Temporalne baze podataka

Perčinlić, Andrea

Master's thesis / Diplomski rad

2015

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:447255

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Andrea Perčinlić

TEMPORALNE BAZE PODATAKA

Diplomski rad

Voditelj rada:
prof. dr. sc. Robert Manger

Zagreb, rujan 2015.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem svojem mentoru, prof. dr. sc. Robertu Mangeru,
na pomoći pri izradi ovog diplomskog rada.

Posebno zahvaljujem svojoj majci na ljubavi i potpori.

Sadržaj

Uvod

U većini baza podataka za svaki podatak pohranjuje se samo njegova najnovija vrijednost. Kod svake promjene vrijednost se ažurira, a stara vrijednost se gubi. No ponekad nas zanimaju prethodne, a možda čak i buduće vrijednosti istog podatka. Temporalne baze podataka su takve baze podataka koje uključuju posebnu podršku za vremensku dimenziju podataka. Drugim riječima, to je sustav koji omogućuje posebne radnje poput pohranjivanja, pretraživanja i ažuriranja prethodnih i/ili budućih vrijednosti podataka. Temporalne baze dopuštaju postavljanje temporalnih upita gdje uvjeti pretraživanja ovise o vremenu. Današnji sustavi za upravljanje bazom podataka pružaju malu ili nikakvu podršku za temporalne podatke. Međutim, ta se situacija počinje mijenjati iz sljedećih razloga:

1. Diskovi i drugi sekundarni mediji za pohranjivanje podataka postali su dovoljno jeftini tako da je čuvanje velike količine temporalnih podataka moguće.
2. Kao posljedica 1., postojanje uskladištenih podataka je u porastu.
3. Korisnici skladišta podataka nailaze na probleme s temporalnim podacima i traže rješenja za te probleme.
4. Kako bi se riješili neki od tih problema, određene temporalne značajke uključene su u zadnju verziju standarda za SQL 2011. godine.
5. Kao posljedica 4., proizvođači konvencionalnih sustava za upravljanje bazom podataka počinju dodavati podršku za temporalne podatke u svoje proizvode.

U nastavku se navode primjeri scenarija gdje je potrebno postojanje podrške za temporalne podatke. Primjeri su preuzeti iz [4]:

1. Interna revizija zahtijeva da financijska institucija prikaže promjene na klijentovom računu u zadnjih pet godina.
2. Zakon propisuje da bolnica ponovno ocijeni kliničko stanje pacijenta prije nego što se započne novi tretman liječenja.
3. Trgovac mora osigurati najviše jednu vrijednost popusta na određeni proizvod tijekom nekog razdoblja.

4. Klijent osporava odluku osiguravajuće agencije o prometnoj nesreći. Agencija mora utvrditi uvjete osiguranja koji su vrijedili u trenutku kada se nesreća dogodila.
5. Putnička agencija želi otkriti nekonzistentnosti u planu puta. Na primjer, ako netko rezervira smještaj u hotelu u Rimu na sedam dana te automobil u Madridu na tri dana od tih sedam, putnička agencija treba ponovno ispitati tu situaciju.

Temporalne baze podataka postaju predmet istraživanja oko 1980. godine. Većina tih pristupa pokazala se neučinkovitima, postoje logičke nekonzistentnosti ili su nezadovoljavajuća iz nekog drugog razloga. Nije jednostavno upravljati vremenom u današnjim relacijskim bazama, no to je pristup koji najviše obećava. Zato se predstavlja u nastavku radu. Rad sadržajno slijedi koncepte za upravljanje temporalnim podacima koji su izloženi u [1]. Relacijski model zahtijeva da se baza podataka sastoji od skupa pravokutnih tablica – relacija. Svaka relacija ima svoje ime po kojem je razlikujemo od ostalih u istoj bazi. Jedan stupac relacije obično sadrži vrijednost jednog atributa. Nadalje, predstavljamo *bazu podataka o dobavljačima i pošiljkama* koja je temelj za sve primjere u radu. Primjeri su većinom preuzeti iz [1]. Formalne definicije primjera zapisane su u jeziku *Tutorial D*. Tutorial D je jezik za postavljanje upita u relacijskim bazama podataka zasnovan na relacijskoj algebri. Ne rabi se izravno u praksi, no važan je za teoriju.

S

SNO	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Pariz
S3	Blake	30	Pariz
S4	Clark	20	London
S5	Adams	30	Atena

SP

SNO	PNO
S1	P1
S1	P2
S1	P3
S1	P4
S1	P5
S1	P6
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4
S4	P5

Slika 0.1: Baza podataka o dobavljačima i pošiljkama

Zamislimo situaciju koja odgovara stanju baze podataka o dobavljačima i pošiljkama. Neka je poslodavac tvrtka koja se bavi izradom i dostavom poslovnih darova. Tvrtka ima nekoliko svojih podružnica diljem svijeta. Kako bi bolje ogranizirala poslovanje, tvrtka vodi evidenciju o svojim zaposlenicima i njihovim zaduženjima. Neki zaposlenici sklapaju ugovor o radu na neodređeno, a drugi na određeno vrijeme. Svakom zaposleniku dodjeljuje se jedinstveni broj koji ga razlikuje od drugih. Posao se sastoji od izrade poslovnih darova i/ili dostave tih pošiljaka klijentima. Određen je broj darova koje svaki zaposlenik mora izraditi. Zaposlenik nije u mogućnosti dostavljati pošiljke ako je zauzet radom u podružnici. Neki zaposlenici mogu biti premješteni u druge podružnice radi povećanja obujma posla. Svaka pošiljka jedinstveno je određena brojem klijenta i brojem zaposlenika koji je zadužen za njezinu dostavu. Obično je dogovoreno vrijeme kada će se pošiljke dostavljati. U nastavku se svi zaposlenici nazivaju dobavljačima.

Relacija S predstavlja dobavljače s ugovorom. Svaki dobavljač ima jedinstvenu oznaku SNO, ime SNAME, brojčanu vrijednost STATUS i lokaciju CITY. Atribut SNO čini primarni ključ. Relacija SP predstavlja potencijalne pošiljke dobavljača s oznakom SNO klijentima s oznakom PNO. Kombinacija atributa SNO i PNO jedinstveno određuje potencijalnu pošiljku. Slijedi zapis primjera u jeziku Tutorial D i kratko objašnjenje zapisa:

```
TYPE SNO ... ;
TYPE PNO ... ;

VAR S BASE RELATION
    { SNO SNO, SNAME NAME, STATUS INTEGER, CITY CHAR }
    KEY { SNO } ;

VAR SP BASE
    RELATION { SNO SNO, PNO PNO }
    KEY { SNO, PNO }
    FOREIGN KEY { SNO } REFERENCES S ;
```

Tipovi SNO, PNO i NAME su korisnički definirani tipovi. Ključna riječ VAR označava varijablu, a BASE definira vrstu varijable. Riječ RELATION označava relacijski tip varijable. Dalje se definiraju pripadni atributi. KEY označava ključ za relacijsku varijablu, a klauzulom FOREIGN KEY ... REFERENCES predstavlja se strani ključ u dotičnoj relaciji i ime relacije u kojoj taj atribut predstavlja ključ.

Poglavlje 1

Osnovni koncepti temporalnih baza podataka

1.1 Vrijeme u bazama podataka

Temporalne baze podataka

Temporalne baze podataka mogu se zamisliti kao baze podataka koje sadrže povijesne podatke umjesto trenutnih podataka ili kao dodatak trenutnim podacima. Takve baze podataka istražuju se od 1980. godine. Neka od tih istraživanja temelje se na ekstremnom pristupu da se podaci koji su jednom pohranjeni u takvu bazu podataka nikada ne bi trebali mijenjati ili brisati. S druge strane, konvencionalne baze podatake temelje se na drugom pristupu: podaci se mijenjaju ili brišu čim prestanu vrijediti. Te baze podataka su eksplicitno netemporalne, odnosno sadrže samo podatke koji trenutačno vrijede. Netemporalne baze podataka u literaturi se ponekad nazivaju *snapshot* baze podataka jer podatke prikazuju u određenom trenutku. Međutim, taj je naziv primjenjiv i na druge baze podataka, temporalne i sl.

Također, baza podataka o dobavljačima i pošiljkama iz uvodnog poglavlja je netemporalna u tom smislu. Na primjer, Slika 0.1 prikazuje da je status dobavljača S5 jednak 30. Temporalna verzija te baze mogla bi prikazivati da je trenutačni status dobavljača S1 jednak 30, ali i da je jednak 30 od 1. srpnja 2015. te da je možda iznosio 25 od 1. travnja do 30. lipnja 2015. godine itd.

Način na koji se administriraju i koriste podaci u temporalnim bazama podataka razlikuje se od načina na koji se administriraju i koriste u netemporalnim bazama. Prepoznatljiva značajka temporalnih baza podataka je vrijeme. Tijekom istraživanja temporalnih baza

mnogo se raspravljalo o samoj prirodi vremena. Slijede neka pitanja koja su se razmatrala:

1. Ima li vrijeme početak i kraj?
2. Je li vrijeme kontinuum ili se sastoji od diskretnih jedinica?
3. Kako najbolje opisati koncept *sada*?

Koncept *sada* u literaturi je poznat pod nazivom *the moving point now*. U nastavku se koriste samo razumne pretpostavke o vremenu. Pretpostavlja se da vrijeme ima početak i kraj. Vrijeme je konačan niz diskretnih točaka. Početna točka nema prethodnika, a završna točka nema sljedbenika.

Tvrdnje s vremenskim žigom

Relacija S može se opisati sljedećim iskazom ili predikatom: *Dobavljač SNO je pod ugovorom, zove se SNAME, ima status STATUS i nalazi se u gradu CITY*.

Svaki redak u S predstavlja određenu tvrdnju ili propoziciju. Na primjer:

1. Dobavljač S1 je pod ugovorom, zove se Smith, ima status 20 i nalazi se u gradu Londonu.
2. Dobavljač S2 je pod ugovorom, zove se Jones, ima status 10 i nalazi se u gradu Parizu.

Na analogan način definiraju se iskaz i tvrdnje za relaciju SP.

Ako podatke zamislimo kao kodirane reprezentacije tvrdnji, tada temporalne podatke možemo opisati kao kodirane reprezentacije tvrdnji s vremenskim žigom, tj. tvrdnji koje imaju neku vremensku oznaku. Dakle, temporalne baze podataka su baze podataka čiji podaci su temporalni, no ne i isključivo temporalni.

Promotrimo sljedeće tvrdnje:

1. Dobavljač S1 je sklopio ugovor 1. srpnja 2015. godine.
2. Dobavljač S1 ima ugovor **od** 1. srpnja 2015. godine.
3. Dobavljač S1 je sklopio ugovor koji vrijedi **od** 1. srpnja 2015. **do** danas.

Svaka od prethodnih tvrdnji predstavlja moguću interpretaciju nekog retka *r* u nekoj relaciji. Redak *r* ima dva atributa, SNO i FROM, čije su vrijednosti broj dobavljača S1 i vremenski žig 1. srpnja 2015.

SNO	FROM
S1	1. srpnja 2015.

Uočimo da prva tvrdnja izražava vrijeme kada se nešto dogodilo, dok druga i treća tvrdnja izražavaju vremensko razdoblje tijekom kojeg je neko stanje postojano. Konvencionalne tehnologije mogu vrlo uspješno upravljati nekim vremenskim trenutkom, ali gotovo uopće ne mogu manipulirati nekim vremenskim periodom. Zaključujemo da iz perspektive temporalnih podataka prva tvrdnja nije zanimljiva. Očito je da su druga i treća tvrdnja logički ekvivalentne, no njihov oblik se značajno razlikuje. Pripadni predikati su:

1. Dobavljač Sx ima ugovor od datuma d.
2. Dobavljač Sx je sklopio ugovor koji vrijedi od datuma b do datuma e.

Treća tvrdnja eksplicitno određuje vremenski period koji ima početnu i završnu točku, a druga tvrdnja određuje samo početni vremenski trenutak. Kao posljedica, oblik druge tvrdnje ne može se koristiti za povijesne zapise, dok se oblik treće tvrdnje može iskoristiti ako se *danas* zamijeni nekim određenim datumom, npr. 1. srpnja 2016. Iz navedenoga zaključujemo da je koncept **od – do** ili **tijekom** vrlo bitan za povijesne zapise. Temporalne baze podataka mogu sadržavati i informacije o budućnosti. Na primjer, želimo dokumentirati da će dobavljač S1 biti pod ugovorom od datuma b do datuma e. U nastavku se podrazumijeva da izrazi *temporalni* i *povijesni* uključuju i budućnost u tom smislu.

Stvarno i transakcijsko vrijeme

Podaci se mogu mijenjati u netemporalnim bazama podataka. No ako se povijesni podaci mogu mijenjati, suočavamo se s mogućnošću da se i povijest može mijenjati. Bitno je naglasiti da baze podataka ne sadrže podatke o stvarnom svijetu, nego naše znanje o njemu. Sukladno tomu zaključujemo da se prošlost ne može promijeniti, ali naša saznanja o njoj – mogu. U kontekstu baza podataka govorimo o *ažuriranju povijesti*, no zapravo mislimo na ažuriranje naših saznanja o povijesti. Kako bi se naglasila ta razlika, u literaturi se koriste termini *stvarno vrijeme* i *transakcijsko vrijeme*.

Promotrimo primjer u nastavku. Neka je p propozicija: *Dobavljač S1 je sklopio ugovor*. Pretpostavimo da ugovor vrijedi od 1. svibnja 2015. do 30. travnja 2016. godine. Tada je odgovarajući redak r u nekoj bazi podataka jednak:

SNO	FROM	TO
S1	1. svibnja 2015.	30. travnja 2016.

Uočimo da redak *r* ne odgovara propoziciji *p*, već njenom proširenju vremenskim žigom: *Dobavljač S1 je sklopio ugovor* – to je *p – koji vrijedi od **1. svibnja 2015. do 30. travnja 2016. godine***. Stvarno vrijeme za propoziciju *p* je vremenski žig, tj. razdoblje od 1. svibnja 2015. do 30. travnja 2016. godine. Općenito, stvarno vrijeme za danu propoziciju sastoji se od skupa vremenskih intervala. Pretpostavimo sada da smo saznali da je dobavljač S1 sklopio ugovor 1. travnja, a ne 1. svibnja 2015. Redak *r* u bazi podataka potrebno je zamijeniti sljedećim:

SNO	FROM	TO
S1	1. travnja 2015.	30. travnja 2016.

Ta promjena nema utjecaja na propoziciju *p*, jedino se mijenja pripadni vremenski žig. Stvarno vrijeme za propoziciju *p* sada je interval od 1. travnja 2015. do 30. travnja 2016. godine. Upravo su *ažurirana naša saznanja o povijesti*, no to ne mijenja činjenicu da je baza podataka prethodno prikazivala da je dobavljač S1 sklopio ugovor 1. svibnja. Na kraju, pretpostavimo da se dogodila pogreška te da dobavljač S1 nikada nije sklopio ugovor. Potrebno je obrisati redak *r* iz baze podataka. Stvarno vrijeme za propoziciju *p* sada je prazan skup vremenskih intervala.

Rad korisnika s bazom podataka u pravilu se svodi na pokretanje tzv. transakcija. Iz korisničke perspektive jedna transakcija predstavlja jednu nedjeljivu cjelinu, no ona se obično realizira kao niz od nekoliko elementarnih zahvata u samoj bazi. Osnovno svojstvo transakcije je da prevodi bazu podataka iz jednog konzistentnog stanja u drugo. Ako pretpostavimo da je originalni redak *r* u bazu podataka dodan u vremenu *t1*, zamijenjen u vremenu *t2* i obrisan u vremenu *t3*, tada se interval od *t1* do *t2* naziva *transakcijsko vrijeme* za proširenje od *p* sa stvarnim vremenom od 1. svibnja 2015. do 30. travnja 2016. godine. Interval od *t2* do *t3* je transakcijsko vrijeme za proširenje od *p* sa stvarnim vremenom od 1. travnja 2015. do 30. travnja 2016. godine. Općenito, transakcijska vremena mogu biti skupovi takvih intervala.

Potrebno je istaknuti činjenicu da se stvarno vrijeme može ažurirati, no transakcijsko vrijeme ne može. Stvarno vrijeme reflektira naše znanje o povijesti. S druge strane, transakcijsko vrijeme predstavlja povijest. Kasnije u radu ponovno ćemo se osvrnuti na ove koncepte.

1.2 Temporalne značajke u netemporalnoj bazi podataka

Pretpostavimo da vrijedi sljedeće:

1. Dobavljač ne može raskinuti ugovor jedan dan i sklopiti novi ugovor odmah drugi dan.
2. Dobavljač ne može imati dva različita ugovora u isto vrijeme.
3. Dobavljač može imati ugovor koji vrijedi do daljnjega, odnosno završetak ugovora trenutačno nije poznat.

Baza podataka o dobavljačima i pošiljkama je netemporalna, odnosno sadrži samo podatke koji trenutačno vrijede. Ilustrirat ćemo neke od problema koji se pojavljuju kada netemporalnim bazama podataka pokušamo dodati neke temporalne značajke. Promatramo pojednostavljeni prikaz baze podataka o dobavljačima i pošiljkama.

S

SNO
S1
S2
S3
S4
S5

SP

SNO	PNO
S1	P1
S1	P2
S1	P3
S1	P4
S1	P5
S1	P6
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4
S4	P5

Slika 1.1: Pojednostavljeni prikaz baze podataka o dobavljačima i pošiljkama

Pojednostavili smo relaciju S, a relacija SP je ostala nepromijenjena. Sada se S može opisati predikatom: *Dobavljač SNO je pod ugovorom*. Dalje proučavamo primjere upita i ograničenja na bazu podataka s prethodne slike koju ćemo proširiti nekim temporalnim značajkama. Promjene koje se odnose na bazu podataka vidljive su na Slikama 1.2 i 1.3. Ograničenja se odnose na uvjete koje korektni i konzistentni podaci u bazi podataka moraju zadovoljavati, a upiti će biti analogoni upita u nastavku:

- **Upit A**: Pronađi brojeve dobavljača koji trenutačno imaju ugovor s barem jednim klijentom.

Zapis u jeziku Tutorial D: SP { SNO }.

- **Upit B**: Pronađi brojeve dobavljača koji trenutačno nemaju ugovor s nekim klijentom.
  Zapis u jeziku Tutorial D: S { SNO } MINUS SP { SNO }.

Uočimo da Upit A sadrži jednostavnu projekciju, a Upit B razliku takvih projekcija.

Polutemporalna verzija početnog primjera

S_SINCE

SNO	SINCE
S1	d04
S2	d07
S3	d03
S4	d04
S5	d02

SP_SINCE

SNO	PNO	SINCE
S1	P1	d04
S1	P2	d05
S1	P3	d09
S1	P4	d05
S1	P5	d04
S1	P6	d06
S2	P1	d08
S2	P2	d09
S3	P2	d08
S4	P2	d06
S4	P4	d04
S4	P5	d05

Slika 1.2: Pojednostavljeni prikaz baze podataka o dobavljačima i pošiljkama – polutemporalna verzija

Modificirali smo bazu podataka sa Slike 1.1 tako da smo relacijama S i SP dodali atribut SINCE te ih preimenovali u S_SINCE i SP_SINCE (Slika 1.2). Atribut SINCE predstavlja vremensku oznaku, a simboli oblika *d01*, *d02* itd., označavaju dan 1, dan 2 itd. Pritom, prepostavljamo da dan 1 neposredno prethodi danu 2, dan 2 neposredno prethodi danu 3 itd. Predikat za relaciju S_SINCE glasi: *Dobavljač SNO je pod ugovorom od dana SINCE*. Relacija SP_SINCE može se opisati predikatom: *Dobavljač SNO ima mogućnost dostavljanja pošiljaka klijentu PNO od dana SINCE*. Ključevi i strani ključ jednaki su onima u originalnoj netemporalnoj bazi podataka sa Slike 1.1. Dakle, definicije relacija u jeziku Tutorial D mogle bi izgledati ovako:

```
VAR S_SINCE BASE
    RELATION { SNO SNO, SINCE DATE }
    KEY { SNO } ;

VAR SP_SINCE BASE
    RELATION { SNO SNO, PNO PNO, SINCE DATE }
    KEY { SNO, PNO }
    FOREIGN KEY { SNO } REFERENCES S ;
```

Međutim, navedene definicije nisu dovoljne same po sebi za čuvanje integriteta baze podataka sa Slike 1.2. U [2] je opisano: Čuvati integritet baze podataka znači čuvati korektnost i konzistentnost podataka. Korektnost znači da svaki pojedini podatak ima ispravnu vrijednost, a konzistentnost da su različiti podaci međusobno usklađeni. Dakle, zahvaljujući klauzuli FOREIGN KEY ... REFERENCES uvedeno je ograničenje za čuvanje integriteta stranog ključa, no potrebno je uvesti i dodatno ograničenje koje će osigurati da dobavljač ne može dostavljati pošiljke klijentu prije nego što je sklopio pripadni ugovor. Slijedi formulacija ograničenja u jeziku Tutorial D:

```
CONSTRAINT XST1
    IS_EMPTY ( ( ( SP_SINCE RENAME { SINCE AS SPS } ) JOIN
    ( S_SINCE RENAME { SINCE AS SS } ) ) WHERE SPS < SS ) ;
```

Navedena formulacija opisuje sljedeće: Ako se redak *sp* u SP_SINCE referira na redak *s* u S_SINCE, tada vrijednost od SINCE u *sp* ne smije biti manja od one u *s*. Zaključujemo da polutemporalne baze podataka poput one sa Slike 1.2 zahtijevaju uvođenje mnogo ograničenja koja su nerijetko prilično glomazna. Iz tog razloga ubrzo će nam zatrebati prikladne pokrate.

Sada promatramo polutemporalne analogone Upita A i B:

- **Upit C**: Pronađi brojeve dobavljača koji trenutačno imaju ugovor s barem jednim klijentom i datume od kada su u mogućnosti dostavljati pošiljke.
- **Upit D**: Pronađi brojeve dobavljača koji trenutačno nemaju ugovor s nekim klijentom i datume od kada više nisu u mogućnosti dostavljati pošiljke.

Ako dobavljač S*x* trenutačno ima mogućnost dostavljati pošiljke svim klijentima, tada očito S*x* ima ugovor s barem jednim klijentom od najranijeg SINCE datuma za tog dobavljača u relaciji SP_SINCE. Na primjer, ako je S*x* S1, tada je najraniji SINCE datum jednak *d04*. Slijedi zapis Upita C u jeziku Tutorial D:

```
EXTEND SP_SINCE { SNO } : { SINCE := MIN ( !!SP_SINCE, SINCE ) }.
```

Ovo je rezultat Upita C:

SNO	SINCE
S1	d04
S2	d08
S3	d08
S4	d04

U našim podacima samo je jedan dobavljač, dobavljač S5, koji trenutačno nije u mogućnosti dostavljati pošiljke nekom klijentu, no ne možemo otkriti datum od kada to više nije u mogućnosti jer podaci u bazi podataka ne sadrže dovoljno informacija. Baza podataka je samo polutemporalna. Na primjer, pretpostavimo da je trenutačno dan 10. Možda je dobavljač S5 bio u mogućnosti dostavljati pošiljke od dana 2, kada je sklopio pripadni ugovor, do dana 9, ili nikada nije bio u mogućnosti dostavljati pošiljke klijentima. Kako bismo mogli odgovoriti na Upit D, potrebno je promijeniti bazu podataka tako da pokazuje koji dobavljač je kada (od – do) bio u mogućnosti dostavljati pošiljke klijentima. Ovim pitanjem bavimo se u nastavku.

Temporalna verzija početnog primjera

Slika 1.3 prikazuje temporalnu verziju baze podataka o dobavljačima i pošiljkama. Atribut SINCE je postao atribut FROM te je svaka relacija proširena dodatnom vremenskom oznakom – atributom TO. Sukladno tomu relacije su preimenovane u S_FROM_TO i SP_FROM_TO. Atributi FROM i TO zajedno predstavljaju vremenski interval tijekom kojeg je neka propozicija istinita. U početnim razmatranjima pretpostavili smo da je vrijeme u bazama podataka konačno, zato smo kao završnu točku uzeli dan 10. Međutim, ta pretpostavka potiče nas da se zapitamo kojim mehanizmom će se *d10* zamijeniti s *d11* u ponoć na dan 10. Ovim problemom detaljnije ćemo se baviti kasnije u radu. Budući da sada pamtimo i povijesne zapise, u bazi podataka sa Slike 1.3 više je redaka nego u njenim prethodnicama. Ova temporalna verzija baze uključuje sve podatke iz polutemporalne verzije sa Slike 1.2, osim što smo za dvije pošiljke dobavljača S4 za vrijednost atributa TO stavili datume koji prethode danu 10. Te dvije pošiljke pretvorili smo iz trenutne u povijesnu informaciju. Također, ova baza podataka uključuje i povijesnu informaciju koja se tiče ranijeg vremenskog intervala, od *d02* do *d04*, tijekom kojeg je dobavljač S2 pod ugovorom i u mogućnosti dostavljati pošiljke.

Predikat za relaciju S_FROM_TO glasi: *Dobavljač SNO je pod ugovorom tijekom perioda od dana FROM do dana TO*. Predikat za relaciju SP_FROM_TO je: *Dobavljač SNO ima mogućnost dostavljanja pošiljaka klijentu PNO tijekom perioda od dana FROM do dana TO*.

S_FROM_TO

SNO	FROM	TO
S1	d04	d10
S2	d02	d04
S2	d07	d10
S3	d03	d10
S4	d04	d10
S5	d02	d10

SP_FROM_TO

SNO	PNO	FROM	TO
S1	P1	d04	d10
S1	P2	d05	d10
S1	P3	d09	d10
S1	P4	d05	d10
S1	P5	d04	d10
S1	P6	d06	d10
S2	P1	d02	d04
S2	P1	d08	d10
S2	P2	d03	d03
S2	P2	d09	d10
S3	P2	d08	d10
S4	P2	d06	d09
S4	P4	d04	d08
S4	P5	d05	d10

Slika 1.3: Pojednostavljeni prikaz baze podataka o dobavljačima i pošiljkama – temporalna verzija s atributima FROM i TO

Dalje razmatramo ograničenja kojima se čuva integritet promatrane baze podataka. Prvo što moramo osigurati je da je vrijednost atributa TO veća od pripadne vrijednosti atributa FROM:

```
CONSTRAINT S_FROM_TO_OK IS_EMPTY
        ( S_FROM_TO WHERE TO < FROM ) ;

CONSTRAINT SP_FROM_TO_OK IS_EMPTY
        ( SP_FROM_TO WHERE TO < FROM ) ;
```

Uočimo da je atribut FROM uključen u primarni ključ za obje relacije sa Slike 1.3. Jednako tako mogli smo uključiti atribut TO umjesto atributa FROM. Zato relacije S_FROM_TO

i SP_FROM_TO obje imaju dva ključa i ne postoji očiti razlog zbog kojeg bismo jednog od njih odabrali kao primarni, no radi jednostavnosti odlučujemo se za prvu varijantu s atributom FROM. Slijede definicije relacija:

```
VAR S_FROM_TO BASE
    RELATION { SNO SNO, FROM DATE, TO DATE }
    KEY { SNO, FROM }
    KEY { SNO, TO } ;

VAR SP_FROM_TO BASE
    RELATION { SNO SNO, PNO PNO, FROM DATE, TO DATE }
    KEY { SNO, PNO, FROM }
    KEY { SNO, PNO, TO } ;
```

Dosadašnja dva ograničenja *_FROM_TO_OK* i četiri ograničenja koja se odnose na ključeve nisu dovoljna da obuhvate sve što želimo. Promotrimo zato relaciju S_FROM_TO. Ako u relaciji za dobavljača S*x* postoji redak gdje je vrijednost atributa FROM *f*, a vrijednost atributa TO *t*, tada u istoj relaciji za dobavljača S*x* ne smije postojati redak koji pokazuje da je S*x* sklopio ugovor na dan koji neposredno prethodi *f* ili na dan koji je neposredni sljedbenik od *t*. Na primjer, za dobavljača S1 u relaciji S_FROM_TO postoji samo jedan redak (s vrijednostima *f* = *d04* i *t* = *d10*):

SNO	FROM	TO
S1	d04	d10

Činjenica da je { SNO, FROM } ključ za relaciju S_FROM_TO nedovoljna je da spriječi pojavu dodatnih redaka za dobavljača S1 gdje se interval *from – to* preklapa s nekim postojećim u bazi. Slijedi ilustracija takvog retka s vrijednostima *f* = *d02* i *t* = *d06*:

SNO	FROM	TO
S1	d02	d06

Ta dva retka želimo sažeti u jedan s vrijednostima *f* = *d02* i *t* = *d10*:

SNO	FROM	TO
S1	d02	d10

Vidimo da je ideja sažimanja redaka vrlo bitna, inače bismo imali situaciju u kojoj dvaput opisujemo isto: *Dobavljač S1 je pod ugovorom na dan 4, 5 i 6*. Također, želimo izbjeći pojavu redaka u kojima se interval *from – to naslanja* na neki postojeći u bazi. Na primjer, to je redak s vrijednostima *f = d02* i *t = d03*:

SNO	FROM	TO
S1	d02	d03

Očito želimo da se i ta dva retka sažmu u jedan. Rezultat sažimanja isti je kao i prije:

SNO	FROM	TO
S1	d02	d10

Dakle, potrebno nam je ograničenje koje osigurava da su intervali FROM – TO za dobavljača S*x* međusobno disjunktni:

```
CONSTRAINT XFT1 IS_EMPTY (
    ( ( S_FROM_TO RENAME { FROM AS F1, TO AS T1 } ) JOIN
    ( S_FROM_TO RENAME { FROM AS F2, TO AS T2 } ) )
    WHERE (T1 ≥ F2 AND T2 ≥ F1) OR (F2 = T1+1 OR F1 = T2+1) ) ;
```

Vidimo da je ograničenje prilično složeno, no ono dobro ilustrira složenost ograničenja koja se pojavljuju u temporalnim bazama podataka poput one sa Slike 1.3. Ne smijemo zanemariti ni činjenicu da smo upotrijebili izraz T1+1 kako bismo označili neposrednog sljedbenika od T1. Naime, što će se dogoditi ako T1 označava *kraj vremena*? U poglavljima koja slijede više se bavimo ovim problemom.

