaju postojanja `fixed` ili `default` u deklaraciji atributa, deklaracija se proširuje atributom `use` postavljenim na `"required"`. Ako su `fixed` ili `default` prisutni u deklaraciji elementa, njegova se deklaracija se proširuje atributom `minOccurs` postavljenim na `"1"`.

Brisanje oznake sužavanja tipa elementa. Složeni se elementi se mogu izvesti proširenjem (`extension`) ili sužavanjem (`restriction`) s obzirom na neki drugi podelement. U slučaju proširivanja, na sve postojeće podelemente i atribute dodaju se novi. U slučaju sužavanja potrebno je navesti sve podelemente i sve atribute koji će biti zadržani. Kako je za izradu grafa sheme nevažno da jedan tip potječe od drugog ako su svi podelementi i atributi već navedeni u njegovoj definiciji, podatak da se radi o sužavanju drugog tipa može se ukloniti. Kod proširenja se u novoj definiciji navodi samo dio strukture, pa informacija o proširenju mora ostati.

Nakon izvršavanja ovih transformacija koje su se obavljale uklanjanjem dijelova XML Scheme i dodavanjem novih, može se pristupiti izradi grafa sheme, strukture oblika nepravog stabla.

6.6. Izrada grafa sheme

Graf sheme izgrađuje se postupnim ugnježđivanjem malih stabala definiranih složenim tipovima elemenata. Ugniježđena struktura nastaje kad se u definiciji složenog tipa nalaze podelementi drugih složenih tipova. Dodajući na te podelemente stabla drugih složenih tipova sukcesivno se formira cjelokupna struktura ukorijenjena u globalnom elementu.

Korijenski element stabla sadrži sve globalne elemente pa se graf sheme na prvi pogled sastoji od više nezavisnih nepravih podstabala. Najčešće ipak postoji globalni element koji u svom podstablu uključuje sve ostale globalne elemente. U tom se slučaju radi o jedinstvenoj strukturi, a zadatak je projektanta skladišta podataka uočiti je. U primjeru narudžbe robe (slika 6.19) već je ranije pokazano (poglavlje 6.4) kako je globalni element *comment* sadržan na više mjesta unutar podstabla globalnog elementa *purchaseOrder*.

Uz graf su pohranjene informacije o svim imenicima koji se pojavljuju u shemi.

U nastavku će biti opisani koraci u izradi grafa sheme.

6.6.1. Popisivanje imenika

Prvi korak u izgradnji grafa je izrada popisa svih imenika koji se spominju u XML Schemama radnog projekta. Zapisivanje se provodi datoteku po datoteku. Imenici koji se koriste u XML Schemi zapisani su posebnom vrstom čvora kao potčvorovi korijenskog elementa `schema` (detaljan opis dan je u ranijim poglavljima 2.3.2 i 5.1.2).

Na temelju pregleda elemenata `import` za svaku se XML Schemu (tj. pojedinačnu datoteku) doznaje kako ona ovisi o ostalim XML Schemama iz radnog projekta. U projektu uvijek postoji jedna ili više XML Schema koje ne koriste strukture definirane izvan matične datoteke. One se obrađuju na početku, a dobivene strukture koriste u obradi ostalih XML Schema. Za sve se imenike jedne XML Scheme zapišu parovi URI-prefiks te imenik u koji se spremaju u njoj definirane strukture (ciljni imenik, *target namespace*).

Aplikacija ne podržava definiranje istog imenika u više različitih datoteka (za razliku od elementa `import` koji služi unosu drugih imenika, za unos dijelova imenika iz drugih datoteka koristi se `include`, također izravni podelement korijenskog elementa `schema`).

Kad se prilikom stvaranja novog radnog projekta ispituje jesu li sve odabrane XML Scheme međusobno kompatibilne (poglavlje 5.1.2), zapravo se vrši stvaranje grafa sheme. Stvoreni graf sheme će potom biti prikazan u aplikaciji u sklopu novog projekta.

6.6.2. Popisivanje globalnih struktura

Sljedeći korak u izgradnji grafa sheme je izrada odvojenih popisa svih jednostavnih tipova, složenih tipova, grupa atributa i elemenata, globalnih atributa, globalnih elemenata te primarnih i stranih ključeva. Na popis jednostavnih odnosno složenih tipova te zajednički popis grupa atributa i grupa elemenata odmah se dodaju globalno definirani tipovi i grupe, dok se bezimeni tipovi dodaju kasnije, prilikom same izgradnje stabla. Globalni atributi i elementi odmah se stavljaju na svoje popise, dok je popis primarnih i stranih ključeva početno prazan.

6.6.3. Izgradnja podstabala

Nakon što su popisom evidentirane sve globalne strukture prelazi se na njihovo povezivanje u složene stablaste strukture i zapisivanje kardinalnosti veza.

U načelu se stablo gradi od listova prema korijenu, dodajući već poznate strukture podelemenata na novostvoreni nadređeni element. Postupak se provodi rekurzivno i tako od malih stabala ugnježđivanjem grade veća. Stvaranje podstabala tipova, grupa te globalnih atributa i elemenata najbolje je provesti onim redom koji će u prosjeku izazvati najmanje nailazaka na nedefinirane složene strukture.

Stoga se najprije definiraju globalni atributi jer su jednostavnog tipa. Slijede definicije stabala za grupe atributa, jer sadrže samo lokalne ili već obrađene globalne atribute. Nakon toga određuju se grupe elemenata pa složeni tipovi. Globalni elementi predstavljaju vrh stablaste strukture pa se određuju posljednji. Graf sheme sastoji se od međusobno nepovezanih struktura globalnih elemenata. Na samom se završetku obrađuju reference na globalne elemente koje omogućuju da se globalni element s cijelom svojom strukturom ugradi u drugi globalni element kao dio njegove strukture (u primjeru narudžbe robe iz dodatka A1, u definiciji strukture globalnog elementa `purchaseOrder` koriste se reference na globalni element `comment`:

```
<xsd:element  ref="comment"  minOccurs="0"/>.
```

6.6.4. Označavanje ključeva

Nakon što su obrađene reference na globalne elemente, vrši se obrada primarnih ključeva. Već se u fazi izgradnje malih podstabala za svaku deklaraciju elementa provjerava sadrži li `element` podelement `key` ili `keyref` te se postavlja zastavica da element sadrži definiciju primarnog ili stranog ključa. Važno je da se ovdje ne radi o elementu koji jest primarni ili strani ključ već o elementu koji unutar čije podstrukture vrijedi ograničenje primarnog ključa odnosno referencijalni integritet stranog ključa. Prave ključeve možemo označiti tek nakon formiranja cjelokupne strukture grafa sheme. Ključ se označava zastavicom (primarnog odnosno stranog ključa).

Kod prepoznavanja stranih ključeva postupa se jednako kao u slučaju primarnih, no za strani je ključ potrebno dodatno provjeriti postojanje primarnog ključ na koji se odnosi strani te podudaranje tipova i redoslijeda tipova između stranog i primarnog ključa.

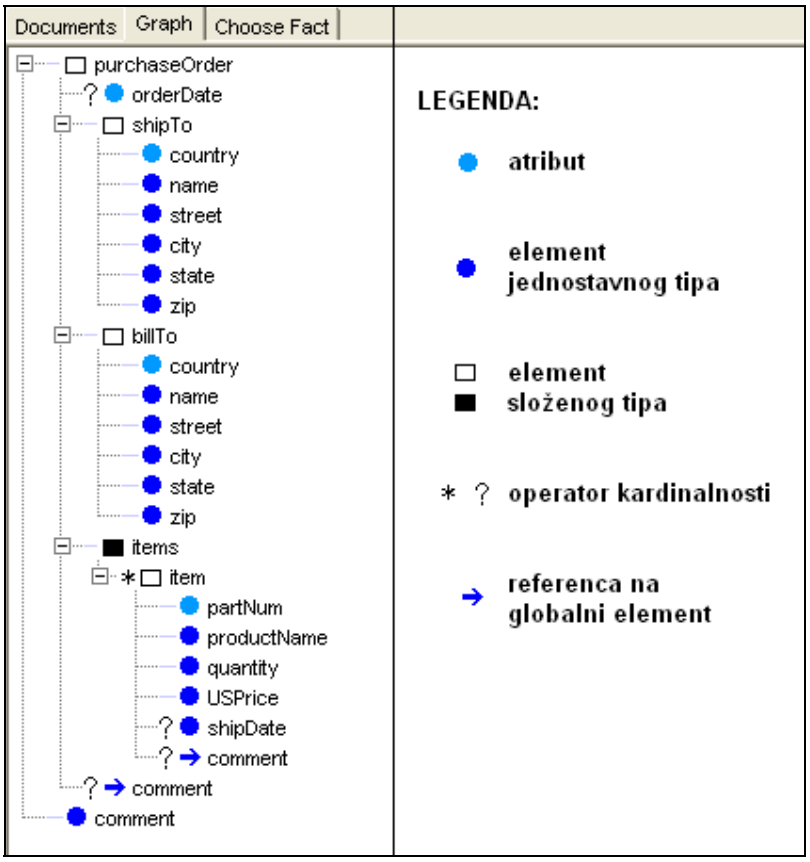
6.6.5. Prikaz grafa sheme u programskom sustavu

Slika 6.28 prikazuje graf sheme za narudžbu robe u grafičkom sučelju aplikacije odnosno *appleta*. Kako je naglašeno u poglavlju 6.4, u programskom je grafičkom sučelju znatno jednostavniji i pregledniji prikaz u obliku pravog stabla u odnosu na nepravo stablo koje koristi u "ručnoj" skici (slika 6.19). Zbog toga su atribut i podelementi od `shipTo` i `billTo` prikazani odvojeno, makar se radi o istovjetnoj strukturi. Svi čvorovi u grafu sheme su jednaki s matematičko-logičkog stajališta, ali se u programu razlikuju u grafičkom prikazu (ikoni) kako bi korisnik programa imao jasan uvid o kakvom se tipu čvora XML Scheme radi.

Korijen grafa sheme u prikazu je skriven budući da nije niti element niti atribut, pa se vidljivi dio grafa sastoji od međusobno odvojenih struktura globalnih elemenata `purchaseOrder` i `comment`.

Atributi su prikazani svjetloplavim krugom (`orderDate`, `country`, `partNum`), a elementi jednostavnog tipa tamnoplavim krugom (globalni element `comment`, lokalni

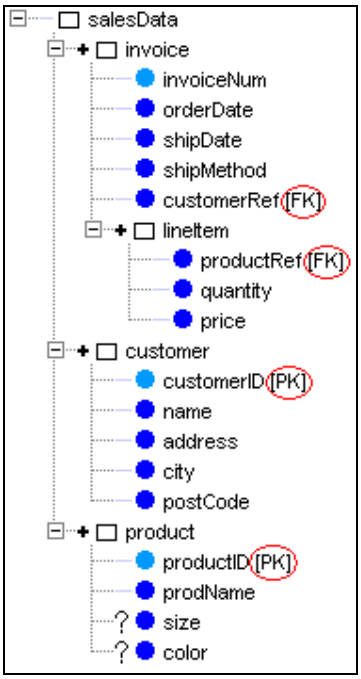
elementi `name`, `street`, `productName` itd.). Elementi složenog tipa, bez vlastitog tekstualnog čvora, prikazani su pravokutnikom bijele (`purchaseOrder`, `shipTo`, `billTo`, `item`) ili crne boje (`items`). Bijeli pravokutnik označava "prazninu", nepostojanje sadržaja u obliku tekstualnog čvora. Razlika između elemenata prikazanih bijelim odnosno crnim pravokutnikom bit će objašnjena u narednom poglavlju, 6.7.1.



Slika 6.28. Prikaz grafa sheme za narudžbu robe u aplikaciji/*appletu*

Graf sheme sadrži i operatore kardinalnosti (npr. element `item` je uvjetno pridružen elementu `items` s kardinalnošću prema-više). U grafu sheme za podatke o trgovini (skica na, prikaz u programu na slici 6.29) elementi i atributi koji predstavljaju primarni ključ imaju oznaku PK (`customerId`, `productID`), a oni koji predstavljaju strani ključ oznaku FK (`productRef`, `customerRef`).

U nastavku algoritma koristit će se samo graf sheme u obliku pravog stabla.



Slika 6.29. Prikaz grafa sheme za podatke o trgovini u aplikaciji/*appletu*

6.7. Odabir činjenice

Odabir činjenice predstavlja ključni korak u procesu konceptualnog oblikovanja skladišta podataka. On se ne može izvesti automatski, već ovisi o projektantovom razumijevanju semantike XML Scheme. To je i prvi korak u algoritmu za oblikovanje područnog skladišta podataka koji se ne izvodi u potpunosti automatski.

Programski sustav ne razumije značenje pojedinih čvorova u grafu sheme. On mora omogućiti korisniku da za činjenicu odabere bilo koji čvor u grafu sheme (što u modelu entitet-veza predstavlja entitet) ili pak vezu dvaju čvorova tj. granu u grafu sheme (što u modelu entitet-veza predstavlja vezu).

6.7.1. Izuzimanje čvorova iz skupa kandidata za činjenicu

U grafu sheme mogu postojati čvorovi koji zbog svojih svojstava sigurno ne mogu biti odabrani za činjenicu. To su čvorovi koji nemaju vlastiti semantički sadržaj. Pod semantičkim sadržajem čvora podrazumijevaju se vrijednosti svih njegovih atributa i svih njemu podređenih elemenata bez obzira na razinu ugniježđenosti, zajedno s vrijednostima svojih atributa. Element čiji je semantički sadržaj jednak semantičkom sadržaju jednog mu podređenog elementa nema vlastiti semantički sadržaj, već preuzima semantički sadržaj tog podelementa.

Opisano svojstvo vrijedi za elemente koji imaju jedan jedini podelement složenog tipa (bez obzira na kardinalnost pridruživanja) i niti jedan atribut. Takvi elementi u XML dokumentu postoje kako bi njegovu strukturu učinili logički jasnijom. Oni su u [VBR03a] nazvani kontejnerima (engl. *containers*). Kontejneri čiji jedini podelement ima kardinalnost pridruživanja prema-više česti su u XML Schemama. Slučaj u kojem bi podelement imao kardinalnost prema-jedan je izuzetno rijetka jer u tom slučaju kontejner nema niti bilo kakvo logičko značenje.

Drugi tip kontejnera je korijenski element XML dokumenta koji ima više od jednog podelemenata, ali su mu svi pridruženi s kardinalnošću prema-više. Razlog zbog

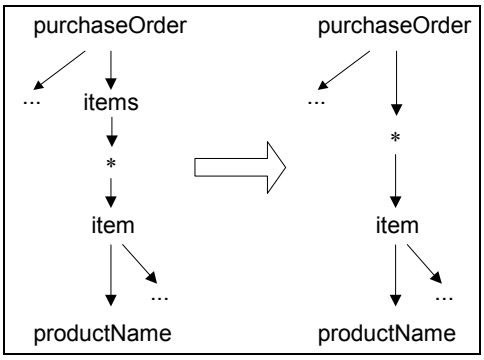
kojeg ovakav element postaje nepotrebni kontejner bit će razumljiv nakon što se opiše proces izgradnje stabla ovisnosti (poglavlje 6.8).

Općenito, za svaki element koji ima barem jedan atribut ili više od jednog podelementa može se zaključiti da nije kontejner jer niti jedan podelement nema semantički ekvivalentan sadržaj. Ova konstatacija ne vrijedi za specijalni slučaj kontejnera drugog tipa.

U XML Schemi za narudžbu robe kontejner je element `items`. Iz grafa sheme na slici 6.19 vidljivo je da `items` ima samo jedan podelement, `item`, pridružen s kardinalnošću prema-više. `items` nema nikakav sadržaj već služi da se cjelina u kojoj će se navoditi puno proizvoda (`item`) naglašeno odvoji od ostatka dokumenta.

Izbacivanjem kontejnera sve njegove podelemente dobiva njemu nadređeni element kao vlastite podelemente. Ako je nadređeni element kontejner, on se također izbacuje. Podelementi kontejnera će novom nadređenom elementu biti pridruženi s istom kardinalnošću kao što su bili pridruženi kontejneru. Kontejner se u grafu sheme razlikuje od ostalih elemenata složenog tipa. Dok su oni predstavljeni pravokutnicima bijele boje, kontejnere označavaju crni pravokutnici.

U primjeru XML Scheme za narudžbu robe izbacuje se element `items`, a njegov jedini podelement `item` pridružuje se elementu `purchaseOrder`, koji je bio nadređen elementu `items` (slika 6.30). Element `item` pridružen je elementu `purchaseOrder` s kardinalnošću prema-više kao što je bio pridružen i elementu `items`.



Slika 6.30. Uklanjanje kontejnera

Kontejner drugog tipa uočava se u grafu sheme koji opisuje poslovanje trgovine (odnosno XML Schemu `salesdata.xsd` iz dodatka B1), a prikazan je na slici 6.15 odnosno slici 6.29. Kontejner je sam element `salesdata` kojem su sva tri podelementa: `customer`, `product` i `invoice` pridruženi s kardinalnošću prema-više. Uklanjanjem tog kontejnera graf sheme raspada se u tri nezavisne cjeline koje kasnije ipak postaju zavisne uslijed djelovanja mehanizma primarnog i stranog ključa.

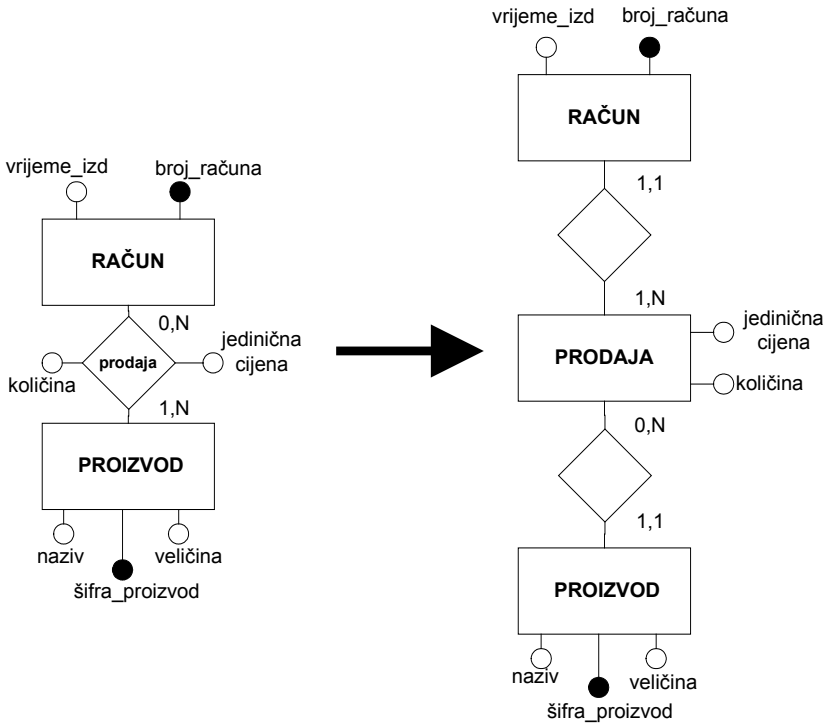
6.7.2. Mogućnosti izbora činjenice

Programski sustav projektantu skladišta za činjenicu nudi sve čvorove u grafu (sve atribute i sve elemente) osim izbačenih elementa-kontejnera te sve veze među čvorovima.

U višedimenzijskom konceptualnom modelu činjenični entitet tvori vezu prema-jedan sa svakom od dimenzija, dok one s njim tvore vezu prema-više. Promatra li se

njegova izvedba u relacijskoj bazi, svakom zapisu u činjeničnoj tablici odgovara samo jedan zapis u svakoj dimenziji. Bilo kojem zapisu u bilo kojoj dimenzijskoj tablici može odgovarati (a u praksi i odgovara) više zapisa u činjeničnoj tablici.

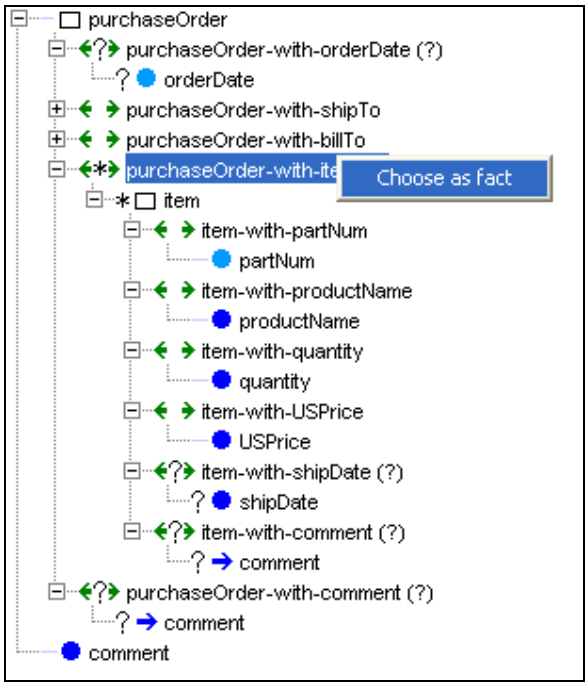
Stoga se u modelu entitet-veza za činjenicu bira ili entitet s kojim drugi entiteti tvore vezu prema-više, a on s njima vezu prema-jedan, ili pak veza više-prema-više. Naime, veza više-prema-više (M:N) se može pretvoriti u novi entitet kojem su oba osnovna entiteta pridružena s kardinalnošću prema-jedan, a on njima s kardinalnošću prema-više, tako da je novi entitet odličan izbor za činjenicu. U modelu entitet-veza za maloprodajno poduzeće vezu više-prema-više čine račun i proizvod (slika 1.1). Veza se zove prodaja. Ona se pretvara u entitet prodaja (slika 6.31). Prodaja s entitetima račun i proizvod tvori vezu prema-jedan.



Slika 6.31. Pretvaranje veze više-prema-više u zasebni entitet

Programski alat pretvara sve veze u grafu sheme u dodatne čvorove grafa i nudi ih, zajedno s čvorovima koji predstavljaju elemente i atribute, projektantu kao moguće činjenice. Veze pretvorene u čvorove opisane su ikonama sa zelenim strelicama i operatorom kardinalnosti.

U grafu sheme uočljiva je veza prema-više između narudžbe (`purchaseOrder`) i naručenog proizvoda (`item`). Veza od proizvoda prema narudžbi nije definirana u XML Schemi, no projektant skladišta zna da se isti proizvod može nalaziti u mnogo narudžaba. Stoga se radi o vezi više-prema-više i upravo ta veza postaje činjenica (slika 6.32). Nužno potrebnu vremensku dimenziju daje `orderDate`. U XML dokumentu sadržani su parametri cijene (`USPrice`) i naručene količine (`quantity`) koji će biti mjere. Njihov umnožak je prihod od prodaje, `income`, najvažnija mjera područnog skladišta podataka.



Slika 6.32. Odabir činjenice pomoću grafičkog sučelja.

U grafu sheme s podacima o trgovini (slika 6.29) za činjenicu se odabire element `lineItem` (stavka računa tj. jedan redak na računu koji opisuje kupnju pojedinog artikla). `lineItem` je preko stranog ključa `productRef` povezan s proizvodom tj. elementom `product` (jedna stavka računa sadrži jedan proizvod, ali se proizvod može naći na puno računa odnosno stavki računa), a preko elementa `invoice` (račun) na potpuno isti način s kupcem tj. elementom `customer`. Vremensku dimenziju daje datum izdavanja računa (`orderDate`). `lineItem` sadrži elemente `price` i `quantity` koji će postati mjere, kao i njihov umnožak, ostvareni prihod.

6.8. Izgradnja grafa ovisnosti

Graf ovisnosti (engl. *dependency graph*) je osnova činjenične sheme kao konceptualnog modela područnog skladišta podataka. Prema algoritmu za izgradnju konceptualnog skladišta podataka iz XML izvora (poglavlje 6.2) graf ovisnosti izgrađuje se poluautomatski, uz pomoć programskog alata (korak 4.1). Činjenica predstavlja korijenski čvor i u grafu ovisnosti i činjeničnoj shemi. Preostali čvorovi grafa ovisnosti su kandidati za mjere te dimenzijske i opisne atribute.

6.8.1. Načela izgradnje grafa ovisnosti

Čvorovi grafa ovisnosti dobivenog poluautomatskim postupkom izgradnje čine podskup čvorova grafa sheme. Proces izgradnje grafa ovisnosti zapravo je rekurzivno preslikavanje grafa sheme koje počinje od čvora određenog za činjenicu i nastavlja se čvor po čvor, dok god grane predstavljaju funkcijske ovisnosti. U n-tom koraku rekurzije promatra se svaki čvor koji je u graf ovisnosti postavljen u (n-1)-om koraku. Ispituju se svi čvorovi s kojima promatrani čvor tvori vezu, a koji još nisu u stablu. Pritom je nebitan smjer grane koja predstavlja vezu. Ukoliko veza prema tim čvorovima ima kardinalnost prema-jedan riječ je o funkcijskoj ovisnosti pa ti čvorovi ulaze u graf ovisnosti. Oni će potom biti obrađeni u (n+1)-om koraku rekurzivnog procesa. U prvom je koraku rekurzije promatrani čvor činjenica. Valja naglasiti da

graf sheme koji se u ovom procesu preslikava u graf ovisnosti ima oblik pravog stabla i ne postoji više od jednog puta između dva čvora.

Postupak izgradnje grafa ovisnosti bit će objašnjen na primjeru grafa sheme s podacima o trgovini (`salesdata`). Graf sheme prikazan je na slici 6.29. Za činjenicu je odabran čvor `lineItem`. Na početku prvog koraka rekurzivnog procesa graf ovisnosti uvijek sadrži samo činjenični čvor, u ovom slučaju `lineItem`. Ovaj čvor ima tri potčvora: `productRef`, `quantity` i `price` te jedan nadređeni čvor: `invoice`. Veze između `lineItem` i triju potčvorova mogu se pročitati iz grafa sheme. Sve tri veze imaju kardinalnost prema-jedan pa ulaze u graf ovisnosti. Veza od čvora `lineItem` prema čvoru `invoice` je veza od podređenog elementa prema nadređenom te je poznata samo kardinalnost u suprotnom smjeru. Kardinalnost od podređenog elementa prema nadređenom može se odrediti ispitivanjem sadržaja XML dokumenata što će detaljno biti objašnjeno u poglavlju 6.8.4. U konkretnom slučaju veza će imati kardinalnost prema-jedan pa će `invoice` postati čvor grafa ovisnosti. U sljedećem koraku obradit će se čvorovi `productRef`, `quantity`, `price` i `invoice`. Čvor `productRef` je strani ključ na čvor `productID`. Na taj način jednoznačno se može povezati `productRef` i dio strukture grafa sheme oko `productID`. `productID` se može smatrati primarnim ključem strukture koju čine `product` i njegovi potčvorovi (uz `productId` to su još `productName`, `size` i `color`). Oni će također postati dio grafa sheme. Ovaj proces detaljno će biti objašnjen u poglavlju 6.8.5.

U ovom primjeru koriste se sva tri osnovna, po svojoj biti različita mehanizma kojima se grafu ovisnosti na promatrani čvor u grafu sheme dodaju novi čvorovi [GRV01, VBR03a, VBR03b, Vrd04]:

- izgradnja grafa ovisnosti od promatranog čvora prema potčvorovima: izgradnja u smjeru "prema dolje",
- izgradnja grafa ovisnosti od promatranog čvora prema nadređenom čvoru: izgradnja u smjeru "prema gore",
- dodavanje podstabla primarnog ključa na mjesto stranog ključa.

U slučaju da je za činjenicu izabrana veza dvaju čvorova, u graf ovisnosti najprije se dodaju dva čvora koje je ta veza povezivala. Kako je objašnjeno u poglavlju 6.7.2, kad se bilo koja binarna veza (a u praksi veza više-prema-više) pretvori u dodatni entitet, tom su novom entitetu oba osnovna entiteta pridružena s kardinalnošću prema-jedan (slika 6.31). Za čvor koji je u grafu sheme podređen onom drugom (u primjeru narudžbe robe čvor `item`) provodi se postupak izgradnje "prema dolje". Za čvor koji je u grafu sheme nadređen (u primjeru `purchaseOrder`) provodi se postupak izgradnje "prema dolje" ukoliko ima više od jednog potčvora (`purchaseOrder` osim `item` ima potčvorove `shipTo`, `billTo` i `comment`) te postupak izgradnje "prema gore" ukoliko taj čvor ima nadređeni element (`purchaseOrder` ga nema).

6.8.2. Ispitivanje veze prema nadređenom čvoru

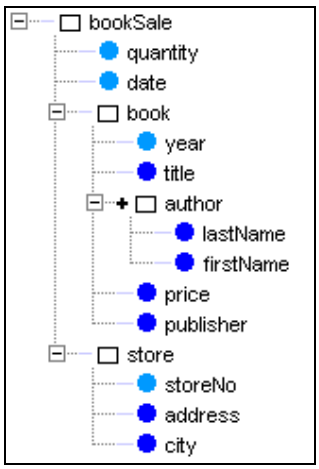
Prilikom izgradnje grafa "prema gore" potrebno je znati kardinalnost veze od nekog promatranog čvora prema njemu nadređenom čvoru. Kardinalnost veze nije zadana XML Schemom pa se nastoji utvrditi iz sadržaja XML dokumenata. Pronađe li se ispitivanjem sadržaja da je riječ o vezi prema-više, kardinalnost je određena. Naprotiv, ako se ispitivanjem sadržaja ne može ustanoviti da je veza kardinalnosti

prema-više, to još uvijek ne znači da ne postoje ili se ne mogu načiniti dokumenti na temelju čijeg sadržaja bi se ustanovilo da veza jest prema-više. U tom slučaju za kardinalnost se pita projektanta skladišta koji odluku mora donijeti na temelju razumijevanja semantike elemenata i atributa.

Ispitivanje je li veza od podređenog elementa prema nadređenom kardinalnosti prema više svodi se na ispitivanje postoji li jedan ili više dokumenata takvih da se za isti sadržaj podređenog elementa u tim dokumentima razlikuju sadržaji nadređenih elemenata. Pritom treba osobitu pažnju obratiti na pojam sadržaja elementa. Postoji sadržaj podelemenata i atributa koji ključno određuje element (u nekim slučajevima on se formalno specificira kao primarni ključ) te preostali sadržaj nevažan za identifikaciju. Primjerice, proizvod u trgovini ključno je određen svojim identifikacijskim brojem pa se u tom slučaju ispituje jedino sadržaj elementa ili atributa koji predstavlja identifikacijski broj.

