Loeng 5. Disaini täpsustamine (järg) ning andmebaasi häälestamine (algus)

Kitsenduste (piirangute) täpsustamine ja lisamine

Kitsendused on vajalikud eelkõige selleks, et garanteerida andmete täielikkus andmebaasi tasemel, s.t. vältida vigaste andmete sattumist andmebaasi. Kitsendused garanteerivad andmete täielikkuse ka siis, kui andmeid ei lisata ekraanivormist, vaid muul moel otse andmebaasi (garanteeririb andmete täielikkuse programmikoodi väliselt).

Kitsenduste liigid on järgmised:

- Tühjade veergude lubamine / mittelubamine (kohustuslikud / mittekohustuslikud veerud),
- Primaarsed võtmed,
- Unikaalsed võtmed,
- Välisvõtmed,
- Projekteerija poolt kirjutatud kontrollkitsendused.

Võtmetest oli juttu teemas "Esmane andmete disain". Lisada võiks seda, et vajalik on võtmete väärtustamine nii serveri kui ka kliendi tasemel, et:

- Garanteerida andmete täielikkus ka otse andmebaasi sisestamisel (selle garanteerib väärtustamine serveris),
- Vähendada võrgu koormust (selle garanteerib väärtustamine kliendi tasemel). Viimane saavutatakse sellega, et andmed väärtustatakse juba rakenduses ning vigaseid andmeid ei hakata üldse üle võrgu serverisse saatma. Kui puuduks väärtustamine rakenduses, siis võidakse saata serverisse ka vigaseid andmeid ning alles sealt saadakse vastus andmete mittetäielikkuse kohta. Vastuse saatmine läbi võrgu põhjustab omakorda veelgi võrgu koormuse kasvu.

Välisvõtmetega seotud lisakitsendustest

Siin on mõeldud välisvõtmetega seotud tabelitega sooritatavaid tegevusi ridade muutmisel, kustutamisel ühes seotud tabelitest. See tähendab, kas:

1) kustutatakse (muudetakse välisvõtme veerge) vastavad read sõltuvast tabelist, kui kustutatakse (muudetakse primaarvõtme veerge) peremees tabeli rida (DELETE / UPDATE CASCADE) või
2) keelatakse peremees - tabeli rea kustutamine (primaarvõtme veergude muutmine) seni, kuni eksisteerib ridu sõltuvas tabelis (DELETE / UPDATE RESTRICTED).
3) Lisaks eespool toodud variantidele võidakse veel muuta sõltuva tabeli vastavates ridades välisvõtme veerud tühjaks (või anda neile mingi muu ette määratud kindel väärtus), kui peremees - tabeli vastav rida kustutatakse (või muudetakse väärtust primaarvõtme veerus).

Nimetatud variantide vahel valiku tegemisel võiks lähtuda järgmistest printsiipidest:

- Primaarvõtme veergude muutmise puhul lähtuda variandist 2), kuna:
  - Valdav enamus primaarvõtme veerge on genereeritud puhtalt ridade unikaalsuse tagamiseks ning ei oma kasutaja jaoks mingit sisulist tähendust (vt. teema "Numbrijada generaatorite kasutamine") ja seega puudub ka vajadus väärtuste muutmiseks neis veergudes
  - Vastavat tüüpi kitsendus on rakendatav ka andmebaasi tasandil (väärtustamine nii serveris kui rakenduses) ning kitsendustega seotud koodi genereerib server automaatselt (sel moel genereeritud kood on serveri poolt ka juba optimeeritud ning annab lisaks võidule süsteemi lihtsuses võidu ka töökiirust silmas pidades.

    Vastasel juhul, lähtudes variandist 1), genereeritakse kitsendus automaatselt vaid rakenduse poolel. Serveri poolelt tuleks primaarvõtmete veergudes tehtavate muutuste edasi kandmiseks kirjutada andmebaasi trigerid, kuid sellega kaotatakse nii süsteemi lihtsuses kui ka töökiiruses.