Također, primijetimo da { SNO, FROM } nije strani ključ u relaciji SP_FROM_TO. Moguće je da kombinacije atributa SNO i FROM koje se pojavljuju u relaciji SP_FROM_TO nisu u S_FROM_TO. S druge strane, mora se osigurati da je skup dobavljača u SP_FROM_TO podskup skupa dobavljača u S_FROM_TO:

```
CONSTRAINT XFT2 SP_FROM_TO { SNO } ⊆ S_FROM_TO { SNO } ;
```

Ovo ograničenje možemo shvatiti kao generalizaciju ograničenja koje se odnosi na strane ključeve u relaciji. Međutim, još uvijek nismo osigurali da ako relacija SP_FROM_TO prikazuje da je dobavljač u mogućnosti dostavljati pošiljke tijekom nekog intervala, tada i relacija S_FROM_TO mora prikazivati da je taj isti dobavljač pod ugovorom tijekom tog istog razdoblja. Slijedi formulacija ograničenja:

```
CONSTRAINT XFT3
    COUNT ( SP_FROM_TO { SNO, FROM, TO } ) = COUNT (
    ( (SP_FROM_TO RENAME { FROM AS SPF, TO AS SPT }) {SNO, SPF, SPT}
    JOIN (S_FROM_TO RENAME { FROM AS SF, TO AS ST }) )
    WHERE SF ≤ SPF AND ST ≥ SPT ) ;
```

Uočimo da su sva ograničenja, koja smo prethodno razmatrali, uključena u zadnje. Ono intuitivno iskazuje da ako u relaciji SP_FROM_TO postoji redak koji pokazuje da je dobavljač S*x* u mogućnosti dostavljati pošiljke nekom klijentu od dana *spf* do dana *spt*, tada relacija S_FROM_TO mora sadržavati točno jedan redak koji pokazuje da je dobavljač S*x* pod ugovorom tijekom tog razdoblja. Opet zaključujemo da su ograničenja za čuvanje integriteta temporalne baze podataka poput one sa Slike 1.3 iznimno složena.

Okrenimo se sada temporalnim analogonima Upita A i B:

- **Upit E**: Pronađi trojke (SNO, FROM, TO) za dobavljače koji su u mogućnosti trenutačno dostavljati pošiljke barem jednom klijentu te gdje vrijednosti FROM i TO označavaju najveće vremensko razdoblje (ili maksimalni interval) tijekom kojeg je dobavljač SNO u mogućnosti dostavljati pošiljke.
- **Upit F**: Pronađi trojke (SNO, FROM, TO) za dobavljače koji trenutačno nisu u mogućnosti dostavljati pošiljke nekom klijentu tijekom nekog razdoblja i gdje vrijednosti FROM i TO označavaju najveće vremensko razdoblje (ili maksimalni interval) tijekom kojeg dobavljač SNO nije bio u mogućnosti dostavljati pošiljke.

Ovi upiti mogu se izraziti u jeziku Tutorial D, no taj je zadatak jako težak. Još jednom moramo primijetiti kako bi postojanje odgovarajućih pokrata za relacijske operacije bitno olakšalo taj posao. Kasnije u radu opisat ćemo neke od njih.

Dakle, osnovni problem s temporalnim podacima je taj što brzo vode do ograničenja i upita koji su nerazumljivi i pretjerani, osim ako sustav za upravljanje bazom podataka pruža odgovarajuću podršku. Nažalost, to dostupni sustavi obično ne čine.

1.3 Vremenski intervali

Kako bismo pravilno postupali s temporalnim podacima, bitno je uočiti potrebu za vremenskim intervalima umjesto parova FROM i TO vrijednosti. Pritom, razmatramo vremenske intervale u kontekstu baza podataka. U nastavku se bavimo značenjem *intervala od dana 4 do dana 10*. Očito je da su dani 5, 6, 7, 8 i 9 uključeni, no dani 4 i 10 ne moraju biti. Ako za neki interval *i* kažemo da se *proteže* od *b* do *e*, točke *b* i *e* mogu pripadati ili ne pripadati intervalu *i*. Ako točka *b* pripada *i*, tada kažemo da je *i* zatvoren na početku,

inače je otvoren na početku. Slično, ako točka *e* pripada *i*, tada kažemo da je *i* zatvoren na kraju, inače je otvoren na kraju. Interval *i* intuitivno shvaćamo kao uređeni niz točaka te označavamo pomoću istaknutih točaka *b* i *e* koje su odvojene dvotočkom i odgovorajućom zagradom na početku i kraju ovisno o tome je li *i* otvoren ili zatvoren na početku i kraju. Notacija je slična matematičkoj: *[d04:d10]*, *[d04:d10)*, *(d04:d10]*, *(d04:d10)*. Sada kada interval kao što je *[d04:d10]* možemo promatrati kao zasebnu cjelinu, potpuno je smisleno kombinaciju atributa FROM i TO iz obje relacije sa Slike 1.3 zamijeniti jednim atributom DURING čije su vrijednosti intervali.

S_DURING

SNO	DURING
S1	[d04:d10]
S2	[d02:d04]
S2	[d07:d10]
S3	[d03:d10]
S4	[d04:d10]
S5	[d02:d10]

SP_DURING

SNO	PNO	DURING
S1	P1	[d04:d10]
S1	P2	[d05:d10]
S1	P3	[d09:d10]
S1	P4	[d05:d10]
S1	P5	[d04:d10]
S1	P6	[d06:d10]
S2	P1	[d02:d04]
S2	P1	[d08:d10]
S2	P2	[d03:d03]
S2	P2	[d09:d10]
S3	P2	[d08:d10]
S4	P2	[d06:d09]
S4	P4	[d04:d08]
S4	P5	[d05:d10]

Slika 1.4: Pojednostavljeni prikaz baze podataka o dobavljačima i pošiljkama – temporalna verzija s atributom DURING

Relacija S_DURING može se opisati sljedećim predikatom: *Dobavljač SNO je pod ugovorom tijekom perioda od dana koji je početna točka od DURING do dana koji je završna točka od DURING*. Predikat za relaciju SP_DURING glasi: *Dobavljač SNO ima mogućnost dostavljanja pošiljaka klijentu PNO tijekom perioda od dana koji je početna točka od DURING do dana koji je završna točka od DURING*. Zamjena atributa FROM i TO atributom DURING ima mnogo prednosti:

1. Relacija S_FROM_TO ima dva ključa, { SNO, FROM } i { SNO, TO }, no relacija S_DURING ima samo jedan, { SNO, DURING }, koji možemo proglasiti primarnim ključem bez nepoželjnog biranja. Slično, relacija SP_FROM_TO ima dva ključa, ali relacija SP_DURING ima samo jedan, { SNO, PNO, DURING }, koji možemo proglasiti primarnim ako želimo. U svakom slučaju, redukcija broja ključeva je pozitivna promjena.
2. Vrijednosti atributa FROM i TO određuju interval *from – to* koji se može interpretirati kao zatvoreni ili otvoreni. U prethodnim razmatranjima podrazumijevalo se da su vremenski intervali zatvoreni. Uvođenjem atributa DURING postoje četiri različite reprezentacije istog intervala te možemo koristiti onu koja je u tom trenutku najprikladnija. Na primjer, *[d04:d10]*, *[d04:d11)*, *(d03:d10]* ili *(d03:d11)*.
3. Eksplicitna ograničenja oblika FROM ≤ TO više nisu potrebna, već su zamijenjena efikasnijim ograničenjima koja se implicitno primjenjuju na intervale i tipove intervala.

Ograničenja za čuvanje integriteta i analogoni Upita A i B mogu se promatrati za bazu podataka sa Slike 1.4, no zasad nećemo ulaziti u detalje. Bitno je naglasiti da vremenski intervali koje proučavamo nemaju vidljive komponente, odnosno početnoj i završnoj točki nekog intervala može se pristupiti tek pomoću odgovarajućih funkcija.

U prethodnim razmatranjima opisali smo kako intuitivno shvaćamo vremenske intervale, u nastavku ćemo malo formalnije pristupiti tom pojmu. Vratimo se još jednom na početni interval *[d04:d10]*. Nazovimo ga kratko interval *i*. Jasno je da su točke koje čine interval *i* dani, tj. *d04*, *d05*, *d06*, *d07*, *d08*, *d09* i *d10*. Pretpostavimo da su te točke tipa DATE koji, kao i prije, reprezentira kalendarski datum. Kažemo da je tip DATE *tip točaka* u intervalu *i*.

Znamo da interval *i* sadrži početnu i završnu točku, tj. *d04* i *d10*. Također, znamo da su točke u *i* uređene po nekom pravilu. Kako bismo odredili sve točke u *i*, prvo moramo odrediti točku koja je neposredni sljedbenik početne točke *d04* u skladu s tim pravilom. Točka *d04+1* je sljedbenik od *d04*, a određuje se pomoću odgovarajuće *funkcije sljedbenika*. U našem slučaju funkcija sljedbenika je *sljedeći dan*, odnosno *dodaj jedan dan danom datumu*. Uočimo još da ako je *d04+1* sljedbenik od *d04*, tada je *d04* prethodnik od *d04+1*. Sada kada smo utvrdili da je *d04+1* sljedbenik od *d04*, sljedeće što moramo odrediti je dolazi li *d04+1* nakon završne točke *d10* u skladu s funkcijom sljedbenika. Ako ne dolazi, tada je točka *d04+1* u intervalu *i* te dalje na analogan način promatramo točku *d04+2*. Ponavljajući ovaj postupak možemo otkriti sve točke u nizu *i = [d04:d10]*. Primijetimo još da ako je *e* završna točka tipa DATE u intervalu *i*, tada izraz *e+1* nije definiran. Ako je *b* početna točka tipa DATE u intervalu *i*, tada izraz *b−1* nije definiran. Slijedi definicija tipa točaka u intervalu *i*.

Definicija 1.3.1. Neka su T neki tip podataka i i interval. Kažemo da je T **tip točaka** u intervalu i ako vrijedi sljedeće:

1. Za svaka dva različita elementa tipa T *v1* i *v2* vrijedi točno jedno: *v1* > *v2* ili *v2* > *v1*.
2. Definirani su operatori *first* i *last* koji vraćaju najmanju i najveću vrijednost tipa T u skladu s totalnim uređajem pod 1.
3. Definirani su operatori *next* i *prior* koji vraćaju sljedbenika i prethodnika (ako postoje) za danu vrijednost tipa T u skladu s totalnim uređajem pod 1.

Operator *next* je funkcija sljedbenika. Također, možemo pretpostaviti da su svi operatori usporedbe dostupni za bilo koji par točaka tipa T: $=, \neq, >, \geq, <, \leq$.

Vratimo se ponovno intervalu *i = [d04:d10]*. Vrijednost promatranog intervala je interval od jedne vrijednosti DATE do druge gdje je prva vrijednost manja ili jednaka drugoj. Drugim riječima, vrijednost *i = [d04:d10]* je posebni tip podataka koji se zove INTERVAL_DATE. Intervali ovoga tipa sastoje se od točaka koje su tipa DATE. Konačno slijedi definicija pojma *interval*.

Definicija 1.3.2. Neka je T tip točaka. Tada je vrijednost intervala ili **interval *i*** tipa INTERVAL_T vrijednost za koju su definirani unarni operatori BEGIN i END te binarni operator $\in$ tako da vrijedi sljedeće:

1. BEGIN(i) i END(i) vraćaju vrijednost tipa T.
2. BEGIN(i) $\leq$ END(i).
3. Ako je p vrijednost tipa T, tada je $p \in i$ ako i samo ako je BEGIN(i) $\leq p$ te $p \leq$ END(i).

Uočimo da interval nikad nije prazan. Uvijek postoji barem jedna točka u bilo kojem intervalu. Na kraju, slijede nepotpune definicije relacija S_DURING i SP_DURING:

```
VAR S_DURING BASE
    RELATION { SNO SNO, DURING INTERVAL_DATE }
    KEY { SNO, DURING } ;

VAR SP_DURING BASE
    RELATION { SNO SNO, PNO PNO, DURING INTERVAL_DATE }
    KEY { SNO, PNO, DURING } ;
```

1.4 Operatori za rad s intervalima

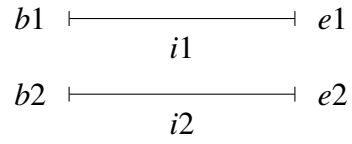
U nastavku se bavimo opisom operatora za rad s intervalima. Prisjetimo se prvo ključnih zaključaka iz prethodnog potpoglavlja:

1. Neka su *T* tip točaka i *p* neka vrijednost tipa *T*. Izrazom *p+1* označavamo vrijednost sljedbenika od *p*. Ova notacija je samo neformalna, pravi jezik trebao bi imati eksplicitni operator *next* koji ćemo nazvati NEXT_*T*. NEXT_*T*(*p*) vraća vrijednost *p+1*. Također, izrazom *p−1* neformalno označavamo vrijednost čiji je sljedbenik *p*. Pravi jezik trebao bi imati eksplicitni operator *prior* koji ćemo nazvati PRIOR_*T*. PRIOR_*T*(*p*) vraća vrijednost *p−1*. Potrebni su i operatori FIRST_*T* i LAST_*T* koji vraćaju najmanju i najveću vrijednost tipa *T*.
2. Neka je INTERVAL_*T* tip intervala koji odgovara tipu točaka *T*. Neformalno koristimo izraz *[p1:pn]* kako bismo označili interval koji sadrži točke *p1*, *p1+1*, *p1+2*, ..., *pn*. Pravi jezik trebao bi sadržavati neku vrstu eksplicitne sintakse za *[p1:pn]*. Na primjer, INTERVAL_*T*(*[p1:pn]*).
3. Postoje različiti stilovi za intervale. Na primjer, neka je INTEGER tip točaka. Tada je pripadni tip intervala INTERVAL_INTEGER. Svi (neformalni) izrazi *[2:4]*, *[2:5)*, *(1:4]* i *(1:5)* odnose se na interval koji sadrži točke 2, 3 i 4. Odgovarajući formalni analogoni tih izraza su: INTERVAL_INTEGER(*[2:4]*), INTERVAL_INTEGER(*[2:5)*), INTERVAL_INTEGER(*(1:4]*) i INTERVAL_INTEGER(*(1:5)*).
4. Neka su *i = [b:e]* interval tipa INTERVAL_*T* i *p* vrijednost tipa *T*. Tada operator BEGIN(*i*) vraća *b*; operator END(*i*) vraća *e*; $p \in i$ vraća TRUE ako i samo ako je $b \leq p$ i $p \leq e$.

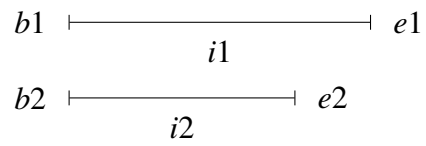
Također, uz pretpostavke pod 4 definiramo: operator PRE(*i*) vraća *b−1*; POST(*i*) vraća *e+1*; $i \ni p$ (*i sadrži kao člana p*) vraća TRUE ako i samo ako je $p \in i$ istinito. Uočimo da PRE(*i*) i POST(*i*) možemo zapisati na sljedeći način: PRIOR_*T*(BEGIN(*i*)) i NEXT_*T*(END(*i*)). U literaturi postoje i drugačija imena za ove operatore. Konačno, neka je *i = [p:p]* interval tipa INTERVAL_*T*. Definiramo operator POINT FROM *i* koji vraća vrijednost točke *p*. Ako *i* nije interval oblika *[p:p]*, tada je taj operator nedefiniran.

Sada se bavimo opisom nekoliko operatora za usporedbu intervala. Operatori koje razmatramo u literaturi su često poznati pod nazivom Allenovi operatori. James F. Allen 1983. godine prvi je definirao operatore za usporedbu intervala u temporalnim upitima. Pretpostavimo da su intervali *i1 = [b1:e1]* i *i2 = [b2:e2]* tipa INTERVAL_*T*. Uočimo da intervali nužno moraju biti istog tipa.

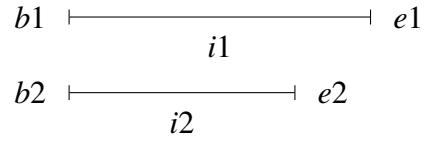
- **Operator =**: *i1 = i2* je istinito ako i samo ako je *b1 = b2* i *e1 = e2*.



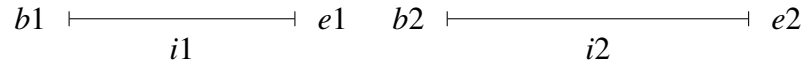
- **Operatori ⊇ i ⊆** : *i1* ⊇ *i2* je istinito ako i samo ako je *b1* ≤ *b2* i *e1* ≥ *e2*; *i2* ⊆ *i1* je istinito ako i samo ako je *i1* ⊇ *i2* istinito.



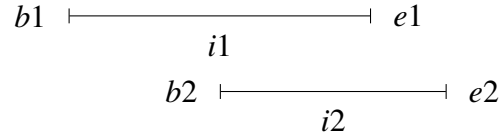
- **Operatori ⊃ i ⊂**: *i1* ⊃ *i2* je istinito ako i samo ako je *i1* ⊇ *i2* istinito i *i1* = *i2* nije istinito; *i2* ⊂ *i1* je istinito ako i samo ako je *i1* ⊃ *i2* istinito.



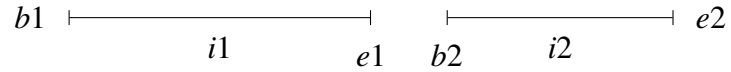
- **Operatori BEFORE i AFTER**: *i1* BEFORE *i2* je istinito ako i samo ako je *e1* < *b2*; *i2* AFTER *i1* je istinito ako i samo ako je *i1* BEFORE *i2* istinito.



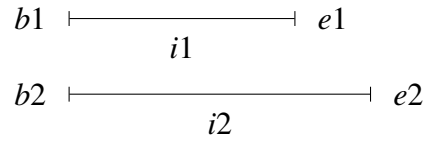
- **Operator OVERLAPS**: *i1* OVERLAPS *i2* je istinito ako i samo ako je *b1* ≤ *e2* i *b2* ≤ *e1*. Operator OVERLAPS je komutativan.



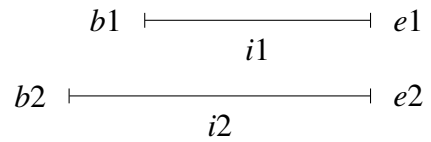
- **Operator MEETS**: *i1* MEETS *i2* je istinito ako i samo ako je *b2* = *e1+1* ili *b1* = *e2+1*. Operator MEETS je komutativan.



- **Operator MERGES**: *i1* MERGES *i2* je istinito ako i samo ako je *i1* OVERLAPS *i2* istinito ili je *i1* MEETS *i2* istinito. Operator MERGES je komutativan. Ključna riječ MERGES možda nije dovoljna intuitivna, no teško je pronaći riječ koja je prikladnija.
- **Operator BEGINS**: *i1* BEGINS *i2* je istinito ako i samo ako je *b1* = *b2* i *e1* ∈ *i2* istinito. Operator BEGINS u literaturi je poznat i pod nazivom *starts*.



- **Operator ENDS**: *i1* ENDS *i2* je istinito ako i samo ako je $e1 = e2$ i $b1 \in i2$ istinito. Operator ENDS u literaturi je poznat i pod nazivom *finishes*.



Negacije navedenih operatora se također mogu definirati. Na primjer, *i1* NOT MEETS *i2* je istinito ako i samo ako *i1* MEETS *i2* nije istinito. Naravno, NOT = je jednostavno $\neq$, NOT $\supseteq$, NOT $\subseteq$, NOT $\supset$ i NOT $\subset$ su redom $\subset$, $\supset$, $\subseteq$ i $\supseteq$.

Operator $\supset$ omogućuje nam da precizno definiramo pojam *maksimalni interval*.

Definicija 1.4.1. Neka je P predikat čiji je primarni ključ tipa nekog intervala. Kažemo da je interval i **maksimalan** obzirom na P ako i samo ako i zadovoljava P i ne postoji interval j takav da je $j \supset i$ koji zadovoljava P.

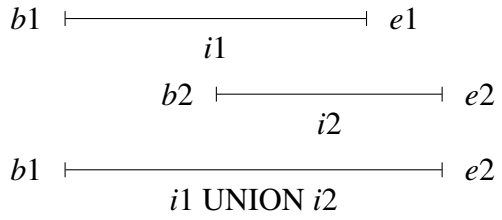
Skupovni operatori UNION, INTERSECT i MINUS mogu se izravno primijeniti na intervale ako ih shvatimo kao skup točaka. Skupovni operatori vraćaju skupove, no bitno je uočiti da njihova povratna vrijednost u ovom slučaju neće uvijek biti interval. Na primjer, ako su intervali *i1* i *i2* međusobno disjunktni, odnosno *i1* NOT OVERLAPS *i2* je istinito, tada je presjek *i1* INTERSECT *i2* prazan, a intervali kao takvi nikad nisu prazni. Zato je potrebno uvesti neka ograničenja na operande.

Definiramo dva pomoćna operatora MIN i MAX. Neka su *p1* i *p2* vrijednosti tipa T i neka je relacija $<$ definirana za taj tip podataka. Tada vrijedi:

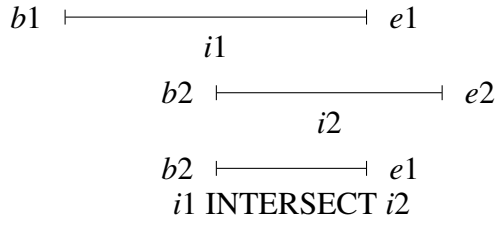
- MAX {*p1*, *p2*} vraća *p2* ako je $p1 < p2$, inače vraća *p1*.
- MIN {*p1*, *p2*} vraća *p1* ako je $p1 < p2$, inače vraća *p2*.

Neka su sada *i1* i *i2* vrijednosti istog tipa intervala. Definiramo skupovne operatore za intervale:

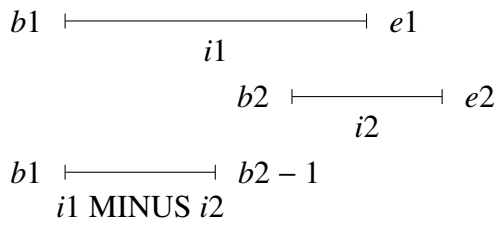
- **UNION**: *i1* UNION *i2* vraća [MIN{*b1*, *b2*}:MAX{*e1*, *e2*}] ako je *i1* MERGES *i2* istinito, inače nije definirano.



- **INTERSECT**: *i1* INTERSECT *i2* vraća [MAX{*b1*, *b2*}:MIN{*e1*, *e2*}] ako je *i1* OVERLAPS *i2* istinito, inače nije definirano.



- **MINUS**: *i1* MINUS *i2* vraća [*b1*:MIN{*b2−1*, *e1*}] ako je $b1 < b2$ i $e1 \leq e2$; [MAX{*e2+1*, *b1*}:*e1*] ako je $b1 \geq b2$ i $e1 > e2$; inače nije definirano.



Konačno, definiramo operator **COUNT**(*i*) čija je povratna vrijednost broj točaka u intervalu *i*. Taj operator se ponekad naziva duljina ili trajanje intervala. Na primjer, ako promatramo interval *i* = *[d02:d05]* koji je tipa INTERVAL_DATE, tada je COUNT(*i*) jednak 4.

Svi operatori za intervale koje smo prethodno opisali mogu se iskoristiti u postavljanju temporalnih upita i mogu se primijeniti na sve moguće intervale. Uzimajući u obzir temporalnu bazu podataka sa Slike 1.4, promotrimo sljedeće upite i njihove moguće zapise u jeziku Tutorial D:

1. Pronađi brojeve dobavljača koji su u mogućnosti dostavljati pošiljke klijentu P2 na dan 8.
   Tutorial D:
   (SP_DURING WHERE PNO = PNO('P2') AND d08 ∈ DURING) {SNO}.
2. Pronađi parove dobavljača koji su u mogućnosti dostavljati pošiljke istim klijentima u isto vrijeme.
   Tutorial D:

```
WITH (
    t1 := SP_DURING RENAME { SNO AS S1, DURING AS D1 } ,
    t2 := SP_DURING RENAME { SNO AS S2, DURING AS D2 } ,
    t3 := t1 JOIN t2 ,
    t4 := t3 WHERE S1 < S2 AND D1 OVERLAPS D2 ) :
t4 { S1, S2 }.
```

3. Pronađi ne samo parove dobavljača koji su u mogućnosti dostavljati pošiljke istim klijentima u isto vrijeme nego i pripadajuće klijente i vrijeme.
   Tutorial D:

```
WITH (
    t1 := SP_DURING RENAME { SNO AS S1, DURING AS D1 } ,
    t2 := SP_DURING RENAME { SNO AS S2, DURING AS D2 } ,
    t3 := t1 JOIN t2 ,
    t4 := t3 WHERE S1 < S2 AND D1 OVERLAPS D2 ) :
    t5 := EXTEND t4 : { DURING := D1 INTERSECT D2 } ) :
t5 { S1, S2, PNO, DURING }.
```

Zapisi upita u jeziku Tutorial D su intuitivno dosta jasni. Primijetimo samo da bi se *d08* u pravom jeziku morao zamijeniti odgovarajućom instancom tipa DATE. Na primjer, ako se dan 8 odnosi na 22. rujna 2015., tada je odgovarajuća instanca u jeziku SQL: DATE '2015-09-22'.

1.5 Operatori EXPAND i COLLAPSE

Za razliku od operatora koje smo prethodno susreli, operatori EXPAND i COLLAPSE ne primjenjuju se na intervale kao takve, nego na skupove intervala. Preciznije, operandi su skupovi intervala istog tipa, a rezultat je ponovno skup intervala istog tipa. Rezultirajući skupovi intervala mogu se smatrati određenom kanonskom formom za ulazni skup intervala. Dvije kanonske forme koje dobivamo primjenom tih operatora imaju svojstvo da je svaka točka iz nekog intervala u ulaznom skupu reprezentirana točno jednom u pripadnom rezultantnom skupu.