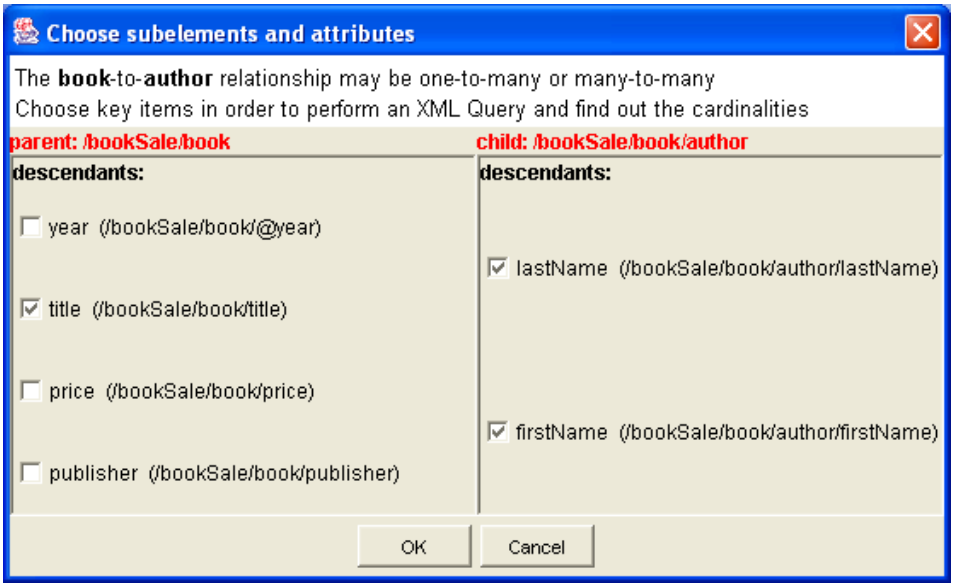
U dodatku C nalazi se XML Schema (dodatak C1) koja opisuje prodaju knjiga te njeni pripadajući dokumenti (dodatak C2, C3, C4, C5). XML Schema preuzeta je iz dokumenta XML Query Use Cases [Xquc03] koji je službeni dodatak radnog nacrta jezika XQuery. Graf sheme prikazuje slika 6.33. Iz njega je vidljivo da knjiga može imati više autora. Međutim, XML Schema ne definira može li više knjiga biti napisano od jednog autora pa je potrebno ispitati sadržaj dokumenata. Projektant skladišta mora odlučiti koji će podelementi i atributi elemenata `author` i `book` biti odabrani kao ključni. Za razlikovanje autora potrebno je znati i ime (`firstName`) i prezime (`lastName`). Budući da dostupni podaci o knjizi ne sadrže jednoznačni identifikator (međunarodni broj isbn) projektant mora odrediti koja će kombinacija preostalih podataka činiti ključ. U obzir dolaze svi potčvorovi od `book`, bez obzira na razinu, osim samog elementa `author`: `title`, `publisher`, `year` i `price`. Promatra li se knjiga sa stajališta autora, odnosno ispituje li se popularnost autora, za knjigu je bitan samo njen naslov. Naprotiv, promatra li se knjiga sa stajališta prodaje, uzima se u obzir kvaliteta papira, tip i kvaliteta uveza te veličina formata (džepno ili standardno izdanje). Tada knjigu određuju i naslov i izdavač i godina izdavanja.

Budući da se ispituje veza autora i knjige, za identifikator knjige uzima se samo njen naslov. Ako se radi o vezi prema-više, tada će za odabranog autora postojati više od jednog naslova knjige.



Slika 6.33. Graf sheme i izgradnja grafa ovisnosti za prodaju knjiga

U grafičkom se sučelju programa za oblikovanje skladišta podataka iz XML izvora unese `title` kao ključ od `book`, a `firstName` i `lastName` kao komponente ključa od `author` (slika 6.34: `title`, `firstName` i `lastName` označeni su kvačicom).



Slika 6.34. Određivanje ključnog sadržaja prilikom ispitivanja sadržaja XML dokumenta

Nakon toga se provodi upit u jeziku XQuery. Zbog ograničenja programske podrške za XQuery potrebno je sve zasebne XML dokumente spojiti u jedan. Novi dokument ima korijenski element `XWPRootElement`, a njegovi su potčvorovi korijenski elementi dokumenata. U prikazanom primjeru riječ je o četiri dokumenta iz dodatka C: C2, C3, C4 i C5, a spojni dokument prikazan je u dodatku D1. Upit za ispitivanje kardinalnosti veze `author→book` prikazuje slika 6.35 [VBR03b], a sadržan je i u dodatku D2. Odgovor na upit je u dodatku D3.

```
<results> {

max(
let $d:=doc("tempdata1.xml")

let $retValue:= for $par in distinct-
                          values($d/XWPRootElement/bookSale/book),
                    $c in distinct-values($par/author)

         return <expression>
                   <parent> { $par/title} </parent>
                   <child>  { $c/lastName}  { $c/firstName}  </child>
                </expression>

for $c in distinct-values($retValue/child)
      let $p:=for $exp in $retValue where deep-equal($exp/child,$c)
      return $exp/parent

return count(distinct-values($p))
)
```

Slika 6.35. Upit za ispitivanje kardinalnosti veze author→book

Sam upit sastoji se od tri dijela. Prvi dio upita predstavlja izraz:

```
let $retValue:= for $par in distinct-
                          values($d/XWPRootElement/bookSale/book),
                    $c in distinct-values($par/author)

return <expression>
          <parent> { $par/title} </parent>
          <child>  { $c/lastName}  { $c/firstName}  </child>
       </expression>
```

Njime se grupiraju različiti (izraz `distinct-values`) parovi čvorova `book-author` koji sadrže samo ključni sadržaj: podelement `title` za `book`, koji se sprema u podelement `parent` od `expression`, i podelemente `firstName` i `lastName` za `author`, koji se spremaju u podelement `child` od `expression`. Svi parovi spremaju se u listu: varijablu `$retValue`. U spojnom dokumentu iz dodatka D1 postoji ukupno 5 takvih čvorova.

U sljedećem dijelu upita:

```
for $c in distinct-values($retValue/child)
      let $p:=for $exp in $retValue where deep-equal($exp/child,$c)
      return $exp/parent
```

za svaku različitu vrijednost ključa autora (zapisanu u `expression/child`; ukupno su 3 autora), navodi se lista knjiga (`$p`) koju je napisao (Iljf i Petrov po 1, Dostojevski 3, pri čemu se dvaput navodi isti naslov "Braća Karamazovi"). Funkcija `deep-equal` ispituje jednakost dvaju elemenata s obzirom za sadržaj i s obzirom na strukturu, ne uzimajući u obzir poredak podstruktura [Xquery04].

U trećem dijelu upita računa se broj različitih članova u listi knjiga za svakog autora: `count(distinct-values($p))`. Za Iljfa i Petrova ta vrijednost je 1, za Dostojevskog 2 (jer se eliminira dvostruki naslov "Braća Karamazovi". Nakon što je za svakog autora dobiven broj različitih naslova koji je napisao, može se zaključiti da veza autor→knjiga ima kardinalnost prema-više ako je bar jedan autor napisao više od jedne knjige. Zato se kao rezultat cjelokupnog upita ispisuje samo broj knjiga onog autora koji ih je napisao najviše (koristi se operator `max`; najviše knjiga, dvije, napisao je Dostojevski). Ako je taj broj veći od 1 veza ima kardinalnost prema-više. Struktura opisanog uvjeta je posve ista bez obzira za koji se element ispituje veza s njemu nadređenim elementom. Razlikuju se samo vrijednosti elemenata i atributa unutar `expression/parent` i `expression/child` koje preko grafičkog sučelja postavlja projektant skladišta.

6.8.3. Izgradnja grafa ovisnosti "prema dolje"

Algoritam se provodi rekurzivno. Ako je u (n-1)-om koraku algoritma čvor V iz grafa sheme dodan u graf ovisnosti kao V', sljedeći, n-ti korak provodi se na svaki potčvor W čvora V u grafu sheme. Rekurzivni algoritam prikazuje slika 6.36.

Ako je promatrani potčvor primarni ili strani ključ obrada se svodi na poziv procedure *obradiPrimarni* koja će biti definirana u poglavlju 6.8.5.

Naiđe li se na potčvor koji nije operator kardinalnosti, radi se o vezi prema-jedan tj. funkcionalnoj ovisnosti i taj se čvor dodaje u graf ovisnosti.

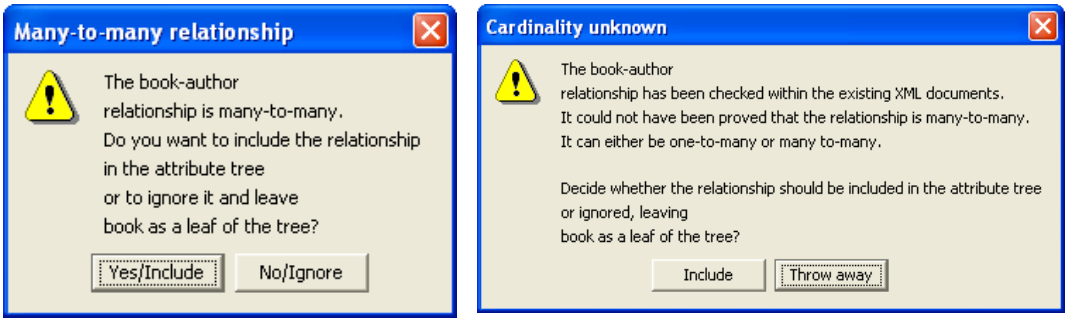
Ako je čvor operator ? njegov se potčvor dodaje u graf ovisnosti s obzirom da se ponovno radi o funkcijskoj ovisnosti, ali je postojanje tog potčvora u grafu ovisnosti uvjetno.

```
dodajDolje(V, V'){
   za svaki potčvor W od V{
     ako (W je roditelj primarnog ili stranog ključa) onda {
        obradiPrimarni(W, V');
     }
     inače{
       ako (W nije operator kardinalnosti) onda{
          STVORI ČVOR W' I DODAJ GA U GRAF OVISNOSTI KAO POTČVOR V'
           BEZUVJETNO;
          dodajDolje(W, W');
       }
       ako ne prethodno i ako(W=?) onda{
          uzmi jedini potčvor X od W;
          stvori čvor X' i dodaj ga u graf ovisnosti kao potčvor V' uvjetno;
          dodajDolje(X, X');
       }
       inače{ // tj. W=+ ili W=*
          uzmi jedini potčvor X od W;
          ispitaj XQueryjem je li veza V-X više-prema-više;
          ako (veza jest više-prema-više) onda{
             pitaj projektanta je li veza interesantna za agregaciju;
             ako (veza jest interesantna za agregaciju) onda{
                stvori čvor X';
                dodaj X' u graf ovisnosti kao potčvor V' bezuvjetno (+) ili uvjetno (*);
                dodajDolje(X, X');
          }
          inače{
             pitaj projektanta je li veza ipak više-prema-više i interesantna za
             agregaciju;
             ako (veza jest interesantna za agregaciju) onda{
                stvori čvor X';
                dodaj X' u graf ovisnosti kao potčvor V' bezuvjetno (+) ili uvjetno (*);
                dodajDolje(X, X');
             }
          }
       }
     }
   }
```

Slika 6.36. Rekurzivni algoritam za izgradnju grafa ovisnosti "prema dolje"

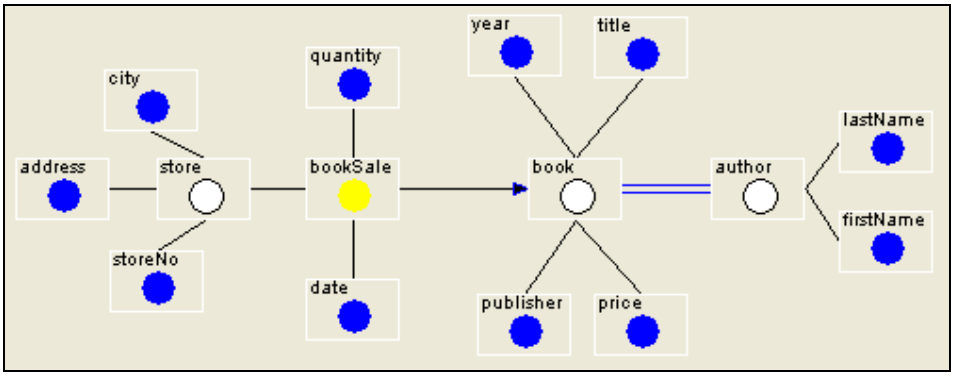
Ako je čvor operator + ili * potrebno je za osnovni čvor i jedini potčvor operatora ispitati je li veza više-prema-više, na način detaljno opisan u poglavlju 6.8.2. Projektant skladišta može apriorno odbaciti ispitivanje sadržaja ocijeni li da je potčvor nije zanimljiv za agregaciju (u grafičkom sučelju na slici 6.34 pritisne se `Cancel`). Utvrdi li se da veza jest više-prema-više, program pita projektanta skladišta treba li potčvor zbilja uključiti u graf ovisnosti (slika 6.37, lijevo). Ukoliko na temelju sadržaja raspoloživih XML dokumenata nije moguće dokazati da se radi o vezi više-prema-više, projektant skladišta o tome mora biti obaviješten (slika 6.37, desno). U

tom slučaju projektant na temelju poznavanje semantike dokumenta zaključuje radi li se o vezi jedan-prema-više ili više-prema-više te je li čvor koji bi se unesao u graf zanimljiv za agregaciju.



Slika 6.37. Program obavještava projektanta skladišta o ishodu ispitivanja sadržaja XML dokumenata.

Na slici 6.38 može se vidjeti konačni graf ovisnosti za primjer prodaje knjiga. Veza više-prema-više prikazana je dvostrukom linijom plave boje. Činjenični čvor je predstavljen krugom žute boje. Čvorovi koji imaju vlastiti (nemješoviti) sadržaj, atributi i elementi jednostavnog tipa, su krugovi plave boje, a elementi složenog tipa bez vlastitog tekstualnog sadržaja bijele boje.



Slika 6.38. Graf ovisnosti za prodaju knjiga izgrađen poluautomatski u sučelju aplikacije/*appleta*

6.8.4. Izgradnja grafa ovisnosti "prema gore"

Izgradnja grafa ovisnosti "prema gore" zapravo je dodavanje čvora koji predstavlja nadređeni element postojećem čvoru. U ovom smjeru XML Schema ne definira kardinalnost veze pa se ona mora pokušati saznati ispitivanjem sadržaja XML dokumenta. Nadređeni element će se u grafu ovisnosti nadovezati na postojeći čvor samo ako je o njemu funkcijski ovisan, odnosno ako je veza od postojećeg čvora prema nadređenom čvoru kardinalnosti prema-jedan. Postupak ispitivanja kardinalnosti veze prema nadređenom elementu detaljno je objašnjen u poglavlju 6.8.2.

Rekurzivni algoritam za izgradnju grafa ovisnosti "prema gore" prikazan je na slici 6.39. Oznaka U\V označava "U komplement V" tj. procedura dodajGore primjenjuje se na čvor U i njegovu cjelokupnu podstrukturu osim čvora V. Kad se to ne bi učinilo, čvor V', koji se već nalazi u grafu ovisnosti bio bi ponovno uključen u graf ako V funkcijski ovisi o U.

```
dodajGore(V, V'){
    pronađi nadređeni čvor U čvora V;
    ako (U operator kardinalnosti) onda{
          uzmi nadređeni čvor od U i postavi taj čvor kao vrijednost varijable U;
    }
    ispitaj XQueryjem je li veza V-U ima kardinalnost prema-više;
    ako (veza nema kardinalnost prema-više) onda{
          pitaj projektanta je li veza ipak prema-više;
          ako (projektant odluči da veza nema kardinalnost prema-više) onda{
              stvori čvor U';
              dodaj U' u graf ovisnosti kao potčvor bezuvjetno;
              dodajDolje(U\V, U')
              dodajGore(U, U');
          }
      }
}
```

Slika 6.39. Rekurzivni algoritam za izgradnju grafa ovisnosti "prema gore"

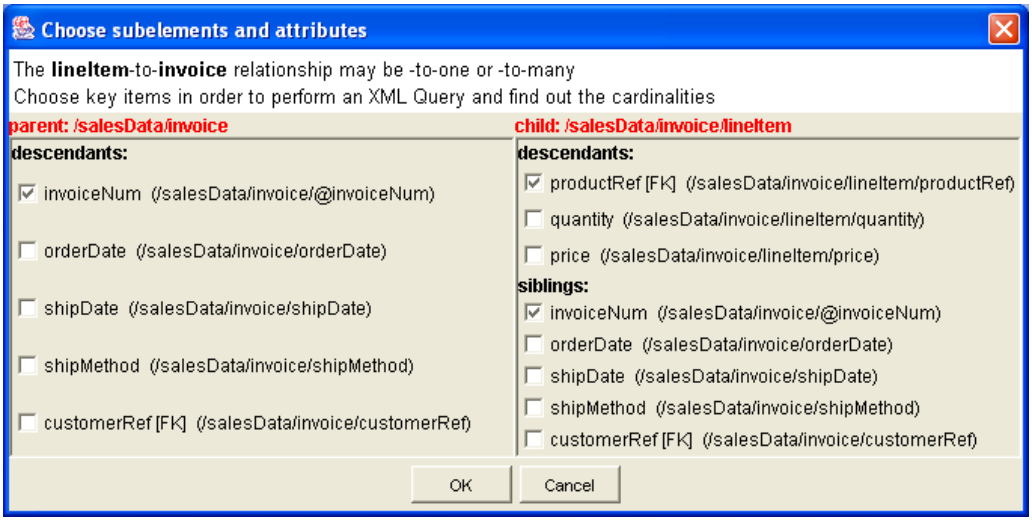
Prilikom ispitivanja kardinalnosti veze nekog elementa s nadređenim elementom (npr. elementa `lineItem` s elementom `invoice` u primjeru podataka o trgovini) potrebno je odrediti koji podelementi i atributi ključno određuju svaki od elemenata. Često se događa da podređeni element predstavlja egzistencijalno slabi entitet s obzirom na entitet opisan nadređenim elementom. To znači da njegovo postojanje nema smisla bez nadređenog elementa. U danom primjeru postojanje retka na računu (`lineItem`) nema smisla bez samog računa. Redak računa jest određen rednim brojem retka, ali samo ako se ujedno zna o kojem se računu radi. Veza egzistencijalno slabog entiteta (`lineItem`) prema entitetu o kojem on ovisi (`invoice`) je kardinalnosti prema-jedan (tj. funkcijska ovisnost).

Kod izbora ključnih podelemenata i atributa za podređeni (niži) element korisniku su ponuđeni svi njegovi atributi i podelementi, ali i svi podelementi i atributi nadređenog elementa, u slučaju da je podređeni element egzistencijalno slabi entitet i ovisi o nadređenom. Ako projektant skladišta podređeni element proglasi egzistencijalno slabim, nije potrebno provoditi ispitivanje upitom u jeziku XQuery.

U primjeru veze od `lineItem` prema `invoice` projektant skladišta pomoću grafičkog sučelja programa za poluautomatsko oblikovanje skladišta podataka mora odabrati ključni sadržaj elemenata `lineItem` i `invoice` (slika 6.40). Ključni sadržaj od `lineItem` ponuđen je s desne strane prozora na slici 6.40: projektant može odabrati među atributima i podelementima svih razina (*descendants* tj. potomci: `productRef`, `quantity`, `price`), ali i svim atributima i podelementima od `invoice`, (*siblings* tj. subraća, budući da su iste razine kao i `lineItem`).

Za ključ od `lineItem` odabiru se oznaka proizvoda `productRef` (u prikazanom modelu jedan proizvod ne može biti u dva različita retka) i `invoiceNum`, broj računa koji je atribut od `invoice`. Na taj način `lineItem` postaje egzistencijalno slabi entitet, veza `invoice` → `lineItem` postaje funkcijska ovisnost, pa `invoice` ulazi u graf ovisnosti.

Projektant skladišta može apriorno odbaciti ispitivanje sadržaja i odlučiti da je kardinalnost veze prema-više (u grafičkom sučelju na slici 6.40 pritisne se `Cancel`). U tom slučaju izgradnja grafa ovisnosti "prema gore" se prekida.



Slika 6.40. Označavanje egzistencijalno slabog entiteta u izgradnji grafa "prema-gore"

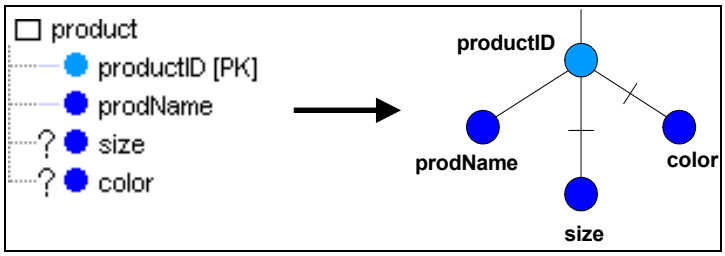
Kad se ispitivala kardinalnost veze u smjeru od podređenog elementa k nadređenom prilikom izgradnje grafa ovisnosti "prema dolje", u grafičkom sučelju nije bilo moguće podređeni čvor eksplicitno proglasiti egzistencijalno slabim entitetom koji ovisi o entitetu koji predstavlja nadređeni čvor. Ovo ograničenje uvedeno je zbog jednostavnosti: ako je riječ o egzistencijalno slabom entitetu, veza neće biti više-prema-više pa je dovoljno pritisnuti `Cancel` i prekinuti rekurzivni proces. Prilikom izgradnje grafa ovisnosti prema gore `Cancel` sugerira automatski prekid rekurzije što je analogno vezi prema-više, dok egzistencijalno slabi entitet izražava upravo suprotno, vezu prema-jedan.

6.8.5. Izgradnja grafa ovisnosti mehanizmom primarnog i stranog ključa

XML Schema omogućuje da element jednostavnog tipa ili atribut postane primarni ključ u cijelom dokumentu ili dijelu dokumenta. U oba će slučaja ključ biti potčvor čvora koji predstavlja element složenog tipa u grafu sheme. Budući da element složenog tipa nema vlastiti sadržaj, može se primijeniti načelo usporedbe s relacijskim modelom podataka pri čemu element složenog tipa odgovara relaciji, a njegovi atributi i podelementi (bez obzira na razinu) atributima relacije. O primarnom ključu funkcijski ovise svi atributi relacije. Prilikom oblikovanja stabla atributa čvor složenog elementa, koji nema tekstualnu vrijednost, može se izbaciti, a njegovo mjesto dolazi čvor primarnog ključa, o kojem funkcijski ovise ostali čvorovi.

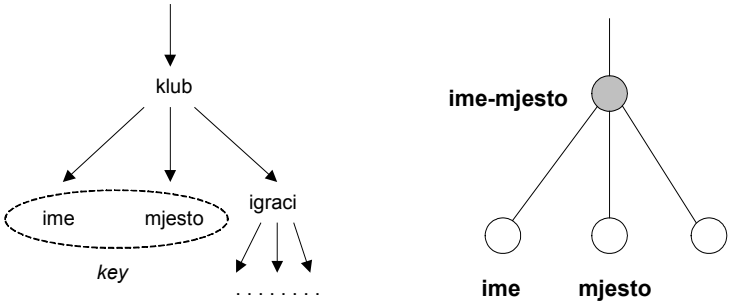
Nakon što je završena izgradnja grafa ovisnosti u oba smjera, prema dolje i prema gore, preostaje analizirati i preurediti ključeve.

Ukoliko se primarni ključ sastoji od samo jednog čvora taj čvor se premješta na ranije mjesto čvora složenog elementa. U primjeru podataka za trgovinu (graf sheme na slici 6.29) čvor `productID` postaje nadređen čvorovima `prodName`, `size` i `color` koji o njemu funkcijski ovise. Uvjetno pridruživanje u grafu ovisnosti prikazuje prekrižena linija preko grane koja povezuje čvorove.



Slika 6.41. Transformacija podstabla jednostavnog primarnog ključa.

U slučaju složenog primarnog ključa uvodi se novi čvor koji predstavlja združenu vrijednost primarnog ključa i zamjenjuje čvor složenog elementa [VBR03a, VBR03b]. Pojedinačne komponente ključa ostaju u grafu ovisnosti kao zasebni čvorovi. Dugotrajno iskustvo u projektiranju skladišta podataka pokazuje kako bi se u suprotnom njihova vrijednost izgubila. Naime, u izvedbi skladišta u relacijskoj bazi podataka za vrijednost primarnog ključa može se koristiti kombinirana vrijednost komponenata ili uvesti umjetni numerički ključ. Autori većinom preporučuju potonji pristup [Kim96]. Za razliku od relacijske baze podataka, gdje je oznaka ključa metapodatak koji dolazi uz tablicu, u grafu ovisnosti ključ se mora izraziti zasebnim čvorom. U primjeru nogometnog kluba iz poglavlja 6.3.3.3, ključ se sastojao od čvora `ime` i čvora `mjesto`. Transformacijom će čvor `klub` iz grafa sheme biti nadomješten novim čvorom `ime-mjesto` u grafu ovisnosti. Svi potčvorovi čvora `klub`, uključujući `ime` i `mjesto` postaju potčvorovi od `ime-mjesto` (slika 6.42).



Slika 6.42. Transformacija podstabla složenog primarnog ključa.

Sada je moguće definirati funkciju obradiPrimarni za obradu primarnog ključa koja se izvodi u potpunosti automatski (bez sudjelovanja projektanta skladišta). Funkcija je prikazana na slici 6.43.

Prvi ulazni parametar, V, odgovara čvoru grafa sheme koji je roditelj primarnog ključa. Drugi parametar, R', je čvor grafa ovisnosti na koji će se dodati struktura primarnog ključa.

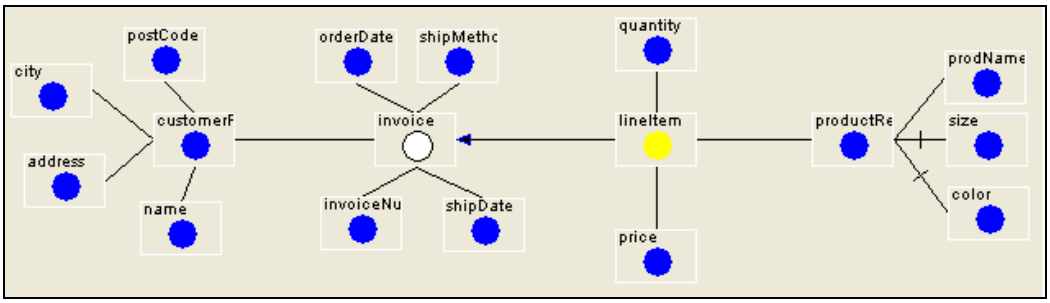
Potrebno je napomenuti da program ne obrađuje složene ključeve čije komponente nemaju jedan zajednički roditeljski čvor u grafu sheme. Takve strukture su u praksi vrlo rijetke.

Kad se prilikom izgradnje grafa ovisnosti naiđe na strani ključ, taj se čvor ugrađuje u stablo, a u posebnu se listu bilježi obaveza kasnije transformacije stranog ključa. Transformacija stranog ključa vrši se nakon što se graf ovisnosti u potpunosti izgrađen.

```
obradiPrimarni(V, R'){
      ako (V je roditelj jednostavnog ključa P) onda {
          stvori čvor P' i dodaj ga u graf ovisnosti kao potčvor R' bezuvjetno;
          za svaki potčvor W od V uz uvjet W≠P{
              stvori čvor W' i dodaj ga u graf ovisnosti kao potčvor R' bezuvjetno;
              dodajDolje(W, W')
          }
      }
      inače {
          stvori čvor P' i dodaj ga u graf ovisnosti kao potčvor R' bezuvjetno;
          za svaki potčvor W od V{
              stvori čvor W' i dodaj ga u graf ovisnosti kao potčvor R' bezuvjetno;
              dodajDolje(W, W')
          }
      }
}
```

Slika 6.43. Algoritam za dodavanje strukture primarnog ključa u graf ovisnosti

Transformacija strukture stranog ključa je operacija slična prirodnom spajanju u relacijskoj bazi podataka [VBR03a, VBR03b, Vrd04]. Ideja za transformaciju potječe iz konceptualnog modeliranja skladišta podataka na temelju dijagrama entitet-veza odnosno [GMR98]. Na čvor koji predstavlja strani ključ dodaje se cjelokupno podstablo primarnog ključa na kojeg se odnosi strani ključ. Pritom čvoru koji predstavlja ključ ostaje ime stranog ključa. U primjeru podataka za trgovinu, *sales data*, prilikom izgradnje stabla ovisnosti nailazi se dvaput na strane ključeve, `customerRef` i `productRef`. Na čvor `productRef` koji se već nalazi u grafu ovisnosti dodaju se svi potčvorovi od `productID`, prikazani na slici 6.41. Oni funkcijski ovise o `productRef`. Konačni graf ovisnosti izgrađen poluautomatskim procesom, a prije procesa preuređivanja grafa, prikazuje slika 6.44. Na desnoj strani slike primjećuje se da su na čvor `productRef` dodani čvorovi `prodName`, `size` i `color`.



Slika 6.44. Graf ovisnosti za podatke o trgovini (*sales data*) izgrađen poluautomatskim postupkom

6.8.6. Konvergencija i dijeljena hijerarhija

Algoritam za poluautomatsko oblikovanje grafa ovisnosti od činjeničnog čvora "prema dolje" odnosno "prema gore" iznesen u poglavljima 6.8.3 i 6.8.4 izgrađuje graf ovisnosti kao pravo stablo. U takvom grafu ne postoje dva puta od jednog čvora do drugog, a u svaki čvor smije ući samo jedna grana.

U XML Schemama često se puta isti složeni tip koristi za nekoliko različitih elemenata u različitom dijelu dokumenta ili se više puta koristi referenca na isti globalni element složenog tipa. Primjerice, u XML Schemi za narudžbu robe (*purchase order*) dva čvora imaju složeni tip `USAddress`: `shipTo` i `billTo`. Zbog toga će oba elementa imati istoimene atribute i podelemente tj. istoimene potčvorove u grafu sheme (kako je i sugerirano u grafu sheme koji je prikazan kao nepravo stablo na slici 6.19).

Načela izvedbe skladišta podataka u sabirničkoj arhitekturi (poglavlje 1.2.6) kao i općenita iskustva logičke izvedbe skladišta podataka u relacijskim bazama sugeriraju objedinjavanje čvorova jednake strukture. U konačnoj realizaciji dvije dimenzije iste strukture (kao što su `shipTo` i `billTo`) mogu biti smještene u jednoj tablici. Opisani slučaj podudaranja čvorova grafa ovisnosti po tipu naziva se dijeljena hijerarhija (engl. *shared hierarchy*).

U posebnim slučajevima može utvrditi da se radi o identičnim podacima zapisanima na dva mjesta pa se oni zbog konzistentnosti mogu zapisati i na samo jednom mjestu. Ovakvo podudaranje čvorova i po tipu i po sadržaju naziva se konvergencijom (engl. *convergence*). Podudarati se mora svaki potčvor (podelement ili atribut) danog čvora.

Čvorovi se grafa ovisnosti koji odgovaraju istom tipu u slučaju konvergencije i dijeljene hijerarhije sažimaju u jedinstveni čvor. Takav graf ovisnosti je nepravo stablo budući da opisani čvor ima više od jedne ulazne grane.