- Ridade kustutamisel kasutada nii punktis 1) kui ka punktis 2) toodud printsiipe, sest siin on mõlemad kitsenduse variandid rakendatavad ka serveri poolelt. Igal konkreetsel juhul lähtutakse variandi valikul:

  - Analüüsi etapi tulemusena teadmusbaasi formuleeritud soovitustest,
  - Andmete kokkukuuluvuse loogikast (vt. järgnevaid näiteid):

a) Objekti kuulumine mingile lepingule (tabel LEPINGU_OBJEKTID) kustutatakse nii vastava objekti kui ka vastava lepingu kustutamisel viimasega koos.
b) Objekti juurde kuuluvad "asjad" ja "tegevused" kustutatakse koos vastava objektiga
c) Tänavate nimistust (tabel TÄNAVAD) tänava kustutamine keelatakse aga seniks, kuni eksisteerib kliente, objekte jms., mis selle tänavaga seotud on.

Kontrollkitsendused

Kontrollkitsendusele on iseloomulik järgnev:

1) Tabeli rida lükatakse tagasi, kui ta annab väärtustamisel tulemuseks vale (FALSE). Kui tulemuseks on MÄÄRAMATA (NULL), siis loetakse rida väärtustatuks (kontrollitav veerg on tühi). Samuti ka siis, kui väärtustamise tulemus on ÕIGE (TRUE).
2) Kasutab väärtustamisel ainult vaadeldava tabeli veerge.
3) On võimas vahend lihtsamat tüüpi piirangute (1 tabeli veerge kasutades) realiseerimiseks andmebaasi poolelt.
4) Keerukamate andmeväärtustusmehhanismide realiseerimiseks sobivad enamusel juhtudest paremini andmebaasi trigerid. Kuid teatud juhtudel osutuvad kontrollkitsendused ka andmebaasi trigeritest võimsamaks andmete kontrolli mehhanismiks. See on juhul, kui tabeli rea väärtustamine nõuab andmete lugemist sama tabeli teistest ridadest (andmebaasi triger, mis käivitub kõigi muudetud (lisatud, kustutatud) tabeli ridade kohta, ei luba päringut tabeli peal, mille muutmine trigeri enda käivitas). Sel juhul saab andmete väärtustamist realiseerida kontrollkitsendusega, mis kasutab andmete kontrollimiseks mingit välisfunktsiooni (on andmebaasis iseseisvalt realiseeritud funktsioon) loogilist tüüpi (ÕIGE / VALE) väljundväärtust.
   Seda tüüpi kontrollkitsendus on oma olemuselt küllalt erandlik võrreldes tavaliste kontrollkitsendustega, kuna ta defineeritakse teadmusbaasis kui kontrollkitsendus, kuid tema põhjal ei genereerita serveris kontrollivat koodi. Teadmusbaasi definitsiooni kasutatakse ainult kontrolli genereerimiseks rakenduse poolel.
5) Erandjuhtudel lubab kasutada veerge ka muudest tabelitest kui väärtustamisele kuuluv tabel. Seda just eelmises alapunktis vaadeldud välisfunktsioonide kasutamise läbi (sisendparameetriks võivad olla siiski vaid väärtustatava tabeli veerud, funktsiooni sees on võimalik aga kasutada ka veerge teistest tabelitest).

Kõige sagedamini kasutatakse kontrollkitsendusi:

- Lubatud väärtuskomplektide määramiseks veergudele,
- Infomudeli kaarte lahendamiseks (vt. "Esmane andmete disain" infomudelis sisalduvate kaarte kohta)
- Lubatud väärtuste kontrollimiseks läbi välisfunktsiooni.

```
ALTER TABLE LEPINGUD ADD (
        CONSTRAINT KONTR_PERIOODI_PÄEVA
        CHECK (
        (arve_periood = 0.5 AND arve_päev >= 1 AND
         arve_päev <= 12) OR arve_periood =1 OR arve_periood = 2 OR
       arve_periood = 3 OR arve_periood = 4 OR arve_periood = 6 OR
       arve_periood = 12
        )
)
```

See oli näide lubatud andmekomplektide kontrollimisest kontrollkitsenduses (arve tegemise päeva ja arve saatmise perioodsuse jaoks lubatud kombinatsioonid).