Definicija 1.5.1. Neka su *X1* i *X2* skupovi intervala istog tipa. Skupovi *X1* i *X2* su **ekvivalentni** ako i samo ako je skup svih točaka koje se nalaze u intervalima iz *X1* jednak skupu svih točaka koje se nalaze u intervalima iz *X2*.

Slijedi primjer ekvivalentnih skupova u skladu s prethodnom definicijom. Neka su *X1* = { *[d01:d01], [d03:d05], [d04:d06]* } i *X2* = { *[d01:d01], [d03:d04], [d05:d05], [d05:d06]* }. Očito *X1* i *X2* nisu jednaki skupovi, no jednostavno je primijetiti da su ekvivalentni. Traženi skupovi svih točaka za *X1* i *X2* su jednaki: { *d01, d03, d04, d05, d06* }. Međutim, ono što nas zanima nisu skupovi svih točaka kao takvi, nego odgovarajući skup koji se

sastoji od intervala kojima pripada samo jedna točka: { *[d01:d01], [d03:d03], [d04:d04], [d05:d05], [d06:d06]* }. Ovaj je skup očito ekvivalentan skupovima *X1* i *X2* te predstavlja *proširenu formu* tih skupova. Nazovimo ga skupom *X3*. Slijedi definicija proširene forme.

Definicija 1.5.2. Neka je *X* skup intervala istog tipa. **Proširena forma** od *X* je skup svih intervala oblika *[p:p]* gdje je *p* točka koja pripada nekom intervalu iz *X*.

Ako je *X* skup intervala istog tipa, jasno je da proširena forma od *X* uvijek postoji i da je ekvivalentna tom skupu. Proširena forma je jedinstvena i predstavlja jednu kanonsku formu za *X*. Ovdje je riječ o jedinstvenom skupu koji je ekvivalentan *X* te sadrži intervale minimalne duljine (jedan). Ako je skup *X* prazan, tada je i proširena forma od *X* prazna. Koncept proširenih formi omogućuje nam da sažmemo Definiciju 1.5.1.

Definicija 1.5.3. Dva skupa intervala istog tipa su **ekvivalentna** ako i samo ako imaju iste proširene forme.

Skupovi *X1*, *X2* i *X3* koje smo prethodno razmatrali imaju različitu kardinalnost. Uočimo da najveću kardinalnost ima skup *X3*, a upravo je *X3* proširena forma skupova *X1* i *X2*. Lako je pronaći neki skup *X4* koji ima istu proširenu formu kao *X1* i *X2* te veću kardinalnost od *X3*. No mnogo je zanimljiviji skup *X5* koji ima istu proširenu formu kao *X1* i *X2* te najmanju moguću kardinalnost: *X5* = { *[d01:d01], [d03:d06]* }. *X5* je *sažeta forma* od *X1, X2, X3* i *X4*. Slijedi definicija sažete forme.

Definicija 1.5.4. Neka je *X* skup intervala istog tipa. **Sažeta forma** od *X* je skup svih intervala *Y* istog tipa tako da vrijedi sljedeće:

1. *X* i *Y* imaju istu proširenu formu.
2. Ne postoje dva različita intervala *i1* i *i2* iz *Y* tako da je *i1* MERGES *i2* istinito. Analogno, ne postoje dva različita intervala *i1* i *i2* iz *Y* tako da je *i1* UNION *i2* definirano.

Primijetimo da se *Y* može izvesti iz *X* tako da se parovi intervala iz *X* zamijene njihovom unijom, pod pretpostavkom da je njihova unija definirana, sve dok daljnje zamjene više nisu moguće. Sažeta forma od *X* je druga moguća kanonska forma za *X*. Riječ je o jedinstvenom skupu koji je ekvivalentan *X* s najmanjom mogućom kardinalnošću. Također, uočimo da sažeta forma od *X* uvijek postoji i da je jedinstvena. Koncept sažetih formi ponovno nam omogućuje da preoblikujemo Definiciju 1.5.1.

Definicija 1.5.5. Dva skupa intervala istog tipa su **ekvivalentna** ako i samo ako imaju iste sažete forme.

Sada možemo definirati operatore EXPAND i COLLAPSE.

Definicija 1.5.6. Neka je X skup intervala istog tipa. Tada EXPAND(X) vraća proširenu formu od X, a COLLAPSE(X) saženu formu od X.

Opisali smo dva operatora, EXPAND i COLLAPSE, koji se primjenjuju na skupove. No način na koji smo opisali ta dva operatora ne pristaje dobro relacijskom modelu budući da se on temelji na radu s relacijama. Kako bismo ostali u okviru relacijskog modela, potrebno je zamijeniti navedene operatore verzijama u kojima su argumenti unarne relacije umjesto skupova. Verzije operatora s unarnim relacijama su slične onima sa skupovima, samo što su umjesto skupova intervala ulaz i izlaz sada unarne relacije koje sadrže te intervale. Na primjer, neka je r ulazna relacija:

DURING
[d04:d09]
[d01:d01]
[d07:d10]
[d03:d08]

Tada EXPAND(r) daje izlaz na lijevoj strani, a COLLAPSE(r) daje izlaz na desnoj strani:

DURING
[d01:d01]
[d03:d03]
[d04:d04]
[d05:d05]
[d06:d06]
[d07:d07]
[d08:d08]
[d09:d09]
[d10:d10]

DURING
[d01:d01]
[d03:d10]

Slično, pojam ekvivalentnosti može se iskazati i za unarne relacije: Dvije unarne relacije su ekvivalentne ako i samo ako imaju iste proširene forme (ili iste sažete forme). U nastavku koristimo samo verzije operatora EXPAND i COLLAPSE koje su definirane za unarne relacije.

1.6 Operatori PACK i UNPACK

Operatori EXPAND i COLLAPSE koje smo prethodno opisali još uvijek nisu dovoljno dobri za uspješno upravljanje temporalnim podacima. Problem je što ti operatori rade s unarnim relacijama, a nama su potrebni operatori koji rade s *n*–arnim relacijama. Zato uvodimo operatore PACK i UNPACK. Operator PACK temelji se na operatoru COLLAPSE, a operator UNPACK na operatoru EXPAND.

Promotrimo kako operatori PACK i UNPACK djeluju na relaciju *r*:

SNO	DURING
S2	[d02:d04]
S2	[d03:d05]
S4	[d02:d05]
S4	[d04:d06]
S4	[d09:d10]

Pakiranjem relacije *r* na atribut DURING dobivamo *pakiranu formu* relacije *r*:

SNO	DURING
S2	[d02:d05]
S4	[d02:d06]
S4	[d09:d10]

Pakiranje se formalno zapisuje kao PACK *r* ON (DURING). Rezultat prikazuje iste informacije kao i originalna relacija *r*, no on je restrukturiran tako da za dva *during* intervala istog dobavljača *i1* i *i2* vrijedi da je *i1* NOT MERGES *i2* istinito. Značajnost operatora COLLAPSE u takvom restrukturiranju relacije *r* je očita.

Analogno, *raspakiravanjem* originalne relacije *r* na atribut DURING, što formalno zapisujemo kao UNPACK *r* ON (DURING), dobivamo *raspakiranu formu* relacije *r*. Rezultat raspakiravanja prikazuje iste informacije kao i originalna relacija *r*, no on je restrukturiran tako da su vrijednosti atributa DURING intervali kojima pripada samo jedna točka.

Važnost operatora EXPAND u takvom restrukturiranju relacije *r* je očita:

SNO	DURING
S2	[d02:d02]
S2	[d03:d03]
S2	[d04:d04]
S2	[d05:d05]
S4	[d02:d02]
S4	[d03:d03]
S4	[d04:d04]
S4	[d05:d05]
S4	[d06:d06]
S4	[d09:d09]
S4	[d10:d10]

Sada ćemo malo detaljnije objasniti operatore PACK i UNPACK, no prisjetimo se prvo Upita E i F iz Potpoglavlja 1.2:

- **Upit E**: Pronađi trojke (SNO, FROM, TO) za dobavljače koji su u mogućnosti trenutačno dostavljati pošiljke barem jednom klijentu te gdje vrijednosti FROM i TO označavaju maksimalni interval tijekom kojeg je dobavljač SNO u mogućnosti dostavljati pošiljke.
- **Upit F**: Pronađi trojke (SNO, FROM, TO) za dobavljače koji trenutačno nisu u mogućnosti dostavljati pošiljke nekom klijentu tijekom nekog razdoblja i gdje vrijednosti FROM i TO označavaju maksimalni interval tijekom kojeg dobavljač SNO nije bio u mogućnosti dostavljati pošiljke.

U Potpoglavlju 1.2 ove upite nismo zapisali u jeziku Tutorial D jer je taj zadatak bio dosta težak. No koristeći operatore PACK i UNPACK, mnogo je jednostavnije pristupiti tom problemu. Upite E i F preoblikovat ćemo tako da odgovaraju temporalnoj bazi podataka s atributom DURING sa Slike 1.4:

- **Upit G**: Pronađi parove SNO i DURING gdje je DURING maksimalni interval tijekom kojeg je dobavljač SNO u mogućnosti dostavljati pošiljke barem jednom klijentu.
- **Upit H**: Pronađi parove SNO i DURING gdje je DURING maksimalni interval tijekom kojeg dobavljač SNO nije bio u mogućnosti dostavljati pošiljke nekom klijentu.

Prikaz rezultata Upita G je na lijevoj strani, a Upita H na desnoj strani:

SNO	DURING
S1	[d04:d10]
S2	[d02:d04]
S2	[d08:d10]
S3	[d08:d10]
S4	[d04:d10]

SNO	DURING
S2	[d07:d07]
S3	[d03:d07]
S5	[d02:d10]

Osvrnimo se prvo na Upit G. Jasno je da se na Upit G može odgovoriti uzimajući u obzir samo relaciju SP_DURING. Rezultat (lijevo) dobivamo evaluacijom sljedećeg izraza:

```
WITH (
    t1 := SP_DURING { SNO, DURING } ,
    t2 := t1 GROUP { DURING } AS X ,
    t3 := EXTEND t2 : { X := COLLAPSE(X) } ) :
t3 UNGROUP { X }.
```

Bilo bi vrlo poželjno kada bismo od relacije *t1* stigli do *rezultata* pomoću samo jedne operacije. Iz tog razloga uvodimo operator PACK čija formalna definicija slijedi u nastavku.

Definicija 1.6.1. Neka je *r* relacija koja ima atribut *A* čije su vrijednosti intervali. Tada izraz PACK *r* ON (*A*) označava pakiranje relacije *r* na atribut *A* te predstavlja pokratu za sljedeće:

```
WITH ( r1 := r GROUP { A } AS X ,
    r2 := EXTEND r1 : { X := COLLAPSE(X) } ) :
r2 UNGROUP { X }.
```

Dakle, sljedeći zapis možemo ponuditi kao razumnu formulaciju Upita G:

```
PACK SP_DURING { SNO, DURING } ON (DURING).
```

Promotrimo sada Upit H. Intuitivno je jasno da se na Upit H može odgovoriti uzimajući u obzir obje relacije sa Slike 1.4. Uočimo da za SNO–DURING parove vrijedi sljedeće:

1. SNO–DURING parovi koji su sadržani u S_DURING pokazuju kada su dobavljači bili pod ugovorom.
2. SNO–DURING parovi koji su sadržani u SP_DURING pokazuju kada su dobavljači bili u mogućnosti dostavljati pošiljke klijentima.

3. Razlika prvog i drugog pokazuje kada su dobavljači bili pod ugovorom, ali ne i u mogućnosti dostavljati pošiljke klijentima.

Dakle, kako bismo odgovorili na Upit H, potrebno je izvesti nekoliko operacija raspakiravanja i uzeti razliku rezultata. Predstavimo prvo operator UNPACK.

Definicija 1.6.2. Neka je *r* relacija koja ima atribut *A* čije su vrijednosti intervali. Tada izraz UNPACK *r* ON (*A*) označava raspakiravanje relacije *r* na atribut *A* te predstavlja pokratu za sljedeće:

```
WITH ( r1 := r GROUP { A } AS X ,
     r2 := EXTEND r1 : { X := EXPAND(X) } ) :
r2 UNGROUP { X }.
```

Primijetimo da je prethodna definicija identična onoj za operator PACK, osim što je operator COLLAPSE zamijenjen operatorom EXPAND u drugoj liniji izraza. Sada možemo ponuditi razumnu formulaciju Upita H:

```
PACK (
     ( UNPACK S_DURING { SNO, DURING } ON (DURING) )
     MINUS
     ( UNPACK SP_DURING { SNO, DURING } ON (DURING) ) )
ON (DURING).
```

Evaluacijom tog izraza dobivamo rezultat na desnoj strani koji smo prije prikazali.

Na kraju, potrebno je istaknuti da su operatori PACK i UNPACK tek pokrate koje omogućuju kompaktniji zapis temporalnih upita, odnosno definirani su u terminima operatora COLLAPSE i EXPAND.

1.7 Generalizacija operatora PACK i UNPACK

Operatori PACK i UNPACK u praksi se najčešće koriste u svom najjednostavnijem obliku, kako smo prethodno opisali, no budući da je moguće da relacija sadrži više atributa čije su vrijednosti intervali, operatore je potrebno prilagoditi tako da se pakiranje i raspakiravanje može provesti na bilo kojem podskupu takvih atributa. Prvi slučaj koji promatramo odnosi se na skupove atributa koji su prazni. Mogućnost pakiranja i raspakiravanja relacije na nijednom atributu pokazat će se iznimno važnom u daljnjim razmatranjima. Sintaksa je slična kao i prije: PACK *r* ON ( ) i UNPACK *r* ON ( ). Definiramo da je rezultat ovih operacija relacija *r*. Drugi slučaj koji promatramo odnosi se na dva ili više atributa čije su

vrijednosti intervali. Intervali ne moraju nužno biti temporalni. Osvrnimo se prvo na raspakiravanje na točno dva atributa. Pretpostavimo da je relacija *r* oblika:

A1	A2
[P1:P1]	[d08:d09]
[P1:P2]	[d08:d08]
[P3:P4]	[d07:d08]

Promotrimo sada izraz: UNPACK (UNPACK *r* ON (A1)) ON (A2). Evaluacijom unutrašnjeg dijela izraza: UNPACK *r* ON (A1), dobivamo sljedeću relaciju:

A1	A2
[P1:P1]	[d08:d09]
[P1:P1]	[d08:d08]
[P2:P2]	[d08:d08]
[P3:P3]	[d07:d08]
[P4:P4]	[d07:d08]

Raspakiravanjem te relacije na atribut A2 dobivamo konačni rezultat:

A1	A2
[P1:P1]	[d08:d08]
[P1:P1]	[d09:d09]
[P2:P2]	[d08:d08]
[P3:P3]	[d07:d07]
[P3:P3]	[d08:d08]
[P4:P4]	[d07:d07]
[P4:P4]	[d08:d08]

Pogledajmo što će se dogoditi ako zamijenimo poredak raspakiravanja: UNPACK (UNPACK *r* ON (A2)) ON (A1). Rezultat unutrašnjeg dijela izraza: UNPACK *r* ON (A2), je relacija:

A1	A2
[P1:P1]	[d08:d08]
[P1:P1]	[d09:d09]
[P1:P2]	[d08:d08]
[P3:P4]	[d07:d07]
[P3:P4]	[d08:d08]

Raspakiravanjem prethodne relacije na atribut *A1* dobivamo konačni rezultat:

A1	A2
[P1:P1]	[d08:d08]
[P1:P1]	[d09:d09]
[P2:P2]	[d08:d08]
[P3:P3]	[d07:d07]
[P4:P4]	[d07:d07]
[P3:P3]	[d08:d08]
[P4:P4]	[d08:d08]

Primijetimo da su rezultati u oba slučaja jednaki, iako su pojedini retci u drugačijem poretku. Zaključujemo da ako je relacija *r* bilo koja relacija s atributima *A1* i *A2* čije su vrijednosti intervali, tada vrijedi sljedeći identitet:

UNPACK (UNPACK *r* ON (A1)) ON (A2) ≡ UNPACK (UNPACK *r* ON (A2)) ON (A1).

Također, lako se vidi da ako je *r* relacija s atributima *A1, A2, . . . , An* čije su vrijednosti intervali, tada raspakiravanje relacije *r* na te atribute u bilo kojem poretku uvijek daje isti rezultat. Slijedi poopćenje definicije operatora UNPACK.

Definicija 1.7.1. Neka su *r* relacija koja ima atribute *A1, A2, . . . , An* čije su vrijednosti intervali (*r* može imati i druge atribute) i n ≥ 0. Tada izraz UNPACK *r* ON (*A1, A2, . . . , An*) označava raspakiravanje relacije *r* na atribute *A1, A2, . . . , An* te predstavlja pokratu za sljedeće: UNPACK (. . . (UNPACK (UNPACK *r* ON (*B1*)) ON (*B2*)) . . .) ON (*Bn*), gdje je niz *B1, B2, . . . , Bn* proizvoljna permutacija niza *A1, A2, . . . , An*.

Slijedi poopćenje definicije operatora PACK iz prethodnog potpoglavlja.

Definicija 1.7.2. Neka su *r* relacija koja ima atribute *A1, A2, . . . , An* čije su vrijednosti intervali (*r* može imati i druge atribute) i n ≥ 0. Tada izraz PACK *r* ON (*A1, A2, . . . , An*) označava pakiranje relacije *r* na atribute *A1, A2, . . . , An* (u tom poretku) te predstavlja pokratu za sljedeće: PACK(. . . (PACK (PACK *r'* ON (*A1*)) ON (*A2*)) . . .) ON (*An*), gdje je *r'* relacija dobivena evaluacijom izraza UNPACK *r* ON (*A1, A2, . . . , An*).

Uočimo da operator PACK djeluje tako da se promatrana relacija prvo raspakira na sve određene atribute u bilo kojem redoslijedu, a tek onda pakira na te iste atribute u točno određenom poretku. Promotrimo sada primjer pakiranja na dva atributa. Neka je *r* relacija s atributima *A1* i *A2* čije su vrijednosti intervali:

A1	A2
[P2:P4]	[d01:d04]
[P3:P5]	[d01:d04]
[P2:P4]	[d05:d06]
[P2:P4]	[d06:d09]

Razmatramo sljedeći izraz: PACK *r* ON (A1, A2). Prvo slijedi rezultat implicitnog raspakiravanja na atribute *A1* i *A2*:

A1	A2
[P2:P2]	[d01:d01]
[P2:P2]	[d02:d02]
[P2:P2]	[d03:d03]
[P2:P2]	[d04:d04]
[P3:P3]	[d01:d01]
[P3:P3]	[d02:d02]
[P3:P3]	[d03:d03]

[P3:P3]	[d04:d04]
[P4:P4]	[d01:d01]
[P4:P4]	[d02:d02]
[P4:P4]	[d03:d03]
[P4:P4]	[d04:d04]
[P5:P5]	[d01:d01]
[P5:P5]	[d02:d02]
[P5:P5]	[d03:d03]

[P5:P5]	[d04:d04]
[P2:P2]	[d05:d05]
[P2:P2]	[d06:d06]
[P3:P3]	[d05:d05]
[P3:P3]	[d06:d06]
[P4:P4]	[d05:d05]
[P4:P4]	[d06:d06]
[P2:P2]	[d07:d07]

[P2:P2]	[d08:d08]
[P2:P2]	[d09:d09]
[P3:P3]	[d07:d07]
[P3:P3]	[d08:d08]
[P3:P3]	[d09:d09]
[P4:P4]	[d07:d07]
[P4:P4]	[d08:d08]
[P4:P4]	[d09:d09]

Pakiranjem te relacije na *A1* dobivamo:

A1	A2
[P2:P5]	[d01:d01]
[P2:P5]	[d02:d02]
[P2:P5]	[d03:d03]

[P2:P5]	[d04:d04]
[P2:P4]	[d05:d05]
[P2:P4]	[d06:d06]

[P2:P4]	[d07:d07]
[P2:P4]	[d08:d08]
[P2:P4]	[d09:d09]

Konačni rezultat dobivamo pakiranjem prethodne relacije na *A2*:

A1	A2
[P2:P5]	[d01:d04]
[P2:P4]	[d05:d09]

Promotrimo sada što će se dogoditi kada zamijenimo redoslijed pakiranja, odnosno razmotrimo sljedeći izraz: PACK *r* ON (A2, A1). Naravno, implicitnim raspakiravanjem kao rezultat dobivamo istu relaciju kao i prije. Pakiranjem te relacije na *A2* dobivamo:

A1	A2
[P2:P2]	[d01:d09]
[P3:P3]	[d01:d09]
[P4:P4]	[d01:d09]
[P5:P5]	[d01:d04]

Konačni rezultat dobivamo pakiranjem prethodne relacije na atribut *A1*:

A1	A2
[P2:P4]	[d01:d09]
[P5:P5]	[d01:d04]

Zadnji rezultat očito nije jednak onome od prije. Zaključujemo da ako je *r* relacija s atributima *A1* i *A2* čije su vrijednosti intervali, tada općenito vrijedi sljedeći identitet:

$$\text{PACK } r \text{ ON (A1, A2)} \not\equiv \text{PACK } r \text{ ON (A2, A1)}.$$

Analogno, može se pokazati da ista tvrdnja vrijedi za proizvoljnu relaciju *r* s *n* atributa čije su vrijednosti intervali te $n > 1$. Zato je redoslijed pakiranja bitan.

Prije nego što nastavimo razmatranje o operatorima PACK i UNPACK, slijede definicije o ekvivalentnosti i redundantnosti relacija.

Definicija 1.7.3. Neka su *r1* i *r2* relacije istog tipa i neka su atributi tih dviju relacija *A1, A2, . . . , An* atributi čije su vrijednosti intervali. Kažemo da su relacije *r1* i *r2* **ekvivalentne** obzirom na *A1, A2, . . . , An* ako i samo ako su relacije koje su rezultati od UNPACK *r1* ON (*A1, A2, . . . , An*) i od UNPACK *r2* ON (*A1, A2, . . . , An*) jednake.

Promotrimo sljedeće dvije relacije:

A1	A2
[P2:P5]	[d01:d04]
[P2:P4]	[d05:d09]

A1	A2
[P2:P4]	[d01:d09]
[P5:P5]	[d01:d04]

Relacije na lijevoj i desnoj strani nisu jednake, ali su ekvivalentne u smislu prethodne definicije. Obje sadrže iste informacije, no prikazuju ih iz dvije različite perspektive.

Definicija 1.7.4. Neka je *r* relacija s atributima *A1, A2, . . . , An* čije su vrijednosti intervali (*r* može imati i druge atribute). Neka je *u* relacija koja je dobivena raspakiravanjem *r* na te atribute. Ako se svaki redak u *u* može izvesti iz točnog jednog retka u *r*, tada kažemo da je relacija *r* **neredundantna** obzirom na *A1, A2, . . . , An*. Inače, kažemo da je relacija *r* **redundantna** obzirom na *A1, A2, . . . , An*.

Posljedice generalizacije operatora PACK i UNPACK, tako da se pakiranje i raspakiravanje može primijeniti na bilo koji skup atributa neke relacije čije su vrijednosti intervali, nisu uvijek odmah očite. Jednu od tih posljedica opisujemo u nastavku. Izrazi:

1. PACK *r* ON (*A1, A2, . . . , An*),
2. PACK (. . . (PACK (PACK *r* ON (*A1*)) ON (*A2*)) . . .) ON (*An*),

općenito nisu logički ekvivalentni. Prvi izraz uključuje potpuno raspakiravanje relacije *r* na atribute *A1, A2, . . . , An*, dok drugi ne. Definicija 1.7.2 za operator PACK jamči eliminiranje suvišnih informacija iz rezultantne relacije, a mnogo prirodnija definicija za taj operator, ona u skladu s drugim izrazom, to ne čini. Zato je operator PACK u Definiciji 1.7.2 definiran tako da uključuje početno raspakiravanje relacije na promatrane atribute. U prethodno se lako uvjeriti ako ponovno promotrimo relaciju *r*:

A1	A2
[P2:P4]	[d01:d04]
[P3:P5]	[d01:d04]
[P2:P4]	[d05:d06]
[P2:P4]	[d06:d09]

Evaluacijom izraza PACK *r* ON (A1, A2) dobivamo rezultat koji je jednak onome koji dobivamo evaluacijom izraza PACK (PACK *r* ON (A1)) ON (A2). Međutim, evaluacijom izraza PACK *r* ON (A2, A1) i PACK (PACK *r* ON (A2)) ON (A1) dobivamo različite rezultate:

A1	A2
[P2:P4]	[d01:d09]
[P5:P5]	[d01:d04]

A1	A2
[P2:P4]	[d01:d09]
[P3:P5]	[d01:d04]

Relacija na lijevoj strani dobivena evalucijom prvog izraza sadrži interval *[P5:P5]*, a relacija na desnoj strani dobivena evaluacijom drugog izraza sadrži interval *[P3:P5]*. Druga relacija dvaput nam kazuje da se klijenti P3 i P4 pojavljuju u kombinaciji s danima 1, 2, 3 i 4. Drugim riječima, sadrži redundantne podatke, dok s prvom relacijom to nije slučaj. Dakle, relacija na lijevoj strani je neredundantna, a relacija na desnoj strani je redundantna u skladu s Definicijom 1.7.4.

1.8 U_ operatori

U prethodnim potpoglavljima vidjeli smo kako izrazi koji su prilično složeni i veliki u terminima originalne relacijske algebre postaju intuitivniji i jednostavniji u terminima operatora PACK i UNPACK. Operatori PACK i UNPACK mogu se iskoristiti kao temelj za definiranje daljnjih operatora koji omogućuju postavljanje temporalnih upita na jednostavniji način. U nastavku predstavljamo te operatore. U ovom trenutku zgodno je sjetiti se Slike 1.4 i ponovno prikazati Upit H:

- **Upit H**: Pronađi parove SNO i DURING gdje je DURING maksimalni interval tijekom kojeg dobavljač SNO nije bio u mogućnosti dostavljati pošiljke nekom klijentu.

Vidjeli smo da je razumna formulacija Upita H jednaka sljedećem izrazu:

```
PACK (
    ( UNPACK S_DURING { SNO, DURING } ON (DURING) )
    MINUS
    ( UNPACK SP_DURING { SNO, DURING } ON (DURING) ) )
ON (DURING).
```

Primijetimo da prethodni izraz zahtijeva izvođenje sljedećih operacija:

1. Raspakiraj oba operanda.
2. Uzmi razliku.
3. Pakiraj rezultat.

Upiti u kojima se materijaliziraju raspakirane relacije, izvodi odgovarajuća relacijska operacija i pakira rezultat, mogu se izvršavati predugo ili nam može ponestati memorije. Posebno se to odnosi na slučajeve s intervalima velike kardinalnosti. Ako sve zahtjeve zapišemo u obliku jedne operacije, postoje efikasne implementacije operatora koje ne zahtijevaju da se raspakiravanje relacija uvijek fizički izvodi. Obrazac koji je ilustriran ovim primjerom pojavljuje se vrlo često u praksi te je smisleno uvesti određene pokrate koje pružaju mogućnost poboljšanja performansi upita. Jasno je da se takve pokrate mogu definirati za sve uobičajene relacijske operatore. Međutim, posebnu važnost imaju binarni operatori: UNION, INTERSECT, MINUS, D_UNION, I_MINUS, MATCHING i NOT MATCHING. U nastavku definiramo niz U_operatora. *U* potječe od *USING* ili od *UNPACKING* budući da je raspakiravanje najbitniji korak u obrascu koji smo prethodno opisali.

Definicija 1.8.1. Neka su *r1* i *r2* relacije koje imaju istu shemu *T*. Neka je *ACL* niz atributa koji su komponente od *T* i čije su vrijednosti intervali. Tada izraz USING (*ACL*) : *r1* MINUS *r2* nazivamo **U_MINUS** te predstavlja pokratu za sljedeće:

PACK ((UNPACK *r1* ON (*ACL*))
MINUS (UNPACK *r2* ON (*ACL*)) **) ON (*ACL*)**.