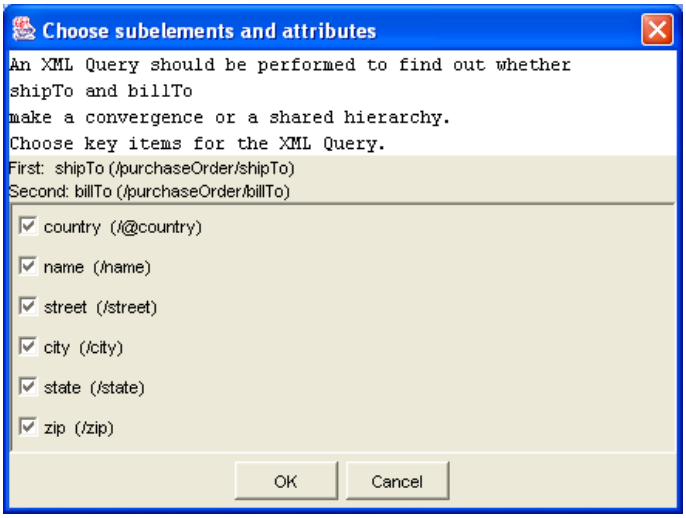
Graf ovisnosti izgrađen poluautomatskim procesom prema algoritmima detaljno opisanim u poglavljima 6.8.3, 6.8.4 i 6.8.5 ima oblik pravog stabla i u njemu može postojati više čvorova istog složenog tipa koje je potrebno svesti u jedinstveni čvor i time stvoriti konačni graf ovisnosti nepravo stablo. Program automatski otkriva sve složene tipove kojima odgovara više čvorova u grafu ovisnosti te svako korištenje globalnog elementa složenog tipa. Svi čvorovi istog složenog tipa spajaju se u jedinstveni zajednički čvor, a ispitivanjem sadržaja utvrđuje se radi li se o konvergenciji ili dijeljenoj hijerarhiji. Redom se uspoređuju sadržaji svakog para istoimenih potčvorova. Ako se i za jedan par potčvorova naiđe na nepodudarnost sadržaja nije riječ o konvergenciji.

Postoji li u grafu ovisnosti više čvorova istog složenog tipa, razvijeni program će o tome obavijestiti projektanta skladišta pomoću grafičkog sučelja i zatražiti odabir potčvorova na osnovi čijeg sadržaja će se utvrditi konvergencija.

Iako je za ispitivanje postojanja konvergencije između dvaju čvorova istog složenog tipa potrebno ispitati cjelokupni sadržaj, tj. sve potčvorove, program omogućuje da se odaberu samo neki potčvorovi te ispita samo njihov sadržaj. Ovako dokazana konvergencija odnosi se samo na dio strukture i nije prava, no može postati prava ako se tijekom kasnijeg preuređenja grafa uklone nekonvergentni čvorovi. Ispitivanje se vrši upitom u jeziku XQuery.

Za primjer narudžbe robe, *purchase order* (uz XML Schemu u dodatku A1 dana su tri pripadajuća XML dokumenta u dodacima A2, A3 i A4), ispituju se sadržaji svih potčvorova: `name`, `street`, `city`, `state`, `zip` i `country`. Zbog nepodudaranja imena, `name`, već je iz dokumenta u dodatku A2 vidljivo da se ne može uspostaviti konvergencija nego samo dijeljena hijerarhija.

Grafičko sučelje programa za odabir čvorova prikazuje slika 6.45. Odabir čvora vrši se postavljanjem kvačice uz ime čvora. Vidljivo je da se uz ime svakog čvora pojavljuje kvačica što znači da su svi čvorovi odabrani za ispitivanje.



Slika 6.45. Grafičko sučelje za ispitivanje sadržaja čvorova kandidata za konvergenciju

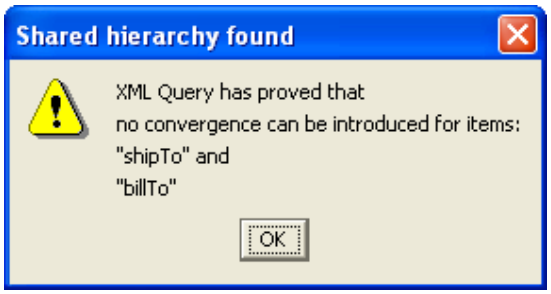
Nakon odabira šest označenih čvorova izvodi se upit u jeziku XQuery prikazan na slici 6.46.

```
<results> {
let $d:=doc("tempdata2.xml")
let $retValue:= for $parent in distinct-
                          values($d/XWPRootElement/purchaseOrder)
    let $child1:= $parent/shipTo
    let $child2:= $parent/billTo
    return  <expression>
              <first>
                <content>
                   {$child1/name}  {$child1/street}  {$child1/city}
                   {$child1/state}  {$child1/zip} {$child1/@country}
                </content>
              </first>
              <second>
                <content>
                   {$child2/name}  {$child2/street}  {$child2/city}
                   {$child2/state}  {$child2/zip} {$child2/@country}
                </content>
              </second>
            </expression>

let $x:= for $c in  $retValue where not(deep-
                         equal($c/first/content,$c/second/content))
return $c return count($x)

} </results>
```

Slika 6.46. Upit u XQueryju za utvrđivanje postojanja konvergencije u primjeru narudžbe robe

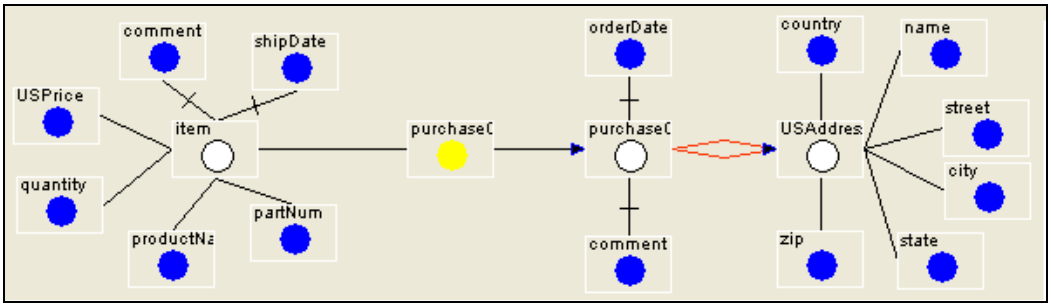
U element `first/content` zapisuju se označeni potčvorovi prvog čvora, a u element `second/content` istoimeni potčvorovi drugog čvora. Ako je broj

nepodudarnih parova `first-second` veći od nule čvorovi nisu konvergentni već samo čine dijeljenu hijerarhiju (slika 6.47). Ne pronađu li se nepodudarni parovi, to još uvijek ne znači da nije moguće napraviti dokumente gdje neće biti konvergencije. Stoga projektant skladišta mora odgovoriti na pitanje je li riječ o konvergenciji i u semantičkom i logičkom smislu.



Slika 6.47. Neuspješan rezultat ispitivanja konvergencije

Konačni je graf sheme za narudžbu robe dan na slici 6.48. Za razliku od ostalih čvorova koje u grafičkom sučelju obilježavaju njihova imena, čvor `USAddress` nosi oznaku tipa koji objedinjuje `shipTo` i `billTo` (`shipTo` i `billTo` nazivaju se instancama tipa `USAddress`). U čvor `USAddress` ulaze dvije grane, obje iz čvora `purchaseOrder`. Radi isticanja označene su crvenom bojom.



Slika 6.48. Graf ovisnosti za primjer narudžbe robe (*purchase order*) dobiven poluautomatskim postupkom

6.9. Preuređivanje grafa ovisnosti

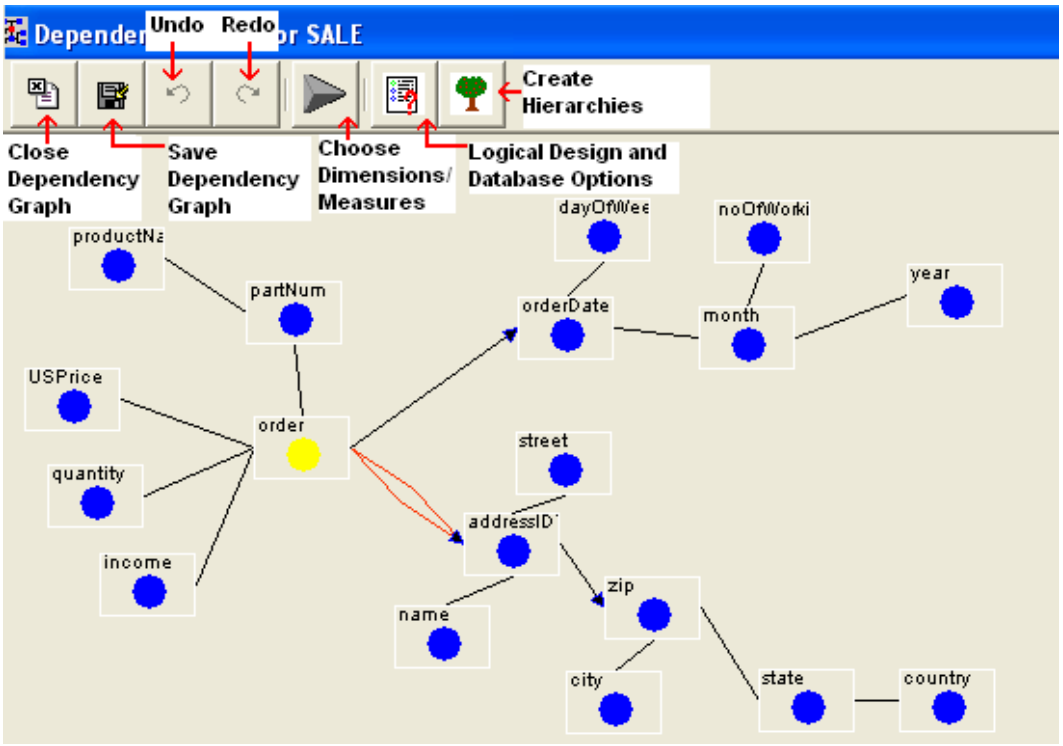
U koraku 4.1 algoritma za konceptualno oblikovanje područnog skladišta podataka iz XML izvora poluautomatskim postupkom automatiziranim do najveće moguće mjere stvara se graf ovisnosti. Svaki čvor u grafu ovisnosti jednoznačno odgovara nekom čvoru u grafu sheme. Takav graf ovisnosti po strukturi još uvijek nema pravi izgled konceptualne sheme područnog skladišta podataka. Neke čvorove potrebno je ukloniti, a druge dodati, dok je trećima nužno promijeniti poziciju ili ulogu u grafu ovisnosti.

6.9.1. Graf ovisnosti u grafičkom sučelju

Graf ovisnosti u obje se verzije programskog sustava za oblikovanje skladišta podataka iz XML izvora (aplikaciji i *appletu*) nalazi u zasebnom prozoru. Program pamti svaki korak u preuređivanju grafa ovisnosti pa ga može poništiti (***Undo***), a nakon poništenja i ponovno izvršiti (***Redo***). Preostale funkcije u sučelju su zatvaranje prozora s grafom ovisnosti (***Close Dependency Graph***), određivanje dimenzija i mjera (***Choose Dimensions/Measures***) te stvaranje hijerarhija (***Create Hierarchies***). Sustav

omogućuje opciju *undo-redo* i u koraku određivanja mjera i dimenzija. Stvaranje hijerarhija i logičko oblikovanje provodi se u jednom, zajedničkom koraku nakon čega se mogu stvoriti tablice u relacijskoj bazi podataka. Projektant skladišta određuje hijerarhije, nakon čega se logičko oblikovanje izvodi automatski.

Aplikacija u svakom trenutku omogućuje pohranu grafa ovisnosti (***Save Dependency Graph***) u datoteku. Zajedno s grafom ovisnosti pohranjuje se čitav radni projekt kako bi korisnik dobio uvid u XML Schemu i njen graf te pripadajuće XML dokumente. Graf ovisnosti moguće je učitati odabirom opcije *Open Graph* u glavnom sučelju aplikacije (poglavlje 4.6). Prozor za prikaz grafa ovisnosti je na slici 6.49.



Slika 6.49. Prozor za prikaz grafa ovisnosti

6.9.2. Čvorovi jednostavnog i složenog tipa u grafu ovisnosti

Nakon što je graf sheme izgrađen poluautomatskim postupkom, pojmovi jednostavnog i složenog tipa više nemaju jednako značenje kao u XML Schemi. Čvor bez vlastitih potčvorova i dalje označava čvor jednostavnog tipa, a složeni tip vezan je uz čvorove koji imaju vlastite potčvorove. Gubi se značenje uz elemente XML Scheme: čvor složenog tipa više ne mora biti prazan tj. bez sadržaja.

Budući da je čvor predstavljen svojom grafičkom komponentom, u aplikaciji se jednostavni čvor naziva jednostavnom komponentom (*simple component*) i predstavlja list u grafu, a složeni čvor je složena komponenta (*complex component*) te ima potčvorove.

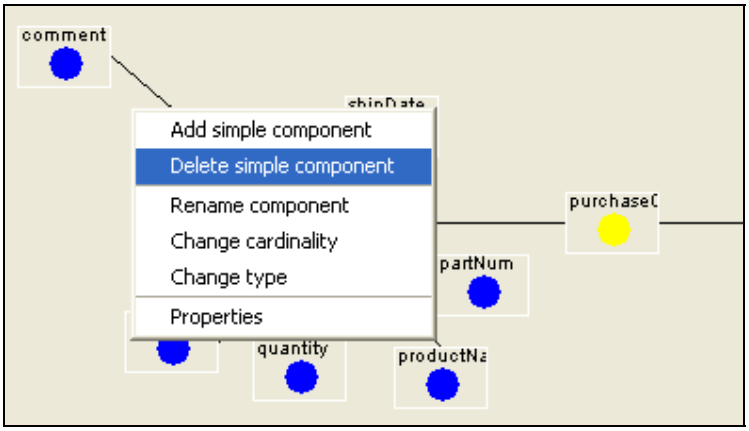
U konačnom grafu ovisnosti postojat će samo čvorovi koji imaju sadržaj.

Pritiskom miša na čvor u grafičkom sučelju moguće je odabrati jednu od brojnih opcija preuređivanja. Razlikuju se opcije preuređivanja za čvorove složenog i jednostavnog tipa te čvorove koji predstavljaju dijeljenu hijerarhiju ili konvergenciju. Svaki čvor (komponenta) ima opciju ***Properties*** kojom se ispisuju osnovni podaci o

čvoru (komponenti): ime, tip, roditeljski čvor i kardinalnost veze od strane roditeljskog čvora. U slučaju dijeljene hijerarhije i konvergencije ispisuju se svi roditeljski čvorovi i kardinalnosti veza kao i imena instanci čvorova.

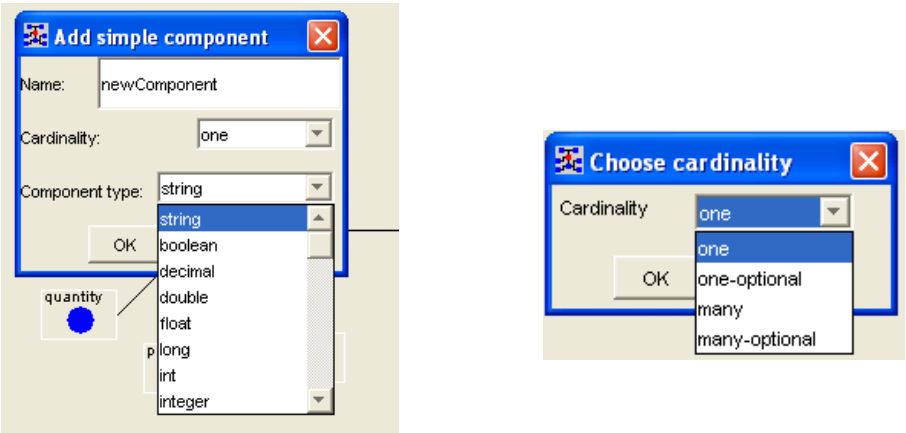
6.9.3. Preuređivanje čvora jednostavnog tipa

Program nudi sljedeće operacije preuređivanja jednostavnog čvora (slika 6.50):



Slika 6.50. preuređivanje jednostavnog čvora u grafu ovisnosti.

- ***Add simple component***. Čvoru se može dodati novi jednostavni potčvor. U tom slučaju promatrani čvor postaje složen. Za razliku od čvorova dobivenih poluautomatskim postupkom, novostvoreni potčvor ne može automatski biti pridružen nekom čvoru grafa sheme (tj. XML dokumenta). Pridruživanje će se obaviti tijekom oblikovanja procesa izvlačenja, transformacije i učitavanja podataka iz izvora u skladišta. Novom jednostavnom čvoru mora se zadati ime, tip i kardinalnost pridruživanja roditeljskom čvoru (slika 6.51, lijevo).
- ***Delete simple component***. Obavlja brisanje jednostavnog čvora.
- ***Rename component***. Promjena imena čvora.
- ***Change cardinality***. Promjena kardinalnosti kojom je dani čvor pridružen svom roditeljskom čvoru (slika 6.51, desno).
- ***Change type***. Promjena tipa čvora. Moguće je odabrati bilo koji od 44 osnovna tipa koje definira XML Schema.



Slika 6.51. Programska izvedba dodavanja novog čvora (lijevo) i kardinalnosti veze (desno)

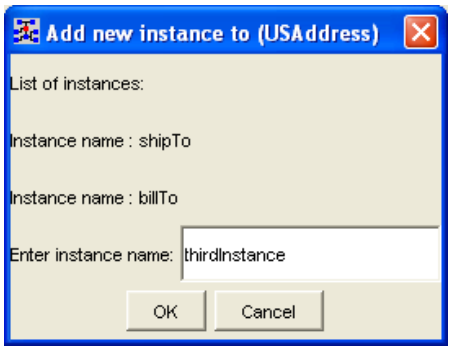
6.9.4. Preuređivanje čvora složenog tipa

Za preuređivanje složenog čvora nude se operacije analogne već opisanima za jednostavni čvor: *Add simple component*, *Delete complex component*, *Rename component* i *Change cardinality*. *Change type* je ponuđen samo ako složeni čvor ima sadržaj.

U primjeru grafa ovisnosti za narudžbu (*purchase order*) čvor `USAddress` predstavlja dijeljenu hijerarhiju s dvije instance: `shipTo` i `billTo`. Kad čvor predstavlja konvergenciju ili dijeljenu hijerarhiju, postoje dodatne funkcije za preuređivanje grafa ovisnosti koje se odnose na samo jednu instancu čvora:

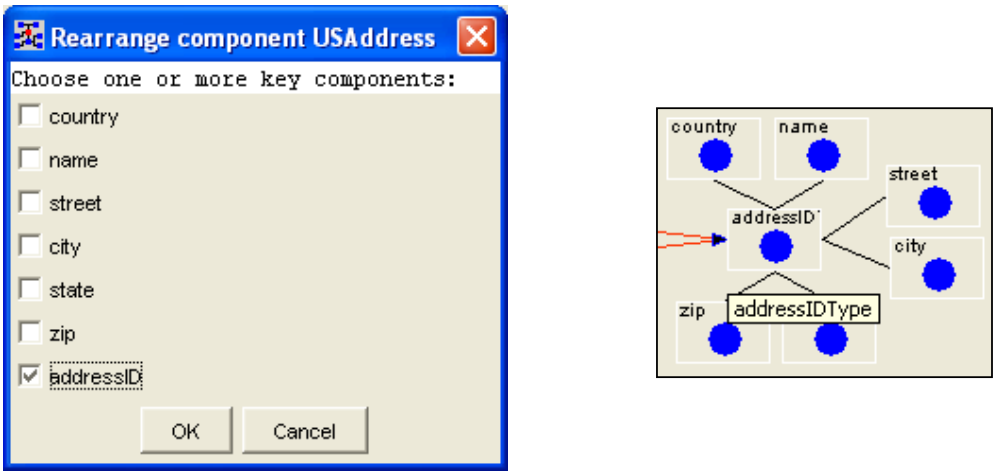
- ***Add new instance***. Dodavanje nove instance složenom čvoru (slika 6.52).
- ***Remove instance***. Uklanjanje jedne instance složenog čvora. Ova opcija dozvoljena je samo za čvorove s dvije ili više instanci.
- ***Rename instance***. Mijenjanje imena instance.

Ukoliko složeni čvor ima više instanci, opcija *Change cardinality* razlikuje se o odnosu na istu opciju kod jednostavnog čvora. U primjeru narudžbe, posebno se mijenja kardinalnost veze `purchaseOrder-shipTo`, a posebno `purchaseOrder-billTo`.



Slika 6.52. Dodavanje nove instance složenom čvoru.

Opcija ***Rearrange component*** preuređuje strukturu složenog čvora po načelu strukture primarnog ključa. Među potčvorovima promatranog čvora koji imaju sadržaj (odnosno, plave su boje) bira se ključ njegove složene strukture. U primjeru narudžbe potrebno je ukloniti čvor `USAddress` koji nema sadržaj i izvršiti transformaciju njegove strukture. Čvor opisuje adresu, ali u njoj je sadržano i ime osobe, pa se može reći da opisuje i osobu (u jednom slučaju kupca, u drugom slučaju osobu kojoj se dostavlja roba; vrlo često je to ista osoba). Nijedan njegov potčvor (`name`, `address`, `city`, `state`, `zip`, `country`) ne može sam za sebe biti ključ strukture nego je to zajednička kombinacija svih čvorova. U tom slučaju najbolje je upotrijebiti umjetni ključ pa se prije preuređivanja strukture čvoru `USAddress` doda novi potčvor `addressID` tipa `positiveInteger` (tj. prirodni broj). Potom se pokrene opcija *Rearrange component*. U grafičkom sučelju odabire se čvor koji postaje ključ strukture, `addressID` (slika 6.53, lijevo). Preuređena struktura nekadašnjeg čvora `USAddress` može se vidjeti na slici 6.53, desno. Čvor `AddressIdType` preuzeo je instance `shipTo` i `billTo` čvora `USAddress`. Za razliku od `USAddress`, koji je bez sadržaja, ovaj čvor ima sadržaj tipa `positiveInteger`.



Slika 6.53. Transformacija strukture složenog čvora. Lijevo: odabir ključa strukture. Desno: izgled preuređene strukture

6.9.5. Preuređivanje činjeničnog čvora i preusmjeravanje veza

Jedine operacije preuređivanja koje se mogu vršiti nad činjeničnim čvorom su njegovo preimenovanje (*Rename component*) ili dodavanje novog jednostavnog potčvora (*Add simple component*). Činjenični čvor potekao je od veze čvorova `purchaseOrder` i `item` u grafu sheme i nosi ime `purchaseOrder-with-item`. Njemu se dodjeljuje novo ime, tipično za činjenicu: `order` (narudžba).

Svakom čvoru u grafu ovisnosti (osim korijenskom činjeničnom čvoru, koji ga nema), može se promijeniti roditeljski čvor što je analogno preusmjeravanju veze između čvorova. Preusmjeravanje veze aktivira se pritiskom miša na liniju koja u grafičkom sučelju predstavlja vezu. U primjeru narudžbe čvor `quantity`, koji je potčvor od `item`, postaje izravni potčvor činjenice budući da treba postati mjera područnog skladišta.

Kod dijeljene hijerarhije ili konvergencije svaka instanca ostvaruje zasebnu vezu sa svojim roditeljskom čvorom. Veza svake instance može se preusmjeriti neovisno o drugim instancama.

6.10. Određivanje mjera i dimenzija

Nakon što je projektant skladišta preuredio graf ovisnosti, moguće je odrediti mjere i dimenzije te definirati hijerarhijske razine čime je konceptualno oblikovanje u potpunosti dovršeno. Sadašnja verzija razvijene programske podrške za oblikovanje skladišta podataka iz polustrukturiranih izvora ne uključuje oblikovanje konkretnih agregacija, već samo hijerarhijskih razina koje nude mogućnost agregacije. Dodavanje funkcionalnosti za oblikovanje agregacija bit će tema daljnjeg proširenja ovog programskog sustava, razvijenog na Zavodu za telekomunikacije Fakulteta elektrotehnike i računarstva u Zagrebu.

Programski sustav ne dozvoljava da se započne s određivanjem mjera i činjenica prije nego su svi čvorovi bez sadržaja uklonjeni iz grafa ovisnosti. Dimenzijom ili mjerom može postati samo čvor čiji roditelj je činjenica. Čvor koji treba postati mjera uz to ne smije imati niti jedan potčvor. Program također zahtijeva da se svaki čvor kojem je roditelj činjenica proglasi ili mjerom ili dimenzijom. Nakon toga moguće je definirati hijerarhije (i agregacijske tvrdnje) u dimenzijama.

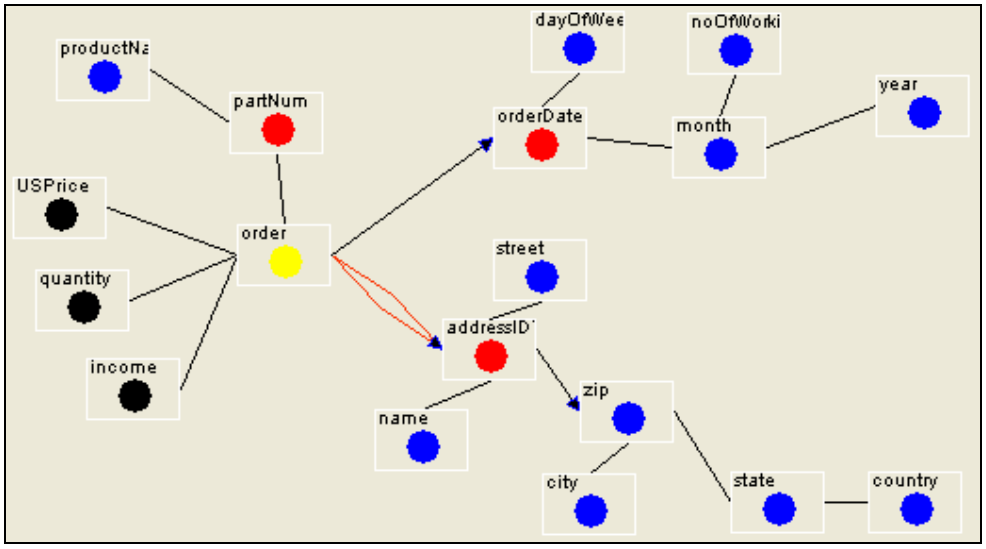
Mjere su u grafičkom sučelju prikazane crvenom bojom, a ključevi (korijeni) dimenzija crnom bojom.

Graf ovisnosti za narudžbu robe s definiranim mjerama dimenzijama prikazuje slika 6.54. U odnosu na graf ovisnosti prikazan na slici 6.48 bilo je potrebno izvršiti promjene. Osnovni problem predstavlja uvjetno postojanje čvora `orderDate` koji daje vremensku dimenziju. Skladište podataka nužno ima vremensku dimenziju. Stoga se veza `orderDate` i njemu nadređenog čvora treba proglasiti bezuvjetnom, a sve dokumente koji nemaju `orderDate` izbaciti iz procesa skladištenja. Dokument *purchase order* nalazi se u specifikaciji XML Scheme [Sch0-04] i samo približno odgovara pravim dokumentima za narudžbe. U praktičnom slučaju svaka XML Schema za izdani račun ili narudžbu obavezno bi sadržavala datum.

Iz grafa ovisnosti izbačeni su čvorovi `comment` i `shipDate`. Čvorovi `price` i `quantity`, kao mjere, postaju potčvorovi činjenice. Dodan je čvor `income` koji predstavlja njihov umnožak. Čvoru `orderDate` dodani su `dayOfWeek` (dan u tjednu), `month` (mjesec, izražen brojem), `noOfWorkingDays` (broj radnih dana u mjesecu) i `year` (godina, izražena brojem). Svi su funkcijski ovisni o `orderDate` i mogu se jednoznačno dobiti iz datuma pretvorbenim funkcijama koje su najčešće standardno ugrađene u sustave za upravljanje bazama podataka ili alate za izgradnju skladišta podataka. Potčvorovi od `addressIDType` su pregrupirani kako bi oblikovali hijerarhiju. Preuređeni graf ovisnosti prije označavanja mjera i dimenzija vidi se na slici 6.49, a s označenim mjerama i dimenzijama na slici 6.54.

Ako se za korijen dimenzije odabere dijeljena hijerarhija, zapravo se definira onoliko dimenzija koliko je instanci složenog tipa u čvoru. U primjeru narudžbe čvor `addressIDType` dat će dvije dimenzije, `shipTo` i `billTo`. Zbog toga u konačnom grafu ovisnosti na slici 6.54 postoje ukupno 4 dimenzije:

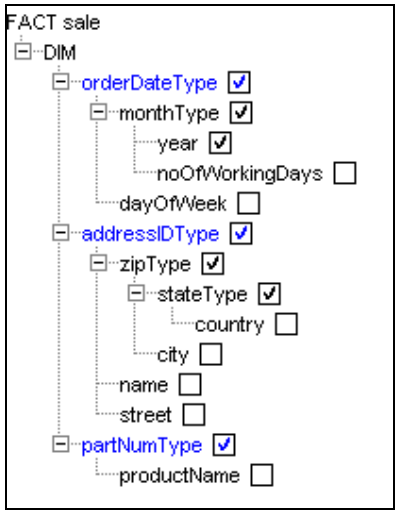
- vremenska dimenzija (korijen u `orderDate`),
- dimenzija proizvoda (korijen u `partNum`) ,
- dimenzija kupca (korijen u instanci `billTo` čvora `addressIDType`),
- dimenzija dostave (korijen u instanci `shipTo` čvora `addressIDType`).



Slika 6.54. Konačni graf ovisnosti za narudžbu robe (*purchase order*)

U primjeru narudžbe postojale su tri hijerarhijske razine u vremenskoj dimenziji (`orderDate-month-year`) te po tri hijerarhijske razine u dimenzijama `shipTo` i `billTo` (`addressID-zip-state`). Kad se narudžba ne bi odnosila samo na Sjedinjene Američke Države (`country` je stalno postavljen na vrijednost "US"), hijerarhija bi mogla dobiti i četvrtu razinu: `addressID-zip-state-country`, pri čemu `state` može označavati federalnu ili sličnu državnu podjedinicu. U dimenziji proizvoda postoji samo jedna hijerarhijska razina. Programski sustav općenito dozvoljava istodobno stvaranje više hijerarhija u jednoj dimenziji.

Hijerarhijske se razine definiraju u grafičkom sučelju (slika 6.55) u kojem su dimenzije iskazane stablom. Temeljni atributi svake hijerarhijske razine označavaju se kvačicom (kvačica se može postaviti i ukloniti). Korijeni dimenzije unaprijed su označeni kvačicom i označavaju najdetaljniju hijerarhijsku razinu (kvačica na toj razini se ne može ukloniti).

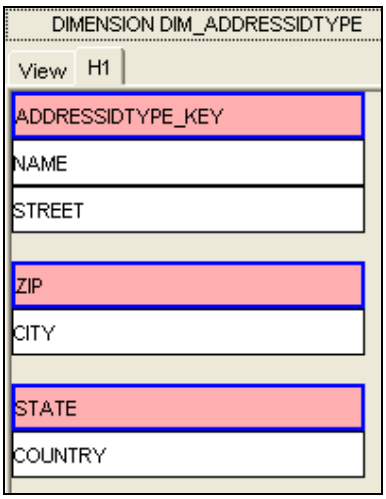


Slika 6.55. Definiranje hijerarhijskih razina

U primjeru narudžbe robe na slici 6.55 u vremenskoj dimenziji (s korijenom `orderDateType`) projektant skladišta uz unaprijed određenu hijerarhijsku razinu `orderDateType` definira još i hijerarhijske razine `monthType` i `year`. Analogno se može postupiti u preostalim dimenzijama. Svi čvorovi koji funkcijski ovise o čvoru-korijenu hijerarhijske razine nalaze se na toj razini, osim ako i sami nisu temelj druge hijerarhijske razine. Primjerice, čvorovi `noOfWorkingDays` i `year` funkcijski ovise o čvoru `month`. Čvor `noOfWorkingDays` (broj radnih dana u mjesecu) je u razini mjeseca (kojoj je temeljni čvor `month`) dok `year` čini osnovu hijerarhijske razine godine.