Andmebaasi trigerid

Andmebaasi trigereid iseloomustavad järgnevad faktid:

1. Avarad kasutamisvõimalused andmete töötlemiseks serveri tasandil. Sellest tulenevalt on andmebaasi trigerite tüüpilised kasutusalad:
    - Keerukamate andmekontrollimehhanismide realiseerimine (selliste, mida ei õnnestu realiseerida kitsenduste abil),
    - Analüüsi poolt ette antud nõudmiste realiseerimine andmebaasis,
    - Tuletatavate väärtuste leidmine (vt. arve numbri genereerimine teema "Numbrijada generaatorite kasutamine" juures).
    - Veergude algväärtustamine

2. Teatud kitsenduste olemasolu andmebaasi trigerite kasutamisel, mistõttu teatud juhtudel sobivad andmete väärtustamiseks paremini kontrollkitsendused:
    - Iga rea kohta käivituvad trigerid (seda tüüpi piirangud ei kehti 1 kord tegevuse (lisamine, muutmine, kustutamine) kohta käivituvate trigerite puhul) ei saa lugeda ega muuta andmeid tabelis, mida

muudetakse (muudetakse, kustutatakse, lisatakse) samal ajal käsu poolt, milline käivitas ka trigeri enese (s.o. vaadeldavas tabelis eneses).
    - Iga rea kohta käivituvad trigerid ei saa lugeda ega muuta andmeid tabelites, millised alluvad muutustele trigeri käivitanud käsu poolt tänu välisvõtmetega seotud lisakitsendustele (DELETE CASCADE).
    - Iga rea kohta käivituvad trigerid ei saa muuta väärtust primaarse-, unikaalse- ega välisvõtme veerus tabelites, mida trigerit käivitanud käsk loeb kas otse või võtmetega seotud kitsenduste kontrollimiseks.
3. Andmebaasi trigereid hoitakse andmebaasis lähtekoodina. Esimesel käivitamisel vastava trigeri lähtekood kompileeritakse ning loetakse mällu. Kui triger mingil põhjusel on mälust maha loetud, siis tuleb järgmisel käivitamisel trigeri lähtekood jälle kompileerida. Seni kuni trigerit mälus hoitakse, pole tema lähtekoodi kompileerida vaja. Sellest lähtuvalt on mõttekas viia maksimaalne osa trigeri koodist üle trigeri poolt välja kutsutavatesse protseduuridesse ja funktsioonidesse (jättes trigerisse ideaaljuhul vaid vastava alamprogrammi väljakutse), milliseid hoitakse andmebaasis kompileeritud kujul. Kuna mitu erinevat trigerit võib kasutada sarnast loogikat (teha sarnaseid tegevusi), siis annab taoline samm lisaks võidule töökiiruses ka võidu süsteemi lihtsust silmas pidades (koodi muutmine toimub vaid ühes kohas).

Abitabelite lisamine

Disaini käigus on tihtipeale vaja projekteerida ja luua uusi andmetabeleid, millele analüüsi tulemustes viited puuduvad. Tüüpiliselt on need tabelid, millised:

- Ei kajastu olemi-suhte mudelil,
- Ei oma reeglina seoseid olemi-suhte mudelil kajastuvate tabelitega,
- Sisaldavad ajutisi (vahepealseid) andmeid,
- On vajalikud keerukamate protsesside loogika realiseerimise lihtsustamiseks (sellega kaasneb ka rakenduste töökiiruse suurenemine)

Tüüpilises rakenduses kuuluvad lisatavad abitabelid järgmistesse kategooriatesse:

1. Parameetrite tabel, milles sisaldub informatsioon, millele analüüsi etapi funktsioonide kirjeldustes on viidatud kui parameetritele. Need on andmed:
    - , mille kohta informatsioon muutub väga harva (kord aastas või veelgi harvem)
    - , mis omavad ühte ja sama väärtust igal ajahetkel üle kogu süsteemi
    - , millised on kirjeldatavad 1 reana andmetabelis.

    Sellisteks andmeteks võivad olla:
        - ✓ kehtiv käibemaksu protsent
        - ✓ riiklikud pühad
        - ✓ DEM kurss Eesti krooni suhtes
        - ✓ Jne.

2. Ajutisi andmeid sisaldavad abitabelid mõningate protsesside läbiviimise toetamiseks. Näiteks:
    - ✓ Abitabelid arvete paketi koostamise ja trükkimise toetamiseks, millest 1 sisaldab iga käivitamisega uuendatavat moodustatud arvete nimistut ning teine samadel tingimustel uuendatavat vigaste lepingute (arvet ei õnnestu moodustada) nimistut koos vastavate veateadetega.
    - ✓ Abitabel, mis sisaldab järgnevas arve moodustamise protsessis kasutamiseks välja valitud klientide nimistut.