Upit H sada možemo formulirati na sljedeći način:

USING (DURING) :
S_DURING { SNO, DURING } MINUS SP_DURING { SNO, DURING }.

Definicija 1.8.2. Neka su *r1* i *r2* relacije koje imaju istu shemu *T*. Neka je *ACL* niz atributa koji su komponente od *T* i čije su vrijednosti intervali. Tada izraz USING (*ACL*) : *r1* UNION *r2* nazivamo **U_UNION** te predstavlja pokratu za sljedeće:

PACK ((UNPACK *r1* ON (*ACL*))
UNION (UNPACK *r2* ON (*ACL*) **) ON (*ACL*)**.

Zapravo, u ovom slučaju nema potrebe za pripremnim raspakiravanjem. Zato se U_UNION može pojednostaviti na sljedeći izraz: PACK (*r1* UNION *r2*) ON (*ACL*). Slična pojednostavljenja postoje i za druge U_ operatore.

Definicija 1.8.3. Neka su *r1* i *r2* relacije koje imaju istu shemu *T*. Neka je *ACL* niz atributa koji su komponente od *T* i čije su vrijednosti intervali. Tada izraz USING (*ACL*) : *r1* INTERSECT *r2* nazivamo **U_INTERSECT** te predstavlja pokratu za sljedeće:

PACK ((UNPACK *r1* ON (*ACL*))
INTERSECT (UNPACK *r2* ON (*ACL*)) **) ON (*ACL*)**.

U_ verzije od D_UNION i I_MINUS svakako imaju smisla, iako su nazivi pomalo nespretni.

Definicija 1.8.4. Neka su *r1* i *r2* relacije koje imaju istu shemu *T*. Neka je *ACL* niz atributa koji su komponente od *T* i čije su vrijednosti intervali. Tada izraz USING (*ACL*) : *r1* D_UNION *r2* nazivamo **disjunktni U_UNION** te predstavlja pokratu za sljedeće:

PACK ((UNPACK *r1* ON (*ACL*))
D_UNION (UNPACK *r2* ON (*ACL*)) **) ON (*ACL*)**.

Definicija 1.8.5. Neka su *r1* i *r2* relacije koje imaju istu shemu *T*. Neka je *ACL* niz atributa koji su komponente od *T* i čije su vrijednosti intervali. Tada izraz USING (*ACL*) : *r1* I_MINUS *r2* nazivamo **uključujući U_MINUS** te predstavlja pokratu za sljedeće:

PACK ((UNPACK *r1* ON (*ACL*))
I_MINUS (UNPACK *r2* ON (*ACL*)) **) ON (*ACL*)**.

Definicija 1.8.6. Neka su *r1* i *r2* relacije koje imaju barem jedan zajednički atribut. Neka je *ACL* niz atributa koji se pojavljuju u objema relacijama *r1* i *r2* te čije su vrijednosti intervali. Tada izraz USING (*ACL*) : *r1* JOIN *r2* nazivamo **U_JOIN** te predstavlja pokratu za sljedeće:

PACK ((UNPACK *r1* ON (*ACL*))
JOIN (UNPACK *r2* ON (*ACL*)) **) ON (*ACL*)**.

Definicija 1.8.7. Neka su *r1* i *r2* relacije koje imaju barem jedan zajednički atribut. Neka je *ACL* niz atributa koji se pojavljuju u objema relacijama *r1* i *r2* te čije su vrijednosti intervali. Tada izraz USING (*ACL*) : *r1* MATCHING *r2* nazivamo **U_MATCHING** te predstavlja pokratu za sljedeće:

PACK ((UNPACK *r1* ON (*ACL*))
MATCHING (UNPACK *r2* ON (*ACL*)) **) ON (*ACL*)**.

Definicija 1.8.8. Neka su *r1* i *r2* relacije koje imaju barem jedan zajednički atribut. Neka je *ACL* niz atributa koji se pojavljuju u objema relacijama *r1* i *r2* te čije su vrijednosti intervali. Tada izraz USING (*ACL*) : *r1* NOT MATCHING *r2* nazivamo **U_NOT MATCHING** te predstavlja pokratu za sljedeće:

PACK ((UNPACK *r1* ON (*ACL*))
NOT_MATCHING (UNPACK *r2* ON (*ACL*)) *) ON (ACL)*.

Operatori U_UNION, U_INTERSECT, disjunktni U_UNION i U_JOIN, koje smo prethodno definirali, imaju i svoje *n*–arne analogone. U_ verzije tih operatora mogu se definirati tako da se raspakira svih *n* relacija, primijeni regularni *n*–arni operator na tih *n* raspakiranih formi, te na kraju pakira rezultat. Slijedi primjer definicije *n*–arnog operatora U_JOIN.

Definicija 1.8.9. Neka su *r1, r2, . . . , rn* relacije koje imaju barem jedan zajednički atribut. Neka je *ACL* niz atributa koji se pojavljuju u svakoj od relacija *r1, r2, . . . , rn*, te čije su vrijednosti intervali. Tada izraz USING (*ACL*) : JOIN {*r1, r2, . . . , rn*} nazivamo *U_JOIN* te predstavlja pokratu za sljedeće:

PACK (JOIN {
UNPACK *r1* ON (*ACL*) ,
UNPACK *r2* ON (*ACL*) ,
. . .
UNPACK *rn* ON (*ACL*) } **) ON (*ACL*)**.

Definicije za ostale operatore slijede isti obrazac.

Osvrnimo se sada na U_ verzije unarnih operatora RENAME, restrikcije, projekcije, EXTEND, GROUP i UNGROUP.

Definicija 1.8.10. Neka je *r* relacija. Neka je *ACL* niz atributa koji se pojavljuju u *r* te čije su vrijednosti intervali. Neka *r* ima atribut *A* koji nije u *ACL* i neka nema atribut koji se zove *B*. Tada izraz USING (*ACL*) : *r* RENAME {*A* AS *B*} nazivamo **U_RENAME** te predstavlja pokratu za sljedeće:

PACK ((UNPACK *r* ON (*ACL*)) RENAME { *A* AS *B* } **) ON (*ACL*)**.

Naravno, taj se izraz može reducirati na: PACK (*r* RENAME { *A* AS *B* }) ON (*ACL*). Ne očekuje se da će se ovaj operator često koristiti u praksi.

Definicija 1.8.11. Neka je *r* relacija. Neka je *ACL* niz atributa koji se pojavljuju u *r* te čije su vrijednosti intervali. Neka je *bx* uvjet na *r*. Tada izraz USING (*ACL*) : *r* WHERE *bx* nazivamo **U_RESTRICT** te predstavlja pokratu za sljedeće:

PACK ((UNPACK *r* ON (*ACL*)) WHERE *bx* **) ON (*ACL*)**.

Primijetimo da se početno raspakiravanje ovdje primjenjuje na relaciju *r*, a ne na restrikciju *r* WHERE *bx*.

Definicija 1.8.12. Neka je *r* relacija. Neka je *ACL* niz atributa koji se pojavljuju u *r* te čije su vrijednosti intervali. Neka je *BCL* niz atributa takav da se svaki atribut iz *ACL* pojavljuje u *BCL*. Tada izraz USING (*ACL*) : *r* {*BCL*} nazivamo **U_PROJECT** te predstavlja pokratu za sljedeće:

PACK ((UNPACK *r* ON (*ACL*)) { *BCL* } **) ON (*ACL*)**.

Definicija 1.8.13. Neka je *r* relacija. Neka je *ACL* niz atributa koji se pojavljuju u *r* te čije su vrijednosti intervali. Tada izraz USING (*ACL*) : EXTEND *r* : {*A* := *exp*} nazivamo **U_EXTEND** te predstavlja pokratu za sljedeće:

PACK (((EXTEND (UNPACK *r* ON (*ACL*)) : { *A* := *exp* } **) ON (*ACL*)**.

Definicija 1.8.14. Neka je *r* relacija. Neka su *ACL* niz atributa koji se pojavljuju u *r* te čije su vrijednosti intervali; *BCL* niz atributa koji se pojavljuju u *r*, a nisu u *ACL*; i neka je *X* ime atributa koje je različito od svih imena atributa koji se pojavljuju u *r*, osim možda onih u *BCL*. Tada izraz USING (*ACL*) : *r* GROUP {*BCL*} nazivamo **U_GROUP** te predstavlja pokratu za sljedeće:

PACK (((UNPACK *r* ON (*ACL*)) GROUP { *BCL* } AS *X* **) ON (*ACL*)**.

Definicija 1.8.15. Neka je *r* relacija koja ima atribut *B* čije su vrijednosti relacije. Neka je *ACL* niz atributa koji se pojavljuju u *r* i čije su vrijednosti intervali. Tada izraz USING (*ACL*) : *r* UNGROUP *B* nazivamo **U_UNGROUP** te predstavlja pokratu za sljedeće:

PACK (((UNPACK *r* ON (*ACL*)) UNGROUP *B* **) ON (*ACL*)**.

Ako se relacije *r1* i *r2* sastoje od nekog atributa čije su vrijednosti intervali, tada često želimo usporediti raspakirane verzije tih relacija. Dakle, preostaje nam definirati U_ verzije operatora za usporedbu relacija.

Definicija 1.8.16. Neka su *r1* i *r2* relacije koje imaju istu shemu *T*. Neka je *ACL* niz atributa koji su komponente od *T* i čije su vrijednosti intervali. Tada izraz USING (*ACL*) : *r1 compop r2*, gdje je *compop* bilo koji operator za usporedbu ($=, \neq, \subseteq, \subset, \supseteq, \supset$), nazivamo **U_comparison** te predstavlja pokratu za sljedeće:

(UNPACK *r1* ON (*ACL*)) *compop* (UNPACK *r2* ON (*ACL*)).

Primijetimo da završno pakiranje ovdje nije potrebno budući da rezultat usporedbe nije relacija, već vrijednost *true* ili *false*.

Promotrimo još jednom operator U_MINUS koji smo ranije definirali. Izraz USING (*ACL*) : *r1* MINUS *r2* je pokrata za sljedeće:

PACK ((UNPACK *r1* ON (*ACL*))
MINUS (UNPACK *r2* ON (*ACL*))) ON (*ACL*).

Pretpostavimo da je *ACL* prazan niz atributa. Tada U_MINUS postaje izraz USING ( ) : *r1* MINUS *r2*, odnosno:

PACK ((UNPACK *r1* ON ( ))
MINUS (UNPACK *r2* ON ( ))) ON ( ).

Prisjetimo se da smo prije definirali da je UNPACK *r* ON ( ) i PACK *r* ON ( ) samo *r*. Dakle, cijeli izraz se može reducirati na samo: *r1* MINUS *r2*. Drugim riječima, regularni relacijski operator MINUS je poseban slučaj od U_MINUS. To nam omogućuje da redefiniramo sintaksu regularnog operatora MINUS kako slijedi. Izraz [USING (*ACL*) :] je opcionalan:

[USING (*ACL*) :] *r1* MINUS *r2*.

Prethodna zapažanja mogu se primijeniti na sve ostale regularne operatore. Regularni operatori dopuštaju, ali ne zahtijevaju, dodatni operand kada se primjenjuju na relacije u kojima se pojavljuju atributi čije su vrijednosti intervali. Zaključujemo da su U_ operatori tek poopćenja svojih regularnih analogona.

Na kraju, potrebno je istaknuti da je operator UNPACK ključna konceptualna komponenta našeg pristupa upravljanju temporalnim podacima u relacijskom modelu za bazu podataka.

Poglavlje 2

Napredni koncepti temporalnih baza podataka

2.1 Oblikovanje temporalnih baza podataka

Relacije S_DURING i SP_DURING omogućile su nam da uočimo potrebu za posebnim tipom podataka u temporalnim bazama podataka, intervalima, te za posebnim operatorima za upravljanje takvim podacima. Međutim, te su relacije iznimno jednostavne strukture, zato se u nastavku okrećemo originalnoj netemporalnoj bazi podataka s relacijama S i SP. Slijedi prikaz definicija originalnih relacija:

```
VAR S BASE RELATION
    { SNO SNO, SNAME NAME, STATUS INTEGER, CITY CHAR }
    KEY { SNO } ;

VAR SP BASE RELATION
    { SNO SNO, PNO PNO }
    KEY { SNO, PNO }
    FOREIGN KEY { SNO } REFERENCES S ;
```

Pretpostavimo da želimo kreirati temporalne analogone ovih netemporalnih relacija. Najjednostavnije je netemporalnim relacijama dodati odgovarajući atribut koji predstavlja neku vremensku oznaku. Na primjer, relaciju S proširit ćemo atributom SINCE u slučaju polutemporalnog oblikovanja ili atributom DURING u slučaju temporalnog oblikovanja:

```
VAR SSSC_SINCE BASE RELATION
    { SNO SNO, SNAME NAME, STATUS INTEGER, CITY CHAR, SINCE DATE }
    KEY { SNO } ;
```

```
VAR SSSC_DURING BASE RELATION
    { SNO SNO, PNO PNO, DURING INTERVAL_DATE }
    KEY { SNO, PNO, DURING }
    FOREIGN KEY { SNO } REFERENCES S ;
```

Relaciju S preoblikovali smo u relacije SSSC_SINCE i SSSC_DURING. U nastavku ćemo koristiti nazive *since relacija* i *during relacija* kako bismo se referirali na relacije poput prethodnih. Intuitivno, since relacija je relacija koja sadrži podatke koji trenutačno vrijede, a during relacija je relacija koja sadrži povijesne podatke. Nije teško primijetiti da su relacije SSSC_SINCE i SSSC_DURING obje loše oblikovane. Jednostavno dodavanje atributa koji predstavlja neku vremensku oznaku netemporalnoj relaciji nije dobar način temporalnog dizajniranja. Postoje tri različita pristupa temporalnom oblikovanju: samo since relacije, samo during relacije, ili kombinacija prethodnih. U praksi se može koristiti bilo koja kombinacija, no u literaturi se najčešće proučava pristup koji se temelji na samo during relacijama.

Samo since relacije

Relacija SSSC_SINCE je polutemporalna i sadrži podatke koji trenutačno vrijede, no ta karakterizacija nije u potpunosti točna. Takve relacije mogu sadržavati neku vrstu informacija o prošlosti i budućnosti. Relaciju SSSC_SINCE možemo opisati sljedećim predikatom:

1. *Dobavljač SNO je pod ugovorom od dana SINCE.*
2. *Dobavljač SNO se zove SNAME od dana SINCE.*
3. *Dobavljač SNO ima status STATUS od dana SINCE.*
4. *Dobavljač SNO se nalazi u gradu CITY od dana SINCE.*

Odmah iz formulacije predikata trebalo bi biti jasno da relacija nije dobro oblikovana. Na primjer, pretpostavimo da promatrana relacija sadrži sljedeći redak:

SNO	SNAME	STATUS	CITY	SINCE
S1	Smith	20	London	d04

Također, pretpostavimo da je danas dan 10 te da se status dobavljača S1 mora promijeniti u 30. Tada prethodni redak moramo zamijeniti sljedećim:

SNO	SNAME	STATUS	CITY	SINCE
S1	Smith	30	London	d10

Između ostaloga, izmjenom retka izgubili smo informaciju da se dobavljač S1 od dana 4 nalazio u Londonu. Trebalo bi biti jasno da je relacija SSSC_SINCE definirana tako da nije u mogućnosti prikazivati bilo koje informacije o dobavljaču koje prethode danu zadnje promjene. No taj se problem može lako riješiti tako da postojeći atribut SINCE zamijenimo četirima takvim atributima, po jednim za svaki originalni atribut. Skica:

SSSC_SINCE {
SNO, SNO_SINCE,
SNAME, SNAME_SINCE,
STATUS, STATUS_SINCE,
CITY, CITY_SINCE }
KEY { SNO }.

Sada relaciju SSSC_SINCE možemo opisati ovim predikatom: *Dobavljač SNO je pod ugovorom od dana SNO_SINCE, zove se SNAME od dana SNAME_SINCE, ima status STATUS od dana STATUS_SINCE i nalazi se u gradu CITY od dana CITY_SINCE*. Slijedi tipični primjer retka u skladu s prethodnim predikatom za relaciju SSSC_SINCE:

SNO	SNO_SINCE	SNAME	SNAME_SINCE
S1	d04	Smith	d04

STATUS	STATUS_SINCE	CITY	CITY_SINCE
30	d10	London	d04

Prethodni redak između ostaloga prikazuje da dobavljač S1 ima status 30 od dana 10, ali i da se nalazi u Londonu od dana 4. Tako smo riješili problem promjene statusa. Naravno, čak i s promijenjenim dizajnom ova relacija reprezentira samo informacije koje trenutačno vrijede budući da je ta baza podataka ipak samo polutemporalna. Na primjer, ako se na dan 10 status dobavljača S1 promijeni u 30, tada redak s vrijednostima STATUS = 20 i STATUS_SINCE = *d04*, moramo zamijeniti drugim retkom s vrijednostima STATUS = 30 i STATUS_SINCE = *d10*. Tako smo izgubili informaciju da je status dobavljača S1 iznosio 20 od dana 4 do dana 9. Drugim riječima, polutemporalni dizajn baze podataka ne može reprezentirati čisto povijesne informacije. Tu mislimo na informacije koje su vrijedile u prošlosti, ali sada više ne, no ako je danas dan 10 i relacija pokazuje da se dobavljač S1 nalazi u Londonu od dana 4, tada ona očito sadrži neku vrstu povijesne informacije.

Istaknimo još neke zaključne opaske:

1. Za svaki atribut u relaciji nije nužno imati i odgovarajući SINCE atribut. Na primjer, ako se ime dobavljača nikada ne mijenja, ili se mijenja, ali nas te promjene ne zanimaju, tada atribut SNAME_SINCE očito nije potreban.

2. Ako je vrijednost atributa SNO_SINCE u nekom retku iz SSSC_SINCE jednaka d, te ako je vrijednost nekog drugog since atributa u istom retku jednaka d', tada mora vrijediti da je $d' \geq d$. Dakle, pretpostavljamo da relacija ne sadrži podatke koji prethode danu od kada je ugovor postao važeći.
3. Relacija SSSC_SINCE sadrži podatke koji trenutačno vrijede, no ona može sadržavati i podatke koji se eksplicitno odnose na budućnost. Na primjer, vrijednost atributa SNO_SINCE može biti neki budući datum d koji indicira da će dobavljač biti pod ugovorom na taj dan u budućnosti. Atributu SNO_SINCE bi u tom slučaju bolje odgovaralo ime SNO_FROM.
4. Relacije poput SSSC_SINCE nužno sadrže podatke koji se implicitno odnose na budućnost. Na primjer, ako relacija sadrži redak koji kazuje da je dobavljač S1 pod ugovorom od dana 4, to znači da će dobavljač S1 biti pod ugovorom od dana 4 do *zadnjega dana*. Dakle, stvarno vrijeme je interval od dana 4 do *zadnjega dana*.
5. Sve informacije koje su vezane uz dobavljača S1 bit će izbrisane iz baze podataka kada ugovor prestane vrijediti. Bilo koje stvarno vrijeme koje uključuje interval kojemu završna točka nije *kraj vremena* ili uključuje dva ili više različitih intervala, ne može biti reprezentirano u bazi podataka koja je polutemporalna. Dakle, stvarna vremena oblika { *[d10:d25]* } nisu moguća.

Samo during relacije

During relacije intuitivno možemo shvatiti kao relacije koje sadrže povijesne informacije, no one mogu sadržavati i eksplicitne informacije koje se odnose na budućnost (kao i na prošlost). Na primjer, relacija SSSC_DURING može sadržavati redak koji pokazuje da je dobavljač Sx pod ugovorom od dana *di* do dana *dj*, gdje su *di* i *dj* neki datumi u budućnosti. Ako je *dj* u budućnosti, a *di* u prošlosti ili je današnji datum, tada relacija sadrži i podatke koji trenutačno vrijede.

Predikat za relaciju SSSC_DURING glasi: *DURING predstavlja maksimalni vremenski interval tijekom kojeg su sljedeće tvrdnje istinite:*

1. *Dobavljač SNO je pod ugovorom.*
2. *Dobavljač SNO se zove SNAME.*
3. *Dobavljač SNO ima status STATUS.*
4. *Dobavljač SNO se nalazi u gradu CITY.*

Kao i u slučaju relacije SSSC_SINCE, iz formulacije predikata trebalo bi biti jasno da relacija SSSC_DURING nije dobro oblikovana. Promjene koje ćemo uvesti drastičnije su nego

u slučaju relacije SSSC_SINCE. During relacije moramo podvrgnuti *vertikalnoj* i/ili *horizontalnoj dekompoziciji*. Vertikalna dekompozicija odnosi se na činjenicu da se različiti atributi istog entiteta razlikuju na nekoj razini. Horizontalna dekompozicija temelji se na činjenici da postoje logičke razlike između trenutnih i povijesnih podataka. Na primjer, pretpostavimo da relacija SSSC_DURING sadrži sljedeći redak:

SNO	SNAME	STATUS	CITY	DURING
S2	Jones	10	Pariz	[d02:d04]

Tada iz prethodnog retka možemo iščitati da je dobavljač S2 bio pod ugovorom, zvao se Jones, imao status 10 i nalazio se u Parizu tijekom istog perioda *[d02:d04]*. Ova situacija je možda moguća, no sigurno nije uobičajena u praksi. Primjerenije je da su ti intervali različiti. Atribut DURING predstavlja vremensku oznaku za kombinaciju četiri samostalna predikata umjesto jednog, zato ćemo originalnu relaciju SSSC_DURING zamijeniti četirima relacijama s vlastitim vremenskim oznakama kako je prikazano u nastavku. Skica:

S_DURING { SNO, DURING } KEY { SNO, DURING }
S_NAME_DURING { SNO, SNAME, DURING } KEY { SNO, DURING }
S_STATUS_DURING { SNO, STATUS, DURING } KEY { SNO, DURING }
S_CITY_DURING { SNO, CITY, DURING } KEY { SNO, DURING }.

Prethodni primjer prikazuje vertikalnu dekompoziciju relacije SSSC_DURING. U skladu s tom dekompozicijom relacija S_DURING prikazuje kada su koji dobavljači pod ugovorom; relacija S_NAME_DURING prikazuje kada su koji dobavljači imali određeno ime; relacija S_STATUS_DURING prikazuje kada su koji dobavljači imali određeni status; te relacija S_CITY_DURING prikazuje kada su koji dobavljači bili smješteni u nekom gradu. U nastavku slijedi prikaz mogućih redaka:

SNO	DURING
S2	[d01:d09]

SNO	SNAME	DURING
S2	Jones	[d01:d04]

SNO	SNAME	DURING
S2	Johns	[d05:d05]

SNO	STATUS	DURING
S2	10	[d01:d09]

SNO	CITY	DURING
S2	Pariz	[d01:d09]

Primijetimo da je rezultat dekompozicije relacija S_DURING koju smo upoznali u prethodnom poglavlju. Relacija S_DURING zapravo nije potrebna u ovom pristupu za oblikovanje temporalnih baza podataka budući da se sve informacije koje sadrži mogu izvesti iz preostalih relacija. Međutim, radi potpunosti je uključujemo u naš dizajn.

Šesta normalna forma

Teorija normalizacije zasnovana je na pojmu *normalnih formi*. U [2] je opisano: Svaka normalna forma predstavlja određeni *zahtjev na kvalitetu* koji bi relacija trebala zadovoljavati. Zahtjevi su stroži što je normalna forma viša.

Vertikalna dekompozicija relacije SSSC_DURING u četiri zasebne relacije podsjeća na klasičnu normalizaciju. Točnije, vertikalna dekompozicija podudara se s klasičnom teorijom normalizacije u pogledu dekompozicijskog operatora (projekcija) i pripadnog kompozicijskog operatora (JOIN). Ideja dekompozicije relacija motivirana je željom za redukcijom na ireducibilne komponente. Dakle, cilj je izvršiti dekompoziciju relacije na njene projekcije tako da sve informacije iz relacije ostanu sačuvane. U netemporalnim bazama podataka potreba za redukcijom relacija je malena, no u slučaju temporalnih baza podataka ona je mnogo izraženija. Na primjer, ime dobavljača, status i grad razlikuju se ovisno o vremenu. Moguće je i da se ime dobavljača gotovo nikada ne mijenja, lokacija se mijenja povremeno, a status relativno često. Bilo bi prilično nezgodno svaki put ponavljati podatke o imenu i gradu kada se status dobavljača promijeni, a i povijest imena, povijest statusa te povijest grada za pojedinog dobavljača su vjerojatno zanimljiviji koncepti nego gledajući povijest kombinacije ime–status–grad. Iz tih razloga potrebna je vertikalna dekompozicija. Također, nakon vertikalne dekompozicije relacije zanimljiviji upiti mogu se lakše izraziti, no oni manje zanimljivi teže.

Osvrnimo se sada na petu normalnu formu koja je u klasičnoj teoriji normalizacije posljednja normalna forma. Peta normalna forma temelji se na konceptu ovisnosti spoja (engl. *join dependency*) koji definiramo u nastavku.

Definicija 2.1.1. Neka je H zaglavlje neke relacije. Tada je **ovisnost spoja** (JD) obzirom na H izraz oblika $\bowtie\{X1, X2, \ldots, Xn\}$, gdje su $X1, X2, \ldots, Xn$ neprazni podskupovi od H čija je unija jednaka H. $X1, X2, \ldots, Xn$ nazivamo komponentama od JD.

Neki primjeri ovisnosti spoja u skladu sa zaglavljem relacije S iz uvodnog poglavlja:

1. ⋈{ { SNO, SNAME }, { SNO, STATUS }, { SNO, CITY } }
2. ⋈{ { SNO, SNAME, STATUS }, { SNAME, CITY } }
3. ⋈{ { SNO, SNAME, CITY }, { CITY, STATUS } }

Definicija 2.1.2. Neka je r relacija sa zaglavljem H i neka je J ovisnost spoja obzirom na H $\bowtie\{X1, X2, \ldots, Xn\}$. Ako je r jednaka prirodnom spoju njenih projekcija na $X1, X2, \ldots, Xn$, tada kažemo da r **zadovoljava** J; inače r **ne zadovoljava** J. (Ovdje mislimo na n–arnu verziju operatora JOIN.)

Na primjer, relacija S iz uvodnog poglavlja zadovoljava ovisnost spoja: ⋈{ { SNO, SNAME }, { SNO, STATUS }, { SNO, CITY } }, ali ne zadovoljava ovisnost spoja: ⋈{ { SNO, SNAME, CITY }, { CITY, STATUS } }.

Definicija 2.1.3. Neka je *r* relacija sa zaglavljem *H*. Kažemo da je *r* u **petoj normalnoj formi** (oznaka: 5NF) ako i samo ako svaka netrivijalna ovisnost spoja obzirom na *H* koju *r* zadovoljava ima svojstvo da sve njene komponente sadrže ključ od *r*. (Ovisnost spoja je trivijalna ako i samo ako barem jedna njena komponenta u cijelosti sadrži zaglavlje *H*.)

Na primjer, relacija SSSC_DURING je u petoj normalnoj formi.

Šesta normalna forma je u teoriju normalizacije uvedena 2002. godine i od posebne je važnosti za temporalne baze podataka. Prvi su je definirali C. J. Date, Hugh Darwen i Nikos A. Lorentzos. Operatore U_PROJECT i U_JOIN definirali smo kao generalizirane verzije projekcije i operatora JOIN. Analogno, postoje poopćenja pojmova koje smo prethodno predstavili.

Definicija 2.1.4. Neka su *H* zaglavlje neke relacije te *ACL* niz atributa koji su komponente od *H* i čije su vrijednosti intervali. Tada je **U_JD** (od engl. *U_ join dependency*) obzirom na *ACL* i *H* izraz oblika USING (*ACL*) : ⋈{*X1, X2, . . . , Xn*}, gdje su *X1, X2, . . . , Xn* neprazni podskupovi od *H* čija je unija jednaka *H*. *X1, X2, . . . , Xn* nazivamo komponentama od U_JD.