Svaki potčvor korijena dimenzije može biti određen kao identifikator hijerarhijske razine. Uz ime svakog čvora postoji prazan kvadratić u koji se mišem može unijeti kvačica, označivši na taj način promatrani čvor kao identifikator hijerarhijske razine. U slučaju da se projektant skladišta predomisli, kvačicu je moguće ukloniti. Korijeni dimenzija napisani su plavom bojom i po definiciji predstavljaju identifikator najdetaljnije hijerarhijske razine. Zbog tog se njihova kvačica, također označena plavom bojom, ne može ukloniti.

Svaka dimenzija može se pregledati u zasebnom prikazu u grafičkom sučelju. Na slici 6.56 je dan pregled dimenzije kupca odnosno dostave (s korijenom u `addressIDType`).



Slika 6.56. Pregledni prikaz hijerarhije u grafičkom sučelju

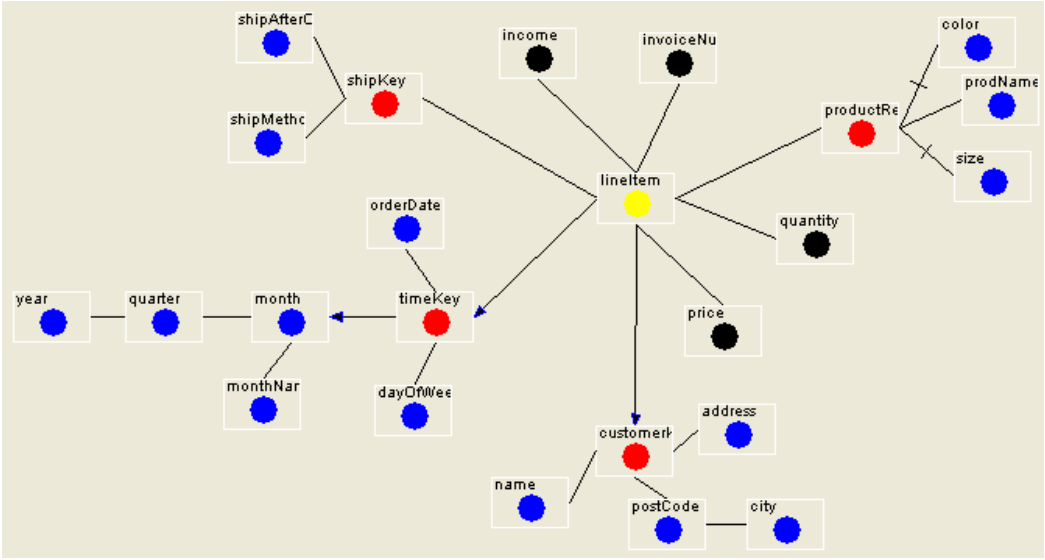
Hijerarhijske razine su razdvojene. Najdetaljnija razina nalazi se na vrhu. Ružičastom bojom označeni su ključni atributi hijerarhijskih razina, a bijelom bojom ostali atributi.

Odabir hijerarhija posljednji je korak oblikovanja koji obavlja projektant skladišta. Njime završava konceptualno oblikovanje (s obzirom da programski sustav u sadašnjoj verziji ne podržava odabir agregacija), a logičko se izvodi automatski.

U primjeru podataka o prodaji (*sales data*) iz grafa ovisnosti dobivenog poluautomatskim postupkom (slika 6.44) izbacuje se čvor `shipDate` (tipa date), a umjesto njega uvodi čvor `shipAfterOrder` (tipa `integer`) u kojem je naveden broj dana od narudžbe do isporuke (brisanje i dodavanje čvorova opisano je u poglavljima 6.9.3 i 6.9.4). Dostava će biti izvedena kao posebna dimenzija pa `shipMethod` i `shipAfterOrder` dobivaju ključ dimenzije `shipKey`. Uvode se čvorovi povezani s datumom: umjetni vremenski ključ `timeKey` bit će tipa `positiveInteger`, ali jednoznačno povezan s pravim datumom `orderDate`. Uvode se i čvorovi `month`, `monthName`, `quarter` i `year`. `invoiceNum` je direktan potčvor činjenice. Hijerarhije postoje u vremenskoj dimenziji (`timeKey-month-quarter-year`) i dimenziji kupca (`customerKey-postCode`).

U konačnom se grafu ovisnosti nalaze četiri mjere: zbrojiva mjera prihoda (`income`), poluzbrojiva mjera količine prodanog proizvoda (`quantity`), poluzbrojiva mjera cijene (`price`) te nezbrojiva mjera broj računa (`invoiceNum`). U sljedećem poglavlju detaljno će se razjasniti razlozi postojanja nezbrojive mjere. Konačna konceptualna shema prikazana je na slici 6.57.

Konkretan broj dimenzija koje opisuju činjenicu, hijerarhije te odabir parametara koji će biti sadržani u dimenzijama ovise o korisničkim zahtjevima i razlikuju se za svako pojedinačno skladište. Univerzalnost se u postupku konceptualnog oblikovanja može postići pri oblikovanju vremenske dimenzije čiji je temelj gotovo uvijek datum, a ostali dimenzijski atributi izvode se iz njega.



Slika 6.57. Konačni graf ovisnosti za podatke o trgovini (*sales data*)

7. Izvedba logičkog oblikovanja područnog skladišta podataka iz XML izvora

7.1. Načela logičkog oblikovanja

7.1.1. Granica konceptualnog i logičkog oblikovanja

Kako je istaknuto u poglavlju 1.2.1, konceptualni model je u potpunosti nezavisan o sustavu za upravljanje bazom podataka i računalu na kojem se baza podataka nalazi, dok logički model predstavlja njegovu izvedbu u konkretnom sustavu za upravljanje bazom podataka. U ovom radu razvija se logički model za relacijsku bazu podataka temeljen na spoju zvijezda. Konceptualno i logičko oblikovanje predstavljaju jedinstveni zajednički proces višedimenzijskog oblikovanja i ne mogu se strogo razdvojiti. Dapače konceptualno i logičko (kod Kimballa [Kim96] implementacijsko) oblikovanje na početku povijesnog razvoja skladišta podataka čine nedjeljivu isprepletenu cjelinu, da bi se konceptualno oblikovanje izdvojilo tek naknadno.

Kao posljedica neraskidive veze konceptualnog i logičkog oblikovanja moguće je zamijetiti brojna načela konceptualnog oblikovanja koja nemaju uzrok i primjenu u samom konceptualnom oblikovanju, ali su nužna kako bi se kasnije uspješno izvelo logičko oblikovanje. Načela logičkog oblikovanja skladišta za izvedbu u relacijskoj tehnologiji koja utječu i na konceptualno oblikovanje opisana su u nastavku poglavlja 7.1. Iznose se samo ona načela koja su primijenjena u ovom radu, u konkretnim primjerima narudžbe robe (*purchase order*), podataka o prodaji (*sales data*) i prodaje knjiga (*book sale*) te naposljetku u studijskom primjeru u poglavlju 8. Objašnjava se načelo dijeljenih dimenzija koje opravdava postojanje nezbrojivih mjera u konceptualnom modelu skladišta podataka. Navodi se razlog zbog kojeg je cijena proizvoda u primjeru prodaje mjera, a ne atribut u dimenziji proizvoda, iako je sa stajališta konceptualnog oblikovanja moguće oboje. Na kraju se daje pregled zbrojivih, poluzbrojivih i nezbrojivih dimenzija.

7.1.2. Degenerirane dimenzije

U grafu sheme za podatke o trgovini čvor `invoice` imao je za potčvorove broj računa (`invoiceNum`), datume izdavanja računa (`orderDate`) i dostave robe (`shipDate`), načina dostave (`shipMethod`) te šifru kupca (`customerRef`). U početnom grafu ovisnosti dobivenom poluautomatskim postupkom čvor `invoice` zadržava ovih pet potčvorova (slika 6.44), ali se u postupku preuređivanja sam `invoice` izbacuje, `customerRef` s potčvorovima izdvaja u posebnu dimenziju kupca, `shipDate` i `shipMethod` ulaze u dimenziju dostave, a `orderDate` je osnova vremenske dimenzije. Čvor `invoiceNum` ne može se smjestiti niti u jednu od četiriju dimenzija (vrijeme, kupac, proizvod, dostava), ali može biti vrlo koristan za ograničenje upita po računu. Za ograničenje upita općenito služe dimenzije. Račun ustvari i jest zasebna dimenzija skladišta koja se sastoji od jednog jedinog atributa, budući da su svi ostali podaci vezani uz račun postali dijelovi drugih dimenzija.

U relacijskoj bazi podataka tablica činjenica sadrži strane ključeve na dimenzije, a među njima bi bio i strani ključ na dimenziju računa. Međutim, kako dimenzija računa ima samo jedan atribut, spajanjem činjenične tablice i takve dimenzijske tablice ne bi se dobili novi podaci pa se takva dimenzijska tablica ne izvodi. Stoga njen ključ u činjeničnoj tablici više nije strani ključ već običan atribut tj. mjera.

Opisana dimenzija-mjera naziva se degeneriranom dimenzijom (engl. *degenerated dimension*).

Broj računa je umjetno stvorena šifra, nema nikakvu vrijednost za sumiranje i predstavlja potpuno nezbrojivu mjeru. Formalno bi se mogao izraziti i nenumeričkim tipom podataka.

Može se zaključiti da se degenerirana dimenzija u konceptualnom modelu smije predočiti i kao dimenzija. Ona se, međutim, u relacijskoj bazi podataka neće realizirati kao zasebna tablica pa njezin ključ u činjeničnoj tablici nije strani ključ nego nezbrojiva mjera. Stoga se u konceptualnom oblikovanju ili mora zabraniti da se čvorovi bez vlastitih potčvorova odabiru za dimenzije, ili program za automatsko pretvaranje konceptualnog modela u logički model za relacijsku bazu podataka mora prepoznati degeneriranu dimenziju.

7.1.3. Vremenska promjenjivost podataka u dimenzijama

Podaci zapisani u dimenzijskim tablicama mogu biti vremenski promjenjivi. Tipičan primjer ovakvih podataka je bračni status ili dob u dimenziji koja opisuje osobe (npr. kupce trgovine, vlasnike police osiguranja itd.). Promjena bračnog statusa (primjerice, ako je osoba sklopila brak) ne može se registrirati jednostavnom promjenom vrijednosti u retku dimenzijske tablice. Prilikom sumiranja po bračnom statusu sve bi se ranije vrijednosti iz činjenične tablice dok je osoba bila nevjenčana zbrajale s podacima vjenčanih osoba. Za parametre koje se mijenjaju rijetko (u slučaju bračnog statusa do nekoliko puta u životu) dodaje se novi zapis u dimenzijsku tablicu, a stari prestaje biti aktualan.

Parametri čije se vrijednosti mijenjaju relativno često (više puta na godinu ili češće) stavljaju se, kao mjera, u činjeničnu tablicu gdje u svaki zapis smiju poprimiti drukčiju vrijednost. Cijena proizvoda tipičan je primjer ovakvog podatka i uvijek se izražava kao mjera. Osnovni problem ovakvog rješenja je porast veličine činjenične tablice koja je puno puta veća od bilo koje dimenzijske tablice. Za atribute koji se mijenjaju rjeđe od cijene, ali češće nego bračni status treba procijeniti povećanje činjenične tablice zbog eventualnog uvođenja nove mjere

7.1.4. Pregled potpuno zbrojivih, poluzbrojivih i nezbrojivih mjera

Potpuno zbrojive mjere mogu se zbrajati (sumirati) po svim razinama svih dimenzija. Poluzbrojive mjere mogu se sumirati samo po nekim dimenzijama, a nezbrojive mjere ne mogu se sumirati ni po jednoj dimenziji.

Potpuno zbrojive mjere

Potpuno zbrojiva (engl. *additive*) mjera je prihod (zarada) u primjeru podataka o trgovini (*sales data*). Prihod je moguće sumirati po svim četirima dimenzijama: vremenu (npr. grupiranje po mjesecima), kupcu (prema mjestu stanovanja kupca ili dobnim skupinama), proizvodu (u primjeru postoji samo jedna razina u dimenziji proizvod, dok u realnim skladištima postoji hijerarhija prema potkategorijama i kategorijama proizvoda: jogurt-mliječni proizvod-prehrambeni proizvod) ili dostavi (prema načinu dostave: poštom ili vozilom). Primjer zbrojivih mjera su, uz iste ili slične dimenzije i troškovi proizvodnje ili kupnje te profit (razlika prihoda i troškova).

Poluzbrojive mjere

Poluzbrojive (engl. *semiadditive*) mjere mogu se zbrajati samo po nekim dimenzijama skladišta podataka.

Neke se poluzbrojive mjere primjenom dodatnih matematičkih funkcija mogu napraviti zbrojivima po dimenzijama gdje obično sumiranje nije imalo smisla. Takva je skupina mjera čije vrijednosti nisu rastavljive na pribrojnike s obzirom na diskretne vremenske intervale, pa su stoga i inherentno nezbrojive po vremenskoj dimenziji. Stanje računa u banci primjer je takve mjere. Zbroje li se vrijednosti završnih dnevnih stanja računa tijekom mjesec dana, taj zbroj nema nikakvo značenje. Stanje računa ima uvijek samo trenutnu vrijednost, pa tako problem predstavlja čak i određivanje samog pojma dnevnog stanja računa. U praksi se može uzeti stanje u trenutku zatvaranja poslovnice, koje se očitava svaka 24 sata.

Agregacija stanja računa po vremenu ipak dobiva smisao ako se računa prosjek za neki vremenski interval. Zbroje se vrijednosti stanja računa za taj interval i podijele s brojem vremenskih jedinica u intervalu. Jednak je slučaj u skladištu podataka za stanje u spremištu robe. Zbrajanje količina pojedinog proizvoda na policama spremišta ima smisla tek kad se računa prosječna količina.

Kod druge vrste poluzbrojivih mjera zbrajanje po nekoj dimenziji nema nikakvog smisla čak niti uz primjenu bilo kakvih matematičkih operacija. Takva mjera je, primjerice, broj izdanih računa u skladištu podataka trgovine s dimenzijama vrijeme, proizvod i trgovina. Vrijednost tog stupca u činjeničnoj tablici opisuje koliko je računa promatranog dana u promatranoj trgovini izdano za promatrani proizvod. Mjera je zbrojiva po vremenu (prekjučer je za proizvod X u trgovini Y izdano 5, a jučer 7 računa; ukupno je u ta dva dana izdano 12 računa). Mjera je zbrojiva i po dimenziji trgovine, ali nije po dimenziji proizvod. Nekog dana je u nekoj trgovini za proizvod A izdano 5 računa, a za proizvod B 8 računa. Oba proizvoda mogu se i ne moraju nalaziti na istim računima. Računa je bilo najviše 13, a najmanje 8, ali se iz same mjere ne može odrediti koliko. Zbog toga se umjesto broja izdanih računa koristi nezbrojiva mjera šifre računa (kao u primjeru *sales data*) koja je primjenom agregacijskih funkcija unutar upita u potpunosti nadomješta.

Izvjesnu nezbrojivost u semantičkom smislu pokazuje i mjera količine prodanog proizvoda u skladištu u kojem uz vremensku dimenziju još mogu biti i dimenzije trgovine, kupca i sl. Dimenzija proizvod može imati hijerarhijske razine proizvod-potkategorija proizvoda-kategorija proizvoda ili proizvod-proizvođač(marka)-država proizvodnje. Količinu proizvoda ima smisla zbrajati po potkategorijama i kategorijama, ako je mjera količine zajednička (kilogram luka i tri kilograma krumpira čine četiri kilograma povrća; tri čašice kiselog mlijeka i dvije čašice jogurta čine pet čašica mliječnih proizvoda). Problem predstavlja slučaj kad se mjere količine ne mogu usporediti, čemu na višim razina agregacije raste vjerojatnost. Četiri boce piva po 0.5 litara i dvije boce vina po 1 litru ukupno daju 6 boca s 4 litre alkoholnog pića, pri čemu je mjera u litrama objektivna, za razliku od boca koje se razlikuju. Do besmislenih rezultata može doći u općenitom slučaju, kad je količina proizvoda opisana je paketima, "komadima" pojedinog proizvoda. Primjerice, nema korisne informacije zbroji li se 5 konzervi sardina, dva paketa toaletnog papira i brisač za automobilski vjetrobran te izrazi ukupna količina od 8 pakiranja.

Nezbrojive mjere

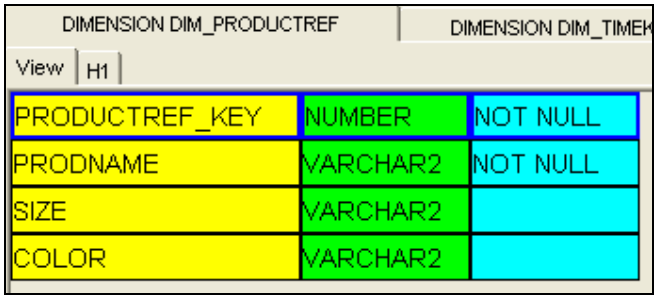
Vrijednosti nezbrojivih (engl. *nonadditive*) mjera ne mogu se zbrajati niti po jednoj dimenziji. Nezbrojive mjere zapravo su degenerirane dimenzije opisane u poglavlju 7.1.2, a njihova svrha je postavljanje ograničenja nad upitima, što i jest temeljna uloga dimenzije. Osim toga, mogu služiti i za prebrojavanje. Nezbrojive mjere ne opisuju kontinuirane mjerljive veličine, već su im vrijednosti samo oblik zapisa šifri, što bi se

moglo izvesti i u drugom, nebrojčanom tipu podataka. Tipične nezbrojive dimenzije su serijski i kontrolni brojevi: šifra računa (engl. *invoice number*) ili šifra retka na računu (engl. *line item number*).

7.2. Izvedba logičkog oblikovanja u programskom sustavu

Nakon što projektant skladišta podataka odredi dimenzijske hijerarhije, program automatski stvara naredbe u SQL-u za izradu dimenzijskih tablica i činjenične tablice u relacijskoj bazi podataka. Dimenzijske tablice mogu se, istodobno s hijerarhijama, vidjeti u grafičkom sučelju (slika 7.1). Vide se imena atributa, tip podataka kojim se atribut predstavlja u bazi (nužno je definirati pretvaranje tipova XML Scheme u tip bez podataka) i dozvola da atribut poprimi nul-vrijednost.

Izvodi se standardna naredba za izradu tablica `CREATE TABLE` u kojoj se navode imena atributa tablice, tip podataka kojem pripada svaki atribut te dozvola postavljanja atributa na nul-vrijednost. Ako je neki čvor uvjetno pridružen nadređenom čvoru, XML Schema dozvoljava da se on u dokumentima ne pojavi. U tom slučaju u tablici skladišta podataka treba ga zamijeniti nul-vrijednošću. Ako se element ili atribut u XML dokumentima pojavljuje obavezno (bezuvjetno), za atribut kojim je predstavljen u tablici skladišta postavlja se zabrana nul-vrijednosti.



Slika 7.1. Prikaz dimenzijske tablice proizvoda u grafičkom sučelju

Konceptualna shema područnog skladišta za trgovinu (*sales data*) dana je na slici 6.57. Naredba kojom se u bazi podataka Oracle 9i Release2 stvara dimenzija proizvoda s ključnim atributom `productRef` kojem je obavezno pridružen atribut `productName` i uvjetno atributi `size` i `color` imala bi sljedeći izgled:

```
CREATE TABLE "DIM_PRODUCTREF"(
"PRODUCTREF_KEY" NUMBER(10) NOT NULL,
"PRODNAME" VARCHAR2(30) NOT NULL,
"SIZE" VARCHAR2(30),
"COLOR" VARCHAR2(30)
);
```

Program ključu dimenzijske tablice automatski dodaje nastavak `_KEY`. Tako `productRef` iz konceptualne sheme u tablici postaje `PRODUCTREF_KEY`. Dimenzijskoj se tablici dodjeljuje ime po ključu dimenzije s prefiksom `DIM_`: `DIM_PRODUCTREF`. Ključni atribut `PRODUCTREF_KEY` po definiciji ne smije poprimiti nul-vrijednost. `PRODNAME` je ključu pridružen obavezno pa ne smije poprimiti nul-vrijednost, dok uvjetno pridruženi `SIZE` i `COLOR` smiju. Potrebno je još izvesti naredbu `ALTER TABLE` (koja općeniti služi za promjene strukturnih obilježja tablice) za postavljanje primarnog ključa na `PRODUCTREF_KEY`:

```
ALTER TABLE "DIM_PRODUCTREF" ADD CONSTRAINT
"PRIMARY_KEY_DIM_PRODUCTREF" PRIMARY KEY("PRODUCTREF_KEY");
```

Na sličan način izgrađuju se ostale dimenzijske tablice. Činjenična se tablica u bazi stvara nakon što već postoje sve dimenzijske tablice budući da sadrži strani ključ na dimenzijske tablice. Najprije se naredbom `CREATE TABLE` stvara činjenična tablica koja sadrži četiri atributa, strana ključa na dimenzije, te četiri mjere (`invoiceNum`, `quantity`, `price`, `income`), a potom se naredbama `ALTER TABLE` postavljaju strani ključevi:

```
CREATE TABLE "FACT_LINEITEM"(
"FOR_KEY_PRODUCTREF_KEY" NUMBER(10) NOT NULL,
"FOR_KEY_TIME_KEY" NUMBER(10) NOT NULL,
"FOR_KEY_SHIP_KEY" NUMBER(10) NOT NULL,
"FOR_KEY_CUSTOMER_KEY" NUMBER(10) NOT NULL,
"MEASURE_INVOICENUM" NUMBER(20) NOT NULL,
"MEASURE_QUANTITY" NUMBER(20) NOT NULL,
"MEASURE_PRICE" NUMBER(20, 2) NOT NULL,
"MEASURE_INCOME" NUMBER(20, 2) NOT NULL);

ALTER TABLE "FACT_LINEITEM" ADD CONSTRAINT "FK_DIM_PRODUCTREF"
FOREIGN KEY ("FOR_KEY_PRODUCTREF_KEY") REFERENCES
"DIM_PRODUCTREF"("PRODUCTREF_KEY");

ALTER TABLE "FACT_LINEITEM" ADD CONSTRAINT "FK_DIM_TIMEKEY"
FOREIGN KEY ("FOR_KEY_TIME_KEY") REFERENCES
"DIM_TIMEKEY"("TIME_KEY");

ALTER TABLE "FACT_LINEITEM" ADD CONSTRAINT "FK_DIM_SHIPKEY"
FOREIGN KEY ("FOR_KEY_SHIP_KEY") REFERENCES
"DIM_SHIPKEY"("SHIP_KEY");

ALTER TABLE "FACT_LINEITEM" ADD CONSTRAINT
"FK_DIM_CUSTOMERKEY" FOREIGN KEY ("FOR_KEY_CUSTOMER_KEY")
REFERENCES "DIM_CUSTOMERKEY"("CUSTOMER_KEY");
```

Imena svih mjera u činjeničnoj tablici dobivaju prefiks `MEASURE_`, a imena svih atributa koji predstavljaju strane ključeve prefiks `FOR_KEY_` nakon čega slijedi ime ključnog atributa dimenzije.

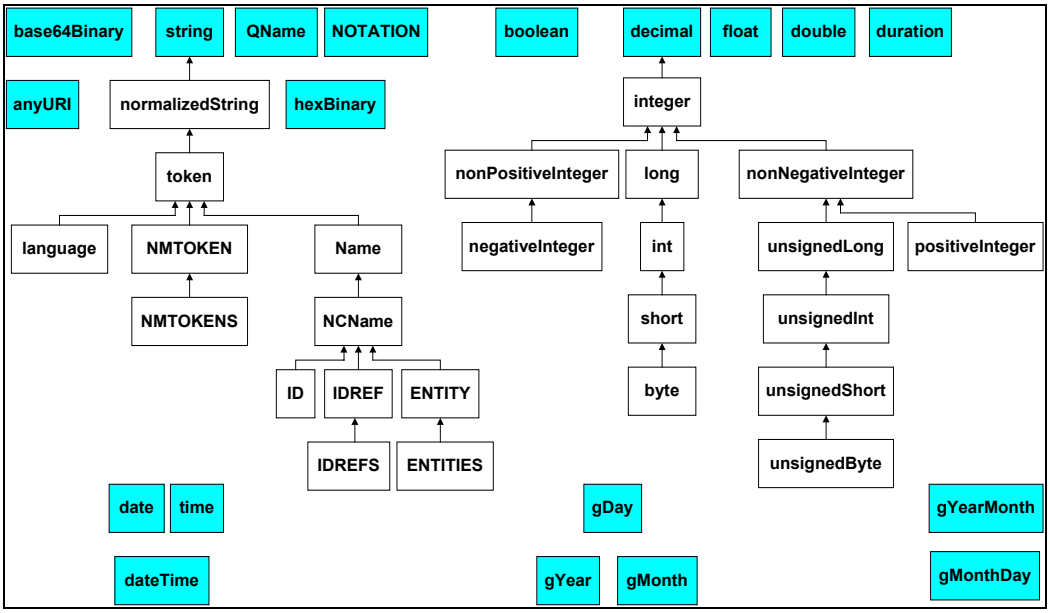
7.2.1. Pretvaranje tipova XML Scheme u tipove baze podataka

Sadržaj elemenata i atributa čine tekstualni čvorovi odnosno vrijednosti atributa. Oni predstavljaju niz znakova. DTD ne omogućuje detaljnije definiranje ograničenja nad nizom znakova, dok XML Schema definira čak 44 osnovna tipa, od toga 19 primitivnih tipova [Sch2-04]. Primitivni tipovi ne mogu se izvesti jedan iz drugoga, dok je preostalih 25 osnovnih tipova u konačnici izvedeno iz primitivnih tipova `string` i `decimal`. Način na koji se 25 izvedenih osnovnih tipova izvode jedan iz drugog [Sch2-04] prikazuje slika 7.2. Primitivni tipovi prikazani su plavim, a izvedeni bijelim pravokutnicima.

Tijekom izrade grafa sheme za svaki čvor jednostavnog elementa ili atributa pamti se tip podataka kojem pripada. Za čvorove izvedenog tipa pamti se i osnovni tip iz kojeg je tip izveden.

Prilikom stvaranja tablica u skladištu podataka svakom je atributu potrebno definirati tip podataka. Atributu se nastoji dodijeliti tip podataka u bazi koji je najsrodniji originalnom tipu XML Scheme. SQL definira različite tipove podataka među kojima

se posebno ističu brojčani, znakovni i datumski tip. U na tržištu trenutno vodećim komercijalnim bazama podataka (Oracle 9i, IBM DB2, Microsoft SQL Server 2000) uz navedene osnovne tipove postoje i različiti dodatni tipovi, primjerice `boolean`, `float` i `double`, koji odgovaraju standardnim tipovima u programskim jezicima.



Slika 7.2. Primitivni i izvedeni osnovni tipovi

Odabir odgovarajućih tipova baze podataka za tipove XML Scheme može se provesti na dva načina:

- za sve atribute u bazi koristiti samo brojčani, znakovni i datumski tip
- koristiti sve postojeće tipove koje nudi pojedina baza podataka, uključujući i definiranje novih tipova

Isključivo korištenje triju temeljnih tipova koje definira svaka baza (brojčani, znakovni, datumski) omogućuje jednostavni zajednički koncept pretvaranja tipova za sve sustave za upravljanje bazom podataka. Korištenje svih tipova koje nudi neka baza podataka zahtijeva da se za svaku bazu podataka zasebno definira pridruživanje SQL-tipa XML-tipu.

Autorovo je mišljenje da treba iskoristiti potencijal koji preciznim i kvalitetnim definiranjem tipova podataka nudi proizvođač baze kako ne bi došlo do gubitka dijela podataka, njihove preciznosti ili ponovne iskoristivosti (engl. *reusability*). Neke tipove podataka (`float` i `double`) najbolje je pohraniti u njihovom originalnom načinu zapisivanja. Pritom treba paziti da se ne naruši zbrojivost mjera ili postavi premali broj mjesta u memoriji. Prilikom odabira tipova podataka korisno je pogledati već postojeća skladišta podataka organizacije kako bi se vidjelo koji se tipovi podataka koriste u njemu. Usprkos tome, programski alat mora imati mogućnost izvođenja tablica samo uz korištenje brojčanog, znakovnog i datumskog tipa.

Tablice skladišta podataka stvaraju se u bazi podataka Oracle 9i Release 2. U sustavu upravljanja bazom podataka Oracle brojčani tip ima naziv `NUMBER`, znakovni `VARCHAR2`, a datumski `DATE`. Kod brojčanog tipa navodi se najveći ukupan broj znamenki (*precision*) te najveći broj decimalnih mjesta (*scale*), a kod znakovnog

najveći dozvoljeni broj znakova. Nenavođenje broja decimalnih mjesta pretpostavlja cijeli broj. Općenito se količina memorije potrebna za spremanje pojedinog atributa može doznati ispitivanjem sadržaja izvornih XML dokumenata. U praksi se ostavlja još znamenki odnosno znakova za slučaj da budući dokumenti čiji će se sadržaj pohraniti u skladište sadrže dulje riječi ili veće brojeve od početnih. Program po definiciji ne dozvoljava da atributi znakovnog tipa imaju prostora za pohranu manje od 30 znakova, brojčani dimenzijski ključevi manje od 10, a mjere manje od 20 znamenki.

Tip `string` i svi iz njega izvedeni tipovi (slika 7.2) pretvaraju se u znakovni tip (u bazi Oracle 9i to je `VARCHAR2`). Tip `decimal` sa svim izvedenim tipovima pretvara se u brojčani tip, `NUMBER`.

Tipovi `hexBinary`, `base64Binary`, `QName`, `anyURI` i `NOTATION` sadrže znakove koji nisu znamenke i, osim `hexBinary`, nemaju numeričko značenje. Svi se pretvaraju u niz znakova. `hexBinary` se ne pohranjuje kao broj jer prema specifikaciji [Sch2-04] sadrži općenite podatke kodirane u heksadecimalnom zapisu, a ne nužno brojeve.

`float` i `double` sadrže brojeve. Ukoliko je dozvoljeno koristiti samo osnovni brojčani tip, pretvorit će se u njega. U suprotnom za njihovu pohranu koristit će se odgovarajući tipovi `FLOAT` i `DOUBLE` (koji postoje u bazi Oracle 9i).

Tip `duration` opisuje vremensko trajanje. Njegov zapis izvodi se u mješovitom znakovno-brojčanom formatu. Pohrana je predviđena u brojčanom obliku (pretvaranje godina, mjeseci, dana, sati i minuta u sekunde). U procesu izvlačenja transformacije i učitavanja podataka nužno je zadati pretvorbenu funkciju.

`time` opisuje vrijeme, `date` datum, a `dateTime` kombinira zapis datuma i vremena. U bazi Oracle 9i datumski tip sadrži i oznaku vremena, tako da se sva tri tipa pretvaraju u datumski tip. Pritom je potrebno voditi računa da se u radu sa skladištem kod atributa u tablicama temeljenim na tipu `time` zanemari datum, a kod onih temeljenih na tipu `date` zanemari vrijeme.