Definicija 2.1.5. Neka je *r* relacija sa zaglavljem *H* i neka je *UJ* U_JD obzirom na *ACL* i *H*: USING (*ACL*) : ⋈{*X1, X2, . . . , Xn*}. Ako je *r* jednaka prirodnom spoju njenih projekcija na *X1, X2, . . . , Xn*, tada kažemo da *r* **zadovoljava** *UJ*; inače *r* **ne zadovoljava** *UJ*. (Ovdje mislimo na operatore U_=, U_PROJECT i *n*–arnu verziju operatora U_JOIN.)

Na primjer, relacija SSSC_DURING zadovoljava U_JD: USING (DURING) : ⋈{ S_DURING, S_NAME_DURING, S_STATUS_DURING, S_CITY_DURING }, gdje imena S_DURING, S_NAME_DURING, S_STATUS_DURING, S_CITY_DURING označavaju odgovarajuće projekcije obzirom na DURING:

S_DURING
$\stackrel{\text{def}}{=}$ USING (DURING) : SSSC_DURING { SNO, DURING },
S_NAME_DURING
$\stackrel{\text{def}}{=}$ USING (DURING) : SSSC_DURING { SNO, SNAME, DURING },
S_STATUS_DURING
$\stackrel{\text{def}}{=}$ USING (DURING) : SSSC_DURING { SNO, STATUS, DURING },
S_CITY_DURING
$\stackrel{\text{def}}{=}$ USING (DURING) : SSSC_DURING { SNO, CITY, DURING }.

Budući da relacija SSSC_DURING zadovoljava prethodni U_JD, slijedi da je dekompozicija od SSSC_DURING na gornje četiri U_projekcije takva da su sačuvane sve informacije iz SSSC_DURING. Uočimo da je klasična ovisnost spoja ili JD poseban slučaj poopćene verzije U_JD, stoga u nastavku možemo koristiti pojam *ovisnost spoja* kako bismo se referirali na obje. Međutim, uglavnom mislimo na generaliziranu verziju.

Definicija 2.1.6. Relacija *r* je u **šestoj normalnoj formi** (oznaka: 6NF) ako i samo ako je svaka ovisnost spoja koju *r* zadovoljava trivijalna. (Ovisnost spoja je trivijalna ako i samo ako je barem jedna njena komponenta u cijelosti jednaka odgovarajućem zaglavlju relacije.)

Iz prethodne definicije odmah slijedi: Ako je relacija u 6NF, ona je i u 5NF. Uočimo da relacija SSSC_DURING nije u 6NF jer postoji ovisnost spoja koju SSSC_DURING zadovoljava (kao u prošlom primjeru), ali sigurno nije trivijalna:

USING (DURING) :
⋈{ S_DURING, S_NAME_DURING, S_STATUS_DURING, S_CITY_DURING }.

Konačno, u literaturi se predlaže da se temporalne relacije poput SSSC_DURING koje nisu u 6NF bez gubitka informacija dekomponiraju u U_projekcije koje jesu. U našem slučaju, pripadne U_projekcije od SSSC_DURING su relacije S_DURING, S_NAME_DURING, S_STATUS_DURING i S_CITY_DURING koje su sve u 6NF.

Koncept *sada*

U prethodnim razmatranjima pretpostavljali smo da se vremenski intervali u during relacijama *protežu* od neke točke *b* do neke točke *e*, gdje je $b \leq e$. Te relacije nisu sadržavale eksplicitne podatke koji se odnose na budućnost, odnosno *b* i *e* nisu bili u budućnosti. Također, pretpostavljali smo da je sadašnjost reprezentirana nekom eksplicitnom vrijednošću, npr. *d10*. Ta pretpostavka uopće nije razumna jer sugerira, na primjer, da se u ponoć na dan 10 sve vrijednosti *d10*, koje reprezentiraju sadašnjost, zamijene vrijednošću *d11*. Dakle, ta pretpostavka zahtijeva da se baza podataka odgovarajuće ažurira. Ako su vremenski intervali finije strukture, takva ažuriranja mogla bi se izvršavati svake milisekunde. Također, upotreba eksplicitne vrijednosti poput *d10* je dvosmislena jer ne postoji način da utvrdimo označava li uistinu dan 10 ili se pretpostavlja da znači *do daljnjega* kao u većini prethodnih primjera. U literaturi se ponekad predlaže upotreba posebne oznake koju ćemo nazvati NOW. Osnovna je ideja dopustiti upotrebu te varijable kada je željena interpretacija upravo *do daljnjega*. S druge strane, mnogi istraživači temporalnih baza podataka smatraju da je uvođenje varijable NOW vrlo nerazumno i da odudara od osnovnih relacijskih principa. U nastavku navodimo neka pitanja koja proizlaze iz upotrebe varijable NOW:

1. Pretpostavimo da u relaciji S_DURING postoji jedinstveni redak za dobavljača S1 gdje je vrijednost atributa DURING interval *[d04:NOW]*. Tada bi rezultat upita: *Kada ugovor dobavljača S1 prestaje vrijediti?*, trebao biti NOW u smislu: *Vrijedi do daljnjega*. Ako sustav za upravljanje bazom podataka evaluira taj NOW u trenutku kada je upit izvršen i vrati vrijednost, npr. *d20*, tada je taj odgovor očito netočan jer ugovor dobavljača S1 još uvijek vrijedi. Postavlja se pitanje koju vrijednost dodijeliti varijabli NOW.
2. Pretpostavimo da je *t* redak koji sadrži interval *[NOW:d12]* i da je danas dan 9. Tada redak *t* možemo zamisliti kao pokratu za četiri različita retka koji sadrže intervale oblika *[d09:d09]*, *[d10:d10]*, *[d11:d11]* i *[d12:d12]*. Kada nastupi ponoć na dan 9, tada će se prvi od tih redaka automatski izbrisati. Slično za dan 10 i 11, no što će se točno dogoditi u ponoć na dan 12?
3. Koji je rezultat usporedbe *d15* = NOW?
4. Koje su povratne vrijednosti izraza NOW−1 i NOW+1?
5. Ako su *i1* i *i2* intervali *[d01:NOW]* i *[d05:d10]*, u kakvom su odnosu *i1* i *i2*?

Gotovo je nemoguće dati suvisle odgovore na ova pitanja, zato se okrećemo pristupu oblikovanju koji se ne oslanja na takve koncepte. Naravno, ako se ograničimo na temporalno oblikovanje koje se sastoji od samo during relacija, tada moramo uvesti neku oznaku koju ćemo interpretirati kao *do daljnjega*. Ako ponovno razmotrimo upit sa završetkom ugovora, sukladno prethodnom možemo izrazu END (DURING) dodijeliti umjetnu vrijednost *zadnji dan*. Tada će korisnik tu vrijednost interpretirati kao *do daljnjega*. Sada možemo isključiti pretpostavku da interval *[b:e]* nije takav da su *b* i *e* u budućnosti.

Kombinacija since i during relacija

Dosad smo predstavili dva pristupa temporalnom oblikovanju. Prvi se temelji na samo since relacijama, a drugi na samo during relacijama. U oba pristupa susrećemo se s određenim problemima. Problem u prvom pristupu je nemogućnost čuvanja *čisto* povijesnih zapisa, a u drugom pristupu problem je koncept *sada* za koji ne postoji adekvatno rješenje. U nastavku predstavljamo pristup koji se temelji na kombinaciji since i during relacija. Uočimo da postoji jasna razlika između trenutnih i povijesnih informacija. U povijesnim informacijama poznate su početna i završna točka, a u trenutnim informacijama poznata je samo početna točka. Ta razlika nam sugerira da bismo trebali imati relacije koje bilježe trenutno stanje i relacije koje bilježe povijest. Ako primijenimo ovaj pristup, tada je skica početnog primjera kao u nastavku:

S_SINCE { SNO, SNO_SINCE, SNAME, SNAME_SINCE,
STATUS, STATUS_SINCE, CITY, CITY_SINCE } KEY { SNO }

S_DURING { SNO, DURING } KEY { SNO, DURING }
S_NAME_DURING { SNO, SNAME, DURING } KEY { SNO, DURING }
S_STATUS_DURING { SNO, STATUS, DURING } KEY { SNO, DURING }
S_CITY_DURING { SNO, CITY, DURING } KEY { SNO, DURING }.

Relacija S_SINCE je jedina since relacija i u potpunosti je jednaka relaciji SSSC_SINCE iz *Samo since relacije*. Uočimo da je relacija S_SINCE u 5NF, ali ne i u 6NF. Ostale relacije su during relacije i odgovaraju relacijama koje smo predstavili u *Samo during relacije*, samo što ovdje sigurno ne sadrže retke s intervalima u kojima se završna točka može interpretirati kao *do daljnjega*. Takve informacije sada sadrži samo relacija S_SINCE. Podjela temporalnih podataka na since i during dijelove, na način koji smo maloprije opisali, naziva se horizontalnom dekompozicijom. Ako pretpostavimo da na početku imamo relaciju SSSC_DURING koja sadrži podatke koji se odnose na prošlost, sadašnjost i budućnost, tada proces horizontalne dekompozicije možemo opisati prvim dvama koracima, a konačni dizajn dobivamo primjenom trećeg:

1. Uvedi since relaciju S_SINCE s atributima koji odgovaraju atributima u relaciji SSSC_DURING, osim atributa DURING.
2. Svakom atributu u S_SINCE pridruži odgovarajući atribut SINCE.
3. Relaciju SSSC_DURING vertikalno dekomponiraj u 6NF projekcije. U ovom koraku relacija SSSC_DURING sadrži samo povijesne informacije.

Dosad smo razmatrali samo relaciju koja opisuje dobavljače. U skladu sa zadnjim pristupom temporalnom oblikovanju slijedi skica preporučenog dizajna za pošiljke:

SP_SINCE { SNO, PNO, SINCE }
    KEY { SNO }
    FOREIGN KEY { SNO } REFERENCES S_SINCE
SP_DURING { SNO, PNO, DURING }
    KEY { SNO, DURING }.

Primijenili smo horizontalnu dekompoziciju kako je opisano, a budući da su ove relacije obje u 6NF, vertikalna dekompozicija nije potrebna.

Zaključno, koji pristup temporalnom oblikovanju odabrati, ovisi o danoj situaciji. Općenito se predlaže pristup koji kombinira since i during relacije. Tada treba postupati u skladu sa sljedećim: Preporuča se horizontalna dekompozicija početne relacije kako bi se odvojile trenutne i povijesne informacije. Predlaže se da se sve since relacije normaliziraju u skladu s klasičnom teorijom normalizacije do 5NF. Na kraju, preporuča se vertikalna dekompozicija svih during relacija u ireducibilne 6NF komponente.

2.2 U_KEY

U nastavku se bavimo ograničenjima za čuvanje integriteta temporalne baze podataka. Pretpostavimo da je baza podataka o dobavljačima i pošiljkama oblikovana u skladu s pristupom koji kombinira since i during varijable. Tada baza podataka sadrži sedam relacija: S_SINCE, S_DURING, S_NAME_DURING, S_STATUS_DURING, S_CITY_DURING, SP_SINCE i SP_DURING. Bez smanjenja općenitosti, izbacimo parove atributa { SNAME, SNAME_SINCE } i { CITY, CITY_SINCE } iz S_SINCE te relacije S_NAME_DURING i S_CITY_DURING u cijelosti. Skica baze podataka slijedi u nastavku (izostavljena su sva ograničenja, osim onih koja se odnose na ključeve i strane ključeve):

S_SINCE { SNO, SNO_SINCE, STATUS, STATUS_SINCE }
    KEY { SNO }
SP_SINCE { SNO, PNO, SINCE }
    KEY { SNO, PNO }
    FOREIGN KEY { SNO } REFERENCES S_SINCE
S_DURING { SNO, DURING }
    KEY { SNO, DURING }
S_STATUS_DURING { SNO, STATUS, DURING }
    KEY { SNO, DURING }
SP_DURING { SNO, PNO, DURING }
    KEY { SNO, PNO, DURING }

Navodimo predikate za relacije iz pojednostavljenog prikaza:

1. **S_SINCE**: Dobavljač SNO je pod ugovorom od dana SNO_SINCE i ima status STATUS od dana STATUS_SINCE.
2. **SP_SINCE**: Dobavljač SNO je u mogućnosti dostavljati pošiljke klijentu PNO od dana SINCE.
3. **S_DURING**: DURING označava maksimalni interval tijekom kojeg je dobavljač SNO bio pod ugovorom.
4. **S_STATUS_DURING**: DURING označava maksimalni interval tijekom kojeg je dobavljač SNO imao status STATUS.
5. **SP_DURING**: DURING označava maksimalni interval tijekom kojeg je dobavljač SNO bio u mogućnosti dostavljati pošiljke klijentu PNO.

U temporalnim bazama podataka, poput one koju ovdje promatramo, mogu se pojaviti tri problema ako izostavimo odgovarajuća ograničenja za čuvanje integriteta. Radi se o *problemima redundancije*, *okolišanja* i *kontradikcije*.

Promotrimo relaciju S_STATUS_DURING. Slijedi formalna definicija u jeziku Tutorial D:

```
VAR S_STATUS_DURING BASE RELATION
    { SNO SNO, STATUS INTEGER, DURING INTERVAL_DATE }
    KEY { SNO, DURING } ;
```

Prvo ilustriramo problem redundancije. Ograničenje koje se odnosi na ključ, KEY, je logički ispravno, ali nije u potpunosti prikladno jer omogućuje da relacija S_STATUS_DURING sadrži sljedeće retke u isto vrijeme:

SNO	STATUS	DURING
S4	25	[d04:d05]

SNO	STATUS	DURING
S2	25	[d05:d06]

Jasno je da prethodna dva retka sadrže redundanciju, odnosno oba kazuju da je status dobavljača S4 na dan 5 jednak 25. Pojava tih redaka je jednaka loša kao i pojava duplikata u relaciji. Dakle, htjeli bismo ta dva retka zamijeniti sljedećim:

SNO	STATUS	DURING
S4	25	[d04:d06]

Primijetimo da je zadnji redak *pakirana* verzija prva dva.

Nadalje, ilustriramo problem okolišanja. Ograničenje koje se odnosi na ključ nije prikladno iz još jednoga razloga. Naime, ono ne sprječava da relacija S_STATUS_DURING sadrži sljedeće retke u isto vrijeme:

SNO	STATUS	DURING
S4	25	[d04:d04]

SNO	STATUS	DURING
S2	25	[d05:d06]

Ovdje je prisutno određeno okolišanje u smislu da dva retka kazuju nešto što bi se bolje izrazilo samo jednim pakiranim retkom:

SNO	STATUS	DURING
S4	25	[d04:d06]

Kako bismo riješili probleme redundancije i okolišanja, koje smo prethodno opisali, moramo uvesti sljedeće ograničenje: Ako u bilo kojem trenutku relacija S_STATUS_DURING sadrži dva retka koja se podudaraju, osim u vrijednostima atributa DURING *i1* i *i2*, tada *i1* MERGES *i2* ne smije biti istinito. Ograničenje ćemo uvesti tako da osiguramo da je relacija S_STATUS_DURING cijelo vrijeme pakirana na atribut DURING. U literaturi [1] predlaže se nova specifikacija PACKED ON sa sintaksom PACKED ON (*ACL*), gdje je *ACL* niz atributa kao i prije:

```
VAR S_STATUS_DURING BASE RELATION
    { SNO SNO, STATUS INTEGER, DURING INTERVAL_DATE }
    PACKED ON (DURING)
    KEY { SNO, DURING } ;
```

Specifikacija PACKED ON (DURING) ovdje implicira da će svi pokušaji ažuriranja relacije na način da krajnji rezultat nije pakiran na atribut DURING biti odbijeni. Tako se rješavaju problemi redundancije i okolišanja.

Na kraju, ilustriramo problem kontradikcije. Ograničenja PACKED ON i KEY zajedno nisu dovoljna za čuvanje integriteta relacije S_STATUS_DURING. Naime, ona ne sprječavaju pojavu sljedećih redaka u relaciji:

SNO	STATUS	DURING
S4	10	[d04:d06]

SNO	STATUS	DURING
S4	25	[d05:d07]

Iz prethodnih redaka iščitavamo informaciju da je status dobavljača S4 jednak 10 i 25 na dane 5 i 6. Naravno, takva situacija je nemoguća, stoga imamo kontradikciju. Kako bismo izbjegli kontradikcije poput prethodne, moramo uvesti dodatno ograničenje: Ako u bilo kojem trenutku relacija S_STATUS_DURING sadrži dva retka u kojima se podudaraju vrijednosti atributa SNO, a razlikuju vrijednosti atributa STATUS, tada njihove vrijednosti atributa DURING *i1* i *i2* moraju biti takve da *i1* OVERLAPS *i2* nije istinito. Ključno je uočiti da je { SNO, DURING } ključ za raspakiranu formu relacije S_STATUS_DURING budući da dobavljač pod ugovorom ima točno jedan status u nekom trenutku. Kao rješenje problema kontradikcije u literaturi [1] predlaže se nova specifikacija WHEN / THEN sa sintaksom kao u nastavku:

```
VAR S_STATUS_DURING BASE RELATION
    { SNO SNO, STATUS INTEGER, DURING INTERVAL_DATE }
    PACKED ON (DURING)
    WHEN UNPACKED ON (DURING) THEN KEY { SNO, DURING }
    KEY { SNO, DURING } ;
```

Specifikacija WHEN UNPACKED ON (DURING) THEN KEY { SNO, DURING } implicira da će svaki pokušaj ažuriranja relacije na način da raspakirana forma rezultata narušava funkcionalnu ovisnost { SNO, DURING } $\rightarrow$ { STATUS } biti odbijen. Ograničenja WHEN / THEN, PACKED ON i KEY zajedno su dovoljne da riješe probleme koje smo ovdje opisali. U praksi se često koriste temporalne relacije poput relacije S_STATUS_DURING na koje se primjenjuju navedena ograničenja. Zato se predlaže sintaksa koja kombinira funkcionalnost svih triju ograničenja: USING (*ACL*) : KEY { *K* }, gdje je *K* ključ, a *ACL* niz

atributa iz *K*. Ovdje { *K* } kraće nazivamo **U_KEY**. Prethodna specifikacija je pokrata za sljedeće:

PACKED ON (*ACL*)
WHEN UNPACKED ON (*ACL*) THEN KEY { *K* }
KEY { *K* }.

Sada definiciju relacije S_STATUS_DURING možemo zapisati na sljedeći način:

VAR S_STATUS_DURING BASE RELATION
{ SNO SNO, STATUS INTEGER, DURING INTERVAL_DATE }
USING (DURING) : KEY { SNO, DURING } ;

Lako se uvjeriti da su regularne specifikacije oblika KEY { *K* } pokrata za specifikaciju oblika USING ( ) : KEY { *K* }. Dakle, regularna ograničenja KEY su poseban slučaj prethodno predložene sintakse. U skladu s tim možemo redefinirati sintaksu regularne specifikacije KEY: [USING (*ACL*) :] KEY { *K* }. Izraz u [. . .] je opcionalan, a možemo ga izostaviti ako i samo ako je *ACL* prazan. Tada svaki ključ postaje U_KEY.

Napomena 2.2.1. Ograničenje WHEN . . . THEN KEY { *K* } nema logički utjecaj na relaciju *r* ako je *K* cijelo zaglavlje od *r*. Ograničenje PACKED ON (*ACL*) nema logički utjecaj na relaciju *r* ako je skup atributa iz *r* koji nije uključen u *ACL* složen ključ od *r* o kojem su potpuno funkcionalno ovisni svi ostali atributi.

Primijetimo da su ograničenja WHEN UNPACKED ON (DURING) : KEY { SNO, DURING } i WHEN UNPACKED ON (DURING) : KEY { SNO, PNO, DURING } trivijalna u skladu s Napomenom 2.2.1.

Razmotrimo ukratko slučaj stranih U_ključeva. Ako se baza podataka sastoji od samo during relacija, tada definicija bilo koje relacije *r2* smije sadržavati sljedeću specifikaciju:

USING (*ACL*) : FOREIGN KEY { *FK* } REFERENCES *r1*.

Značenje specifikacije je sljedeće: Ako su relacije *r1* i *r2* obje raspakirane na atribute navedene u *ACL*, tada *FK* predstavlja strani ključ u *r2* i U_KEY u relaciji *r1*.

U nastavku predstavljamo utjecaj prethodnog razmatranja na skicu baze podataka s početka potpoglavlja:

S_SINCE { SNO, SNO_SINCE, STATUS, STATUS_SINCE }
KEY { SNO }

```
SP_SINCE { SNO, PNO, SINCE }
    KEY { SNO, PNO }
    FOREIGN KEY { SNO } REFERENCES S_SINCE

S_DURING { SNO, DURING }
    USING (DURING) : KEY { SNO, DURING }

S_STATUS_DURING { SNO, STATUS, DURING }
    USING (DURING) : KEY { SNO, DURING }

SP_DURING { SNO, PNO, DURING }
    USING (DURING) : KEY { SNO, PNO, DURING }
```

2.3 Ostala ograničenja za čuvanje integriteta

Ovdje razmatramo sva ograničenja za čuvanje integriteta temporalne baze podataka o dobavljačima i pošiljkama. Ograničimo se ponovno na pojednostavljeni prikaz te baze iz prethodnog potpoglavlja. Temporalni podaci zahtijevaju posebnu pažnju. Često moraju zadovoljavati ograničenja koja se odnose na *gustoću*, u smislu da određeni zahtjevi moraju biti zadovoljeni u svakoj točki nekog intervala. U svrhu jednostavnijeg razumijevanja, ograničenja zapisana u prirodnom jeziku zvat ćemo zahtjevima, a definicije tih zahtjeva u formalnom jeziku Tutorial D ograničenjima. U nastavku navodimo zahtjeve koje temporalni podaci u promatranoj bazi podataka moraju zadovoljavati:

1. Ako baza podataka prikazuje da je dobavljač S*x* pod ugovorom na dan *d*, tada mora sadržavati točno jedan redak s tim podatkom.
2. Ako baza podataka prikazuje da je dobavljač S*x* pod ugovorom na dan *d* i dan *d+1*, tada mora sadržavati točno jedan redak s tim podatkom.
3. Ako baza podataka prikazuje da je dobavljač S*x* pod ugovorom na dan *d*, tada također mora prikazivati da je dobavljač S*x* imao neki status na dan *d*.

Primijetimo da se prvi zahtjev odnosi na problem redundancije, drugi na problem okolišanja, a treći na problem gustoće koji smo opisali u uvodnom dijelu.

4. Ako baza podataka prikazuje da dobavljač S*x* ima neki status na dan *d*, tada mora sadržavati točno jedan redak s tim podatkom.
5. Ako baza podataka prikazuje da dobavljač S*x* ima isti status na dan *d* i dan *d+1*, tada mora sadržavati točno jedan redak s tim podatkom.
6. Ako baza podataka prikazuje da dobavljač S*x* ima neki status na dan *d*, tada također mora prikazivati da je dobavljač S*x* pod ugovorom na dan *d*.

Četvrti zahtjev vezan je uz probleme redundancije i kontradikcije, peti uz problem okolišanja, a šesti uz problem gustoće.

7. Ako baza podataka prikazuje da je dobavljač S*x* u mogućnosti dostavljati pošiljke nekom klijentu P*y* na dan *d*, tada mora sadržavati točno jedan redak s tim podatkom.
8. Ako baza podataka prikazuje da je dobavljač S*x* u mogućnosti dostavljati pošiljke istom klijentu P*y* na dan *d* i dan *d+1*, tada mora sadržavati točno jedan redak s tim podatkom.
9. Ako baza podatak prikazuje da je dobavljač S*x* u mogućnosti dostavljati pošiljke nekom klijentu na dan *d*, tada također mora prikazivati da je dobavljač S*x* pod ugovorom na dan *d*.

Sedmi zahtjev odnosi se na problem redundancije, osmi zahtjev na problem okolišanja, a deveti na problem gustoće. U nastavku se bavimo njihovim utjecajem na oblikovanje temporalne baze podataka o dobavljačima i pošiljkama, no primijetimo da se analogoni ovih devet zahtjeva mogu primijeniti na bilo koju temporalnu bazu podataka.

Promotrimo prvo slučaj temporalnog oblikovanja koje se temelji na samo since relacijama. Tada se baza podataka u pitanju sastoji od dvije since relacije. Budući da se u tim relacijama ne pojavljuju atributi čije su vrijednosti intervali, ograničenja PACKED ON i WHEN / THEN nisu potrebna:

```
S_SINCE { SNO, SNO_SINCE, STATUS, STATUS_SINCE }
    KEY { SNO }
SP_SINCE { SNO, PNO, SINCE }
    KEY { SNO, PNO }
    FOREIGN KEY { SNO } REFERENCES S_SINCE.
```

Naravno, ova baza podataka je samo polutemporalna. Ne može sadržavati informacije o prošlosti, no može one o budućnosti. Ako u shemu uključimo ograničenja na bazu podataka koja odgovaraju iznesenim zahtjevima, tada je ona potpuno oblikovana:

```
S_SINCE { SNO, SNO_SINCE, STATUS, STATUS_SINCE }
    KEY { SNO }
CONSTRAINT SR6 IS_EMPTY
    ( S_SINCE WHERE STATUS_SINCE < SNO_SINCE ) ;
SP_SINCE { SNO, PNO, SINCE }
    KEY { SNO, PNO }
    FOREIGN KEY { SNO } REFERENCES S_SINCE
CONSTRAINT SR9 IS_EMPTY
    ( ( SP_SINCE JOIN S_SINCE ) WHERE SINCE < SNO_SINCE ) ;
```

Dalje promatramo slučaj temporalnog oblikovanja koje se temelji na samo during relacijama:

```
S_DURING { SNO, DURING }
    PACKED ON (DURING)
    KEY { SNO, DURING }
S_STATUS_DURING { SNO, STATUS, DURING }
    PACKED ON (DURING)
    WHEN UNPACKED ON (DURING) THEN KEY { SNO, DURING }
    KEY { SNO, DURING }
SP_DURING { SNO, PNO, DURING }
    PACKED ON (DURING)
    KEY { SNO, PNO, DURING }.
```

Baza podataka koja se sastoji od samo during relacija je potpuno temporalna. Pretpostavimo da horizontalna dekompozicija nije izvršena, odnosno relacije u ovom oblikovanju mogu reprezentirati informacije koje se odnose na prošlost, sadašnjost i budućnost. SQL pretpostavlja ovakvo oblikovanje temporalne baze podataka. Ako u shemu uključimo ograničenja na bazu podataka koja odgovaraju iznesenim zahtjevima, tada je ona potpuno oblikovana:

```
S_DURING { SNO, DURING }
    USING (DURING) : KEY { SNO, DURING }
    USING (DURING) : FOREIGN KEY { SNO, DURING }
            REFERENCES S_STATUS_DURING
S_STATUS_DURING { SNO, STATUS, DURING }
    USING (DURING) : KEY { SNO, DURING }
    USING (DURING) : FOREIGN KEY { SNO, DURING }
            REFERENCES S_DURING
SP_DURING { SNO, PNO, DURING }
    USING (DURING) : KEY { SNO, PNO, DURING }
    USING (DURING) : FOREIGN KEY { SNO, DURING }
            REFERENCES S_DURING.
```

Preostao je slučaj temporalnog oblikovanja koje se temelji na kombinaciji since i during relacija:

```
S_SINCE { SNO, SNO_SINCE, STATUS, STATUS_SINCE }
    KEY { SNO }
SP_SINCE { SNO, PNO, SINCE }
    KEY { SNO, PNO }
    FOREIGN KEY { SNO } REFERENCES S_SINCE
```

```
S_DURING { SNO, DURING }
    PACKED ON (DURING)
    KEY { SNO, DURING }
S_STATUS_DURING { SNO, STATUS, DURING }
    PACKED ON (DURING)
    WHEN UNPACKED ON (DURING) THEN KEY { SNO, DURING }
    KEY { SNO, DURING }
SP_DURING { SNO, PNO, DURING }
    PACKED ON (DURING)
    KEY { SNO, PNO, DURING }.
```

Za ovu bazu podataka eksplicitno su navedena ograničenja PACKED i WHEN / THEN. Zahtjevi koje mora zadovoljavati su složeniji nego prethodno jer se odgovarajuća ograničenja moraju referirati na kombinaciju since i during relacija. Ako u shemu uključimo ograničenja na bazu podataka koja odgovaraju iznesenim zahtjevima, tada je ona potpuno oblikovana:

```
S_SINCE { SNO, SNO_SINCE, STATUS, STATUS_SINCE } KEY { SNO }
SP_SINCE { SNO, PNO, SINCE } KEY { SNO, PNO }
    FOREIGN KEY { SNO } REFERENCES S_SINCE
S_DURING { SNO, DURING } USING (DURING) : KEY { SNO, DURING }
S_STATUS_DURING { SNO, STATUS, DURING }
    USING (DURING) : KEY { SNO, DURING }
SP_DURING { SNO, PNO, DURING }
    USING (DURING) : KEY { SNO, PNO, DURING }
CONSTRAINT BR12 IS_EMPTY (
    (S_SINCE JOIN S_DURING) WHERE SNO_SINCE ≤ POST (DURING) ) ;
CONSTRAINT BR36 WITH (
    t1 := EXTEND S_SINCE : { DURING := INTERVAL_DATE
            ( [SNO_SINCE : LAST_DATE( )] ) },
    t2 := t1 { SNO, DURING },
    t3 := t2 UNION S_DURING,
    t4 := EXTEND S_SINCE : { DURING := INTERVAL_DATE
            ( [STATUS_SINCE : LAST_DATE( )] ) },
    t5 := t4 { SNO, STATUS, DURING },
    t6 := t5 UNION S_STATUS_DURING,
    t7 := t6 { SNO, DURING } ) :
    USING (DURING) : t3 = t7 ;
CONSTRAINT BR3X S_DURING { SNO } ⊆ S_STATUS_DURING { SNO } ;
CONSTRAINT BR4 IS_EMPTY (
    ( S_SINCE JOIN S_STATUS_DURING { SNO, DURING } )
    WHERE STATUS_SINCE < POST (DURING) ) ;
CONSTRAINT BR5 IS_EMPTY ( ( S_SINCE JOIN S_STATUS_DURING )
    WHERE STATUS_SINCE = POST (DURING) ) ;
```

```
CONSTRAINT BR78 IS_EMPTY ( ( SP_SINCE JOIN SP_DURING )
    WHERE SINCE ≤ POST (DURING) ) ;
CONSTRAINT BR9 WITH (
    t1 := EXTEND S_SINCE : { DURING := INTERVAL_DATE
            ( [SNO_SINCE : LAST_DATE( )] ) },
    t2 := t1 { SNO, DURING },
    t3 := t2 UNION S_DURING,
    t4 := EXTEND SP_SINCE : { DURING := INTERVAL_DATE
            ( [STATUS_SINCE : LAST_DATE( )] ) },
    t5 := t4 { SNO, DURING },
    t6 := SP_DURING { SNO, DURING },
    t7 := t5 UNION t6 ) :
    USING (DURING) : t3 ⊇ t7 ;
```

Zahtjevi i odgovarajuća ograničenja detaljno su razrađeni u [1]. Uočimo da temporalno oblikovanje koje uključuje since i during relacije rezultira najsloženijim ograničenjima za čuvanje integriteta temporalne baze podataka. Navedena ograničenja predstavljaju svojevrsni obrazac za ograničenja u drugim temporalnim bazama. Autori u [1] predlažu uvođenje sintaktičkih pokrata na temelju kojih bi sustav za upravljanje bazom podataka sam uključio Ograničenja BR12, BR36, BR3X, BR4, BR5, BR78 i BR9. Tada se ta ograničenja ne bi morala eksplicitno navoditi.

Zaključimo potpoglavlje s nekoliko opaski:

1. U temporalnom kontekstu malo je vjerojatno da će doći do brisanja during relacija jer je smisao baze podataka da sačuva povijesne zapise. Since relacije mogu se izbrisati, no nužno je da se prvo sve informacije koje sadrže premjeste u odgovarajuće during relacije. Brisanje bilo koje vrste relacije vjerojatno zahtijeva izmjenu postojećih ograničenja.
2. U temporalnim bazama podataka ne preporuča se dodavanje novog atributa u during relacije jer svaka takva relacija reprezentira povijest samo jednog *svojstva*. Dodavanje novog atributa u since relacije je smislenije, no ono zahtijeva uvođenje dodatnog *since* atributa te prateću during relaciju. Vjerojatno su potrebna i nova ograničenja.
3. Ograničenja za čuvanje integriteta baze podataka mijenjaju se s vremenom, no ona ograničenja koja smo ovdje predstavili manje su podležna promjenama.

2.4 Temporalni upiti

Postavljanje upita u temporalnim bazama podataka je dosta složeno. Međutim, postavljanje upita mogu olakšati predefinirane kolekcije virtualnih relacija, koje nazivamo *pogledima*,

za bazu podataka u pitanju. Općenito govoreći, virtualne relacije poništavaju djelovanje horizontalne i vertikalne dekompozicije. Važno je razumijeti da se dekompozicije samo *konceptualno* poništavaju. Na primjer, ako promatramo neku temporalnu bazu podataka, tada možemo definirati prirodni spoj njenih relacija kao virtualnu relaciju koja, iako će vjerojatno narušavati principe normalizacije, omogućuje postavljanje određenih upita na jednostavniji način. S druge strane, pojedine temporalne upite teže je izraziti koristeći poglede. Primjeri nekih temporalnih upita u jeziku SQL mogu se vidjeti u zadnjem poglavlju.

Ažuriranje temporalnih baza podataka

Osvrnimo se posebno na upite kojima se ažurira temporalna baza podataka. Kao i obično, primjere temeljimo na trima verzijama temporalne baze podataka o dobavljačima i pošiljkama koje smo opisali u Potpoglavlju 2.1 te pretpostavljamo da su ugrađena sva ograničenja za čuvanje integriteta koja su iznesena u Potpoglavljima 2.2 i 2.3.

Promotrimo prvo slučaj polutemporalne baze podataka sa samo since relacijama. Ta baza podataka sadrži relacije S_SINCE i SP_SINCE.

S_SINCE

SNO	A	STATUS	B
S1	d04	20	d06
S2	d07	10	d07
S3	d03	30	d03
S4	d14	20	d14
S5	d02	30	d02

SP_SINCE

SNO	PNO	SINCE
S1	P1	d04
S1	P2	d05
S1	P3	d09
S1	P4	d05
S1	P5	d04
S1	P6	d06
S2	P1	d08
S2	P2	d09
S3	P2	d08
S4	P5	d14

Slika 2.1: Polutemporalna baza podataka za Upite 1 – 3

Radi jednostavnosti atributi SNO_SINCE i STATUS_SINCE na Slici 2.1 i Slici 2.3 označeni su s A i B. U nastavku koristimo uobičajene nazive.

Bitno je napomenuti da bazu podataka shvaćamo kao skup propozicija koje su trenutačno istinite. Ažuriranje baze podataka možemo shvatiti kao ubacivanje, promjenu ili brisanje propozicija. U nastavku slijede primjeri upita u jeziku Tutorial D koji se odnose na ažuriranje naše baze podataka.

Upit 1: Dodaj propoziciju koja kazuje da je dobavljač S9 pod ugovorom počevši od danas sa statusom 15.

```
INSERT S_SINCE RELATION
    { TUPLE { SNO SNO( 'S9' ), SNO_SINCE TODAY( ) ,
    STATUS 15, STATUS_SINCE TODAY( ) } } ;
```

Pretpostavimo da postoji operator TODAY čija je povratna vrijednost današnji datum. Ako je danas dan 10, naredba INSERT dodaje sljedeći redak u relaciju S_SINCE:

SNO	SNO_SINCE	STATUS	STATUS_SINCE
S9	d10	15	d10

Upit 2: Izbriši propoziciju koja kazuje da je dobavljač S5 pod ugovorom.

```
DELETE S_SINCE WHERE SNO = SNO( 'S5' ) ;
```

Naredbom DELETE uklanja se sljedeći redak iz relacije S_SINCE:

SNO	SNO_SINCE	STATUS	STATUS_SINCE
S5	d02	30	d02

Upit 3: Zamijeni propoziciju koja kazuje da je dobavljač S1 pod ugovorom od dana 4, propozicijom koja kazuje da je pod ugovorom od dana 3.

```
UPDATE S_SINCE WHERE SNO = SNO( 'S1' ) : { SNO_SINCE := d03 } ;
```

Naredbom UPDATE prvo se uklanja, a onda dodaje odgovarajući redak u relaciju S_SINCE:

SNO	SNO_SINCE	STATUS	STATUS_SINCE
S1	d03	20	d06

Primijetimo da se ažuriranje polutemporalne baze podataka ne razlikuje značajnije od ažuriranja bilo koje netemporalne baze podataka.

Dalje se bavimo slučajem temporalne baze podataka sa samo during relacijama. Pretpostavimo da horizontalna dekompozicija nije izvršena (*d99* je vrijednost koja označava *zadnji dan*). Baza podataka sastoji se od relacija S_DURING, S_STATUS_DURING i SP_DURING.

S_STATUS_DURING

SNO	STATUS	DURING
S2	5	[d02:d02]
S2	10	[d03:d04]
S6	5	[d03:d04]
S6	7	[d05:d05]
S7	15	[d03:d08]
S7	20	[d09:d99]

SP_DURING

SNO	PNO	DURING
S2	P1	[d02:d04]
S2	P2	[d03:d03]
S2	P5	[d03:d04]
S6	P3	[d03:d05]
S6	P4	[d04:d04]
S6	P5	[d04:d05]
S7	P1	[d03:d04]
S7	P1	[d06:d07]
S7	P1	[d09:d99]

S_DURING

SNO	DURING
S2	[d02:d04]
S6	[d03:d05]
S7	[d03:d99]

Slika 2.2: Temporalna baza podataka za Upite 4 – 11

Upit 4: Dodaj propoziciju koja kazuje da je dobavljač S2 bio u mogućnosti dostavljati pošiljke klijentu P4 na dan 2.

```
INSERT SP_DURING RELATION
    { TUPLE { SNO SNO ( 'S2' ), PNO PNO ( 'P4' ),
    DURING INTERVAL_DATE( [d02:d02] ) } } ;
```

Pretpostavimo da smo umjesto P4 imali P5. Kada bismo odgovarajući redak jednostavno dodali u relaciju SP_DURING, tada bi ona sadržavala sljedeća dva retka:

SNO	PNO	DURING
S2	P5	[d02:d02]

SNO	PNO	DURING
S2	P5	[d03:d04]

Time se krši ograničenje PACKED ON na relaciju SP_DURING. Kako bismo riješili problem kršenja ograničenja PACKED ON, uvodimo operator U_INSERT.

Definicija 2.4.1. Neka su *r* i *R* relacije koje imaju istu shemu *T*. Neka je *ACL* niz atributa koji su komponente od *T* i čije su vrijednosti intervali. Tada izraz USING (*ACL*) : INSERT *R r* nazivamo **U_INSERT** te predstavlja pokratu za sljedeće: *R* := USING (*ACL*) : *R* UNION *r*.

Drugim riječima, ažuriranje se provodi tako da se relacije *r* i *R* obje raspakiraju na *ACL*, formira se unija raspakiranih relacija koja se potom pakira na *ACL*, a pakirani rezultat se pridružuje *R*. Ako je *ACL* prazan, tada oznaku USING i znak : možemo izostaviti, a U_INSERT se reducira na regularni INSERT. Ova opaska može se primijeniti na sve operatore koje ćemo definirati u nastavku. Sada možemo ispravno formulirati Upit 4:

```
USING (DURING) : INSERT SP_DURING RELATION
    { TUPLE { SNO SNO( 'S2' ), PNO PNO( 'P5' ),
    DURING INTERVAL_DATE( [d02:d02] ) } } ;
```

Upit 5: Izbriši propoziciju koja kazuje da je dobavljač S6 bio u mogućnosti dostavljati pošiljke klijentu P3 od dana 3 do dana 5.

Prije nego što objasnimo kako postići željeni učinak brisanja propozicije iz Upita 5, prisjetimo se razmatranja o stvarnom i transakcijskom vremenu. Ako je baza podataka jednom sadržavala propoziciju *p* koja je istinita, tada se ona smatra povijesnim zapisom, a povijesni zapisi bi se u temporalnoj bazi podataka trebali čuvati zauvijek s odgovarajućim transakcijskim vremenom koje indicira kada je *p* bila istinita. Međutim, *p* se ne smatra povijesnim zapisom ako naknadno utvrdimo da je netočna, možda je napravljena pogreška i uopće nije trebala biti reprezentirana u bazi podataka, tek tada se *p* može ukloniti. Slijedi formulacija Upita 5:

```
DELETE SP_DURING WHERE SNO = SNO( 'S6' )
    AND PNO = PNO( 'P3' )
    AND DURING = INTERVAL_DATE( [d03:d05] ) ;
```

Ako pretpostavimo da imamo interval od dana 4 do dana 5, tada odgovarajući DELETE neće imati utjecaja na relaciju jer ne postoji redak za dobavljača S6 i klijenta P3 s vrijednošću atributa DURING *[d04:d05]*. Zato definiramo operator U_DELETE_WHERE.

Definicija 2.4.2. Neka je *R* relacija koja ima shemu *T*. Neka je *ACL* niz atributa koji su komponente od *T* i čije su vrijednosti intervali. Tada izraz USING (*ACL*) : DELETE *R* WHERE *bx* nazivamo **U_DELETE_WHERE** te predstavlja pokratu za sljedeće: *R* := USING (*ACL*) : *R* WHERE NOT (*bx*).

Uočimo da je operator U_DELETE_WHERE definiran pomoću operatora U_RESTRICT. Ažuriranje relacije se provodi tako da se relacija *R* raspakira na *ACL*, a iz raspakirane relacije formira se nova relacija koja sadrži retke koji zadovoljavaju uvjet NOT (*bx*). Zatim se nova relacija pakira na *ACL*, a pakirani rezultat pridružuje *R*. Dakle, ispravna formulacija za upit s intervalom *[d04:d05]* je sljedeća:

```
USING (DURING) : DELETE SP_DURING WHERE SNO = SNO( 'S6' )
    AND PNO = PNO( 'P3' )
    AND DURING ⊆ INTERVAL_DATE( [d04:d05] ) ;
```

Utjecaj prethodne operacije na relaciju SP_DURING prikazan je u nastavku:

SP_DURING

SNO	PNO	DURING
S2	P1	[d02:d04]
S2	P2	[d03:d03]
S2	P5	[d03:d04]
S6	**P3**	**[d03:d03]**
S6	P4	[d04:d04]
S6	P5	[d04:d05]
S7	P1	[d03:d04]
S7	P1	[d06:d07]
S7	P1	[d09:d99]

Primijetimo da u našem konkretnom slučaju brisanje propozicija ne može kršiti ograničenje PACKED ON jer se raspakiravanje i pakiranje vrše na samo jednom atributu. Općenito, ova tvrdnja ne vrijedi za slučaj s dva ili više atributa.

Dodatno možemo definirati operator U_DELETE kojemu je temelj operator U_MINUS umjesto operatora U_RESTRICT.

Definicija 2.4.3. Neka su *r* i *R* relacije koje imaju istu shemu *T*. Neka je *ACL* niz atributa koji su komponente od *T* i čije su vrijednosti intervali. Tada izraz USING (*ACL*) : DELETE *R r* nazivamo **U_DELETE** te predstavlja pokratu za sljedeće: *R* := USING (*ACL*) : *R* MINUS *r*.

Budući da se u praksi češće susreću izrazi oblika USING (*ACL*) : DELETE *R* WHERE *bx* nego oni oblika USING (*ACL*) : *R* MINUS *r*, u nastavku koristimo naziv U_DELETE kako

bismo se referirali na prvi oblik.

Upit 6: Zamijeni propoziciju koja kazuje da je dobavljač S2 bio u mogućnosti dostavljati pošiljke klijentu P5 od dana 3 do dana 4, propozicijom koja kazuje da je to bio u mogućnosti od dana 2 do dana 4.

Pitanje ažuriranja propozicija u temporalnim bazama podataka analogno je pitanju brisanja propozicija koje smo prethodno opisali. Upit 6 možemo formulirati na sljedeći način:

```
UPDATE SP_DURING WHERE SNO = SNO( 'S2' )
    AND PNO = PNO( 'P5' )
    AND DURING = INTERVAL_DATE( [d03:d04] ) :
    { DURING := INTERVAL_DATE( [d02:d04] ) } ;
```

Ako pretpostavimo da promatramo propozicije *Dobavljač S2 je bio u mogućnosti dostavljati pošiljke klijentu P5 na dan 3* i *Dobavljač S2 je bio u mogućnosti dostavljati pošiljke klijentu P5 na dan 2*, tada odgovarajući INSERT nema utjecaja na relaciju SP_DURING jer SP_DURING ne sadrži redak za dobavljača S2 i klijenta P5 s vrijednošću atributa DURING *[d03:d03]*. Upit možemo ispravno formulirati pomoću operatora U_UPDATE.

Definicija 2.4.4. Neka je *R* relacija koja ima shemu *T*. Neka je *ACL* niz atributa koji su komponente od *T* i čije su vrijednosti intervali. Tada izraz USING (*ACL*) : UPDATE *R* WHERE *bx* : { *pridruživanje vrijednosti* } nazivamo **U_UPDATE** te predstavlja pokratu za sljedeće:

```
WITH (
t1 := UNPACK R ON (ACL) ,
t2 := t1 WHERE NOT (bx) ,
t3 := t1 MINUS t2 ,
t4 := EXTEND t3 : { pridruživanje vrijednosti } ,
t5 := t2 UNION t4 ) :
R := PACK t5 ON (ACL).
```

Formulacija upita slijedi u nastavku:

```
USING (DURING) : UPDATE SP_DURING WHERE SNO = SNO( 'S2' )
    AND PNO = PNO( 'P5' )
    AND DURING = INTERVAL_DATE( [d03:d03] ) :
    { DURING := INTERVAL_DATE( [d02:d02] ) } ;
```

Alternativno rješenje temelji se na specifikaciji PORTION koju ćemo opisati u nastavku.

Upit 7 (modificirana verzija Upita 5): Izbriši propoziciju koja kazuje da je dobavljač S6 bio u mogućnosti dostavljati pošiljke klijentu P3 od dana 4 do dana 5.

Vidjeli smo da se Upit 7 može formulirati na sljedeći način:

```
USING (DURING) : DELETE SP_DURING WHERE SNO = SNO( 'S6' )
    AND PNO = PNO( 'P3' )
    AND DURING ⊆ INTERVAL_DATE( [d04:d05] ) ;
```

No postoji formulacija u terminima specifikacije PORTION koja je također ispravna:

```
DELETE SP_DURING WHERE SNO = SNO( 'S6' )
    AND PNO = PNO( 'P3' ) :
    PORTION { DURING { INTERVAL_DATE( [d04:d05] ) } } ;
```

Prethodni upit možemo objasniti pomoću sljedeće definicije.

Definicija 2.4.5. Neka je *R* relacija koja ima shemu *T*. Neka je *A* atribut koji je komponenta od *T* i čije su vrijednosti intervali nekog tipa *I*. Neka je *ix* vrijednost intervala tipa *I*. Tada izraz DELETE *R* WHERE *bx* : PORTION { *A* { *ix* } } nazivamo **PORTION DELETE** te predstavlja pokratu za sljedeće:

```
WITH (
t1 := R WHERE (bx) AND A OVERLAPS (ix) ,
t2 := R MINUS t1 ,
t3 := UNPACK t1 ON (A) ,
t4 := t3 WHERE NOT (A ⊆ (ix) ) ,
t5 := t2 UNION t4 ) :
R := PACK t5 ON (A).
```

Budući da relacije mogu sadržavati više atributa čije su vrijednosti intervali, razumno je dopustiti izraze oblika:

DELETE *R* WHERE *bx* : PORTION { *A1* { *ix1* }, *A2* { *ix2* }, . . . , *An* { *ixn* } } ;

Također, razumno je dopustiti i izraze sljedećeg oblika:

DELETE *R* WHERE *bx* : PORTION { *A* { *ix1*, *ix2*, . . . , *ixn* } } ;

Slijedi jednostavan primjer:

```
DELETE SP_DURING WHERE SNO = SNO( 'S7' ) :
    PORTION { DURING { INTERVAL_DATE( [d04:d06] ),
    INTERVAL_DATE( [d12:d99] ) } } ;
```

Rezultat upita prikazan je u nastavku:

SP_DURING

SNO	PNO	DURING
S2	P1	[d02:d04]
S2	P2	[d03:d03]
S2	P5	[d03:d04]
S6	P3	[d03:d05]
S6	P4	[d04:d04]
S6	P5	[d04:d05]
S7	**P1**	**[d03:d03]**
S7	**P1**	**[d07:d07]**
S7	**P1**	**[d09:d11]**

Upit 8 (modificirana verzija Upita 6): Zamijeni propoziciju koja kazuje da je dobavljač S2 bio u mogućnosti dostavljati pošiljke klijentu P5 na dan 3, propozicijom koja kazuje da je to bio u mogućnosti na dan 2.

```
UPDATE SP_DURING WHERE SNO = SNO( 'S2' ) AND PNO = PNO( 'P5' ) :
    PORTION { DURING { INTERVAL_DATE( [d03:d03] ) } } :
    { DURING := INTERVAL_DATE( [d02:d02] ) } ;
```

Intuitivno bi trebalo biti jasno kako radi prethodni primjer.

Upit 9: Dodaj propozicije koje kazuju da je dobavljač S9 pod ugovorom počevši od danas sa statusom 15.

```
INSERT S_DURING RELATION { TUPLE {
    SNO SNO( 'S9' ),
    DURING INTERVAL_DATE( [TODAY( ):LAST_DAY( )] ) } } ,

INSERT S_STATUS_DURING RELATION { TUPLE {
    SNO SNO( 'S9' ), STATUS 15,
    DURING INTERVAL_DATE( [TODAY( ):LAST_DAY( )] ) } } ;
```

Budući da su ciljne relacije S_DURING i S_STATUS_DURING, potrebne su dvije naredbe INSERT. Primijetimo da su odvojene zarezom pa zajedno predstavljaju jednu izjavu. Obje relacije ažuriraju se u isto vrijeme.

S_SINCE

SNO	A	STATUS	B
S1	d04	20	d06
S2	d07	10	d07
S3	d03	30	d03
S4	d04	20	d08
S5	d02	30	d02

SP_SINCE

SNO	PNO	SINCE
S1	P1	d04
S1	P2	d05
S1	P3	d09
S1	P4	d05
S1	P5	d04
S1	P6	d06
S2	P1	d08
S2	P2	d09
S3	P2	d08
S4	P5	d05

S_DURING

SNO	DURING
S2	[d02:d04]
S6	[d03:d05]

S_STATUS_DURING

SNO	STATUS	DURING
S1	15	[d04:d05]
S2	5	[d02:d02]
S2	10	[d03:d04]
S4	10	[d04:d04]
S4	25	[d05:d07]
S6	5	[d03:d04]
S6	7	[d05:d05]

SP_DURING

SNO	PNO	DURING
S2	P1	[d02:d04]
S2	P2	[d03:d03]
S3	P5	[d05:d07]
S4	P2	[d06:d09]
S4	P4	[d04:d08]
S6	P3	[d03:d03]
S6	P3	[d05:d05]

Slika 2.3: Temporalna baza podataka za Upite 12 – 14

Upit 10: Obriši sve propozicije koje kazuju da je dobavljač S7 pod ugovorom.

```
DELETE SP_DURING WHERE SNO = SNO( 'S7' ) ,
DELETE S_STATUS_DURING WHERE SNO = SNO( 'S7' ) ,
DELETE S_DURING WHERE SNO = SNO( 'S7' ) ;
```

Upit 11: Ugovor dobavljača S7 je upravo raskinut. Odgovarajuće ažuriraj bazu podataka.

```
UPDATE S_DURING WHERE SNO = SNO( 'S7' )
    AND TODAY( ) ∈ DURING : { END (DURING) := TODAY( ) } ,
```

```
UPDATE S_STATUS_DURING WHERE SNO = SNO( 'S7' )
    AND TODAY( ) ∈ DURING : { END (DURING) := TODAY( ) } ,
UPDATE SP_DURING WHERE SNO = SNO( 'S7' )
    AND TODAY( ) ∈ DURING : { END (DURING) := TODAY( ) } ;
```

Na kraju, bavimo se slučajem temporalne baze podataka sa since i during relacijama. Takva baza podataka *trenutne* informacije čuva u since relacijama, a *povijesne* u during relacijama. Baza podataka za Upite 12 – 14 prikazana je na Slici 2.3.

Upit 12: Dodaj propoziciju koja kazuje da je dobavljač S4 u mogućnosti dostavljati pošiljke klijentu P4 od dana 10.

```
INSERT SP_SINCE RELATION { TUPLE {
    SNO SNO( 'S4' ), PNO PNO( 'P4' ), SINCE d10 } } ;
```

Da je u Upitu 12 bio naveden dan 9 umjesto dana 10, relacije SP_SINCE i SP_DURING bi nakon odgovarajućeg INSERT-a sadržavale sljedeće retke:

SNO	PNO	SINCE
S4	P4	d09

SNO	PNO	DURING
S4	P4	[d04:d08]

Tako se krši osmi zahtjev na bazu podataka koji je opisan u Potpoglavlju 2.3. Problem se može riješiti tako da iz SP_DURING izbrišemo redak za S4 i P4 s vrijednošću atributa DURING *[d04:d08]* te u SP_SINCE ubacimo redak za S4 i P4 s vrijednošću atributa SINCE *d04*.

```
DELETE SP_DURING WHERE SNO = SNO( 'S4' )
    AND PNO = PNO( 'P4' )
    AND DURING = INTERVAL_DATE( [d04:d08] ) ,
INSERT SP_SINCE RELATION { TUPLE {
    SNO SNO( 'S4' ), PNO PNO( 'P4' ), SINCE d04 } } ;
```

Upit 13: Od sutra dobavljač S1 više neće biti u mogućnosti dostavljati pošiljke klijentu P1. Odgovarajuće ažuriraj bazu podataka.

```
INSERT SP_DURING RELATION { TUPLE { SNO SNO( 'S1' ) ,
    PNO PNO( 'P1' ), DURING INTERVAL_DATE( [d04:TODAY( ) } } ,
DELETE SP_SINCE WHERE SNO = SNO( 'S1' )
    AND PNO = PNO( 'P1' ) ;
```

Upit 14: Zamijeni propoziciju koja kazuje da je dobavljač S2 u mogućnosti dostavljati pošiljke klijentu P1 od dana 8, propozicijom koja kazuje da je to u mogućnosti od dana 7.

```
UPDATE SP_SINCE WHERE SNO = SNO( 'S2' )
    AND PNO = PNO( 'P1' ) : { SINCE := d07 } ;
```

Ako pretpostavimo da je u Upitu 14 nova vrijednost atributa SINCE dan 5, a ne dan 7, tada odgovarajući UPDATE krši osmi zahtjev na bazu podataka iz Potpoglavlja 2.3. Odnosno, problem predstavljaju sljedeća dva retka:

SNO	PNO	SINCE
S2	P1	d05

SNO	PNO	DURING
S2	P1	[d02:d04]

Dakle, ispravno je postupiti kako slijedi:

```
DELETE SP_DURING WHERE SNO = SNO( 'S2' )
    AND PNO = PNO( 'P1' )
    AND END(DURING) = d04 ,

UPDATE SP_SINCE WHERE SNO = SNO( 'S2' )
    AND PNO = PNO( 'P1' ) : { SINCE := d07 } ;
```

Svi upiti koji su predstavljeni u ovom radu preuzeti su iz [1].