Logički tip, `boolean`, poprima samo vrijednosti točno (koja se zapisuje s `true` ili `1`) te netočno (koje se opisuje s `false` ili `0`). U bazi Orale 9i moguće ga je pohraniti u odgovarajući tip BOOLEAN. Drugi je način pohrane u vidu jednog znaka, koristeći tip za fiksni broj znakova, `CHAR(1`, ili jedne znamenke, `NUMBER(1)`. U procesu izvlačenja transformacije i učitavanja podataka treba odrediti pretvorbenu funkciju te način zapisa `T/F` ili `1/0`.

Tipovi `gDay`, `gMonth` i `gYear` označavaju dan u mjesecu, mjesec, odnosno godinu. Njihovi zapisi sadrži znamenke i znak "-". Korištenjem pretvorbene funkcije mogu se pretvoriti u prave brojeve. Za datum i mjesec potrebne su dvije, a za godinu četiri znamenke.

`gMonthDay` opisuje mjesec i dan u mjesecu bez navođenja godine, a `gYearMonth` godinu i mjesec. Zapis se izvodi u obliku niza znakova, a u skladištu mora biti definirana funkcija koja će iz niza znakova izvlačiti dan, mjesec odnosno godinu. Druga mogućnost je automatsko stvaranje dvaju odvojenih brojčanih atributa za zapis dana i mjeseca odnosno mjeseca i godine. Treća mogućnost, zapis u obliku datuma uz kasnije zanemarivanje dijelova datuma, je nepraktična.

7.2.2. Stvaranje dimenzija i hijerarhija u bazi podataka

Baza Oracle 9i omogućuje da se nad tablicama koje predstavljaju dimenzije stvore dodatni metapodaci o dimenzijskim hijerarhijama. Vremenska hijerarhija u konačnoj konceptualnoj shemi za podatke o trgovini (slika 6.57) ima četiri razine: `timeKey-month-quarter-year`. Stvaranje vremenske dimenzijske tablice izvršava se sljedećim kodom u SQL-u

```
CREATE TABLE "DIM_TIMEKEY"(
"TIME_KEY" NUMBER(10) NOT NULL,
"MONTH" NUMBER(10) NOT NULL,
"QUARTER" NUMBER(10) NOT NULL,
"YEAR" NUMBER(20) NOT NULL,
"MONTHNAME" VARCHAR2(30) NOT NULL,
"ORDERDATE" DATE NOT NULL,
"DAYOFWEEK" VARCHAR2(30) NOT NULL);

ALTER TABLE "DIM_TIMEKEY" ADD CONSTRAINT "PRIMARY_KEY_DIM_TIMEKEY"
PRIMARY KEY("TIME_KEY");
```

Oracleov SQL-kod za stvaranje metapodataka o vremenskoj dimenziji definira sve četiri hijerarhijske razine, te svakom ključnom atributu razine pridružuje ostale, neključne atribute (ključ najdetaljnije razine `H1_L1_DIM_TIMEKEY`, `TIME_KEY` određuje `ORDERDATE` i `DAYOFWEEK`):

```
CREATE DIMENSION "DIMENSION_DIM_TIMEKEY"
LEVEL "H1_L1_DIM_TIMEKEY" IS "DIM_TIMEKEY"."TIME_KEY"
LEVEL "H1_L2_DIM_TIMEKEY" IS "DIM_TIMEKEY"."MONTH"
LEVEL "H1_L3_DIM_TIMEKEY" IS "DIM_TIMEKEY"."QUARTER"
LEVEL "H1_L4_DIM_TIMEKEY" IS "DIM_TIMEKEY"."YEAR"
HIERARCHY H1_DIM_TIMEKEY (
"H1_L1_DIM_TIMEKEY"   CHILD   OF   "H1_L2_DIM_TIMEKEY"   CHILD   OF
"H1_L3_DIM_TIMEKEY" CHILD OF "H1_L4_DIM_TIMEKEY")
ATTRIBUTE "H1_L1_DIM_TIMEKEY" DETERMINES ("ORDERDATE", "DAYOFWEEK")
ATTRIBUTE "H1_L2_DIM_TIMEKEY" DETERMINES ("MONTHNAME");
```

Pregled naredbi u SQL-u za stvaranje svih dimenzijskih tablica, dimenzijskih metapodataka te činjenične tablice dan je u dodatku E1.

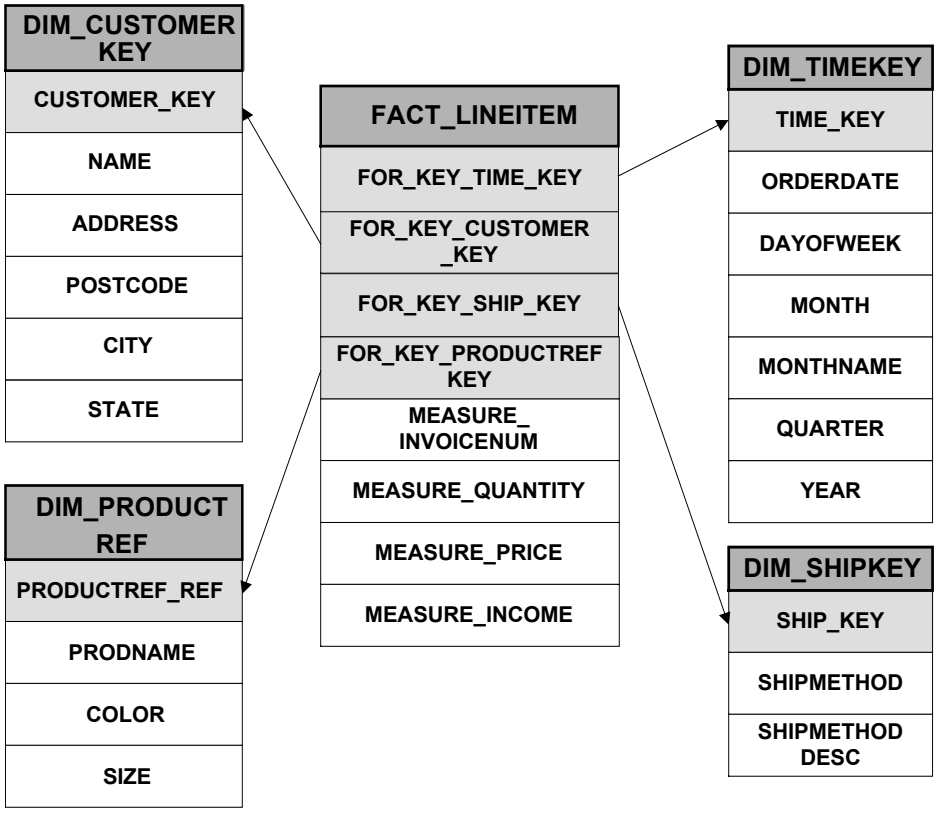
SQL naredbe za stvaranje tablica skladišta podataka izvode se spajanjem na bazu podataka preko sučelja JDBC [Jdbc].

Aplikacija se izravno spaja na bazu podataka, pri čemu njen korisnik, projektant skladišta, navodi podatke o računalu na kojem se nalazi baza (URL), parametre spajanja (TCP *port* i druge parametre specifične za pojedine baze: kod baze Oracle 9i to je *SID*, ime logičke instance baze na računalu budući da je dozvoljeno na istom računalu postaviti veći broj logički nezavisnih instanci baze), korisničko ime i lozinku.

Koristi li se *applet*, spajanje se mora izvesti preko poslužitelja na kojem postoji podrška za JDBC (poglavlje 4.5.2). Korisnik, jednako kao i u slučaju spajanja preko aplikacije, navodi URL računala, parametre spajanja te korisničko ime i lozinku.

Aplikacija ne omogućuje pregled stvorenih tablica. Za tu svrhu se mora koristiti klijentski program baze podataka u kojoj su tablice stvorene.

Logička shema za primjer *sales data* (graf sheme na slici 6.29, početni graf ovisnosti dobiven poluautomatskim postupkom na slici 6.44, konačni graf ovisnosti na slici 6.57) prikazana je na slici 7.3.



Slika 7.3. Logička izvedba područnog skladišta podataka za podatke o trgovini

7.2.3. Izvedba dijeljenih hijerarhija i konvergencije

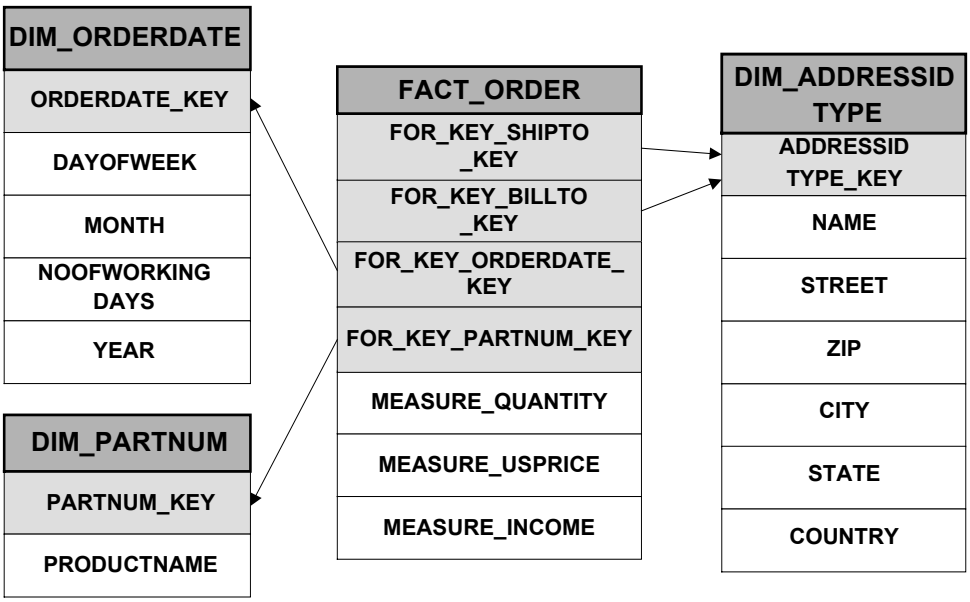
U primjeru narudžbe robe (konceptualna shema na slici 6.54) postojala je dijeljena hijerarhija u dimenzijskom čvoru `AddressIDType` s instancama `shipTo` i `billTo`. Drukčiji se postupak logičkog oblikovanja koristi ako se dijeljena hijerarhija ili konvergencija nalaze u čvoru koji je korijen dimenzije nego kad se nalaze u nekom drugom dimenzijskom čvoru.

Ako je korijen dimenzije konvergentan čvor, postupa se kao da se radi o običnom čvoru. Izgrađuje se jedna dimenzijska tablica koja se referencira jednim stranim ključem iz činjenične tablice.

Dijeljena hijerarhija u korijenu dimenzije podrazumijeva postojanje više dimenzija koje su zbog potpuno jednake strukture pohranjene u jednoj tablici. Izgrađuje se samo jedna dimenzijska tablica, ali se u činjeničnoj tablici stvara onoliko stranih ključeva na tu dimenziju koliko ima instanci u dijeljenoj hijerarhiji. U primjeru narudžbe (graf sheme na slici 6.28 početni graf ovisnosti dobiven poluautomatskim postupkom na slici 6.48, konačni graf ovisnosti na slici 6.54) za čvor `AddressIDType` izgrađuje se jedna dimenzija s primarnim ključem `ADDRESSIDTYPE_KEY`, ali se na nju odnose dva strana ključa u činjeničnoj tablici `FACT_ORDER`: `FOR_KEY_SHIPTO_KEY` i `FOR_KEY_BILLTO_KEY` (slika 7.4). Naredbe u SQL-u za stvaranje svih tablica te dimenzijskih metapodataka za primjer narudžbe robe nalaze se u dodatku E2.

U slučaju da se dijeljena hijerarhija nalazi u potčvoru ključa dimenzije bilo koje razine, s tim da sve staze koje ulaze u taj čvor izlaze iz jednog dimenzijskog ključa, program od projektanta skladišta traži da odredi želi li napraviti posebnu tablicu (u tom slučaju spoj zvijezda postaje spoj pahuljica) ili će se za ukupno M instanci čvora

s N potčvorova načiniti M*(N+1) dimenzijskih atributa (N+1 budući da se radi o čvoru i N potčvorova). Stvaranje zasebne tablice preporučljivo je samo u slučaju da ona sadržava vrlo velik broj atributa. U slučaju konvergencije postupa se kao da se radi o običnom čvoru s N potčvorova te se u dimenzijskoj tablici izvodi N+1 atribut.



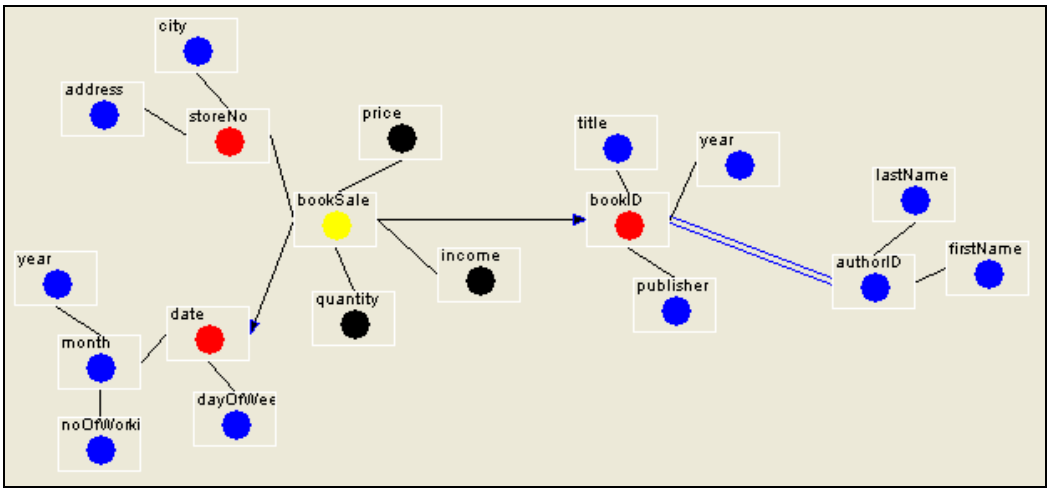
Slika 7.4. Logička izvedba područnog skladišta podataka za narudžbu robe

Ako je čvor dijeljena hijerarhija s N potčvorova, pri čemu staze koje ulaze u čvor izlaze iz različitih dimenzija, odustaje se o koncepta dijeljene dimenzije te se u svakoj dimenziji izvodi N+1 atribut. U slučaju konvergencije projektantu skladišta omogućuje se odabir između realizacije konvergentnog čvora zasebnom tablicom (kad se jedna dimenzija izvodi u dvije ili više tablica govori se o pahuljastom spoju, engl. *snowflake*) ili pak odustajanja od konvergencije i izvođenja N+1 atributa u svakoj dimenziji.

7.2.4. Izvedba veze više-prema-više

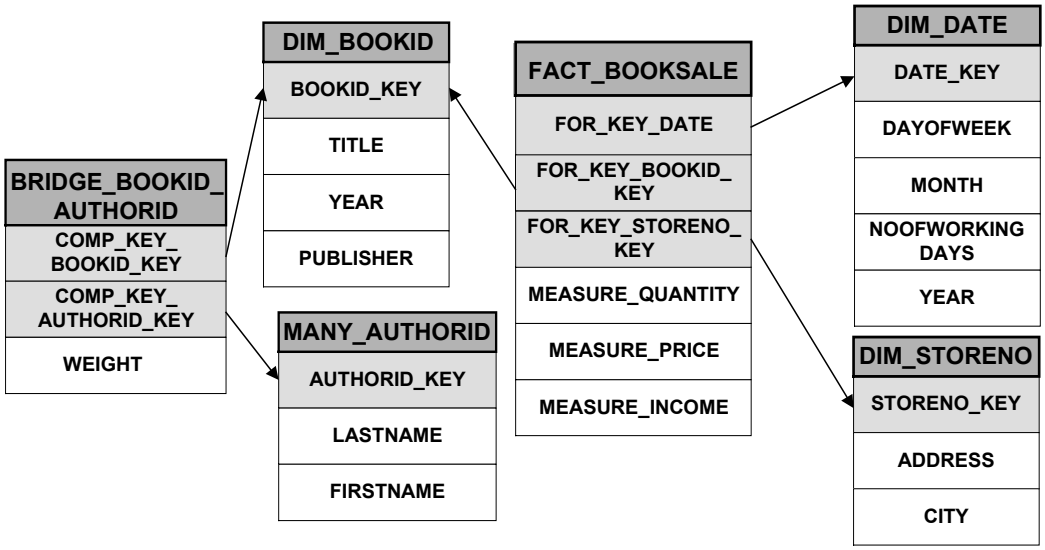
Veza više-prema-više se, kako je već opisano u poglavlju 6.7.2, može se pretvoriti u posebni entitet. Novi entitet sa svakim od početna dva tvori vezu prema-jedan, a oni s njim vezu prema-više. U relacijskim bazama podataka svaki od početna dva entiteta ima svoju tablicu, a povezuje ih tablica koja predstavlja novi, treći entitet. Ona sadrži samo dva atributa, po jedan strani ključ na svaku od tablica [Kim96]. U konkretnom slučaju, vezom više-prema-više povezana su dva čvora u dimenzijskoj hijerarhiji. Čvor koji funkcijski ovisi o dimenzijskom ključu ili je sam dimenzijski ključ (ostat će u dimenzijskoj tablici, dok će drugi čvor biti osnova nove tablice. Na taj način spoj zvijezda prelazi u pahuljasti (*snowflake*) spoj.

Za primjer prodaje knjiga konačna je konceptualna shema na slici 7.5 (početni graf ovisnosti dobiven poluautomatskim postupkom na slici 6.38).



Slika 7.5. Konačna konceptualna shema za prodaju knjiga (*book sale*)

Čvor `bookID` osnova je dimenzijske tablice, a `authorID` zasebne tablice koja je s dimenzijskom spojena preko dodatne treće tablice, `BRIDGE_BOOK_AUTHOR`, namijenjene ostvarivanju veze više-prema-više (slika 7.6). U trećoj tablici uz strane ključeve na tablice `BOOK` i `AUTHOR` postoji atribut `WEIGHT` (težina).



Slika 7.6. Logička izvedba područnog skladišta podataka za prodaju knjiga

U skladištu podataka kod veze više-prema-više dolazi do problema višestrukog zbrajanja prema autorima. Tablica 7.1 prikazuje podatke o prihodu od knjiga čiju prodaju opisuju XML dokumenti dodatku C2, C3, C4 i C5. Ukupan prihod od prodaje knjiga iznosi 700.48 kuna.

Agregacija po ruskim autorima (npr. ukupni prihod svih autora iz Rusije, kad bi u skladištu uz svakog autora još bila navedena i njegova domovina-`COUNTRY` u tablici `AUTHOR`, računao bi se pomoću `GROUP BY AUTHOR.COUNTRY`) daje ukupni prihod od 856.94 kune, što je uzrokovano dvostrukim prebrojavanjem knjige "Zlatno tele" koja ima dva autora (tablica 7.2). U slučaju agregacije po parametru koji je s dimenzijskim ključem povezan vezom više-prema-više nužno je uvesti tzv. težinske faktore. Za knjigu koja ima dva autora, svaki autor dobiva težinski faktor 0.5 (tj. 1/2),

za knjigu koja ima tri autora, svaki autor 0.333 (tj. 1/3). Općenito koautor knjige s ukupno N autora dobiva težinski faktor 1/N. Pretpostavlja se da svaki autor jednako doprinosi prihodu od knjige.

Tablica 7.1. Podaci o prihodu od prodaje knjiga

Naslov	Autor(i)	Izdavač	Godina	Cijena	Količina	Prihod
Zlatno tele	Iljf i Petrov	Školska knjiga	1999	78.23	2	156.46
Idiot	Dostojevski	Školska knjiga	2001	132.90	1	132.90
Braća Karamazovi	Dostojevski	Otokar Keršovani	2003	205.56	1	205.56
Braća Karamazovi	Dostojevski	AGM	2003	205.56	1	205.56
UKUPNO						700.48

Tablica 7.2. Pogrešan ukupni zbroj kod agregacije prihoda po autoru

Autor	Prihod
Iljf	156.46
Petrov	156.46
Dostojevski	544.02
UKUPNO	856.94

Upotrebom težinskih faktora dodjeljuje se Iljfu pola prihoda od "Zlatnog teleta", a Petrovu druga polovica. Na taj način dobiva se ispravni ukupni zbroj (tablica 7.3). Odgovor na pitanje "Koliko je ukupno prodano knjiga autora Iljfa, Petrova i Dostojevskog?" bez upotrebe težinskih faktora dao bi vrijednost 2+2+3=7. Ispravna vrijednost 5 dobiva se korištenjem težinskih faktora.

Tablica 7.3. Ispravno agregacijsko zbrajanje uz upotrebu težinskih faktora

Autor	Prihod knjige	Težinski faktor	Agregirani prihod
Iljf	156.46	0.5	78.23
Petrov	156.46	0.5	78.23
Dostojevski	544.02	1	544.02
UKUPNO			700.48

8. Studijski primjer oblikovanja skladišta podataka iz polustrukturiranih izvora podataka: *OAGIS 7.2.1. Purchase Order*

Načela oblikovanja skladišta podataka iz polustrukturiranih izvora do sada su bila prikazana na jednostavnim primjerima iz specifikacija i radnih nacrta organizacije W3C. Za njihovu je verifikaciju potrebno izraditi studijski primjer koji će se temeljiti na polustrukturiranim (odnosno XML) dokumentima koji se stvarno koriste u mrežnoj razmjeni između poslovnih organizacija.

Studijski primjer [Agr04] izrađen je na osnovi dokumenta ***OAGIS 7.2.1. Purchase Order*** koji poduzeće Agrokor koristi za narudžbu robe od svojih poslovnih partnera. Predložak dokumenta izradila je organizacija Open Applications Group (OAG), neprofitno udruženje kompanija i pojedinaca koje potiče korištenje i standardizaciju u elektroničkom poslovanju te standardizaciju u integraciji programskih sustava. Ta organizacija je donijela specifikaciju OAGIS (Open Applications Group Integration Specification) kojom se definira zajednički koncept XML dokumenata za opis poslovnih procesa. Dokument *OAGIS 7.2.1. Purchase Order*, je dio verzije 7.2.1 ove specifikacije.

8.1. Opis dokumenta OAGIS 7.2.1. Purchase Order

Dokument *OAGIS 7.2.1. Purchase Order* opisuje narudžbu roba ili usluga te njihovu isporuku. OAG je oblikovao dokument s ciljem da posluži u što široj primjeni pa je on stoga izuzetno detaljan i može opisati veliki broj različitih modela narudžbi i isporuke robe. Zbog toga najveći broj elemenata u XML Schemi (*003_process_po_007.xsd*, dodatak F1) postoji uvjetno. XML Schema za *OAGIS 7.2.1. Purchase Order* sadrži i elemente iz XML Schema *OAGIS Fields* (*oagis_segments.xsd*, dodatak F2) i *OAGIS Segments* (*oagis_fields.xsd*, dodatak F3) u kojima su definirani osnovni pojmovi vezani uz poslovanje (partner, adresa, količina, cijena). Oni se koriste i u drugim tipovima dokumenata specifikacije OAGIS 7.2.1. Prefiksi *of* i *os* kod XML elemenata označavaju da dani elementi pripadaju imenicima dokumenata *OAGIS Fields* (*of*) odnosno *OAGIS Segments* (*os*), za razliku od *Purchase Order* čiji imenik ne nosi prefiks.

Kako je ranije naglašeno, model narudžbe koji koristi Agrokor samo je jedan od brojnih modela koji se mogu opisati dokumentom *OAGIS 7.2.1. Purchase Order*. Stoga će u nastavku biti opisani samo oni njegovi elementi i atributi koji se stvarno koriste u Agrokorovim narudžbama (primjer narudžbe je u dodatku F4) i samo će oni biti prikazani u grafu sheme. Svi elementi i atributi koji neće biti opisani navedeni su u XML Schemi s uvjetnim kardinalnostima pridruživanja (`minOccurs=0`) tj. u dokumentu ih je dozvoljeno izostaviti.

Osnovni element dokumenta, `PROCESS_PO_007`, sadrži dva podelementa: `of:CNTROLAREA` (koji sadrži podatke o tipu i vremenu nastanka dokumenta, te sudioniku poslovnog procesa koji ga je stvorio i poslao) i `DATAAREA` (koji sadrži podatke o narudžbi u svom jedinom podelementu `PROCESS_PO`). Narudžba se odnosi na jedan ili više proizvoda.

Element `PROCESS_PO` sadrži:

- zaglavlje narudžbe (`POORDERHDR`),

- jednu ili više stavki računa (`POORDERLIN`) od kojih se svaka odnosi na jedan pojedinačni proizvod.

U zaglavlju narudžbe obavezno se nalaze:

- točno jedan element `of:POID` (*Purchase Order Identifier*, serijski broj narudžbe),
- točno jedan element `of:POTYPE` (*Purchase Order Type*, tip narudžbe; budući da se kod svih Agrokorove narudžbi radi o istoj vrijednosti, 224, ovaj element nije zanimljiv za skladištenje),
- dva ili više podelemenata `os:PARTNER` od kojih svaki označava jednog sudionika u procesu naručivanja. Ulogu partnera u procesu opisuje podelement od `os:PARTNER`, `of:PARTNRTYPE`. Uz obavezne sudionike dobavljača (`Supplier`) i kupca (`SoldTo`, u Agrokorovim narudžbama kupac je uvijek Agrokor) u Agrokorovim dokumentima se kao stranka kojoj se isporučuje roba (`ShipTo`) navode različite lokacije Agrokorovih skladišta.

Podelementi zaglavlja koje XML Schema ne navodi kao obavezne, ali se pojavljuju u svim Agrokorovim narudžbama su:

- `of:ACKREQUEST`, koji daje odgovor na pitanje treba li na dokument *Process PO* poslati kao odgovor dokument *OAGIS 7.2.1. Acknowledge PO* (u svim Agrokorovim dokumentima navodi se vrijednost 0 koja označava da odgovor nije potreban); ovaj element je, međutim, nebitan za skladištenje,
- složeni podelement `os:DATETIME`, koji opisuje vrijeme i sadrži podelemente: `of:YEAR`, `of:MONTH`, `of:DAY`, `of:HOUR`, `of:MINUTE`, `of:SECOND`, `of:SUBSECOND` (opisuje milisekunde), `of:TIMEZONE`. Vrijednost njegovog atributa `minOccurs` u XML Schemi je `0`. Oznaka vremena, međutim, postoji u zaglavlju svake Agrokorove narudžbe, pa se `minOccurs` postavi na `1`.

Svaka stavka računa obavezno sadrži:

- točno jedan element `of:POLINENUM`, redni broj stavke koji se najčešće navodi prirodnim redoslijedom, počevši od 1,
- složeni podelement koji predstavlja naručenu količinu, `os:QUANTITY`; element količine sadrži podelemente: `of:VALUE` (iznos količine, bez decimalne oznake), `of:NUMOFDEC` (broj decimalnih mjesta u polju `of:VALUE`), `of:SIGN` (pozitivni ili negativni predznak) te `of:UOM` (*Unit of Measure*, količinske jedinice, npr. komadi, parovi, jedinice duljine ili mase).

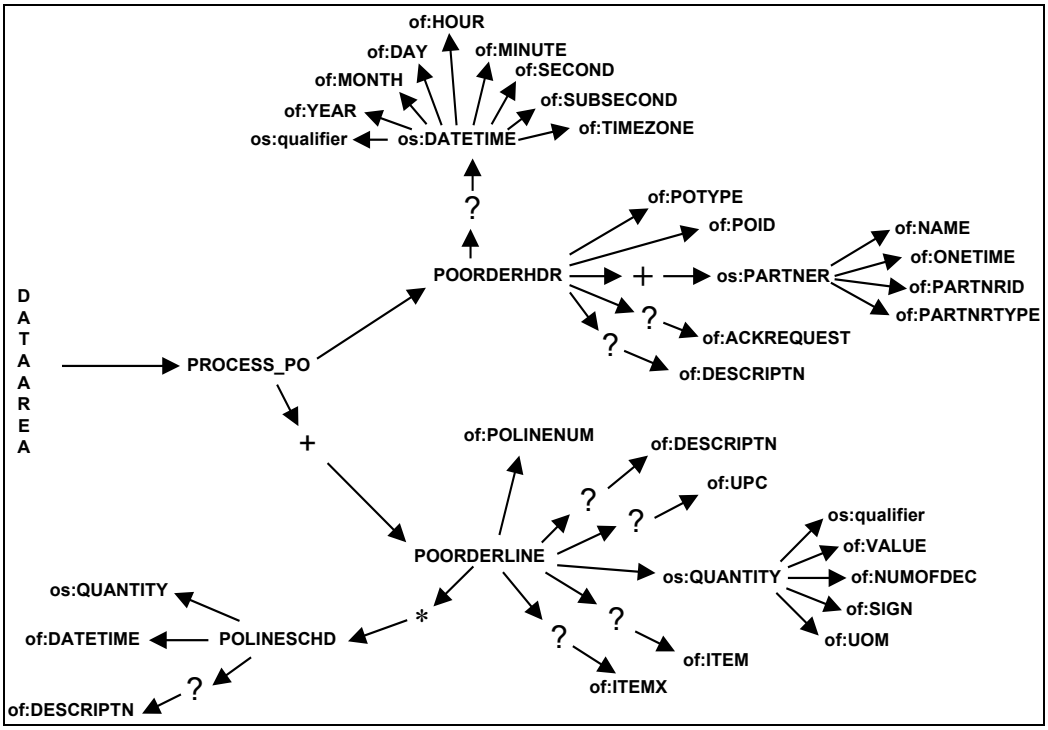
Proizvod čiju narudžbu opisuje stavka može se definirati njezinima četirima podelementima (od kojih u stavci mora biti barem jedan, ali niti jedan nije obavezan):

- `of:UPC` (*Universal Product Code*, kao vrijednost ovog polja unosi se međunarodna identifikacijska oznaka proizvoda tj. robe),
- `of:ITEM` (identifikator robe ili usluge),
- `of:ITEMX` (identifikator proizvoda koji koristi dobavljač) te
- `of:DESCRIPTN` (tekstualni opis robe ili usluge).

U Agrokorovim dokumentima koristi se međunarodna oznaka `of:UPC` te opis proizvoda na hrvatskom jeziku kao vrijednost podelementa `of:DESCRIPTN`. Zbog toga je potrebno u XML Schemi ovim elementima postaviti vrijednost atributa `minOccurs` na `1`.

Agrokorovi dokumenti još sadrže podatke o isporuci robe, koji po XML Schemi nisu obavezni, a navode se u zasebnim podelementima stavke narudžbe `POLINESCHD` (*Purchase Order Line Schedule*). Svaki element `POLINESCHD` obavezno sadrži vrijeme isporuke (`os:DATETIME`) i isporučenu količinu (`os:QUANTITY`), a može sadržavati i element opisa (`of:DESCRIPTN`).

Graf sheme prikazuje slika 8.1.



Slika 8.1. Graf sheme za studijski primjer narudžbe robe OAGIS 7.2.1. *Purchase Order*

Posebno je zanimljivo istaknuti da u Agrokorovim XML dokumentima-narudžbama nije sadržana cijena naručene robe (prema XML Schemi za svaki proizvod, stavku računa može se, ali i ne mora navesti cijena). Dok su narudžbe robe dostupne širokom krugu djelatnika poduzeća, iznos veleprodajnih nabavnih cijena poznat je užem krugu osoba, pa su one stoga pohranjene na posebnom mjestu (primjerice, u relacijskoj bazi podataka).