2.5 Transakcijsko vrijeme u temporalnim bazama podataka

U Potpoglavlju 1.1 predstavili smo koncept *stvarnog* i *transakcijskog vremena*. U standardu za SQL koriste se pripadni nazivi *aplikacijsko* i *sistemsko vrijeme*. Prisjetimo se kako smo intuitivno objasnili te pojmove. Stvarna vremena odnose se na nešto u što trenutno vjerujemo da je istinito. S druge strane, transakcijska vremena referiraju se na to kada je baza podataka rekla da je nešto bilo istinito. U nastavku detaljnije razrađujemo te koncepte. Stvarna vremena smatramo običnim podacima jer se mogu reprezentirati pomoću atributa u relacijama. Ti atributi ne razlikuju se značajnije od bilo kojih drugih atributa, možemo ih ažurirati i pristupati im putem upita. Iz tog razloga nisu posebno zanimljiva. Međutim, transakcijska vremena ne mogu se ažurirati poput stvarnih vremena, njih održava sustav, niti smo u mogućnosti postavljati upite koji su vezani uz njih. Autori u [1] smatraju da transakcijska vremena moraju biti dostupna u standardnom relacijskom obliku, odnosno potrebne su nam pomoćne relacije koje prikazuje povijest svih transakcija za neku bazu podataka. Te pomoćne relacije možemo shvatiti kao analogone žurnal–datoteka koje se koriste u slučajevima oštećenja baze podataka. Promotrimo sljedeći primjer:

```
VAR S_DURING BASE RELATION {
    SNO SNO, DURING INTERVAL_DATE }
    USING (DURING) : KEY { SNO, DURING }
    LOGGED_TIMES_IN (S_DURING_LOG) ;
```

Upotrebom specifikacije LOGGED_TIMES_IN postiže se da sustav kreira novu relaciju s atributima SNO, DURING i X_DURING: S_DURING_LOG. U toj relaciji retci sadrže transakcijska vremena za sve retke koji su se ikada pojavili u relaciji S_DURING. Na primjer, Slika 2.4 prikazuje moguće vrijednosti za relacije S_DURING i S_DURING_LOG.

S_DURING

SNO	DURING
S2	[d02:d04]
S6	[d03:d05]

S_DURING_LOG

SNO	DURING	X_DURING
S2	[d02:d04]	[d04:d07]
S2	[d02:d04]	[d10:d20]
S2	[d02:d04]	[d50:d70]
S6	[d02:d05]	[d15:d25]
S6	[d03:d05]	[d26:d70]
S1	[d01:d01]	[d20:d30]
S1	[d05:d06]	[d40:d50]

Slika 2.4: Prikaz relacija S_DURING i S_DURING_LOG

Pretpostavimo da je danas dan 70. Relacija S_DURING trenutačno prikazuje da je dobavljač S2 bio pod ugovorom od dana 2 do dana 4. Prvi redak u S_DURING_LOG nam sugerira da je S_DURING ranije prikazivao isti podatak, od dana 4 do dana 7. Drugi redak u S_DURING_LOG nam ponovno sugerira da je S_DURING ranije prikazivao isti podatak, od dana 10 do dana 20. Treći redak u S_DURING_LOG nam kazuje da S_DURING prikazuje isti podatak od dana 50 do danas. Primijetimo da se transakcijska vremena ne mogu odnositi na budućnost. Relacija S_DURING također trenutačno prikazuje da je dobavljač S6 bio pod ugovorom od *dana 3* do dana 5. S druge strane, redak 4 u S_DURING_LOG nam kazuje da je od dana 15 do dana 25 relacija S_DURING sadržavala podatak da je dobavljač S6 bio pod ugovorom od *dana 2* do dana 5. Redak 5 u S_DURING_LOG međutim pokazuje da od dana 26 S_DURING sadrži podatak da je dobavljač S6 bio pod ugovorom od *dana 3* do dana 5. S_DURING trenutačno ne sadrži informacije o dobavljaču S1. No redak 6 u S_DURING_LOG pokazuje da je od dana 20 do dana 30 relacija S_DURING sadržavala podatak da je dobavljač S1 bio pod ugovorom na dan 1. Analogno, redak 7 pokazuje da je od dana 40 do dana 50 S_DURING sadržavao informaciju da je dobavljač S1 bio pod ugovorom od dana 5 do dana 6. Zaključujemo da su informacije o dobavljaču

S1 obrisane iz S_DURING na dan 31 i ponovno na dan 51 (jer je vjerojatno utvrđeno da podaci nisu ispravni).

Iznesimo nekoliko zaključnih opaski koje se mogu općenito primijeniti:

1. Relacija S_DURING_LOG ne mora postojati cijelo vrijeme, dovoljno je da je sustav kreira kada se referencira u nekom upitu.
2. Relaciju S_DURING_LOG ažurira sustav, a ne obični korisnici. Proces zamjene svih *d70* sa *d71* u ponoć na dan 70 automatski provodi sustav.
3. Zaglavlje relacije S_DURING_LOG odgovara zaglavlju relacije S_DURING koje je prošireno dodatnim atributom X_DURING. Vrijednosti atributa X_DURING su intervali.
4. Ako relacija S_DURING zadovoljava ograničenje USING (DURING) : KEY { SNO, DURING }, tada relacija S_DURING_LOG zadovoljava ograničenje USING (DURING, X_DURING) : KEY { SNO, DURING, X_DURING }.
5. Redak koji sadrži jednu vremensku oznaku za stvarno vrijeme, a drugu za transakcijsko vrijeme, u literaturi se naziva *bitemporalnim retkom*.

SNO	DURING	X_DURING
S2	[d02:d04]	[d04:d07]

Slika 2.5: Bitemporalni redak

Promotrimo upit koji se odnosi na transakcijsko vrijeme.

Upit: Kada je baza podataka pokazivala da je dobavljač S6 bio pod ugovorom na dan 4?

Informacije o transakcijskom vremenu prisutne su u relaciji S_DURING_LOG. Ako tu relaciju restringiramo samo na retke koji sadrže podatak da je dobavljač S6 bio pod ugovorom na dan 4, tada je rezultat sljedeći:

SNO	DURING	X_DURING
S6	[d02:d05]	[d15:d25]
S6	[d03:d05]	[d26:d70]

Formulacija upita glasi:

```
WITH (
t1 := S_DURING_LOG WHERE SNO = SNO( 'S6' ) AND d04 ∈ DURING ) :
USING (X_DURING) : t1 { X_DURING }.
```

Rezultat upita je:

X_DURING
[d15:d70]

Poglavlje 3

SQL podrška za temporalne podatke

Određene temporalne značajke uključene su u zadnju verziju standarda za SQL 2011. godine. U ovom poglavlju analiziramo SQL podršku za pristup temporalnim podacima i razne koncepte koji su izloženi u Poglavljima 1 i 2. Temporalne značajke u standardu za SQL mogu se smatrati velikim korakom u upravljanju temporalnim podacima, no nisu dovoljne da riješe sve probleme koje dolaze s njima, a koje smo identificirali u prethodnim razmatranjima.

3.1 Periodi

SQL ne podržava vremenske intervale koje smo predstavili u Potpoglavlju 1.3. Umjesto intervala podržava *periode* koji se sastoje od parova vrijednosti FROM i TO. Obično je riječ o parovima vrijednosti stupaca u SQL tablici. Štoviše, takvi periodi su temporalni po svojoj prirodi. Slika 3.1 prikazuje SQL verziju baze podataka o dobavljačima i pošiljkama. Primijetimo da je u potpuno temporalnom obliku. Baza podataka nalikuje onoj sa Slike 1.3, no stupce FROM i TO preimenovali smo u DFROM i DTO budući da su FROM i TO rezervirane riječi u SQL-u. Vrijednosti *d10* koje su označavale *kraj vremena* zamijenili smo vrijednošću *d99* koja označava 31. prosinca 9999. jer kada je SQL u pitanju, kraj vremena je DATE '9999-12-31', a početak vremena je DATE '0001-01-01'. SQL podržava i finiju zrnatost (engl. *granularity*).

Već smo prije rekli da su SQL periodi predstavljeni uređenim parom vrijednosti *from* i *to*. SQL takve parove interpretira na sljedeći način: vrijednost *from* označava prvu točku u periodu, a vrijednost *to* označava točku koja je neposredni sljedbenik zadnje točke u periodu. Dakle, SQL koristi [:) interpretaciju. Primijetimo da vrijednost *from* nikada ne smije biti jednaka *d99*, a vrijednost *to d100* budući da *d100* nije legalan datum u SQL-u (DATE '10000-01-01').

S_FROM_TO

SNO	DFROM	DTO
S1	d04	d99
S2	d02	d05
S2	d07	d99
S3	d03	d99
S4	d04	d99
S5	d02	d99

SP_FROM_TO

SNO	PNO	DFROM	DTO
S1	P1	d04	d99
S1	P2	d05	d99
S1	P3	d09	d99
S1	P4	d05	d99
S1	P5	d04	d99
S1	P6	d06	d99
S2	P1	d02	d05
S2	P1	d08	d99
S2	P2	d03	d04
S2	P2	d09	d99
S3	P2	d08	d99
S4	P2	d06	d10
S4	P4	d04	d09
S4	P5	d05	d99

Slika 3.1: SQL verzija baze podataka o dobavljačima i pošiljkama

Prikažimo početnu SQL definiciju za relaciju S_FROM_TO:

```
CREATE TABLE S_FROM_TO (
    SNO SNO NOT NULL ,
    DFROM DATE NOT NULL ,
    DTO DATE NOT NULL ,
    PERIOD FOR DPERIOD (DFROM, DTO) ,
    UNIQUE (SNO, DPERIOD WITHOUT OVERLAPS) ) ;
```

Klauzulom PERIOD FOR definira se da stupci DFROM i DTO određuju period DPERIOD. Primijetimo da DPERIOD sam nije stupac, ali dopušta se njegova upotreba u postavljanju upita. Stupci koji su spareni u specifikaciji PERIOD FOR moraju biti istog tipa: DATE ili TIMESTAMP. Ovdje nećemo ulaziti u detalje vezane uz vremenske tipove DATE i TIMESTAMP. Period DPERIOD reprezentira *aplikacijsko vrijeme*. Aplikacijsko vrijeme je SQL-ov izraz za stvarno vrijeme. SQL osigurava ograničenje da je za bilo koji takav period vrijednost *from* strogo manja od vrijednosti *to*. SQL tablica ne može imati više od jednog perioda koji reprezentira aplikacijsko vrijeme.

Operatori za rad s periodima

Neka su [*f, t*), [*f1, t1*) i [*f2, t2*) FROM – TO parovi koji odgovaraju SQL periodima *p*, *p1* i *p2*. Tablica u nastavku prikazuje SQL analogone operatora koje smo opisali u potpoglavlju *Operatori za rad s intervalima*.

Operator	SQL analogon
BEGIN	f
END	t – '1' DAY
PRE	f – '1' DAY
POST	t
POINT FROM	nije podržano
$\in$	nije podržano
$\ni$	p CONTAINS x
=	p1 EQUALS p2
$\supseteq$	p1 CONTAINS p2
$\supset$	nije podržano
$\subseteq$	nije podržano
$\subset$	nije podržano
BEFORE	p1 PRECEDES p2
AFTER	p1 SUCCEEDS p2
OVERLAPS	p1 OVERLAPS p2
MERGES	nije podržano
BEGINS	nije podržano
ENDS	nije podržano

Tablica 3.1: Operatori za rad s periodima

Operator MEETS može se izraziti na sljedeći način: *p1* IMMEDIATELY PRECEDES *p2* OR *p1* IMMEDIATELY SUCCEEDS *p2*. Operatori koji nisu izravno podržani mogu se izraziti pomoću postojećih. Definirali smo i operator COUNT za koji ne postoji odgovarajući SQL analogon, no može se simulirati oduzimanjem vrijednosti *from* od vrijednosti *to* (SQL sintaksa: CAST((*to* – *from*) AS INTEGER). Operatori UNION, INTERSECT i MINUS nemaju svoje SQL analogone.

Sintaksom PERIOD (*f, t*) označavamo operande nekog Allenovog operatora. Izrazi *f* i *t* su istog tipa (DATE ili TIMESTAMP). U nastavku slijedi primjer jednostavnog SQL upita:

```
SELECT DISTINCT SNO FROM S_FROM_TO WHERE PERIOD (DFROM, DTO)
    OVERLAPS PERIOD (DATE '2015-08-25', DATE '2015-12-25') ;
```

3.2 Operatori PACK i UNPACK i U_ operatori

SQL ne podržava operatore PACK i UNPACK. Nažalost, to je ozbiljan propust. U skladu s tim ne postoji ni podrška za generalizirane relacijske operatore poput operatora U_JOIN. Također, ne postoji podrška za operatore U_UPDATE, U_INSERT i U_DELETE.

Operator PACK se u svojem najjednostavnijem obliku (COLLAPSE) može definirati u terminima FORALL, EXISTS, $\in$, FIRST, LAST, PRE i POST. Budući da se sve prethodne

konstrukcije mogu simulirati u SQL-u, sigurno je moguće napisati SQL izraz koji za danu SQL tablicu vraća pakiranu verziju te tablice.

3.3 Oblikovanje temporalnih baza podataka

SQL koristeći tipove DATE ili TIMESTAMP može definirati polutemporalne tablice, no one se ne razlikuju značajnije od uobičajenih netemporalnih tablica. Zanimljivo je jedino prikazati kako se formuliraju SQL ograničenja za čuvanje integriteta baze podataka. Pogledajmo primjer za ograničenje SR6 iz Potpoglavlja 2.3:

```
CREATE ASSERTION SR6 CHECK ( NOT EXISTS
    (SELECT * FROM S_FROM WHERE STATUS_FROM < SNO_FROM) ) ;
```

Okrenimo se sada potpunom temporalnom oblikovanju. SQL pretpostavlja drugi pristup temporalnom oblikovanju sa samo during relacijama. Ne postoji izravna podrška za treći pristup, no ništa nas ne sprječava da odaberemo tu shemu ako želimo. SQL rješava problem koncepta *sada* uvođenjem posebne oznake koja predstavlja *kraj vremena*. Već smo vidjeli SQL definiciju za tablicu S_FROM_TO. Potpuna fizička shema za temporalnu bazu podataka s relacijama S_FROM_TO, S_STATUS_FROM_TO i SP_FROM_TO prikazana je na Slici 3.2.

Specifikacija UNIQUE sprječava pojavu problema redundancije i kontradikcije u tablicama, no ne rješava problem okolišanja. Intuitivno to možemo objasniti na sljedeći način: Neka je *r* redak u tablici S_FROM_TO s vrijednostima *sno*, *df* i *dt*. Ako iz retka *r* izvedemo po jedan redak za svaku DATE vrijednost *d*, gdje je $df \leq d \leq dt$, tada svaki takav redak sadrži vrijednosti *sno* i *d*. Neka je *T* tablica koja sadrži sve izvedene retke za svaki redak u S_FROM_TO. Tada ne postoje dva ista retka u tablici *T*. Prvu klauzulu FOREIGN KEY možemo objasniti na sličan način: Pretpostavimo da smo iz S_FROM_TO i S_STATUS_FROM_TO dobili odgovarajuće tablice *T1* i *T2*. Neka je D stupac u *T1* i *T2* koji sadrži odgovarajuće DATE vrijednosti *d*. Tada je { SNO, D } regularni strani ključ u *T1* i ključ u *T2*. Kao posljedica, tablice S_FROM_TO i S_STATUS_FROM_TO moraju se simultano ažurirati. SQL nema izravnu podršku za simultano ažuriranje kakvo smo vidjeli u Upitima 12 – 14.

SQL ne podržava specifikacije U_KEY ili FOREIGN U_KEY kao takve, no prethodno smo se uvjerili da podržava nešto vrlo slično. Zato se uspijeva nositi sa šest od devet zahtjeva koje smo naveli u Potpoglavlju 2.3. Zahtjevi 2, 5 i 8 nisu zadovoljeni jer tablice nisu nužno u svojoj pakiranoj formi budući da SQL podržava WITHOUT OVERLAPS, no ne i WITHOUT MERGES. Održavanje tablica u pakiranoj formi smatra se korisnikovom zadaćom.

```
CREATE TABLE S_FROM_TO (
SNO SNO NOT NULL ,
DFROM DATE NOT NULL ,
DTO DATE NOT NULL ,
PERIOD FOR DPERIOD (DFROM, DTO) ,
UNIQUE (SNO, DPERIOD WITHOUT OVERLAPS) ) ,
FOREIGN KEY (SNO, PERIOD DPERIOD) REFERENCES
        S_STATUS_FROM_TO (SNO, PERIOD DPERIOD ) ) ;

CREATE TABLE S_STATUS_FROM_TO (
SNO SNO NOT NULL ,
STATUS INTEGER NOT NULL ,
DFROM DATE NOT NULL ,
DTO DATE NOT NULL ,
PERIOD FOR DPERIOD (DFROM, DTO) ,
UNIQUE (SNO, DPERIOD WITHOUT OVERLAPS) ) ,
FOREIGN KEY (SNO, PERIOD DPERIOD) REFERENCES
        S_FROM_TO (SNO, PERIOD DPERIOD ) ) ;

CREATE TABLE SP_FROM_TO (
SNO SNO NOT NULL ,
PNO PNO NOT NULL ,
DFROM DATE NOT NULL ,
DTO DATE NOT NULL ,
PERIOD FOR DPERIOD (DFROM, DTO) ,
UNIQUE (SNO, PNO, DPERIOD WITHOUT OVERLAPS) ,
FOREIGN KEY (SNO, PERIOD DPERIOD) REFERENCES
        S_FROM_TO (SNO, PERIOD DPERIOD ) ) ;
```

Slika 3.2: SQL definicije za temporalnu bazu podataka

3.4 Temporalni upiti

Pogledajmo primjere nekih upita u jeziku SQL.

Upit 1: Pronađi status dobavljač S1 na dan *dn* i pripadni period.

```
SELECT STATUS, DFROM, DTO FROM S_STATUS_FROM_TO
    WHERE SNO = SNO( 'S1' ) AND DPERIOD CONTAINS dn ;
```

WITHOUT OVERLAPS jamči da tablica S_STATUS_FROM_TO sadrži najviše jedan redak koji zadovoljava zadani uvjet. Važno je istaknuti da se periodi koji reprezentiraju aplikacijsko vrijeme ne prenose u rezultantne tablice.

S_STATUS_FROM_TO

SNO	STATUS	DFROM	DTO
S2	5	d02	d03
S2	10	d03	d05
S6	5	d03	d05
S6	7	d05	d06
S7	15	d03	d09
S7	20	d09	d99

S_FROM_TO

SNO	DFROM	DTO
S2	d02	d05
S6	d03	d06
S7	d03	d99

SP_FROM_TO

SNO	PNO	DFROM	DTO
S2	P1	d02	d05
S1	P2	d03	d04
S1	P5	d03	d05
S6	P3	d03	d06
S6	P4	d04	d05
S6	P5	d04	d06
S7	P1	d03	d05
S7	P1	d06	d08
S7	P1	d09	d04

Slika 3.3: Baza podataka za Upite 1 – 4

Upit 2: Pronađi parove dobavljača koji trenutačni status imaju od istog dana.

```
WITH
t1 AS (SELECT SNO, DFROM FROM S_STATUS_FROM_TO
    WHERE DPERIOD CONTAINS CURRENT_DATE) ,
t2 AS (SELECT SNO AS XNO, DFROM FROM t1) ,
t3 AS (SELECT SNO AS YNO, DFROM FROM t1) ,
t4 AS (SELECT XNO, YNO FROM t2 JOIN t3)
SELECT XNO, YNO FROM t4 WHERE XNO < YNO ;
```

Ovdje koristimo SQL-ovu WITH konstrukciju. CURRENT_DATE je SQL-ova sistemska varijabla.

Upit 3: Dodaj propozicije koje kazuju da je dobavljač S9 pod ugovorom počevši od danas sa statusom 15.

```
INSERT INTO S_FROM_TO (SNO, DFROM, DTO)
```

```
        VALUES (SNO( 'S9' ), CURRENT_DATE, DATE '9999-12-31') ;
INSERT INTO S_STATUS_FROM_TO (SNO, STATUS, DFROM, DTO)
        VALUES (SNO( 'S9' ), 15, CURRENT_DATE, DATE '9999-12-31') ;
```

Primijetimo da obje tablica ima strani ključ koji referencira drugu, zato sva ažuriranja moraju biti simultana. Budući da se svaka SQL naredba smatra zasebnom cjelinom, ovaj niz naredbi moramo proglasiti transakcijom. Potvrda da je transakcija uredno završila postiže se naredbom COMMIT.

Upit 4: Obriši propoziciju koja kazuje da je dobavljač S7 bio u mogućnosti dostavljati pošiljke klijentu P1 od dana 4 do dana 11.

```
DELETE FROM SP_FROM_TO FOR PORTION OF
    DPERIOD FROM d04 TO d12
    WHERE SNO = SNO( 'S7' ) AND PNO = PNO( 'P1' ) ;
```

SQL podržava klauzulu **FOR PORTION OF** koja ima sličnu funkcionalnost kao PORTION iz Potpoglavlja 2.4. Opća sintaksa je: FOR PORTION OF *p* FROM *t1* TO *t2*, gdje su *p* aplikacijski period, a *t1* i *t2* odgovarajuće vremenske vrijednosti. Klauzula se može koristiti uz DELETE i UPDATE.

Rezultat Upita 4 prikazan je u nastavku:

SP_FROM_TO

SNO	PNO	DFROM	DTO
S2	P1	d02	d05
S1	P2	d03	d04
S1	P5	d03	d05
S6	P3	d03	d06
S6	P4	d04	d05
S6	P5	d04	d06
S7	**P1**	**d03**	**d04**
S7	**P1**	**d12**	**d99**

3.5 Sistemsko vrijeme

SQL pruža podršku za transakcijsko vrijeme te za njega koristi izraz *sistemsko vrijeme*. Pristup transakcijskom vremenu, opisan u Potpoglavlju 2.5, temelji se na pomoćnim relacijama koje reprezentiraju povijest svih transakcija koje su mijenjale bazu podataka, a generira ih i održava DBMS. S druge strane, SQL dopušta da sama tablica sadrži najviše jedan period koji reprezentira sistemsko vrijeme. Poželjno je da je takva tablica u 6NF. Promotrimo primjer u nastavku:

```
CREATE TABLE XS_STATUS_FROM_TO (
SNO SNO NOT NULL,
STATUS INTEGER NOT NULL,
XFROM TIMESTAMP(12) GENERATED ALWAYS AS ROW START NOT NULL,
XTO TIMESTAMP(12) GENERATED ALWAYS AS ROW END NOT NULL,
PERIOD FOR SYSTEM_TIME (XFROM, XTO),
UNIQUE (SNO),
FOREIGN KEY (SNO) REFERENCES XS_FROM_TO (SNO) )
WITH SYSTEM VERSIONING;
```

Primijetimo sljedeće: Tablica XS_STATUS_FROM_TO ima samo dva stupca, SNO i STATUS, koje korisnici izravno mogu ažurirati. Period koji reprezentira sistemsko vrijeme SYSTEM_TIME određen je stupcima XFROM i XTO. Periodi su oblika [*f, t*). Specifikacija **GENERATED ALWAYS AS ROW START (END)** je potrebna jer sustav dodjeljuje vrijednosti stupcu XFROM, odnosno stupcu XTO. Specifikacija **WITH SYSTEM VERSIONING** je opcionalna, no ako je navedena, tada i PERIOD FOR SYSTEM_TIME mora biti naveden.

Pogledajmo kako se ažurira tablica XS_STATUS_FROM_TO. Naravno, na početku je tablica prazna. Pretpostavimo da želimo izvršiti sljedeću naredbu:

```
INSERT INTO XS_STATUS_FROM_TO (SNO, STATUS)
    VALUES (SNO( 'S2' ), 25) ;
```

Ako pretpostavimo da se INSERT izvodi u vremenu *t02* koje generira sistemski sat, tada ubačeni redak izgleda ovako:

SNO	STATUS	XFROM	XTO
S2	25	t02	t99

t99 označava SQL vrijednost: TIMESTAMP '9999-12-31 23:59:59.999999999999'. Pretpostavimo sada da se u vremenu *t07* koje generira sistemski sat izvodi sljedeća naredba:

```
UPDATE XS_STATUS_FROM_TO SET STATUS = 20
    WHERE SNO = SNO( 'S2' ) ;
```

Tada tablica izgleda ovako:

SNO	STATUS	XFROM	XTO
S2	20	t07	t99
S2	25	t02	t07

Da smo u naredbi UPDATE, u SET dijelu, umjesto STATUS = 20 specificirali STATUS = 25, novi redak bi svejedno bio ubačen. Tada tablica više ne bi bila pakirana na SYSTEM_TIME. Konačno, pretpostavimo da se naredba DELETE izvodi u vremenu *t49* koje generira sistemski sat:

```
DELETE FORM XS_STATUS_FROM_TO WHERE SNO = SNO( 'S2' ) ;
```

Na kraju tablica izgleda ovako:

SNO	STATUS	XFROM	XTO
S2	20	t07	t49
S2	25	t02	t07

Primijetimo da tablica sada sadrži samo povijesne retke. Jako je važno napomenuti da se u tablicama koje uključuju specifikaciju WITH SYSTEM VERSIONING (engl. *system–versioned tables*) mogu ažurirati samo trenutni retci. Povijesni retci su zauvijek u tablici i ne mogu se promijeniti. Istaknimo još da se u takvim tablicama klauzule UNIQUE i FOREIGN KEY odnose samo na trenutne retke.

Upiti se u tablicama sa specifikacijom WITH SYSTEM VERSIONING primjenjuju samo na trenutne retke, a povijesne retke ignoriraju. Neka tablica XS_STATUS_FROM_TO sadrži dva retka:

SNO	STATUS	XFROM	XTO
S2	20	t07	t99
S2	25	t02	t07

Neka je upit oblika:

```
SELECT STATUS FROM XS_STATUS_FROM_TO WHERE SNO = SNO( 'S2' ) ;
```

Tada je rezultat:

STATUS
20

Potrebna nam je klauzula **FOR SYSTEM_TIME** u FROM dijelu upita kako bismo ih mogli primijeniti na sve retke. SQL podržava sljedeće FOR SYSTEM_TIME opcije (*t*, *t1* i *t2* su vremenske oznake):

1. Specifikacija FOR SYSTEM_TIME AS OF *t* selektira retke u kojima sistemsko vrijeme sadrži *t*.
2. FOR SYSTEM_TIME FROM *t1* TO *t2* selektira retke u kojima se period koji reprezentira sistemsko vrijeme preklapa s periodom [*t1, t2*).
3. FOR SYSTEM_TIME BETWEEN *t1* AND *t2* selektira retke u kojima se period koji reprezentira sistemsko vrijeme preklapa s periodom [*t1, t2*].

Na primjer:

```
SELECT STATUS, XFROM, XTO
    FROM XS_STATUS_FROM_TO FOR SYSTEM_TIME AS OF t05
    WHERE SNO = SNO( 'S2' ) ;
```

Rezultat upita je:

STATUS	XFROM	XTO
25	t02	t07

Već smo prije spomenuli da je specifikacija WITH SYSTEM VERSIONING opcionalna. Tablice sa sistemskim vremenom, ali bez navedene specifikacije, sadrže samo trenutne retke. INSERT djeluje jednako kao i na tablice s navedenom specifikacijom. UPDATE ažurira početak sistemskog vremena u odgovarajućim retcima, ali ne ubacuje povijesne retke. DELETE jednostavno uklanja odgovarajuće retke, no ponovno ne ubacuje povijesne retke. Dakle, retci u takvim tablicama samo prikazuju podatak kada su zadnji put ažurirani.

SQL tablica može imati period koji reprezentira aplikacijsko vrijeme i period koji predstavlja sistemsko vrijeme. Takve tablice neslužbeno se nazivaju *bitemporalnim tablicama*.

Sve što smo opisali u ovom poglavlju može se primijeniti i na bitemporalne tablice.

Zaključno, pristup temporalnim podacima koji je predstavljen u Poglavljima 1 i 2 temelji se na apstraktnom pojmu *interval*. Taj je pristup u potpunosti u skladu s fundamentalnim relacijskim principima. Vrijeme kao takvo pojavljuje se jedino prilikom oblikovanja baze podataka i u raspravi o transakcijskom vremenu. SQL-ov pristup temporalnim podacima je specifičan jer se bazira na pojmu *period*. Temporalne značajke ugrađene u SQL dizajnirane su pod pretpostavkom da netemporalnu bazu podataka možemo pretvoriti u temporalnu samo dodavanjem stupaca koji predstavljaju neku vremensku oznaku u postojeće tablice. Pritom se ne vodi računa o tome da rezultantne temporalne tablice moraju biti u šestoj normalnoj formi.