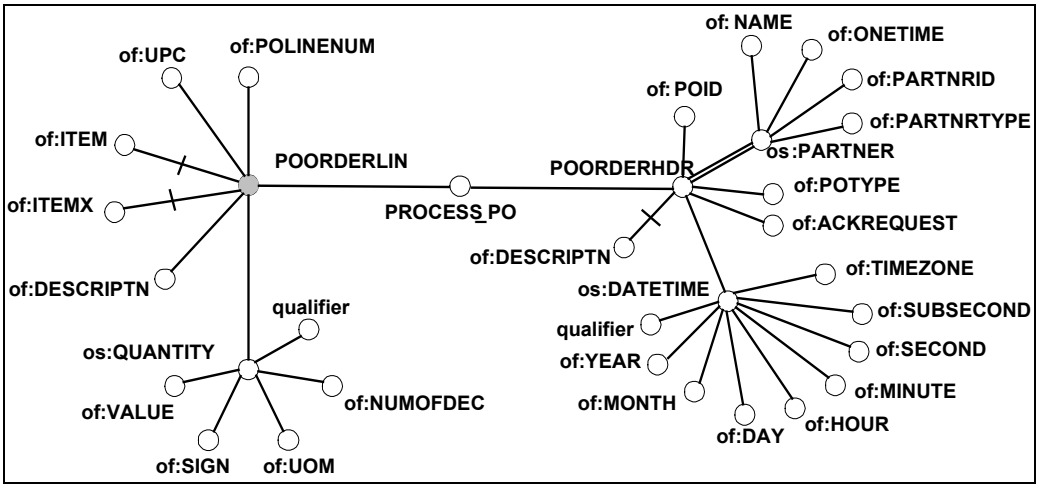
8.2. Konceptualno oblikovanje

8.2.1. Izrada grafa ovisnosti

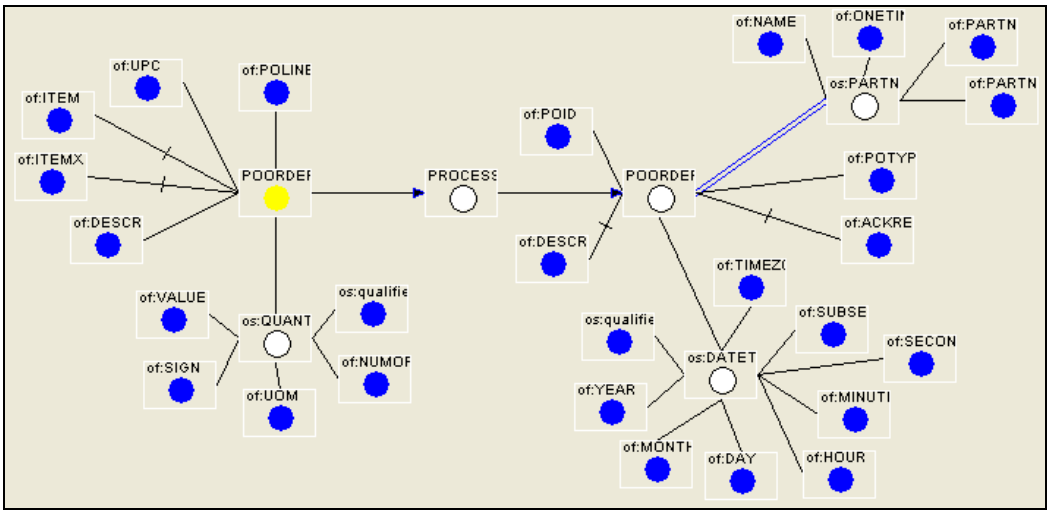
Dokument *OAGIS 7.2.1. Purchase Order* po strukturi je vrlo sličan ranije opisanom primjeru *sales data*. Analogno elementu `lineItem` u *sales data*, opet će za činjenicu biti izabrana stavka narudžbe, `POORDERLIN`. U graf ovisnosti kao potčvorovi korijenskog čvora `POORDERLIN` ulaze njegovi potčvorovi iz grafa sheme: `of:POLINENUM`, `os:QUANTITY` (s vlastitim potčvorovima), `of:UPC`, `of:ITEM`,

of:ITMX i of:DESCRIPTN. Nakon toga potrebno je odrediti kardinalnost veze od POORDERLIN prema PROCESS_PO. Stavka narudžbe je egzistencijalno slabi entitet, ovisan o entitetu narudžbe. Stavka narudžbe jednoznačno je određena kombinacijom broja narudžbe (of:POID) i rednim brojem stavke (of:POLINENUM). Analogno, u primjeru *sales data* za jednoznačnu identifikaciju stavke računa bilo je potrebno znati i broj računa, invoiceNum. Može se zaključiti da je kardinalnost veze od POORDERLIN prema PROCESS_PO prema-jedan. PROCESS_PO i njegov jedini podelement POORDERHDR ulaze u graf ovisnosti. Na POORDERHDR vežu se of:POID i of:POTYPE, os:DATETIME te of:DESCRIPTN. Veza od POORDERHDR prema os:PARTNER ima kardinalnost prema-više. Programski sustav ispituje kardinalnost veze u obrnutom smjeru. Za identifikator zaglavlja uzima se of:POID, a za indentifikator poslovnog partnera of:PARTNRID. Ispitivanjem se ustanovljuje da je riječ o vezi više-prema-više. Podaci o partneru od ključnog su značaja za poslovnu analizu pa su os:PARTNER i njegovi potčvorovi uključeni u graf ovisnosti.

Skica grafa ovisnosti dobivenog poluautomatskim procesom je na slici 8.2. Njegov prikaz u programskom sučelju dan je na slici 8.3.



Slika 8.2. Graf ovisnosti za OAGIS 7.2.1. *Purchase Order* oblikovan poluautomatskim postupkom



Slika 8.3. Graf ovisnosti za OAGIS 7.2.1. *Purchase Order* prikazan u programu

8.2.2. Preuređivanje grafa ovisnosti

Iz grafa ovisnosti uklanjaju se čvorovi PROCESS_PO i POORDERHDR, koji nemaju sadržaj. Na taj način potčvorovi od POORDERHDR postaju izravni potčvorovi činjenice.

Čvor os:DATETIME osnova je vremenske dimenzije. U skladištu podataka za narudžbe kao logičan izbor najveće zrnatosti nameće se dan. Zbog toga se uklanjaju oznake vremenske zone i vremenskih odsječaka kraćih od dana (sati, minute, sekunde i dijelovi sekunde). Kad bi se radilo o izdanom računu, vrijeme izdavanja u toku radnog dana bilo bi od izuzetne važnosti, no u slučaju narudžbe taj je podatak suvišan. Čvor os:DATETIME, koji je bez sadržaja, nadomješta se oznakom datuma, DATE (koji se lako računa iz kombinacije dana mjeseca i godine).

Čvor os:PARTNER nema sadržaj pa se prema načelu opisanom u poglavlju 6.9.4 izbacuje, a na njegovo mjesto dolazi of:PARTNRID. Već je istaknuto da se u svakom Agrokorovom dokumentu nalaze tri partnera. Za pohranu je zanimljiv dobavljač (Supplier) i Agrokorovo skladište u koje se isporučuje roba (ShipTo). Kupac (SoldTo) je uvijek Agrokor, pa se taj podatak, budući da uvijek poprima istu vrijednost, nije zanimljiv u procesu skladištenja. Za partnera se oblikuju dvije instance čvora os:PARTNER, Supplier i ShipTo, koje čine dijeljenu hijerarhiju. Sada čvor of:PARTNRTYPE gubi smisao pa se može ukloniti. Uklanja se i čvor of:ONETIME.

Količina naručene robe na specifičan je način opisana potčvorovima os:QUANTITY. Taj se čvor sa svim svojim potčvorovima uklanja i nadomješta jednim čvorom cjelobrojnog tipa, QUANTITYVALUE.

U Agrokorovim dokumentima identifikator naručenog proizvoda je of:UPC. Stoga preostali čvorovi koji opisuju proizvod (of:DESCRIPTN, of:ITEM, of:ITEMX) postaju potčvorovi of:UPC.

Iz grafa ovisnosti uklanja se of:POLINENUM, redni broj stavke računa koji ne može korisno poslužiti u analizi. Naprotiv, serijski broj narudžbe, of:POID je vrlo značajan za ograničavanje upita (u konačnici će predstavljati dijeljenu dimenziju).

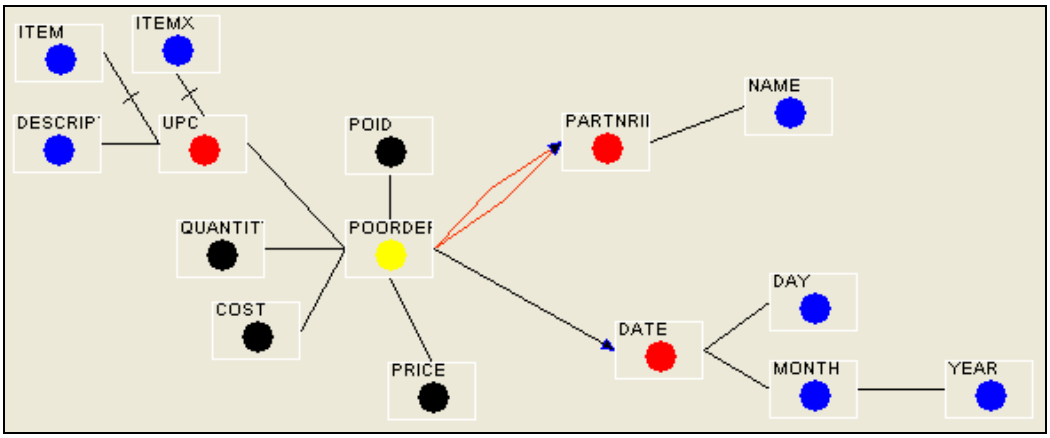
U graf ovisnosti dodaje se čvor PRICE, cijena, čija se vrijednost nije navedena u XML dokumentima, već će biti potrebno pročitati je iz Agrokorove baze podataka. Na ovom realnom primjeru pokazuje se kako skladište podataka može uspješno funkcionirati tek kad se integriraju podaci iz različitih izvora. Uz PRICE se dodaje i čvor COST (trošak kupnje, umnožak cijene i naručene količine).

Imena čvorova su kombinacija imena XML elementa ili atributa koje čvor predstavlja i njihovog imeničkog prefiksa (npr. os:PARTNER). U danom grafu ovisnosti moguće je, u svrhu jednostavnosti, ukloniti prefikse iz imena budući da to neće dovesti do podudaranja po imenu bilo kojih dvaju čvorova.

8.2.3. Određivanje dimenzija i mjera

Konačna konceptualna shema u grafičkom sučelju programa prikazana je na slici 8.4.

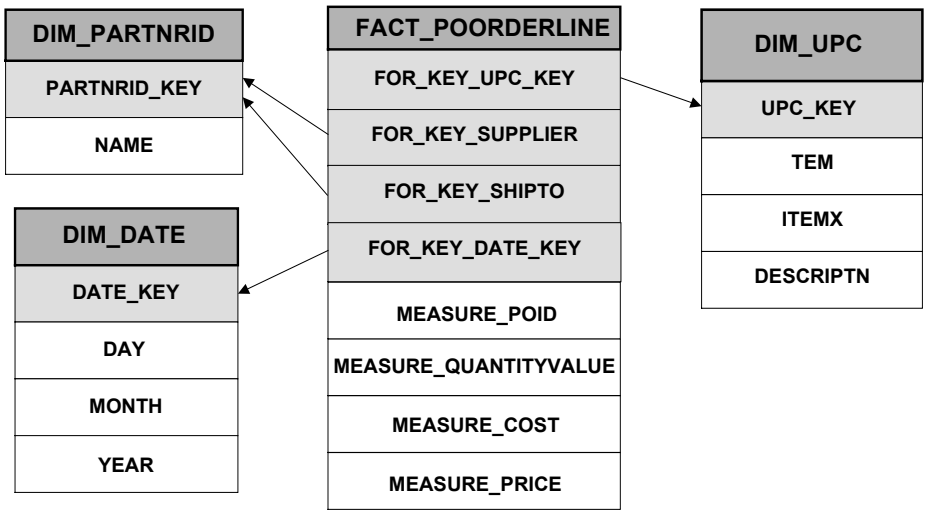
Uz vremensku dimenziju moguće je definirati još tri dimenzije: dobavljač (Supplier), primatelj robe (ShipTo) i proizvod (UPC). Mjere su cijena proizvoda (PRICE), naručena količina (QUANTITYVALUE), trošak (COST) i degenerirana dimenzija POID.



Slika 8.4. Konačna konceptualna shema za studijski primjer OAGIS 7.1.2. *Purchase Order*

8.3. Logičko oblikovanje

Logička shema za studijski primjer pokazana je na slici 8.5. Dimenzijske tablice su **partner** (DIM_ PARTNRID), **vrijeme** (DIM_DATE) i **proizvod** (DIM_UPC). Dijeljena hijerarhija definira jednu dimenzijsku tablicu, DIM_PARTNRID, s primarnim ključem `PARTNRID_KEY` na koji se, međutim, odnose dva strana ključa u činjeničnoj tablici FACT_POORDERLINE: `FOR_KEY_SUPPLIER` i `FOR_KEY_SHIPTO`. Prikaz činjenične tablice u programskom sustavu daje slika 8.6.



Slika 8.5. Logička shema za studijski primjer

FOR_KEY_UPC_KEY	VARCHAR2	NOT NULL
FOR_KEY_SUPPLIER	VARCHAR2	NOT NULL
FOR_KEY_SHIPTO	VARCHAR2	NOT NULL
FOR_KEY_DATE_KEY	DATE	NOT NULL
MEASURE_POID	VARCHAR2	NOT NULL
MEASURE_QUANTITYVALUE	NUMBER	NOT NULL
MEASURE_COST	VARCHAR2	NOT NULL
MEASURE_PRICE	VARCHAR2	NOT NULL

Slika 8.6. Programski prikaz činjenične tablice za studijski primjer

Zaključak

Uspješna primjena skladišta podataka u potpori poslovnom odlučivanju ovisi o relevantnosti i kvaliteti podataka sadržanih u skladištu. Uz transakcijske baze same organizacije, nezaobilazan izvor podataka za skladištenje u današnje vrijeme predstavljaju i polustrukturirani podaci dobiveni mrežnom razmjenom, podaci sadržani na *web*-stranicama te podaci dobiveni korištenjem *web*-usluga. Integracija heterogenih podatkovnih izvora u skladište podataka najkvalitetnije će se ostvariti ako se svaki segment skladišta podataka što izravnije oblikuje prema svom podatkovnom izvoru.

U radu je opisana metodologija integracije polustrukturiranih podataka zapisanih u formatu XML u skladište podataka. Skladište podataka oblikuje se na osnovi strukture i sadržaja samih izvornih XML dokumenata, polazeći od njihovog predloška, XML Scheme. Glavni zadatak bila je izrada programskog sustava koji će izvesti i verificirati iznesena načela oblikovanja. Programski sustav ima dvije verzije: klasičnu aplikaciju pisanu u jeziku Java, te *web*-aplikaciju s reduciranim skupom funkcionalnosti ostvarenu u tehnologiji *Java Applet*.

Konceptualno oblikovanje je poluautomatski proces koji se izvodi u interakciji programskog sustava i projektanta skladišta podataka. Oblikovanje se vrši na temelju analize XML Scheme, a po potrebi i samog sadržaja XML dokumenata.

Prikazani konceptualni model može se prilagoditi za izvedbu u različitim vrstama baza podataka. Logički model opisan u radu omogućuje izvedbu u relacijskoj bazi. Na temelju logičkog modela stvaraju se naredbe u jeziku SQL za izradu tablica, prilagođene konkretnom sustavu za upravljanja bazom podataka.

Programski sustav izrađen u sklopu magistarskog rada omogućuje projektantu skladišta da na brz i jednostavan način stvori konceptualni model skladišta, za koji se potom automatski oblikuje logička shema. Na početku se učitavaju XML Scheme i njima pripadajući XML dokumenti na temelju kojih se vrši oblikovanje. Konceptualno oblikovanje, ključni korak u stvaranju skladišta, program automatizira do najveće moguće razine. To omogućuje projektantu skladišta da se usredotoči na one dijelove oblikovanja koji se ne mogu izvesti automatski. Među njima se ističu odabir činjenice te izbor mjera i dimenzija. U tim se koracima projektantu skladišta preko grafičkog sučelja predočuju rezultati prethodnih, automatski izvedenih dijelova algoritma te opcije za nastavak automatskog postupka oblikovanja.

U radu iznesena načela oblikovanja u verificirana su na studijskom primjeru narudžbi koje koristi poduzeće Agrokor. Izrađeni programski sustav stvorenu konceptualnu shemu pretoči u spoj zvijezda i zapiše tablice u bazu podataka. Na taj je način pokazano da izrađeni tip programskog alata uspješno implementira ranije iznesena teorijska načela oblikovanja skladišta. Istodobno, dokazuje se da su navedena načela oblikovanja uistinu primjenjiva u praktičnom slučaju.

U procesu integracije polustrukturiranih podataka u skladišta podataka opširno su proučeni procesi konceptualnog i logičkog oblikovanja. Interes budućih istraživanja autor vidi u definiranju procesa izvlačenja, transformacije i učitavanja podataka za tablice skladišta podataka definiranih programskom aplikacijom. Nadogradnjom postojeće aplikacije omogućit će se popunjavanje tablica podacima iz XML dokumenata.

Literatura

[Abi97] S. Abiteboul. Querying Semi-Structured Data. *Proc. of the International Conference on Database Theory (ICDT)*, Delphi, Greece, 1997.

[Agr04] ***, B2B elektroničko poslovanje, suradnja Agrokor i FER, završni izvještaj, 2004.

[Ban03] M. Banek. *Aplikacija za konceptualno oblikovanje područnog skladišta podataka iz XML izvora*, diplomski rad, Fakultet elektrotehnike i računarstva, Zagreb, 2003

[Cod70] E.F. Codd. A Relational Model of Data for Large Data Banks. *Communications of the ACM, 13(6)*, pp. 377-387, 1970.

[DFS99] A. Deutsch, M. Fernandez, D. Suciu. Storing Semistructured Data with STORED. *Proc. of the ACM SIGMOD Conf. on Management of Data*, 1999.

[Dom98] Document object model (DOM) level 1 specification. *W3C Recommendation*, 1998. http://www.w3.org/TR/REC-DOM-Level-1

[Ebxml01] ebXML Technical Architecture Specification v1.0.4, *Final Draft*, 2001. http://www.w3.org/TR/REC-DOM-Level-1

[FK99] D. Florescu, D. Kossmann. Storing and Querying XML data using an rdbms. *IEEE Data Engineering Bulletin, 22(3)*, 1999.

[Fra] Fraunhofer IPSI-XQ: The XQuery Demonstrator. http://www.ipsi.fraunhofer.de/oasys/projects/ipsi-xq/overview_e.html

[GMR98] M. Golfarelli, D. Maio, S. Rizzi, The Dimensional Fact Model: a Conceptual Model for Data Warehouses, *International Journal of Cooperative Information Systems, vol. 7*, 1998.

[Gra99] Graph Theory, an Introduction, 1999. http://www.cs.usask.ca/resources/tutorials/csconcepts/1999_8/tutorial/index.html

[GR99] M. Golfarelli, S. Rizzi. Designing the data warehouse: key steps and crucial issues. *Journal of Computer Science and Information Management*, 2(3), 1999.

[GR02] M. Golfarelli, S. Rizzi. Data Warehouse - Teoria e pratica della progettazione. *McGraw-Hill*, 2002.

[GRV01] M. Golfarelli, S. Rizzi, B. Vrdoljak. Data warehouse design from XML sources. *Proc. of ACM 4th Int. Workshop on Data Warehousing and OLAP (DOLAP)*, Atlanta, USA, 2001.

[GRS02] M. M. Golfarelli, S. Rizzi, E. Saltarelli. WAND: A CASE Tool for Workload-Based Design of a Data Mart. *Proc. of Decimo Convegno Nazionale su Sistemi Evoluti Per Basi Di Dati*, Portoferraio, Italy, pp. 422-426, 2002.

[Html97] HyperText Markup Language 4.0 Specification. *W3C Recommendation*, 1997. http://www.w3.org/TR/WD-html40/

[Jdbc] JDBC Technology. http://java.sun.com /products/jdbc

[Jdom03] JDOM News and Status: JDOM Beta 9. 2003.
http://www.jdom.org /news/index.html

[Jdom04] The JDOM Project: JDOM 1.0. 2004.
http://www.jdom.org

[Inm96] W.H. Inmon. Building the Data Warehouse (Second Edition). *John Wiley & Sons*, 1996.

[Kim96] R. Kimball. The Data Warehouse Toolkit. *John Wiley & Sons*, 1996.

[Orj] Oracle JDBC
http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html

[Orx] Oracle XML Developer's Kit for Java (XDK)
http://www.oracle.com/technology/tech/xml/xdk/doc/production/java/index.html

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, J. Widom. Object exchange across heterogeneous information sources. *Proc. of Int. Conf. on Data Engineering (ICDE)*, Taipei, Taiwan, 1995.

[Sch0-04] XML Schema Part 0: Primer Second Edition. *W3C Recommendation*, 2001.
http://www.w3.org/TR/2004/REC-xmlschema-0-20041028

[Sch1-04] XML Schema Part 1: Structures Second Edition. *W3C Recommendation*, 2001.
http://www.w3.org/TR/2004/REC-xmlschema-1-20041028

[Sch2-04] XML Schema Part 2: Datatypes Second Edition. *W3C Recommendation*, 2001.
http://www.w3.org/TR/2004/REC-xmlschema-2-20041028

[SMV00] Z. Skočir, I. Matasić, B. Vrdoljak, *Organizacija obrade podataka*. Sveučilišna skripta, Fakultet elektrotehnike i računarstva, Zagreb, 2000.

[SX] SQL/XML.
http://www.sqlx.org/

[STH+99] J. Shanmugasundaram, K. Tufte, G. He, et al. Relational Databases for Querying XML Documents: Limitations and Opportunities. *Proc. of Very Large Data Bases Conference (VLDB)*, Edinburgh , 1999.

[VBR03a] B. Vrdoljak, M. Banek, S. Rizzi. Automating Conceptual Design of Web Warehouses. *Proc. of the 7th Int'l Conf. on Telecommunications (ConTEL)*, Zagreb, Croatia, 2003.

[VBR03b] B. Vrdoljak, M. Banek, S. Rizzi. Designing Web Warehouses from XML Schemas. *Proc. of Int'l Conf. on Data Warehousing and Knowledge Discovery (DaWaK 2003)*, Prague, Czech Republic. *Lecture notes in Computer Science (LNCS), Springer*, 2003.

[VBS04] B. Vrdoljak, M. Banek, Z. Skočir. A Methodology for Integrating XML Data into Data Warehouses. *Proc. of 27th International Convention MIPRO (MIPRO 2004)*, Opatija, Hrvatska, 2004

[Vrd99] B. Vrdoljak. *Skladište podataka kao potpora u procesu odlučivanja*, magistarski rad, Fakultet elektrotehnike i računarstva, Zagreb, 1999.

[Vrd04] B. Vrdoljak. *Integrating Semi-structured Data into Data Warehousing System*, doctoral thesis, Fakultet elektrotehnike i računarstva, Zagreb, 2004

[Xer] The Apache XML Project: Xerces2 Java Parser http://xml.apache.org/xerces2-j/

[Xml00] Extensible Markup Language (XML) 1.0 (Second Edition). *W3C Recommendation*, 2004 http://www.w3.org/TR/2000/REC-xml-20001006

[Xml04] Extensible Markup Language (XML) 1.0 (Third Edition). *W3C Recommendation*, 2004 http://www.w3.org/TR/2004/REC-xml-20040204/

[XPath99] XML Path Language (XPath). *W3C Recommendation*, 1999. http://www.w3.org/TR/1999/REC-xpath-19991116

[XQuery03] XQuery 1.0: An XML Query Language. *W3C Working Draft 22 August 2003*, 2003. http://www.w3.org/TR/2003/WD-xquery-20030822

[Xquc03] XML Query Use Cases. *W3C Working Draft 12 November 2003*, 2003. http://www.w3.org/TR/2003/WD-xquery-use-cases-20031112

[Xslt99] XSL Transformations (XSLT) Version 1.0. *W3C Recommendation*, 1999. http://www.w3.org/TR/1999/REC-xslt-19991116

Životopis

Rođen sam u Zagrebu 10. veljače 1980., gdje sam pohađao osnovnu školu i prirodoslovno-matematičku gimnaziju. Nakon završene gimnazije, 1998. upisujem Fakultet elektrotehnike i računarstva. Tijekom studija nagrađen sam godišnjim priznanjem "Josip Lončar" za svaku godinu studija. Diplomirao sam s naglaskom na znanstveno-istraživački rad 19. veljače 2003. U travnju 2003. zapošljavam se na Zavodu za osnove elektrotehnike i električka mjerenja Fakulteta elektrotehnike i računarstva kao znanstveni novak na projektu "Umrežena ekonomija" te upisujem poslijediplomski studij. Iste godine dodijeljena mi je brončana plaketa "Josip Lončar" kao najuspješnijem studentu u svojoj generaciji. Moj znanstveni interes obuhvaća skladištenje podataka, polustrukturirane podatke, s naglaskom na XML, te objektno-orijentirano programiranje, osobito u jeziku Java. Koautor sam većeg broja znanstvenih radova na polju skladištenja podataka i polustrukturiranih podataka objavljenih na međunarodnim konferencijama. Aktivno govorim i pišem engleskim i njemačkim jezikom, a služim se i francuskim. Član sam IEEE.

Sažetak

Skladište podataka je baza podataka u kojoj su integrirani podaci iz različitih heterogenih izvora u svrhu potpore procesima donošenja odluka. Polustrukturirani podaci, naročito XML, u današnje su vrijeme važan izvor podataka za skladište. U ovom je radu ranije definirana metodologija za integraciju podataka u XML-u u skladište izvedena u programskom alatu pisanom u jeziku Java. Oblikovanje skladišta podataka započinje od XML dokumenta i njihovih XML Schema. Postoje dvije verzije programa: aplikacija i pojednostavnjena *web*-aplikacija s ograničenim skupom funkcija. Dva su glavna koraka metodologije konceptualno i logičko oblikovanje. Konceptualno oblikovanje se izvodi poluautomatski, počevši od XML Scheme. U nekim se slučajevima mora ispitati sadržaj XML dokumenata koristeći jezik XML Query. Projektant skladišta interaktivno sudjeluje u dijelovima procesa koristeći grafičko sučelje alata. Za dani konceptualni model program automatski predlaže logičku izvedbu za relacijske baza podataka temeljenu na spoju zvijezda. Alat izrađuje i izvodi SQL naredbe za stvaranje tablica, primarnih te stranih ključeva u bazi podataka. Programski je alat uspješno ispitan na primjeru XML podataka za nabavku robe iz stvarnog života, verificirajući na taj način opisanu metodologiju.

Summary

Data warehouse is a database that integrates historical data from different heterogeneous sources in order to support decision-making processes. Semi-structured data, particularly XML, are nowadays a major data source for the warehouse. In this thesis a previously defined methodology for integrating XML data into data warehouses was implemented in a Java prototype tool. The warehouse design starts from the XML documents and their XML Schemas. There are two versions of the program: an application and a lightweight *web*-application with a reduced set of functions. The two main steps of the methodology are the conceptual and the logical design. The conceptual design is performed semi-automatically, starting from the XML Schema. In some cases the content of the XML documents must be examined using XML Query language. The designer participates interactively in the parts of the process that cannot be performed automatically by using the graphical interface of the tool. Given the conceptual model, the program automatically suggests a logical implementation for relational databases based on the star schema. The tool produces and executes SQL statements for creating tables, primary keys and foreign keys in a database. The program tool has been successfully tested on real-life XML data for purchasing goods, thus verifying the described methodology.