Poglavlje 4

Studijski primjer

Potreba za bitemporalnim tablicama javila se u bankarskim sustavima oko 1992. godine, a određene temporalne značajke ugrađene su u SQL standard već 1994. Među proizvođačima sustava za upravljanje bazom podataka temporalne i bitemporalne tablice nisu bile popularne jer ih je bilo iznimno teško efikasno implementirati. Dvadesetak godina kasnije vodeće relacijske tehnologije za upravljanje temporalnim podacima su *Teradata Database*, *Oracle DB*, *IBM DB2*, *PostgreSQL* i *Microsoft SQL Server*. Prethodno spomenuti sustavi implementiraju određena temporalna proširenja, a posebno se izdvaja IBM-ov DB2 koji svoju podršku za temporalne podatke temelji na inačici standarda za SQL iz 2011. godine.

Pogledajmo kako se realizira DB2 temporalna baza podataka i kako manipulirati temporalnim podacima koje sadrži. Baza podataka je realizirana pomoću *IBM DB2 Express-C* tehnologije (verzija 10.5.5). Riječ je o nekomercijalnom izdanju DB2 servera koji ima gotovo jednaku funkcionalnost kao i komercijalno izdanje. Sastavni dio ove tehnologije su proširenja za upravljanje temporalnim podacima koja se nazivaju *Time Travel Query*. Ugrađena podrška za temporalne podatke omogućuje automatsko generiranje povijesti svih transakcija koje su mijenjale bazu podataka i ažuriranje stvarnih aplikacijskih datuma. Takav pristup reducira složenost upita i omogućuje jednostavniju analizu događaja koji ovise o vremenu. DB2 implementira *periode* te koncepte *sistemskog* i *poslovnog vremena*. Poslovno vrijeme je IBM-ov naziv za stvarno ili aplikacijsko vrijeme.

Sve instrukcije u ovom zadatku izvedene su u *DB2 CLP* komandnom prozoru. Postoje alati za upravljanje DB2 bazom podataka koji imaju bogatije korisničko sučelje, npr. *IBM Data Studio*. IBM Data Studio može se besplatno preuzeti. Rad u DB2 okruženju započinjemo izvođenjem sljedećih naredbi:

1. CREATE DATABASE PRIMJER ;

2. CONNECT TO PRIMJER ;
3. CREATE SCHEMA BAZA SET AUTHORIZATION ANDREA ;
4. SELECT SCHEMANAME FROM SYSCAT.SCHEMATA ;
5. SET SCHEMA BAZA ;

Prva naredba izvodi se oko pet minuta, a četvrta naredba je opcionalna, tj. njome možemo provjeriti je li shema iz prethodnog koraka kreirana. Shemu intuitivno shvaćamo kao analogon *namespace*-u u jezicima C++ ili C#. Istaknimo da znak ; ne mora doći iza naredbe. Pretpostavimo prvo da se baza podataka PRIMJER sastoji od tablica BS, BSSTATUS i BSP. Tablica BS predstavlja dobavljače s ugovorom, BSSTATUS predstavlja statuse dobavljača, a BSP potencijalne pošiljke. Neka tablice BS, BSSTATUS i BSP odgovaraju tablicama S_FROM_TO, S_STATUS_FROM_TO i SP_FROM_TO sa Slike 3.2. Odgovarajuće relacije su sve u 6NF. Rad u DB2 okruženju nastavljamo izvođenjem sljedećih naredbi:

```
CREATE TABLE BS (
SNO VARCHAR(5) NOT NULL,
BSTART DATE NOT NULL,
BEND DATE NOT NULL,
PERIOD BUSINESS_TIME (BSTART, BEND),
UNIQUE (SNO, BUSINESS_TIME WITHOUT OVERLAPS) ) ;

CREATE TABLE BSSTATUS (
SNO VARCHAR(5) NOT NULL,
STATUS INT NOT NULL,
BSTART DATE NOT NULL,
BEND DATE NOT NULL,
SSTART TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN NOT NULL,
SEND TIMESTAMP(12) GENERATED ALWAYS AS ROW END NOT NULL,
TSTART TIMESTAMP(12) GENERATED ALWAYS AS
        TRANSACTION START ID IMPLICITLY HIDDEN,
PERIOD BUSINESS_TIME (BSTART, BEND),
PERIOD SYSTEM_TIME (SSTART, SEND),
UNIQUE (SNO, BUSINESS_TIME WITHOUT OVERLAPS) ) ;
```

Uočimo da se sintaksa ne razlikuje značajnije od one koja je predstavljena u Poglavlju 3. Standard za SQL zapravo ne spominje postojanje sistemskog sata, nego posebne vremenske oznake koja označava početak sistemskog vremena. Ovdje je to TSTART. Prisjetimo se SQL definicija za promatrane tablice sa Slike 3.2. Upotrebom klauzule FOREIGN KEY ... REFERENCES čuva se referencijalni integritet baze podataka. Dakle, tablicama BS i BSSTATUS potrebno je dodati odgovarajuća referencijalna ograničenja. To smo pokušali slijedeći standardnu DB2 sintaksu:

```
ALTER TABLE STATUS ADD
    FOREIGN KEY (SNO, BUSINESS_TIME) REFERENCES S ;
```

Međutim, specificiranje perioda BUSINESS_TIME je uzrokovalo grešku. U [3] je opisano da dodavanje ograničenja za čuvanje temporalnog referencijalnog integriteta nije trivijalno, no može se osigurati upotrebom odgovarajućih procedura. Procedure koje su izložene u [3] mogu se iskoristiti kao polazna točka u kodiranju vlastitih. Ovdje se nećemo baviti procedurama, stoga nije smisleno implementirati bazu podataka o dobavljačima i pošiljkama. Ograničimo se samo na tablicu BSSTATUS. Neka tablica BSSTATUS opisuje stanje na nekom računu; SNO je oznaka računa, a STATUS pripadni iznos. Budući da tablica BSSTATUS sadrži sistemski period, sljedeće što moramo napraviti je definirati povijesnu tablicu BSSTATUS_HISTORY, koja ima istu strukturu kao BSSTATUS, i omogućiti da se povijesni zapisi iz BSSTATUS automatski ubacuju u BSSTATUS_HISTORY. To postižemo sljedećim naredbama:

```
CREATE TABLE BSSTATUS_HISTORY LIKE BSSTATUS ;

ALTER TABLE BSSTATUS ADD VERSIONING
    USE HISTORY TABLE BSSTATUS_HISTORY ;
```

Primijetimo da DB2 konstrukcija CREATE TABLE ... LIKE kreira tablicu s povijesnim podacima poput konstrukcije LOGGED_TIMES_IN koju smo predstavili u Potpoglavlju 2.5. U ovom trenutku naša temporalna baza podataka PRIMJER sadrži dvije prazne tablice: BSSTATUS i BSSTATUS_HISTORY. Ubacimo neke podatke u BSSTATUS:

```
INSERT INTO BSSTATUS (SNO, STATUS, BSTART, BEND)
    VALUES ('A111', 1000, '2015-07-01', '9999-12-31'),
    VALUES ('A1212', 1500, '2015-09-01', '9999-12-31') ;
```

Tablica BSSTATUS_HISTORY je i dalje prazna, a tablica BSSTATUS sada sadrži dva retka (radi jednostavnosti ne navodimo puno sistemsko vrijeme):

BSSTATUS

SNO	STATUS	BSTART	BEND	SSTART	SEND
A1111	1000	2015-07-01	9999-12-31	2015-08-31	9999-12-31
A1212	1500	2015-09-01	9999-12-31	2015-08-31	9999-12-31

Poslovni period u BSSTATUS odnosi se na razdoblje tijekom kojeg korisnik SNO ima iznos STATUS na računu, a sistemski period odnosi se na pripadne promjene na računu. Sistemski period održava DB2. Pogledajmo što će se dogoditi ako pokušamo ažurirati prvi redak:

```
UPDATE BSSTATUS
    FOR PORTION OF BUSINESS_TIME FROM '2015-07-01' TO '2015-08-01'
    SET STATUS = 3000 WHERE SNO = 'A1111';
```

Nakon što je ažurirana, tablica BSSTATUS sadrži sljedeće retke:

BSSTATUS

SNO	STATUS	BSTART	BEND	SSTART	SEND
A1111	3000	2015-07-01	2015-08-01	2015-09-01	9999-12-31
A1212	1500	2015-09-01	9999-12-31	2015-08-31	9999-12-31
A1111	1000	2015-08-01	9999-12-31	2015-09-01	9999-12-31

U tablicu BSSTATUS_HISTORY automatski je ubačen povijesni redak iz BSSTATUS:

BSSTATUS_HISTORY

SNO	STATUS	BSTART	BEND	SSTART	SEND
A1111	3000	2015-07-01	9999-12-31	2015-08-31	2015-09-01

Dakle, što se tiče tablice BSSTATUS, DB2 automatski ažurira redak na koji se odnosi navedeni period te dodaje redak koji reprezentira stare vrijednosti za period koji nije uključen u zadnjoj naredbi. Također, automatski su ažurirani pripadni sistemski periodi. S druge strane, u tablicu BSSTATUS_HISTORY DB2 automatski dodaje redak na koji se odnosio UPDATE i kraj pripadnog sistemskog perioda u tom retku ažurira tako da mu dodjeljuje vrijednost početka transakcije koja je mijenjala BSSTATUS. Promotrimo kako naredba DELETE djeluje na naše temporalne tablice. Rezultati su prikazani odmah u nastavku.

```
DELETE FROM BSSTATUS FOR PORTION OF BUSINESS_TIME
    FROM '2015-08-01' TO '2015-09-01' WHERE SNO = 'A1111';
```

BSSTATUS

SNO	STATUS	BSTART	BEND	SSTART	SEND
A1111	3000	2015-07-01	2015-08-01	2015-09-01	9999-12-31
A1212	1500	2015-09-01	9999-12-31	2015-08-31	9999-12-31
A1111	1000	2015-09-01	9999-12-31	2015-09-02	9999-12-31

BSSTATUS_HISTORY

SNO	STATUS	BSTART	BEND	SSTART	SEND
A1111	1000	2015-07-01	9999-12-31	2015-08-31	2015-09-01
A1111	1000	2015-08-01	9999-12-31	2015-09-01	2015-09-02

Pretpostavimo da je netko slučajno izvršio sljedeću naredbu:

```
DELETE FROM BSSTATUS WHERE SNO = 'A1212';
```

U tom slučaju tablice BSSTATUS i BSSTATUS_HISTORY izgledaju ovako:

BSSTATUS

SNO	STATUS	BSTART	BEND	SSTART	SEND
A1111	3000	2015-07-01	2015-08-01	2015-09-01	9999-12-31
A1111	1000	2015-09-01	9999-12-31	2015-09-02	9999-12-31

BSSTATUS_HISTORY

SNO	STATUS	BSTART	BEND	SSTART	SEND
A1111	1000	2015-07-01	9999-12-31	2015-08-31	2015-09-01
A1111	1000	2015-08-01	9999-12-31	2015-09-01	2015-09-02
A1212	1500	2015-09-01	9999-12-31	2015-08-31	2015-09-03

Korisnik *A1212* zatražio je ispis svih promjena na svojem računu u zadnja tri mjeseca. Rezultat sljedeće naredbe sugerira da je došlo do greške:

```
SELECT * FROM BSSTATUS FOR SYSTEM_TIME
    FROM '2015-06-04' TO '2015-09-05'  WHERE SNO = 'A1212';
```

Ispis svih promjena prikazan je u nastavku:

SNO	STATUS	BSTART	BEND	SSTART	SEND
A1212	1500	2015-09-01	9999-12-31	2015-08-31	2015-09-03

Specifikacija FOR SYSTEM_TIME potaknula je automatsko pristupanje sustava DB2 tablici BSSTATUS_HISTORY i dohvaćanje tražene informacije. Pretpostavimo da nas zanimaju iznosi na računu korisnika na dan 3. rujna 2015.

```
SELECT * FROM STATUS FOR BUSINESS_TIME AS OF '2015-09-03' ;
```

Rezultat prethodnog upita odgovara drugom retku zadnje BSSTATUS tablice. Opcije za sistemski period oblika FOR SYSTEM_TIME mogu se upotrijebiti i za poslovni period BUSINESS_TIME, no potonji slučaj ne uključuje povijesne retke. Ilustrirajmo i rad nekog operatora koji se primjenjuje na periode:

```
SELECT * FROM STATUS WHERE BUSINESS_TIME CONTAINS '2015-07-20' ;
```

Prethodna naredba izazvala je grešku. Upiti s ostalim operatorima za rad s periodima iz Tablice 3.1 također se ne mogu realizirati. Pretpostavljamo da operatori za periode nisu podržani u obliku koji je predstavljen u Poglavlju 3. No uočimo da koristeći specifikaciju FOR BUSINESS_TIME prethodni upit možemo preoblikovati tako da dobijemo željeni rezultat. Promotrimo još pitanje pakiranja tablice. Pretpostavimo da smo u BSSTATUS ubacili još jedan redak:

```
INSERT INTO BSSTATUS (SNO, STATUS, BSTART, BEND)
    VALUES ('A2222', 2000, '2015-05-01', '2015-07-01'),
    VALUES ('A2222', 2000, '2015-07-01', '2015-08-01') ;
```

Očekivano dobivamo tablicu koja nije pakirana na BUSINESS_TIME (nema promjena u tablici BSSTATUS_HISTORY):

BSSTATUS

SNO	STATUS	BSTART	BEND	SSTART	SEND
A1111	3000	2015-07-01	2015-08-01	2015-09-01	9999-12-31
A2222	2000	2015-05-01	2015-07-01	2015-09-04	9999-12-31
A1111	1000	2015-09-01	9999-12-31	2015-09-02	9999-12-31
A2222	2000	2015-07-01	2015-08-01	2015-09-04	9999-12-31

U DB2 je ugrađena posebna SET CURRENT TEMPORAL konstrukcija koja omogućuje da rad s bazom podataka obavljamo u nekom trenutku u vremenu. Konstrukcije se mogu primijeniti na BUSINESS_TIME i SYSTEM_TIME, no ne bi se trebale koristiti istovremeno. Pogledajmo primjer u nastavku:

```
SET CURRENT TEMPORAL BUSINESS_TIME = '2015-09-01' ;
```

Formulacija za SYSTEM_TIME je analogna. Pogled na tablicu BSSTATUS_HISTORY je nepromijenjen jer svi poslovni periodi sadrže točku '2015-09-01'. Tablica BSSTATUS prikazana je u nastavku:

BSSTATUS

SNO	STATUS	BSTART	BEND	SSTART	SEND
A1111	1000	2015-09-01	9999-12-31	2015-09-02	9999-12-31

Zaključno, implementacija temporalne baze podataka koja zahtijeva određena ograničenja za čuvanje referencijalnog integriteta ne može se jednostavno implementirati u DB2 sustavu. Također, operatori za rad s periodima nisu dostupni u obliku koji je predstavljen u standardu za SQL iz 2011. godine. Pitanje pakiranja tablice na neki period ostaje otvorenim. S druge strane, bitemporalne tablice mogu se jednostavno implementirati. Pored tablice koja sadrži trenutne retke, DB2 automatski generira tablicu u kojoj će se čuvati povijesni retci. Operacije UPDATE i DELETE su maksimalno pojednostavljene. Sve u svemu, DB2 podrška za manipuliranje temporalnim podacima dobiva prolaznu ocjenu.

Zaključak

U većini baza podataka za svaki podatak pohranjuje se samo njegova najnovija vrijednost, no temporalne baze podataka uključuju posebnu podršku za vremensku dimenziju podataka. Dopušta se pohranjivanje i ažuriranje prethodnih i/ili budućih vrijednosti podataka te postavljanje temporalnih upita gdje uvjeti pretraživanja ovise o vremenu. U radu se predstavljaju napredni koncepti za upravljanje temporalnim podacima u relacijskom modelu za bazu podataka. *Baza podataka o dobavljačima i pošiljkama* temelj je za sve primjere u radu, a formalne definicije primjera zapisane su u jeziku *Tutorial D*.

Temporalne podatke možemo opisati kao kodirane reprezentacije tvrdnji koje imaju neku vremensku oznaku: *Dobavljač S1 je sklopio ugovor koji vrijedi* ***od*** *1. svibnja 2015.* ***do*** *30. travnja 2016. godine.* Prethodna propozicija predstavlja moguću interpretaciju nekog retka *r* u nekoj relaciji. *Stvarno vrijeme* odnosi se na naša saznanja o temporalnim podacima, a koncept *transakcijskog vremena* reprezentira povijest tih saznanja. Na primjer, stvarno vrijeme za prethodnu propoziciju je razdoblje od 1. svibnja 2015. do 30. travnja 2016. godine. Ako pretpostavimo da je originalni redak *r* u bazu podataka dodan u vremenu *t1*, a zamijenjen nekim drugim retkom u vremenu *t2*, tada je transakcijsko vrijeme za danu propoziciju interval od *t1* do *t2*.

SNO	FROM	TO
S1	1. svibnja 2015.	30. travnja 2016.

Slika 4.1: Prikaz retka u temporalnoj relaciji

Kako bismo pravilno postupali s temporalnim podacima, bitno je uočiti potrebu za posebnim tipom podataka u temporalnim bazama podataka – *vremenskim intervalima* – umjesto parova FROM i TO vrijednosti. Interval *i* intuitivno shvaćamo kao uređeni niz točaka nekog tipa *T*. Pritom *T* najčešće reprezentira kalendarski datum. U radu predstavljamo niz operatora za usporedbu intervala u temporalnim upitima: $=$, $\supseteq$, $\subseteq$, $\supset$, $\subset$, BEFORE, AFTER, OVERLAPS, MEETS, MERGES, BEGINS i ENDS. Skupovni operatori UNION, INTERSECT i MINUS mogu se izravno primijeniti na intervale ako ih shvatimo kao skup

točaka. Također, definiramo operator COUNT(*i*) čija je povratna vrijednost broj točaka u nekom intervalu *i*.

Temporalni upiti koji su iznimno složeni u terminima originalne relacijske algebre postaju intuitivniji i jednostavniji u terminima operatora PACK i UNPACK. Pakiranje relacije na atribut čije su vrijednosti intervali možemo opisati kao odgovarajuću zamjenu parova intervala njihovom unijom, pod pretpostavkom da je njihova unija definirana, sve dok daljnje zamjene više nisu moguće. Rezultat raspakiravanja prikazuje iste informacije kao i originalna relacija, no on je restrukturiran tako da su vrijednosti određenog atributa intervali kojima pripada samo jedna točka. Operator PACK djeluje tako da se promatrana relacija prvo raspakira na sve određene atribute u bilo kojem redoslijedu, a tek onda pakira na te iste atribute u točno određenom poretku. Operatori PACK i UNPACK mogu se iskoristiti kao temelj za definiranje U_ operatora koji omogućuju postavljanje temporalnih upita na jednostavniji način. Definicije U_ operatora uglavnom slijede obrazac: Raspakiraj oba operanda, primijeni odgovarajuću relacijsku operaciju, pakiraj rezultat. Dakle, operator UNPACK je ključna konceptualna komponenta našeg pristupa upravljanju temporalnim podacima u relacijskom modelu za bazu podataka.

Postoje tri različita pristupa temporalnom oblikovanju: samo *since relacije*, samo *during relacije*, ili kombinacija prethodnih. Intuitivno, since relacija je relacija koja sadrži podatke koji trenutačno vrijede, a during relacija je relacija koja sadrži povijesne podatke. Problem u prvom pristupu je nemogućnost čuvanja *čisto* povijesnih zapisa, a u drugom pristupu problem je koncept *sada* za koji ne postoji adekvatno rješenje. Općenito se predlaže pristup koji kombinira since i during relacije. Tada treba postupati u skladu sa sljedećim: Preporuča se horizontalna dekompozicija početne relacije kako bi se odvojile trenutne i povijesne informacije. Predlaže se da se sve since relacije normaliziraju u skladu s klasičnom teorijom normalizacije do 5NF. Na kraju, preporuča se vertikalna dekompozicija svih during relacija u ireducibilne 6NF komponente.

U temporalnim bazama podataka mogu se pojaviti četiri problema ako izostavimo odgovarajuća ograničenja za čuvanje integriteta. Radi se o *problemima redundancije*, *okolišanja*, *kontradikcije* i *gustoće*. Uvodi se sintaksa U_KEY koja kombinira funkcionalnost rješenja prvih triju problema. Ostala ograničenja za čuvanje integriteta zahtijevaju posebnu pažnju te je u radu predstavljeno devet zahtjeva na bazu podataka o dobavljačima i pošiljkama čiji se analogoni mogu primijeniti na bilo koju temporalnu bazu podataka. Temporalno oblikovanje koje uključuje since i during relacije rezultira najsloženijim ograničenjima za čuvanje integriteta temporalne baze podataka.

Postavljanje upita u temporalnim bazama podataka je dosta složeno. Posebno su zanimljivi

upiti kojima se ažurira temporalna baza podataka. Prisjetimo se razmatranja o stvarnom i transakcijskom vremenu. Ako je baza podataka jednom sadržavala propoziciju *p* koja je istinita, tada se ona smatra povijesnim zapisom, a povijesni zapisi bi se u temporalnoj bazi podataka trebali čuvati zauvijek s odgovarajućim transakcijskim vremenom koje indicira kada je *p* bila istinita. Međutim, *p* se ne smatra povijesnim zapisom ako naknadno utvrdimo da je netočna, tek tada se *p* može ažurirati. Ažuriranje polutemporalne baze podataka ne razlikuje se značajnije od ažuriranja bilo koje netemporalne baze podataka. Slučaj temporalne baze podataka sa samo during relacijama zahtijeva uvođenje operatora: U_INSERT, U_DELETE, U_UPDATE i PORTION DELETE.

Transakcijska vremena u temporalnim bazama podataka ne mogu se ažurirati poput stvarnih vremena, njih održava sustav, niti smo u mogućnosti postavljati upite koji su vezani uz njih. Smatra se da transakcijska vremena moraju biti dostupna u standardnom relacijskom obliku, odnosno potrebne su nam pomoćne relacije koje prikazuju povijest svih transakcija za neku bazu podataka.

Određene temporalne značajke uključene su u zadnju verziju standarda za SQL 2011. godine. SQL ne podržava vremenske intervale, nego koristi *periode* koji se sastoje od parova vrijednosti stupaca u SQL tablici. Određeni operatori za rad s intervalima koje smo predstavili u Poglavlju 1 imaju svoje SQL analogone koji se primjenjuju na periode. Operatori koji nisu izravno podržani mogu se izraziti pomoću postojećih. Standard za SQL ne podržava operatore PACK i UNPACK, U_ operatore, U_KEY i odgovarajuće U_ operatore za ažuriranje temporalnih baza podataka. Oblikovanje temporalne baze podataka temelji se na drugom pristupu sa samo during relacijama. SQL pruža podršku za transakcijko vrijeme te za njega koristi izraz *sistemsko vrijeme*. SQL dopušta da sama tablica sadrži najviše jedan period koji reprezentira sistemsko vrijeme. SQL tablica koja ima period koji reprezentira aplikacijsko vrijeme i period koji predstavlja sistemsko vrijeme, neslužbeno se naziva *bitemporalnom tablicom*.

Pristup temporalnim podacima koji je predstavljen u Poglavljima 1 i 2 temelji se na apstraktnom pojmu *interval*. Taj je pristup u potpunosti u skladu s fundamentalnim relacijskim principima. SQL-ov pristup temporalnim podacima bazira se na pojmu *period*. Temporalne značajke koje su ugrađene u SQL dizajnirane su pod pretpostavkom da netemporalnu bazu podataka možemo pretvoriti u temporalnu samo dodavanjem stupaca koji predstavljaju neku vremensku oznaku. Temporalne značajke u standardu za SQL mogu se smatrati velikim korakom u upravljanju temporalnim podacima, no nisu dovoljne da riješe sve probleme koji dolaze s njima. U ovom trenutku vodeće relacijske tehnologije za upravljanje temporalnim podacima su *Teradata Database*, *Oracle DB*, *IBM DB2*, *PostgreSQL* i *Microsoft SQL Server*.

Bibliografija

[1] C. J. Date, Hugh Darwen i Nikos A. Lorentzos, *Time and Relational Theory*, Morgan Kaufmann, 2014.

[2] Robert Manger, *Baze podataka*, Element, 2012.

[3] Matthias Nicola i Martin Sommerlandt, *Managing time in DB2 with temporal consistency*, (2012), http://www.ibm.com/developerworks/data/library/techarticle/dm-1207db2temporalintegrity/.

[4] Cynthia Saracco, Matthias Nicola i Lenisha Gandhi, *A Matter of Time: Temporal Data Management in DB2 10*, (2012), http://www.ibm.com/developerworks/data/library/techarticle/dm-1210temporaltablesdb2/.

Sažetak

U većini baza podataka za svaki podatak pohranjuje se samo njegova najnovija vrijednost, no odnedavno je moguće sačuvati i stare vrijednosti. Temporalne baze podataka su takve baze podataka koje uključuju posebnu podršku za vremensku dimenziju podataka. To je sustav koji omogućuje posebne radnje poput pohranjivanja, pretraživanja i ažuriranja temporalnih podataka. Svrha ovoga rada je objasniti razne tehnike koje su vezane uz oblikovanje i korištenje temporalnih baza podataka. Opisali smo neke od problema koji se pojavljuju u temporalnim bazama podataka i pokazali kako se mogu riješiti u relacijskom sustavu za upravljanje bazom podataka. Naš pristup temporalnim podacima temeljio se na korištenju intervala kao takvih. Operator UNPACK bio je ključna konceptualna komponenta našeg pristupa upravljanju temporalnim podacima. Predstavili smo niz naprednih temporalnih koncepata koji bi se mogli primijeniti na oblikovanje i ažuriranje temporalnih baza podataka, te u formulaciji ograničenja za čuvanje integriteta i postavljanju temporalnih upita. Određene temporalne značajke uključene su u zadnju verziju standarda za SQL 2011. godine. U radu smo analizirali SQL podršku za pristup temporalnim podacima.

Summary

Until recently, databases tended to focus on current information only, but now it is possible to keep historical information as well. A temporal database system is one that includes special support for the time dimension of data. It provides special features for storing, querying and updating temporal data. The purpose of this paper is to explain various techniques involved in designing and using temporal databases. We described some of the issues that arise in temporal databases and showed how they might be addressed in a relational database management system. Our approach to temporal data was based on dealing with intervals as such. The UNPACK operator was a crucial conceptual component of our approach to the management of temporal data in particular. We covered a range of advanced temporal concepts that could be applied to temporal database design, temporal database updates, and the formulation of integrity constraints and temporal queries. Temporal database support was added to the SQL standard in the 2011 edition of that standard. Therefore, we introduced some of the temporal features of the SQL standard.

Životopis

Zovem se Andrea Perčinlić. Rođena sam 25. veljače 1992. godine. Formalno obrazovanje započela sam 1998. godine u 1. osnovnoj školi Vrbovec u Vrbovcu. 2006. godine upisala sam program opće gimnazije u Srednjoj školi Vrbovec u Vrbovcu. U srednjoj školi posebno me privukla matematika. 2010. godine upisala sam preddiplomski studij *Matematika* na Matematičkom odsjeku Prirodoslovno–matematičkog fakulteta u Zagrebu. Završetkom preddiplomkog studija stekla sam akademski naziv sveučilišne prvostupnice matematike. 2013. godine veliki interes prema programiranju potaknuo me da upišem diplomski studij *Računarstvo i matematika* na PMF–MO u Zagrebu. Tijekom školovanja usvojila sam vrijedna znanja iz matematike i stekla praktično iskustvo u primjeni aktualne tehnologije u svrhu razvoja softvera. U slobodno vrijeme volim slušati glazbu i igrati tenis.