Ključne riječi

- skladištenje podataka
- oblikovanje skladišta podataka
- višedimenzijski model podataka
- polustrukturirani podaci
- XML
- Java
- *web*-aplikacija
- aplikacija za oblikovanje skladišta podataka

Keywords

- data warehousing
- data warehouse design
- multidimensional data model
- semi-structured data
- XML
- Java
- web-application
- application for data warehouse design

Dodatak: XML Scheme, XML dokumenti te kodovi u XML Queryju i SQL-u

A: Podaci za narudžbu robe (purchase order)

A1. XML Schema za narudžbu robe po.xsd iz specifikacije XML Schema Part 0: Primer

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
  <xsd:documentation xml:lang="en">
   Purchase order schema for Example.com.
   Copyright 2000 Example.com. All rights reserved.
  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

 <xsd:element name="comment" type="xsd:string"/>

 <xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
   <xsd:element name="shipTo" type="USAddress"/>
   <xsd:element name="billTo" type="USAddress"/>
   <xsd:element ref="comment" minOccurs="0"/>
   <xsd:element name="items"  type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
 </xsd:complexType>

 <xsd:complexType name="USAddress">
  <xsd:sequence>
   <xsd:element name="name"   type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city"   type="xsd:string"/>
   <xsd:element name="state"  type="xsd:string"/>
   <xsd:element name="zip"    type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN"
     fixed="US"/>
 </xsd:complexType>

 <xsd:complexType name="Items">
  <xsd:sequence>
   <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="quantity">
       <xsd:simpleType>
        <xsd:restriction base="xsd:positiveInteger">
         <xsd:maxExclusive value="100"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:element>
      <xsd:element name="USPrice"  type="xsd:decimal"/>
      <xsd:element ref="comment"   minOccurs="0"/>
```

```
      <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
     </xsd:sequence>
     <xsd:attribute name="partNum" type="SKU" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <!-- Stock Keeping Unit, a code for identifying products -->
 <xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
 </xsd:simpleType>

</xsd:schema>
```

A2. XML dokument za narudžbu robe po.xml iz specifikacije XML Schema Part 0: Primer

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>PA</state>
        <zip>95819</zip>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <items>
        <item partNum="872-AA">
            <productName>Lawnmower</productName>
            <quantity>1</quantity>
            <USPrice>148.95</USPrice>
            <comment>Confirm this is electric</comment>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <USPrice>39.98</USPrice>
            <shipDate>1999-05-21</shipDate>
        </item>
    </items>
</purchaseOrder>
```

A3. XML dokument za narudžbu robe po2.xml

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-09-20">
    <shipTo country="US">
        <name>Patrick Cunningham</name>
        <street>34 Washington Street</street>
        <city>Atlanta</city>
        <state>GA</state>
```

```
        <zip>33255</zip>
    </shipTo>
    <billTo country="US">
        <name>Stephen Callaghan</name>
        <street>4 Paris Road</street>
        <city>Philadelphia</city>
        <state>PA</state>
        <zip>65819</zip>
    </billTo>
    <items>
        <item partNum="338-AA">
            <productName>Wardrobe</productName>
            <quantity>2</quantity>
            <USPrice>287.95</USPrice>
        </item>
        <item partNum="446-AA">
            <productName>Television Set</productName>
            <quantity>1</quantity>
            <USPrice>103.67</USPrice>
            <shipDate>1999-05-21</shipDate>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <USPrice>39.98</USPrice>
        </item>
    </items>
</purchaseOrder>
```

A4. XML dokument za narudžbu robe po3.xml

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <billTo country="US">
        <name>Alice Smith</name>
        <street>8 Oak Avenue</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </billTo>
    <items>
        <item partNum="872-AA">
            <productName>Lawnmower</productName>
            <quantity>1</quantity>
            <USPrice>148.95</USPrice>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <USPrice>39.98</USPrice>
            <shipDate>1999-05-21</shipDate>
        </item>
    </items>
</purchaseOrder>
```

B: Podaci o prodaji (sales data)

B1. XML Schema za prodaju u trgovini salesdata.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <xsd:element name="salesData">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="invoice" type="InvoiceType"
                                    maxOccurs="unbounded"/>
            <xsd:element name="customer" type="CustomerType"
                                       maxOccurs="unbounded"/>
            <xsd:element name="product" type="ProductType"
                                       maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:key name="prodPKey">
         <xsd:selector xpath="product"/>
         <xsd:field xpath="@prodID"/>
      </xsd:key>
      <xsd:keyref name="prodFKey" refer="prodPKey">
         <xsd:selector xpath="invoice/lineItem"/>
         <xsd:field xpath="productRef"/>
      </xsd:keyref>

      <xsd:key name="custPKey">
         <xsd:selector xpath=".//*"/>
         <xsd:field xpath="@customerID"/>
      </xsd:key>
      <xsd:keyref name="custFKey" refer="custPKey">
         <xsd:selector xpath=".//*"/>
         <xsd:field xpath="customerRef"/>
      </xsd:keyref>

   </xsd:element>

   <xsd:complexType name="InvoiceType">
      <xsd:sequence>
         <xsd:element name="orderDate" type="xsd:date"/>
         <xsd:element name="shipDate" type="xsd:date"/>
         <xsd:element name="shipMethod" type="xsd:string"/>
          <xsd:element name="lineItem" type="LineItemType"
                        maxOccurs="unbounded"/>
         <xsd:element name="customerRef"
                                  type="xsd:positiveInteger"/>
      </xsd:sequence>
      <xsd:attribute name="invoiceNum"
                  type="xsd:positiveInteger" use="required"/>
   </xsd:complexType>


   <xsd:complexType name="CustomerType">
      <xsd:sequence>
         <xsd:element name="name" type="xsd:string"/>
         <xsd:element name="address" type="xsd:string"/>
         <xsd:element name="city" type="xsd:string"/>
         <xsd:element name="postCode" type="xsd:decimal"/>
      </xsd:sequence>
      <xsd:attribute name="customerID"
```

```
                  type="xsd:positiveInteger" use="required"/>
   </xsd:complexType>

   <xsd:complexType name="LineItemType">
      <xsd:sequence>
         <xsd:element name="quantity"
                                 type="xsd:positiveInteger"/>
         <xsd:element name="price" type="xsd:decimal"/>
         <xsd:element name="productRef" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="ProductType">
      <xsd:sequence>
         <xsd:element name="prodName" type="xsd:string"/>
         <xsd:element name="size" type="xsd:string"
                                              minOccurs="0"/>
         <xsd:element name="color" type="xsd:string"
                                                minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="prodID" type="xsd:string"
                                             use="required"/>
   </xsd:complexType>

</xsd:schema>
```

B2. XML dokument za prodaju u trgovini sale1.xml

```
<?xml version="1.0"?>
<salesData>
    <invoice invoiceNum="123456">
        <orderDate>2002-10-20</orderDate>
        <shipDate>2002-11-03</shipDate>
        <shipMethod>car delivery</shipMethod>
        <customerRef>56789</customerRef>
        <lineItem>
            <productRef>456-HW</productRef>
            <quantity>2</quantity>
            <price>37.92</price>
        </lineItem>
        <lineItem>
            <productRef>987-GR</productRef>
            <quantity>2</quantity>
            <price>7.34</price>
        </lineItem>
    </invoice>
    <invoice invoiceNum="234567">
        <orderDate>2003-03-08</orderDate>
        <shipDate>2003-04-01</shipDate>
        <shipMethod>air mail</shipMethod>
        <customerRef>98765</customerRef>
        <lineItem>
            <productRef>234-RT</productRef>
            <quantity>1</quantity>
            <price>36.87</price>
        </lineItem>
        <lineItem>
            <productRef>456-HW</productRef>
            <quantity>2</quantity>
            <price>37.92</price>
        </lineItem>
```

```
    </invoice>
    <customer customerID="56789">
        <name>Sarah Dickinson</name>
        <address>123 Maple Road</address>
        <city>Leicester</city>
        <state>UK</state>
        <postCode>SR763</postCode>
    </customer>
    <customer customerID="98765">
        <name>Juergen Wissermann</name>
        <address></address>
        <city>Freising</city>
        <state>Deutschland</state>
        <postCode>85150</postCode>
    </customer>
    <product prodID="234-RT">
        <prodName>Data on the Web</prodName>
        <size>34cm*20cm</size>
    </product>
    <product prodID="456-HW">
        <prodName>T-Shirt</prodName>
        <size>L</size>
        <color>yellow</color>
    </product>
    <product prodID="987-GR">
        <prodName>Philips headphones</prodName>
        <size>standard american</size>
    </product>
</salesData>
```

C: Podaci o prodaji knjiga (booksale)

C1. XML Schema za prodaju knjiga booksale.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="bookSale">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="book" type="typeBook"/>
         <xsd:element name="store" type="typeStore"/>
      </xsd:sequence>
      <xsd:attribute name="quantity" type="xsd:positiveInteger"
                                                 use="required"/>
      <xsd:attribute name="date" type="xsd:date" use="required"/>
</xsd:complexType>
</xsd:element>

<xsd:complexType name="typeBook">
    <xsd:sequence>
       <xsd:element name="title" type="xsd:string"/>
       <xsd:element name="author" maxOccurs="unbounded">
           <xsd:complexType>
               <xsd:sequence>
                   <xsd:element name="lastName" type="xsd:string"/>
                   <xsd:element name="firstName" type="xsd:string"/>
               </xsd:sequence>
           </xsd:complexType>
       </xsd:element>
       <xsd:element name="price" type="xsd:decimal"/>
```

```
        <xsd:element name="publisher" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="year" type="xsd:gYear" use="required"/>
</xsd:complexType>

<xsd:complexType name="typeStore">
   <xsd:sequence>
      <xsd:element name="address" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
   </xsd:sequence>
   <xsd:attribute name="storeNo" type="xsd:positiveInteger"
                                                    use="required"/>
</xsd:complexType>

</xsd:schema>
```

C2. XML dokument za prodaju knjiga booksale1.xml

```
<bookSale date="2002-06-15" quantity="2">
    <book year="1999">
        <title>Zlatno tele</title>
        <author>
            <lastName>Iljf</lastName>
            <firstName>Ilja A.</firstName>
        </author>
        <author>
            <lastName>Petrov</lastName>
            <firstName>Jevgenij P.</firstName>
        </author>
        <price>78.23</price>
        <publisher>Skolska knjiga Zagreb</publisher>
    </book>
    <store storeNo="4">
        <address>Masarykova 24</address>
        <city>Zagreb</city>
    </store>
</bookSale>
```

C3. XML dokument za prodaju knjiga booksale2.xml

```
<bookSale date="2002-09-10" quantity="1">
    <book year="2001">
        <title>Idiot</title>
        <author>
            <lastName>Dostojevski</lastName>
            <firstName>Fjodor M.</firstName>
        </author>
        <price>132.90</price>
        <publisher>Skolska knjiga Zagreb</publisher>
    </book>
    <store storeNo="10">
        <address>Trg slobode 7</address>
        <city>Varazdin</city>
    </store>
</bookSale>
C4. XML dokument za prodaju knjiga booksale3.xml
<bookSale date="2002-12-19" quantity="1">
    <book year="2001">
        <title>Braca Karamazovi</title>
        <author>
            <lastName>Dostojevski</lastName>
            <firstName>Fjodor M.</firstName>
```

```
        </author>
        <price>205.56</price>
        <publisher>Otokar Kersovani Rijeka</publisher>
    </book>
    <store storeNo="13">
        <address>Gajeva 1</address>
        <city>Zagreb</city>
    </store>
</bookSale>
```

C5. XML dokument za prodaju knjiga booksale4.xml

```
<bookSale date="2002-12-19" quantity="1">
    <book year="2003">
        <title>Braca Karamazovi</title>
        <author>
            <lastName>Dostojevski</lastName>
            <firstName>Fjodor M.</firstName>
        </author>
       <price>205.56</price>
       <publisher>AGM Zagreb</publisher>
   </book>
   <store storeNo="13">
       <address>Gajeva 1</address>
       <city>Zagreb</city>
   </store>
</bookSale>
```

D: Upit u jeziku XQuery za ispitivanje veze više-prema-više na primjeru prodaje knjiga

D1. Skupni podaci o prodaji knjiga

```
<?xml version="1.0" encoding="UTF-8"?>
<XWPRootElement>

<bookSale date="2002-06-15" quantity="2">
    <book year="1999">
        <title>Zlatno tele</title>
        <author>
            <lastName>Iljf</lastName>
            <firstName>Ilja A.</firstName>
        </author>
        <author>
            <lastName>Petrov</lastName>
            <firstName>Jevgenij P.</firstName>
        </author>
        <price>78.23</price>
        <publisher>Skolska knjiga Zagreb</publisher>
    </book>
    <store storeNo="4">
        <address>Masarykova 24</address>
        <city>Zagreb</city>
    </store>
</bookSale>

<bookSale date="2002-09-10" quantity="1">
    <book year="2001">
       <title>Idiot</title>
       <author>
           <lastName>Dostojevski</lastName>
```

```
            <firstName>Fjodor M.</firstName>
        </author>
        <price>132.90</price>
        <publisher>Skolska knjiga Zagreb</publisher>
    </book>
    <store storeNo="10">
        <address>Trg slobode 7</address>
        <city>Varazdin</city>
    </store>
</bookSale>

<bookSale date="2002-12-19" quantity="1">
    <book year="2001">
        <title>Braca Karamazovi</title>
        <author>
            <lastName>Dostojevski</lastName>
            <firstName>Fjodor M.</firstName>
        </author>
        <price>205.56</price>
        <publisher>Otokar Kersovani Rijeka</publisher>
    </book>
   <store storeNo="13">
        <address>Gajeva 1</address>
        <city>Zagreb</city>
   </store>
</bookSale>
<bookSale date="2002-12-19" quantity="1">
   <book year="2003">
        <title>Braca Karamazovi</title>
        <author>
            <lastName>Dostojevski</lastName>
            <firstName>Fjodor M.</firstName>
        </author>
        <price>205.56</price>
        <publisher>AGM Zagreb</publisher>
   </book>
   <store storeNo="13">
      <address>Gajeva 1</address>
      <city>Zagreb</city>
   </store>
</bookSale>

</XWPRootElement>
```

D2. Upit za ispitivanje kardinalnosti veze author→ book

```
<results> {

max(
let $d:=doc("tempdata.xml")

let $retValue:= for $par in distinct-
                          values($d/XWPRootElement/bookSale/book),
                    $c in distinct-values($par/author)

return <expression>
          <parent> { $par/title} </parent>
          <child>  { $c/lastName}  { $c/firstName}  </child>
       </expression>

for $c in distinct-values($retValue/child)
```

```
      let $p:=for $exp in $retValue where deep-equal($exp/child,$c)
      return $exp/parent

return count(distinct-values($p))
)

} </results>
```

D3. Rezultat upita za ispitivanje kardinalnosti veze author→ book

```
<results>2</results>
```

E: Kodovi u jeziku SQL za stvaranje dimenzijskih i činjeničnih tablica

E1. Kod za stvaranje tablica za podatke o trgovini (*sales data*)

```
CREATE TABLE "DIM_PRODUCTREF"(
"PRODUCTREF_KEY" NUMBER(10) NOT NULL,
"PRODNAME" VARCHAR2(30) NOT NULL,
"SIZE" VARCHAR2(30),
"COLOR" VARCHAR2(30));

ALTER TABLE "DIM_PRODUCTREF" ADD CONSTRAINT
"PRIMARY_KEY_DIM_PRODUCTREF"
PRIMARY KEY("PRODUCTREF_KEY");

CREATE DIMENSION "DIMENSION_DIM_PRODUCTREF"
LEVEL "H1_L1_DIM_PRODUCTREF" IS "DIM_PRODUCTREF"."PRODUCTREF_KEY"
HIERARCHY H1_DIM_PRODUCTREF (
"H1_L1_DIM_PRODUCTREF")
ATTRIBUTE "H1_L1_DIM_PRODUCTREF" DETERMINES ("PRODNAME", "SIZE",
"COLOR");


CREATE TABLE "DIM_TIMEKEY"(
"TIME_KEY" NUMBER(10) NOT NULL,
"MONTH" NUMBER(10) NOT NULL,
"QUARTER" NUMBER(10) NOT NULL,
"YEAR" NUMBER(20) NOT NULL,
"MONTHNAME" VARCHAR2(30) NOT NULL,
"ORDERDATE" DATE NOT NULL,
"DAYOFWEEK" VARCHAR2(30) NOT NULL);

ALTER TABLE "DIM_TIMEKEY" ADD CONSTRAINT "PRIMARY_KEY_DIM_TIMEKEY"
PRIMARY KEY("TIME_KEY");

CREATE DIMENSION "DIMENSION_DIM_TIMEKEY"
LEVEL "H1_L1_DIM_TIMEKEY" IS "DIM_TIMEKEY"."TIME_KEY"
LEVEL "H1_L2_DIM_TIMEKEY" IS "DIM_TIMEKEY"."MONTH"
LEVEL "H1_L3_DIM_TIMEKEY" IS "DIM_TIMEKEY"."QUARTER"
LEVEL "H1_L4_DIM_TIMEKEY" IS "DIM_TIMEKEY"."YEAR"
HIERARCHY H1_DIM_TIMEKEY (
"H1_L1_DIM_TIMEKEY" CHILD OF "H1_L2_DIM_TIMEKEY" CHILD OF
"H1_L3_DIM_TIMEKEY" CHILD OF "H1_L4_DIM_TIMEKEY")
ATTRIBUTE "H1_L1_DIM_TIMEKEY" DETERMINES ("ORDERDATE", "DAYOFWEEK")
ATTRIBUTE "H1_L2_DIM_TIMEKEY" DETERMINES ("MONTHNAME");


CREATE TABLE "DIM_SHIPKEY"(
```

```
"SHIP_KEY" NUMBER(10) NOT NULL,
"SHIPMETHOD" VARCHAR2(30) NOT NULL,
"SHIPMETHODDESC" VARCHAR2(30) NOT NULL);

ALTER TABLE "DIM_SHIPKEY" ADD CONSTRAINT "PRIMARY_KEY_DIM_SHIPKEY"
PRIMARY KEY("SHIP_KEY");


CREATE DIMENSION "DIMENSION_DIM_SHIPKEY"
LEVEL "H1_L1_DIM_SHIPKEY" IS "DIM_SHIPKEY"."SHIP_KEY"
HIERARCHY H1_DIM_SHIPKEY (
"H1_L1_DIM_SHIPKEY")
ATTRIBUTE "H1_L1_DIM_SHIPKEY" DETERMINES ("SHIPMETHOD",
"SHIPMETHODDESC");


CREATE TABLE "DIM_CUSTOMERKEY"(
"CUSTOMER_KEY" NUMBER(10) NOT NULL,
"NAME" VARCHAR2(30) NOT NULL,
"ADDRESS" VARCHAR2(30) NOT NULL,
"POSTCODE" NUMBER(20, 2) NOT NULL,
"CITY" VARCHAR2(30) NOT NULL,
"STATE" VARCHAR2(30) NOT NULL);

ALTER TABLE "DIM_CUSTOMERKEY" ADD CONSTRAINT
"PRIMARY_KEY_DIM_CUSTOMERKEY"
PRIMARY KEY("CUSTOMER_KEY");

CREATE DIMENSION "DIMENSION_DIM_CUSTOMERKEY"
LEVEL "H1_L1_DIM_CUSTOMERKEY" IS "DIM_CUSTOMERKEY"."CUSTOMER_KEY"
LEVEL "H1_L2_DIM_CUSTOMERKEY" IS "DIM_CUSTOMERKEY"."POSTCODE"
LEVEL "H1_L3_DIM_CUSTOMERKEY" IS "DIM_CUSTOMERKEY"."CITY"
HIERARCHY H1_DIM_CUSTOMERKEY (
"H1_L1_DIM_CUSTOMERKEY" CHILD OF "H1_L2_DIM_CUSTOMERKEY" CHILD OF
"H1_L3_DIM_CUSTOMERKEY")
ATTRIBUTE "H1_L1_DIM_CUSTOMERKEY" DETERMINES ("NAME", "ADDRESS")
ATTRIBUTE "H1_L2_DIM_CUSTOMERKEY" DETERMINES ("STATE");


CREATE TABLE "FACT_LINEITEM"(
"FOR_KEY_PRODUCTREF_KEY" NUMBER(10) NOT NULL,
"FOR_KEY_TIME_KEY" NUMBER(10) NOT NULL,
"FOR_KEY_SHIP_KEY" NUMBER(10) NOT NULL,
"FOR_KEY_CUSTOMER_KEY" NUMBER(10) NOT NULL,
"MEASURE_INVOICENUM" NUMBER(20) NOT NULL,
"MEASURE_QUANTITY" NUMBER(20) NOT NULL,
"MEASURE_PRICE" NUMBER(20, 2) NOT NULL,
"MEASURE_INCOME" NUMBER(20, 2) NOT NULL);

ALTER TABLE "FACT_LINEITEM" ADD CONSTRAINT "FK_DIM_PRODUCTREF"
FOREIGN KEY ("FOR_KEY_PRODUCTREF_KEY") REFERENCES
"DIM_PRODUCTREF"("PRODUCTREF_KEY");

ALTER TABLE "FACT_LINEITEM" ADD CONSTRAINT "FK_DIM_TIMEKEY" FOREIGN
KEY ("FOR_KEY_TIME_KEY") REFERENCES "DIM_TIMEKEY"("TIME_KEY");

ALTER TABLE "FACT_LINEITEM" ADD CONSTRAINT "FK_DIM_SHIPKEY" FOREIGN
KEY ("FOR_KEY_SHIP_KEY") REFERENCES "DIM_SHIPKEY"("SHIP_KEY");
```

```
ALTER TABLE "FACT_LINEITEM" ADD CONSTRAINT "FK_DIM_CUSTOMERKEY"
FOREIGN KEY ("FOR_KEY_CUSTOMER_KEY") REFERENCES
"DIM_CUSTOMERKEY"("CUSTOMER_KEY");
```

E2. Kod za stvaranje tablica za narudžbu robe (*purchase order*)

```
CREATE TABLE "DIM_ORDERDATE"(
"ORDERDATE_KEY" DATE NOT NULL,
"MONTH" VARCHAR2(30) NOT NULL,
"YEAR" VARCHAR2(30) NOT NULL,
"NOOFWORKINGDAYS" VARCHAR2(30) NOT NULL,
"DAYOFWEEK" VARCHAR2(30) NOT NULL);

ALTER TABLE "DIM_ORDERDATE" ADD CONSTRAINT
"PRIMARY_KEY_DIM_ORDERDATE"
PRIMARY KEY("ORDERDATE_KEY");

CREATE DIMENSION "DIMENSION_DIM_ORDERDATE"
LEVEL "H1_L1_DIM_ORDERDATE" IS "DIM_ORDERDATE"."ORDERDATE_KEY"
LEVEL "H1_L2_DIM_ORDERDATE" IS "DIM_ORDERDATE"."MONTH"
LEVEL "H1_L3_DIM_ORDERDATE" IS "DIM_ORDERDATE"."YEAR"
HIERARCHY H1_DIM_ORDERDATE (
"H1_L1_DIM_ORDERDATE" CHILD OF "H1_L2_DIM_ORDERDATE" CHILD OF
"H1_L3_DIM_ORDERDATE")
ATTRIBUTE "H1_L1_DIM_ORDERDATE" DETERMINES ("DAYOFWEEK")
ATTRIBUTE "H1_L2_DIM_ORDERDATE" DETERMINES ("NOOFWORKINGDAYS");

CREATE TABLE "DIM_ADDRESSIDTYPE"(
"ADDRESSIDTYPE_KEY" NUMBER(10) NOT NULL,
"ZIP" NUMBER(20, 2) NOT NULL,
"STATE" VARCHAR2(30) NOT NULL,
"COUNTRY" VARCHAR2(30) NOT NULL,
"CITY" VARCHAR2(30) NOT NULL,
"NAME" VARCHAR2(30) NOT NULL,
"STREET" VARCHAR2(30) NOT NULL);

ALTER TABLE "DIM_ADDRESSIDTYPE" ADD CONSTRAINT
"PRIMARY_KEY_DIM_ADDRESSIDTYPE"
PRIMARY KEY("ADDRESSIDTYPE_KEY")

CREATE DIMENSION "DIMENSION_DIM_ADDRESSIDTYPE"
LEVEL "H1_L1_DIM_ADDRESSIDTYPE" IS
"DIM_ADDRESSIDTYPE"."ADDRESSIDTYPE_KEY"
LEVEL "H1_L2_DIM_ADDRESSIDTYPE" IS "DIM_ADDRESSIDTYPE"."ZIP"
LEVEL "H1_L3_DIM_ADDRESSIDTYPE" IS "DIM_ADDRESSIDTYPE"."STATE"
LEVEL "H1_L4_DIM_ADDRESSIDTYPE" IS "DIM_ADDRESSIDTYPE"."COUNTRY"
HIERARCHY H1_DIM_ADDRESSIDTYPE (
"H1_L1_DIM_ADDRESSIDTYPE" CHILD OF "H1_L2_DIM_ADDRESSIDTYPE" CHILD OF
"H1_L3_DIM_ADDRESSIDTYPE" CHILD OF "H1_L4_DIM_ADDRESSIDTYPE")
ATTRIBUTE "H1_L1_DIM_ADDRESSIDTYPE" DETERMINES ("NAME", "STREET")
ATTRIBUTE "H1_L2_DIM_ADDRESSIDTYPE" DETERMINES ("CITY");


CREATE TABLE "DIM_PARTNUM"(
"PARTNUM_KEY" VARCHAR2(30) NOT NULL,
"PRODUCTNAME" VARCHAR2(30) NOT NULL);

ALTER TABLE "DIM_PARTNUM" ADD CONSTRAINT "PRIMARY_KEY_DIM_PARTNUM"
PRIMARY KEY("PARTNUM_KEY");

CREATE DIMENSION "DIMENSION_DIM_PARTNUM"
```

```
LEVEL "H1_L1_DIM_PARTNUM" IS "DIM_PARTNUM"."PARTNUM_KEY"
HIERARCHY H1_DIM_PARTNUM (
"H1_L1_DIM_PARTNUM")
ATTRIBUTE "H1_L1_DIM_PARTNUM" DETERMINES ("PRODUCTNAME");

CREATE TABLE "FACT_SALE"(
"FOR_KEY_ORDERDATE_KEY" DATE NOT NULL,
"FOR_KEY_SHIPTO" NUMBER(10) NOT NULL,
"FOR_KEY_BILLTO" NUMBER(10) NOT NULL,
"FOR_KEY_PARTNUM_KEY" VARCHAR2(30) NOT NULL,
"MEASURE_QUANTITY" NUMBER(20) NOT NULL,
"MEASURE_USPRICE" NUMBER(20, 2) NOT NULL,
"MEASURE_INCOME" NUMBER(20, 2) NOT NULL);

ALTER TABLE "FACT_SALE" ADD CONSTRAINT "FK_DIM_ORDERDATE" FOREIGN KEY
("FOR_KEY_ORDERDATE_KEY") REFERENCES
"DIM_ORDERDATE"("ORDERDATE_KEY");

ALTER TABLE "FACT_SALE" ADD CONSTRAINT "FK_DIM_ADDRESSIDTYPE_SHIPTO"
FOREIGN KEY ("FOR_KEY_SHIPTO") REFERENCES
"DIM_ADDRESSIDTYPE"("ADDRESSIDTYPE_KEY");

ALTER TABLE "FACT_SALE" ADD CONSTRAINT "FK_DIM_ADDRESSIDTYPE_BILLTO"
FOREIGN KEY ("FOR_KEY_BILLTO") REFERENCES
"DIM_ADDRESSIDTYPE"("ADDRESSIDTYPE_KEY");

ALTER TABLE "FACT_SALE" ADD CONSTRAINT "FK_DIM_PARTNUM" FOREIGN KEY
("FOR_KEY_PARTNUM_KEY") REFERENCES "DIM_PARTNUM"("PARTNUM_KEY");
```

E3. Kod za stvaranje tablica za prodaju knjiga (*book sale*)

```
CREATE TABLE "DIM_DATE"(
"DATE_KEY" DATE NOT NULL,
"MONTH" NUMBER(20) NOT NULL,
"YEAR" NUMBER(20) NOT NULL,
"NOOFWORKINGDAYS" NUMBER(20) NOT NULL,
"DAYOFWEEK" VARCHAR2(4) NOT NULL);

ALTER TABLE "DIM_DATE" ADD CONSTRAINT "PRIMARY_KEY_DIM_DATE"
PRIMARY KEY("DATE_KEY");

CREATE DIMENSION "DIMENSION_DIM_DATE"
LEVEL "H1_L1_DIM_DATE" IS "DIM_DATE"."DATE_KEY"
LEVEL "H1_L2_DIM_DATE" IS "DIM_DATE"."MONTH"
LEVEL "H1_L3_DIM_DATE" IS "DIM_DATE"."YEAR"
HIERARCHY H1_DIM_DATE (
"H1_L1_DIM_DATE" CHILD OF "H1_L2_DIM_DATE" CHILD OF "H1_L3_DIM_DATE")
ATTRIBUTE "H1_L1_DIM_DATE" DETERMINES ("DAYOFWEEK")
ATTRIBUTE "H1_L2_DIM_DATE" DETERMINES ("NOOFWORKINGDAYS");

CREATE TABLE "DIM_STORENO"(
"STORENO_KEY" NUMBER(10) NOT NULL,
"ADDRESS" VARCHAR2(4) NOT NULL,
"CITY" VARCHAR2(4) NOT NULL);

ALTER TABLE "DIM_STORENO" ADD CONSTRAINT "PRIMARY_KEY_DIM_STORENO"
PRIMARY KEY("STORENO_KEY");

CREATE DIMENSION "DIMENSION_DIM_STORENO"
LEVEL "H1_L1_DIM_STORENO" IS "DIM_STORENO"."STORENO_KEY"
ATTRIBUTE "H1_L1_DIM_STORENO" DETERMINES ("ADDRESS", "CITY");
```

```
CREATE TABLE "DIM_BOOKID"(
"BOOKID_KEY" NUMBER(10) NOT NULL,
"AUTHORID" VARCHAR2(4) NOT NULL,
"YEAR" VARCHAR2(4) NOT NULL,
"TITLE" VARCHAR2(4) NOT NULL,
"PUBLISHER" VARCHAR2(4) NOT NULL);

ALTER TABLE "DIM_BOOKID" ADD CONSTRAINT "PRIMARY_KEY_DIM_BOOKID"
PRIMARY KEY("BOOKID_KEY");

CREATE DIMENSION "DIMENSION_DIM_BOOKID"
LEVEL "H1_L1_DIM_BOOKID" IS "DIM_BOOKID"."BOOKID_KEY"
ATTRIBUTE "H1_L1_DIM_BOOKID" DETERMINES ("AUTHORID", "YEAR", "TITLE",
"PUBLISHER");

CREATE TABLE "MANY_AUTHORID"(
"AUTHORID_KEY" NUMBER(10) NOT NULL,
"LASTNAME" VARCHAR2(4) NOT NULL,
"FIRSTNAME" VARCHAR2(4) NOT NULL);

ALTER TABLE "MANY_AUTHORID" ADD CONSTRAINT
"PRIMARY_KEY_MANY_AUTHORID" PRIMARY KEY("AUTHORID_KEY");

CREATE DIMENSION "DIMENSION_MANY_AUTHORID"
LEVEL "H1_L1_DIM_AUTHORID" IS "MANY_AUTHORID"."AUTHORID_KEY"
ATTRIBUTE "H1_L1_DIM_AUTHORID" DETERMINES ("LASTNAME", "FIRSTNAME")

CREATE TABLE "FACT_BOOKSALE"(
"FOR_KEY_DATE_KEY" DATE NOT NULL,
"FOR_KEY_STORENO_KEY" NUMBER(10) NOT NULL,
"FOR_KEY_BOOKID_KEY" NUMBER(10) NOT NULL,
"MEASURE_QUANTITY" NUMBER(20) NOT NULL,
"MEASURE_PRICE" NUMBER(20, 2) NOT NULL,
"MEASURE_INCOME" VARCHAR2(4) NOT NULL);

ALTER TABLE "FACT_BOOKSALE" ADD CONSTRAINT "FK_DIM_DATE" FOREIGN KEY
("FOR_KEY_DATE_KEY") REFERENCES "DIM_DATE"("DATE_KEY");

ALTER TABLE "FACT_BOOKSALE" ADD CONSTRAINT "FK_DIM_STORENO" FOREIGN
KEY ("FOR_KEY_STORENO_KEY") REFERENCES "DIM_STORENO"("STORENO_KEY");

ALTER TABLE "FACT_BOOKSALE" ADD CONSTRAINT "FK_DIM_BOOKID" FOREIGN
KEY ("FOR_KEY_BOOKID_KEY") REFERENCES "DIM_BOOKID"("BOOKID_KEY");

CREATE TABLE "BRIDGE_BOOKID_AUTHORID" ("COMP_KEY_BOOKID_KEY"
NUMBER(10) NOT NULL, "COMP_KEY_AUTHORID_KEY" NUMBER(10) NOT NULL,
"BRIDGE_WEIGHT" NUMBER(5,4) NOT NULL);

ALTER TABLE "BRIDGE_BOOKID_AUTHORID" ADD CONSTRAINT
"BRIDGE_PKEY_DIM_AUTHORID" PRIMARY KEY("COMP_KEY_BOOKID_KEY",
"COMP_KEY_AUTHORID_KEY");

ALTER TABLE "BRIDGE_BOOKID_AUTHORID" ADD CONSTRAINT
"BRIDGE_FKEY_MANY_AUTHORID" FOREIGN KEY("COMP_KEY_AUTHORID_KEY")
REFERENCES "MANY_AUTHORID"("AUTHORID_KEY");

ALTER TABLE "BRIDGE_DIM_AUTHORID" ADD CONSTRAINT
"BRIDGE_FKEY_DIM_BOOKID" FOREIGN KEY("COMP_KEY_BOOKID_KEY")
REFERENCES "DIM_BOOKID"("BOOKID_KEY");
```

F: Podaci za studijski primjer OAGIS 7.2.1. Purchase Order

F1: XML Schema za glavni dokument: *003_PROCESS_PO_007.xsd*

```
<?xml version="1.0"?>
<xs:schema
targetNamespace="http://www.openapplications.org/003_process_po_007"
xmlns:os="http://www.openapplications.org/oagis_segments"
xmlns:of="http://www.openapplications.org/oagis_fields"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.openapplications.org/003_process_po_007">

<xs:import namespace="http://www.openapplications.org/oagis_segments"
schemaLocation="oagis_segments.xsd"/>

<xs:import   namespace="http://www.openapplications.org/oagis_fields"
schemaLocation="oagis_fields.xsd"/>

<xs:element name="PROCESS_PO_007">
<xs:complexType>
            <xs:sequence>
                  <xs:element ref="os:CNTROLAREA"/>
                  <xs:element ref="DATAAREA" maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="DATAAREA">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="PROCESS_PO"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="PROCESS_PO">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="POORDERHDR"/>
                  <xs:element ref="POORDERLIN"
                                              maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="POORDERHDR">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="os:DATETIME" minOccurs="0"/>
                  <xs:element ref="of:POID"/>
                  <xs:element ref="of:POTYPE"/>
                  <xs:element ref="of:ACKREQUEST" minOccurs="0"/>
                  <xs:element ref="of:DESCRIPTN" minOccurs="0"/>
                  <xs:element ref="os:PARTNER"
    maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="POTERM">
      <xs:complexType>
```

```
            <xs:sequence>
                  <xs:element ref="of:DESCRIPTN" minOccurs="0"/>
                  <xs:element ref="of:TERMID" minOccurs="0"/>
                  <xs:choice>
                        <xs:sequence>
                              <xs:element ref="of:DAYOFMONTH"/>
                              <xs:element ref="of:PROXMONTH"/>
                        </xs:sequence>
                        <xs:element ref="of:DAYSNUM"/>
                  </xs:choice>
                  <xs:choice>
                        <xs:element ref="os:OPERAMT"/>
                        <xs:element ref="os:QUANTITY"/>
                  </xs:choice>
                  <xs:element ref="of:USERAREA" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="POORDERLIN">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="os:QUANTITY"/>
                  <xs:element ref="of:POLINENUM"/>
                  <xs:choice>
                    <xs:sequence>
                        <xs:element ref="of:DESCRIPTN"/>
                        <xs:element ref="of:ITEM" minOccurs="0"/>
                        <xs:element ref="of:ITEMX" minOccurs="0"/>
                        <xs:element ref="of:UPC" minOccurs="0"/>
                    </xs:sequence>
                    <xs:sequence>
                        <xs:element ref="of:ITEM"/>
                        <xs:element ref="of:ITEMX" minOccurs="0"/>
                        <xs:element ref="of:UPC" minOccurs="0"/>
                    </xs:sequence>
                    <xs:sequence>
                        <xs:element ref="of:ITEMX"/>
                        <xs:element ref="of:UPC" minOccurs="0"/>
                    </xs:sequence>
                    <xs:sequence>
                        <xs:element ref="of:UPC"/>
                    </xs:sequence>
                  </xs:choice>
                  <xs:element ref="POLINESCHD" minOccurs="0"
                        maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="POSUBLINE">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="os:QUANTITY"/>
                  <xs:element ref="of:DRAWING" minOccurs="0"/>
                  <xs:element ref="of:ITEMRV" minOccurs="0"/>
                  <xs:element ref="of:ITEMRVX" minOccurs="0"/>
                  <xs:element ref="of:PSBLINENUM" minOccurs="0"/>
                  <xs:choice>
                    <xs:sequence>
                        <xs:element ref="of:DESCRIPTN"/>
```

```
                        <xs:element ref="of:ITEM" minOccurs="0"/>
                        <xs:element ref="of:ITEMX" minOccurs="0"/>
                        <xs:element ref="of:UPC" minOccurs="0"/>
                    </xs:sequence>
                    <xs:sequence>
                        <xs:element ref="of:ITEM"/>
                        <xs:element ref="of:ITEMX" minOccurs="0"/>
                        <xs:element ref="of:UPC" minOccurs="0"/>
                    </xs:sequence>
                    <xs:sequence>
                        <xs:element ref="of:ITEMX"/>
                        <xs:element ref="of:UPC" minOccurs="0"/>
                    </xs:sequence>
                    <xs:sequence>
                        <xs:element ref="of:UPC"/>
                    </xs:sequence>
                </xs:choice>
                <xs:element ref="of:USERAREA" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="POLINESCHD">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="os:DATETIME"/>
                  <xs:element ref="os:QUANTITY"/>
                  <xs:element ref="of:DESCRIPTN" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

</xs:schema>
```

F2: Skraćena XML Schema oagis_segments.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://www.openapplications.org/oagis_segments"
xmlns="http://www.openapplications.org/oagis_segments"
xmlns:of="http://www.openapplications.org/oagis_fields"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">


<xs:import   namespace="http://www.openapplications.org/oagis_fields"
schemaLocation="oagis_fields.xsd"/>

<xs:element name="CNTROLAREA" type="CNTROLAREA"/>

<xs:complexType name="CNTROLAREA">
      <xs:sequence>
            <xs:element ref="BSR"/>
            <xs:element ref="SENDER"/>
            <xs:element ref="DATETIME"/>
      </xs:sequence>
</xs:complexType>

<xs:element name="BSR">
      <xs:complexType>
            <xs:sequence>
```

```
                  <xs:element ref="of:VERB"/>
                  <xs:element ref="of:NOUN"/>
                  <xs:element ref="of:REVISION"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="SENDER">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="of:LOGICALID"/>
                  <xs:element ref="of:COMPONENT"/>
                  <xs:element ref="of:TASK"/>
                  <xs:element ref="of:REFERENCEID"/>
                  <xs:element ref="of:CONFIRMATION"/>
                  <xs:element ref="of:LANGUAGE"/>
                  <xs:element ref="of:CODEPAGE"/>
                  <xs:element ref="of:AUTHID"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="DATETIME" type="DATETIME"
substitutionGroup="DATETIMEANY"/>
<xs:complexType name="DATETIME">
      <xs:sequence>
            <xs:element ref="of:YEAR"/>
            <xs:element ref="of:MONTH"/>
            <xs:element ref="of:DAY"/>
            <xs:element ref="of:HOUR"/>
            <xs:element ref="of:MINUTE"/>
            <xs:element ref="of:SECOND"/>
            <xs:element ref="of:SUBSECOND"/>
            <xs:element ref="of:TIMEZONE"/>
      </xs:sequence>
      <xs:attribute name="qualifier" type="DATETIMEqual"
           use="required"/>
</xs:complexType>

<xs:simpleType name="DATETIMEqual">
      <xs:restriction base="xs:string">
            <xs:enumeration value="ACCOUNTING"/>
            <xs:enumeration value="ACTEND"/>
            <xs:enumeration value="ACTSTART"/>
            <xs:enumeration value="APPREQ"/>
            <xs:enumeration value="APPROVAL"/>
            <xs:enumeration value="AVAILABLE"/>
            <xs:enumeration value="BKTEND"/>
            <xs:enumeration value="BKTSTART"/>
            <xs:enumeration value="CANCEL"/>
            <xs:enumeration value="CHANGEDATE"/>
            <xs:enumeration value="COMPDATE"/>
            <xs:enumeration value="CONSUME"/>
            <xs:enumeration value="CREATION"/>
            <xs:enumeration value="CUMULATIVE"/>
            <xs:enumeration value="DELIVACT"/>
            <xs:enumeration value="DELIVSCHED"/>
            <xs:enumeration value="DISCNT"/>
            <xs:enumeration value="DOCUMENT"/>
            <xs:enumeration value="DUE"/>
            <xs:enumeration value="EARLSTEFF"/>
```

```
            <xs:enumeration value="EARLSTSHIP"/>
            <xs:enumeration value="EFFECTIVE"/>
            <xs:enumeration value="ENGCHG"/>
            <xs:enumeration value="EXECFINISH"/>
            <xs:enumeration value="EXECSTART"/>
            <xs:enumeration value="EXPIRATION"/>
            <xs:enumeration value="FAILDATE"/>
            <xs:enumeration value="FORECASTF"/>
            <xs:enumeration value="FORECASTS"/>
            <xs:enumeration value="FROM"/>
            <xs:enumeration value="GENERATION"/>
            <xs:enumeration value="JOBDUE"/>
            <xs:enumeration value="IMPL"/>
            <xs:enumeration value="INVOICE"/>
            <xs:enumeration value="LABORFINSH"/>
            <xs:enumeration value="LABORSTART"/>
            <xs:enumeration value="LASTUSED"/>
            <xs:enumeration value="LOADING"/>
            <xs:enumeration value="MATCHING"/>
            <xs:enumeration value="MSMENTDATE"/>
            <xs:enumeration value="NEEDDELV"/>
            <xs:enumeration value="OPFINISH"/>
            <xs:enumeration value="OPSTART"/>
            <xs:enumeration value="PAYEND"/>
            <xs:enumeration value="PLANEND"/>
            <xs:enumeration value="PLANSTART"/>
            <xs:enumeration value="PO"/>
            <xs:enumeration value="PROMDELV"/>
            <xs:enumeration value="PROMSHIP"/>
            <xs:enumeration value="PYMTTERM"/>
            <xs:enumeration value="RECEIVED"/>
            <xs:enumeration value="REPORTDATE"/>
            <xs:enumeration value="REPORTNGFN"/>
            <xs:enumeration value="REPORTNGST"/>
            <xs:enumeration value="REQUIRED"/>
            <xs:enumeration value="RESORCDWNF"/>
            <xs:enumeration value="RESORCDWNS"/>
            <xs:enumeration value="RSPDDATE"/>
            <xs:enumeration value="RSPDOCGEN"/>
            <xs:enumeration value="SCHEND"/>
            <xs:enumeration value="SCHSTART"/>
            <xs:enumeration value="SETUPFINSH"/>
            <xs:enumeration value="SETUPSTART"/>
            <xs:enumeration value="SHIP"/>
            <xs:enumeration value="SHIPSCHED"/>
            <xs:enumeration value="STATUSDATE"/>
            <xs:enumeration value="TEARDOWNF"/>
            <xs:enumeration value="TEARDOWNS"/>
            <xs:enumeration value="TO"/>
            <xs:enumeration value="OTHER"/>
      </xs:restriction>
</xs:simpleType>

<xs:element name="OPERAMT" type="OPERAMT"/>
<xs:complexType name="OPERAMT">
            <xs:sequence>
                  <xs:element ref="of:VALUE"/>
                  <xs:element ref="of:NUMOFDEC"/>
                  <xs:element ref="of:SIGN"/>
                  <xs:element ref="of:CURRENCY"/>
                  <xs:element ref="of:UOMVALUE"/>
```

```
                  <xs:element ref="of:UOMNUMDEC"/>
                  <xs:element ref="of:UOM"/>
            </xs:sequence>
            <xs:attribute name="qualifier" use="required">
                  <xs:simpleType>
                        <xs:restriction base="xs:NMTOKEN">
                              <xs:enumeration value="COST"/>
                              <xs:enumeration value="EXTENDED"/>
                              <xs:enumeration value="FREIGHT"/>
                              <xs:enumeration value="UNIT"/>
                              <xs:enumeration value="OTHER"/>
                        </xs:restriction>
                  </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="type" use="required">
                  <xs:simpleType>
                        <xs:restriction base="xs:NMTOKEN">
                              <xs:enumeration value="T"/>
                              <xs:enumeration value="F"/>
                              <xs:enumeration value="OTHER"/>
                        </xs:restriction>
                  </xs:simpleType>
            </xs:attribute>
      </xs:complexType>

<xs:element name="PARTNER" type="PARTNER"/>

<!-- izmjena prema dokumentima: Marko Banek -->
<xs:complexType name="PARTNER">
      <xs:sequence>
            <xs:element ref="of:NAME"/>
            <xs:element ref="of:ONETIME"/>
            <xs:element ref="of:PARTNRID"/>
            <xs:element ref="of:PARTNRTYPE"/>
      </xs:sequence>
</xs:complexType>

<xs:element name="QUANTITY" type="QUANTITY"/>
<xs:complexType name="QUANTITY">
      <xs:sequence>
            <xs:element ref="of:VALUE"/>
            <xs:element ref="of:NUMOFDEC"/>
            <xs:element ref="of:SIGN"/>
            <xs:element ref="of:UOM"/>
      </xs:sequence>
      <xs:attribute name="qualifier" use="required">
            <xs:simpleType>
                  <xs:restriction base="xs:NMTOKEN">
                        <xs:enumeration value="ACCEPTED"/>
                        <xs:enumeration value="ACTDUR"/>
                        <xs:enumeration value="ACTHRS"/>
                        <xs:enumeration value="ALLOCATED"/>
                        <xs:enumeration value="ALLOWEDWT"/>
                        <xs:enumeration value="AVAILABLE"/>
                        <xs:enumeration value="AVGRUNSIZE"/>
                        <xs:enumeration value="BACKORDERD"/>
                        <xs:enumeration value="BATCHSIZE"/>
                        <xs:enumeration value="BATCHTIME"/>
                        <xs:enumeration value="BLOCKED"/>
                        <xs:enumeration value="BREAKTIME"/>
                        <xs:enumeration value="CAPPERCENT"/>
```

```
<xs:enumeration value="CATCHWEGHT"/>
<xs:enumeration value="COMMISSION"/>
<xs:enumeration value="COMPLETED"/>
<xs:enumeration value="CUMULATIVE"/>
<xs:enumeration value="DELIVERED"/>
<xs:enumeration value="DURATION"/>
<xs:enumeration value="DUROVER"/>
<xs:enumeration value="DURUNDER"/>
<xs:enumeration value="EMPLOYEES"/>
<xs:enumeration value="EMPREQD"/>
<xs:enumeration value="ESTDUR"/>
<xs:enumeration value="ESTHRS"/>
<xs:enumeration value="ESTWEIGHT"/>
<xs:enumeration value="FILENAME"/>
<xs:enumeration value="FILESIZE"/>
<xs:enumeration value="FIXEDTIME"/>
<xs:enumeration value="HEIGHT"/>
<xs:enumeration value="INSPECTED"/>
<xs:enumeration value="INSPECTION"/>
<xs:enumeration value="ITEM"/>
<xs:enumeration value="LDTMOFFSET"/>
<xs:enumeration value="LENGTH"/>
<xs:enumeration value="LABOR"/>
<xs:enumeration value="LOADINGWT"/>
<xs:enumeration value="LOTSIZEMAX"/>
<xs:enumeration value="LOTSIZEMIN"/>
<xs:enumeration value="LOTSIZEMLT"/>
<xs:enumeration value="LOWERLIMIT"/>
<xs:enumeration value="MACHINEHRS"/>
<xs:enumeration value="MAXIMUM"/>
<xs:enumeration value="MAXPARLTM"/>
<xs:enumeration value="MINIMUM"/>
<xs:enumeration value="MOVETIME"/>
<xs:enumeration value="MSMENT"/>
<xs:enumeration value="MULTIPLIER"/>
<xs:enumeration value="NETWEIGHT"/>
<xs:enumeration value="OPEN"/>
<xs:enumeration value="ORDERED"/>
<xs:enumeration value="OTHERREJ"/>
<xs:enumeration value="OVERSHIP"/>
<xs:enumeration value="PACKING"/>
<xs:enumeration value="PERCENT"/>
<xs:enumeration value="PERCENTREQ"/>
<xs:enumeration value="PERSHBNOPR"/>
<xs:enumeration value="PERSHWIOPR"/>
<xs:enumeration value="PLNDPRCT"/>
<xs:enumeration value="PRCBRK"/>
<xs:enumeration value="PRIOR"/>
<xs:enumeration value="QUEUETIME"/>
<xs:enumeration value="RATE"/>
<xs:enumeration value="RECEIVED"/>
<xs:enumeration value="REJECTED"/>
<xs:enumeration value="REJFIXED"/>
<xs:enumeration value="REJPERCENT"/>
<xs:enumeration value="REMDUR"/>
<xs:enumeration value="REMHRS"/>
<xs:enumeration value="REQUIRED"/>
<xs:enumeration value="RETURNED"/>
<xs:enumeration value="REWORK"/>
<xs:enumeration value="RUNTIME"/>
<xs:enumeration value="SCRAP"/>
```

```
                        <xs:enumeration value="SETUPTIME"/>
                        <xs:enumeration value="SHELFLIFE"/>
                        <xs:enumeration value="SHIPPED"/>
                        <xs:enumeration value="SHIPUNIT"/>
                        <xs:enumeration value="START"/>
                        <xs:enumeration value="TEARDOWN"/>
                        <xs:enumeration value="TOOLREQD"/>
                        <xs:enumeration value="TOTWEIGHT"/>
                        <xs:enumeration value="TRANSFRLOT"/>
                        <xs:enumeration value="UNDERSHIP"/>
                        <xs:enumeration value="UNIT"/>
                        <xs:enumeration value="UPPERLIMIT"/>
                        <xs:enumeration value="VOLUME"/>
                        <xs:enumeration value="WAITTIME"/>
                        <xs:enumeration value="WEIGHT"/>
                        <xs:enumeration value="WIDTH"/>
                        <xs:enumeration value="OTHER"/>
                  </xs:restriction>
            </xs:simpleType>
      </xs:attribute>
</xs:complexType>

</xs:schema>
```

F3: Skraćena XML Schema oagis_fields.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.openapplications.org/oagis_fields"

targetNamespace="http://www.openapplications.org/oagis_fields"
xmlns:="http://www.openapplications.org/oagis_fields"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">

<xs:element name="VERB" type="VERB"/>
<xs:simpleType name="VERB">
      <xs:restriction base="xs:string">
            <xs:enumeration value="ACKNOWLEDGE"/>
            <xs:enumeration value="ALLOCATE"/>
            <xs:enumeration value="ADD"/>
            <xs:enumeration value="CANCEL"/>
            <xs:enumeration value="CHANGE"/>
            <xs:enumeration value="CONFIRM"/>
            <xs:enumeration value="CREATE"/>
            <xs:enumeration value="GET"/>
            <xs:enumeration value="GETLIST"/>
            <xs:enumeration value="ISSUE"/>
            <xs:enumeration value="LIST"/>
            <xs:enumeration value="LOAD"/>
            <xs:enumeration value="POST"/>
            <xs:enumeration value="PROCESS"/>
            <xs:enumeration value="RECEIVE"/>
            <xs:enumeration value="RESPOND"/>
            <xs:enumeration value="SHOW"/>
            <xs:enumeration value="SYNC"/>
            <xs:enumeration value="TRANSFER"/>
            <xs:enumeration value="UPDATE"/>
      </xs:restriction>
</xs:simpleType>
<xs:simpleType name="NOUN">
      <xs:restriction base="xs:string">
```

```
<xs:enumeration value="ACTIVITY"/>
<xs:enumeration value="BOM"/>
<xs:enumeration value="BOD"/>
<xs:enumeration value="CATALOG"/>
<xs:enumeration value="COA"/>
<xs:enumeration value="CONSUMPTN"/>
<xs:enumeration value="COUNTINFO"/>
<xs:enumeration value="CREDIT"/>
<xs:enumeration value="CUSTOMER"/>
<xs:enumeration value="DELIVERY"/>
<xs:enumeration value="DSPTCHLIST"/>
<xs:enumeration value="ECATALOG"/>
<xs:enumeration value="ENGCHGORDR"/>
<xs:enumeration value="EXCHNGRATE"/>
<xs:enumeration value="FIELD"/>
<xs:enumeration value="INSPECTION"/>
<xs:enumeration value="INVENCOUNT"/>
<xs:enumeration value="INVENTORY"/>
<xs:enumeration value="INVOICE"/>
<xs:enumeration value="ISSUE"/>
<xs:enumeration value="ISSUEINFO"/>
<xs:enumeration value="ITEM"/>
<xs:enumeration value="ITEMCLASS"/>
<xs:enumeration value="ITEMSPECS"/>
<xs:enumeration value="ITEMXREF"/>
<xs:enumeration value="JOURNAL"/>
<xs:enumeration value="LDGRACTUAL"/>
<xs:enumeration value="LDGRBUDGET"/>
<xs:enumeration value="MAINTORDER"/>
<xs:enumeration value="MATCHDOC"/>
<xs:enumeration value="MATCHFAIL"/>
<xs:enumeration value="MATCHOK"/>
<xs:enumeration value="MFGTLCODE"/>
<xs:enumeration value="MISCITEM"/>
<xs:enumeration value="PAYABLE"/>
<xs:enumeration value="PERSONNEL"/>
<xs:enumeration value="PERSONTIME"/>
<xs:enumeration value="PICKLIST"/>
<xs:enumeration value="PLANSCHD"/>
<xs:enumeration value="PLINVOICE"/>
<xs:enumeration value="PO"/>
<xs:enumeration value="PRICELIST"/>
<xs:enumeration value="PRODAVAIL"/>
<xs:enumeration value="PRODORDER"/>
<xs:enumeration value="PRODUCTREQ"/>
<xs:enumeration value="PROJACCTNG"/>
<xs:enumeration value="PROJINFO"/>
<xs:enumeration value="QUOTE"/>
<xs:enumeration value="RECEIVABLE"/>
<xs:enumeration value="REQUISITN"/>
<xs:enumeration value="RESOURCE"/>
<xs:enumeration value="RFQ"/>
<xs:enumeration value="ROUTING"/>
<xs:enumeration value="SALESORDER"/>
<xs:enumeration value="SEQSCHD"/>
<xs:enumeration value="SHIPMENT"/>
<xs:enumeration value="SHIPSCHD"/>
<xs:enumeration value="SITELEVEL"/>
<xs:enumeration value="STATUS"/>
<xs:enumeration value="SUPPLIER"/>
<xs:enumeration value="UOMGROUP"/>
```

```
            <xs:enumeration value="WIPCONFIRM"/>
            <xs:enumeration value="WIPMERGE"/>
            <xs:enumeration value="WIPMOVE"/>
            <xs:enumeration value="WIPRECOVER"/>
            <xs:enumeration value="WIPSPLIT"/>
            <xs:enumeration value="WIPSTATUS"/>
            <xs:enumeration value="WRKSCHDULE"/>
      </xs:restriction>
</xs:simpleType>
<xs:simpleType name="REVISION">
      <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="LOGICALID">
      <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="CONFIRMATION">
      <xs:annotation>
            <xs:documentation>
            0 - No Confirm BOD requested, 1 - Send Confirm BOD only
                              on error, 2 - Send Confirm BOD always
            </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string">
            <xs:enumeration value="0"/>
            <xs:enumeration value="1"/>
            <xs:enumeration value="2"/>
            <xs:enumeration value="NOCONFIRM"/>
            <xs:enumeration value="ONERROR"/>
            <xs:enumeration value="YESCONFIRM"/>
      </xs:restriction>
</xs:simpleType>
<xs:simpleType name="LANGUAGE">
      <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="ITEM">
      <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="PARTNRID">
      <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="PARTNRTYPE">
      <xs:restriction base="xs:string">
            <xs:enumeration value="ShipTo"/>
            <xs:enumeration value="BillTo"/>
            <xs:enumeration value="SoldTo"/>
            <xs:enumeration value="PayFrom"/>
            <xs:enumeration value="Supplier"/>
            <xs:enumeration value="RemitTo"/>
            <xs:enumeration value="Carrier"/>
            <xs:enumeration value="Broker"/>
            <xs:enumeration value="Employee"/>
            <xs:enumeration value="JV"/>
            <xs:enumeration value="Publisher"/>
            <xs:enumeration value="Manufacturer"/>
            <xs:enumeration value="ShipFrom"/>
      </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SCHLINENUM">
      <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="SUBLINENUM">
```

```
      <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="TERMID">
      <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:element name="NOUN" type="NOUN"/>
<xs:element name="REVISION" type="REVISION"/>

<xs:element name="LOGICALID" type="LOGICALID"/>
<xs:element name="COMPONENT" type="xs:string"/>
<xs:element name="TASK" type="xs:string"/>
<xs:element name="REFERENCEID" type="xs:string"/>
<xs:element name="CONFIRMATION" type="CONFIRMATION"/>
<xs:element name="LANGUAGE" type="LANGUAGE"/>
<xs:element name="CODEPAGE" type="xs:string"/>
<xs:element name="AUTHID" type="xs:string"/>
<xs:element name="YEAR" type="xs:gYear"/>
<xs:element name="MONTH" type="xs:integer"/>
<xs:element name="DAY" type="xs:integer"/>
<xs:element name="HOUR" type="xs:integer"/>
<xs:element name="MINUTE" type="xs:integer"/>
<xs:element name="SECOND" type="xs:integer"/>
<xs:element name="SUBSECOND" type="xs:integer"/>
<xs:element name="TIMEZONE" type="xs:string"/>
<xs:element name="ACKREQUEST">
      <xs:simpleType>
            <xs:restriction base="xs:string">
                  <xs:enumeration value="0"/>
                  <xs:enumeration value="1"/>
                  <xs:enumeration value="2"/>
                  <xs:enumeration value="NOACK"/>
                  <xs:enumeration value="ONCHANGE"/>
                  <xs:enumeration value="YESACK"/>
            </xs:restriction>
      </xs:simpleType>
</xs:element>
<xs:element name="CURRENCY" type="xs:string"/>
<xs:element name="DAYOFMONTH" type="xs:integer"/>
<xs:element name="DAYSNUM" type="xs:integer"/>
<xs:element name="DESCRIPTN" type="xs:string"/>
<xs:element name="DRAWING" type="xs:string"/>
<xs:element name="ITEM" type="ITEM"/>
<xs:element name="ITEMX" type="xs:string"/>
<xs:element name="ITEMRV" type="xs:string"/>
<xs:element name="ITEMRVX" type="xs:string"/>
<xs:element name="LINENUM" type="xs:string"/>
<xs:element name="NAME" type="xs:string"/>
<xs:element name="NUMOFDEC" type="xs:integer"/>
<xs:element name="ONETIME" type="xs:boolean"/>
<xs:element name="PARTNRID" type="PARTNRID"/>
<xs:element name="PARTNRTYPE" type="PARTNRTYPE"/>
<xs:element name="POID" type="xs:string"/>
<xs:element name="POLINENUM" type="xs:string"/>
<xs:element name="POTYPE" type="xs:string"/>
<xs:element name="PROXMONTH" type="xs:integer"/>
<xs:element name="PSBLINENUM" type="xs:string"/>
<xs:element name="SIGN" type="xs:string"/>
<xs:element name="TERMID" type="TERMID"/>
<xs:element name="UOM" type="xs:string"/>
<xs:element name="UOMNUMDEC" type="xs:integer"/>
```

```
<xs:element name="UOMVALUE" type="xs:string"/>
<xs:element name="UPC" type="xs:string"/>
<xs:element name="USERAREA" type="USERAREA"/>
<xs:simpleType name="USERAREA">
            <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:element name="VALUE" type="xs:decimal"/>

</xs:schema>
```

F4. Agrokorov XML dokument za narudžbu robe prema XML Schemi OAGIS 7.21. *Purchase Order*

```
<?xml version="1.0" encoding="UTF-8"?>
<PROCESS_PO_007
           xmlns="http://www.openapplications.org/003_process_po_007"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:of="http://www.openapplications.org/oagis_fields"
           xmlns:os="http://www.openapplications.org/oagis_segments">
      <os:CNTROLAREA>
        <os:BSR>
          <of:VERB>PROCESS</of:VERB>
          <of:NOUN>PO</of:NOUN>
          <of:REVISION>007</of:REVISION>
        </os:BSR>
        <os:SENDER>
          <of:LOGICALID>3280756</of:LOGICALID>
          <of:COMPONENT>PURCHASING</of:COMPONENT>
          <of:TASK>POISSUE</of:TASK>
          <of:REFERENCEID>20030415151758</of:REFERENCEID>
          <of:CONFIRMATION>0</of:CONFIRMATION>
          <of:LANGUAGE>HRV</of:LANGUAGE>
          <of:CODEPAGE>UTF-8</of:CODEPAGE>
          <of:AUTHID>b2b.agrokor.hr</of:AUTHID>
        </os:SENDER>
        <os:DATETIME qualifier="CREATION">
          <of:YEAR>2003</of:YEAR>
          <of:MONTH>4</of:MONTH>
          <of:DAY>15</of:DAY>
          <of:HOUR>15</of:HOUR>
          <of:MINUTE>17</of:MINUTE>
          <of:SECOND>58</of:SECOND>
          <of:SUBSECOND>732</of:SUBSECOND>
          <of:TIMEZONE>+0100</of:TIMEZONE>
        </os:DATETIME>
      </os:CNTROLAREA>
      <DATAAREA>
        <PROCESS_PO>
          <POORDERHDR>
            <os:DATETIME qualifier="DOCUMENT">
              <of:YEAR>2003</of:YEAR>
              <of:MONTH>4</of:MONTH>
              <of:DAY>15</of:DAY>
              <of:HOUR>15</of:HOUR>
              <of:MINUTE>17</of:MINUTE>
              <of:SECOND>58</of:SECOND>
              <of:SUBSECOND>732</of:SUBSECOND>
              <of:TIMEZONE>+0100</of:TIMEZONE>
            </os:DATETIME>
            <of:POID>N325985</of:POID>
            <of:POTYPE>224</of:POTYPE>
```

```
        <of:ACKREQUEST>1</of:ACKREQUEST>
        <os:PARTNER>
          <of:NAME>Supplier1</of:NAME>
          <of:ONETIME>0</of:ONETIME>
          <of:PARTNRID>E0001</of:PARTNRID>
          <of:PARTNRTYPE>Supplier</of:PARTNRTYPE>
        </os:PARTNER>
        <os:PARTNER>
          <of:NAME>SoldTo1</of:NAME>
          <of:ONETIME>0</of:ONETIME>
          <of:PARTNRID>HR-ZG-AGROKOR</of:PARTNRID>
          <of:PARTNRTYPE>SoldTo</of:PARTNRTYPE>
        </os:PARTNER>
        <os:PARTNER>
          <of:NAME>ShipTo1</of:NAME>
          <of:ONETIME>0</of:ONETIME>
          <of:PARTNRID>HR-ZG-AGROKOR</of:PARTNRID>
          <of:PARTNRTYPE>ShipTo</of:PARTNRTYPE>
        </os:PARTNER>
      </POORDERHDR>

    <POORDERLIN>
        <os:QUANTITY qualifier="ORDERED">
          <of:VALUE>2880</of:VALUE>
          <of:NUMOFDEC>1</of:NUMOFDEC>
          <of:SIGN>+</of:SIGN>
          <of:UOM>PCE</of:UOM>
        </os:QUANTITY>
        <of:POLINENUM>1</of:POLINENUM>
        <of:DESCRIPTN>KRASTAVAC 1,4KG ST PODRAVKA</of:DESCRIPTN>
        <of:UPC>3850104050220</of:UPC>
        <POLINESCHD>
          <os:DATETIME qualifier="NEEDDELV">
            <of:YEAR>2003</of:YEAR>
            <of:MONTH>4</of:MONTH>
            <of:DAY>15</of:DAY>
            <of:HOUR>15</of:HOUR>
            <of:MINUTE>17</of:MINUTE>
            <of:SECOND>58</of:SECOND>
            <of:SUBSECOND>732</of:SUBSECOND>
            <of:TIMEZONE>+0100</of:TIMEZONE>
          </os:DATETIME>
          <os:QUANTITY qualifier="ORDERED">
            <of:VALUE>3880</of:VALUE>
            <of:NUMOFDEC>1</of:NUMOFDEC>
            <of:SIGN>+</of:SIGN>
            <of:UOM>PCE</of:UOM>
          </os:QUANTITY>
        </POLINESCHD>
      </POORDERLIN></PROCESS_PO>
  </DATAAREA>
</PROCESS_PO_007>
```