Teema: Andmebaasi häälestamine

Taust

Siin keskendutakse eelkõige andmebaasi häälestamise võimalustele, millised on kasutatavad andmebaasi projekteerijate (disainerite) poolt süsteemiarenduse disaini etapil. Lisaks nendele võimalustele on olemas veel rida keerulisemaid ja spetsiifilisemaid (ning üldjuhul väiksema efektiivsusega) meetmeid, mida saavad süsteemi töökiiruse tõstmise huvides rakendada süsteemi administraatorid. Selliste meetodite hulka kuuluvad näiteks:

- Tabelite koondamine kobaratesse (klastritesse),
- Hash võtmete kasutamine,
- Parameetrite faili parameetrite häälestamine (erinevate mälustruktuuride jaoks eraldatavad mälu suurused),
- Võimsama riistvara kasutusele võtmine (on eelnevaga seotud, kuna võimaldab eraldada süsteemi erinevatele mälustruktuuridele suuremaid ressursse),
- Rakenduse hajutamine paljude serverite peale,
- …

Eesmärgid ja sisu

Andmebaasi häälestamise eesmärgiks on vastava andmebaasi peal töötavate rakenduste optimaalse töökiiruse saavutamine olemasoleva andmete struktuuri juures.

Optimaalse töökiiruse tagamiseks on olemas väga palju erinevaid meetmeid, mida on võimalik rakendada süsteemiarenduse erinevates faasides (disaini, realisatsiooni, töötava süsteemi ülalpidamise faasis), erinevate rollide (peamiselt andmebaasi administraatorid, programmeerijad, projekteerijad) esindajate poolt.

Andmebaasi häälestamise meetmete rakendamine disaini etapil:

- omab neist kõige suuremat efekti töökiirusele ning
- tuleb süsteemi kui terviku muutmiseks vajalike ressursside kasutamise seisukohalt välja kõige odavam.

Seda põhjusel, et häälestamise vajadusest tingitud muutuste sisseviimisel disaini käigus on tegemist probleeme ennetavate meetmetega, mitte tekkinud probleemide lappimisega.

Kõige üldisemalt on taolised töökiiruse optimeerimise meetodid järgmised:

- viia minimaalseks võrgu koormus (klient-server arhitektuuri puhul),
- kasutada maksimaalselt ära Oracle'i poolt pakutavate mehhanismide võimalusi:
  - piirangud, kitsendused (on deklaratiivsed, ei nõua programmeerimist, nende käivitamine on Oracle'i poolt optimeeritud),
  - andmebaasi trigerid
  - numbrijada generaatorid
  - serverisse laetud protseduurid ja funktsioonid,
  - Oracle'I poolt pakutavad objektide lukustamismehhanismid
- Optimeerida objektide laadimine andmeblokkidesse,
- Optimeerida andmebaasi objektide paigutamine:
  - Paigutada vajadusel andmebaasi objektid eraldi failidesse erinevatel ketastel,
  - Hajutatud andmebaasi korral (mitu serverit) paigutada objektid kõige sobivamasse serverisse

Osa loetletud tehnikatest (kitsendused, andmebaasi trigerid, numbrijada generaatorid) on käsitletud eelmistes teemades, ülejäänud tulevad järgnevalt.

Serveris hoitavad protseduurid

Andmebaasi peal töötavate rakenduste töökiiruse tõstmise huvides on kasulik koondada maksimaalne osa programmikoodist (rakendusmoodulitest (ekraanivormid, aruanded), andmebaasi trigeritest) serveris hoitavatesse protseduuridesse ja funktsioonidesse.

Serveris hoitavate protseduuride (ja funktsioonide) maksimaalne kasutamine aitab optimeerida rakenduste töökiirust järgmistel põhjustel:

1. Vastava protseduuri käivitamiseks rakendusest kantakse üle võrgu vaid 1 käsk (käsk mooduli väljakutseks). Sama funktsionaalsuse saavutamiseks eraldi SQL lausete abil (ilma PL/SQL mootorit kasutamata) tuleks võrgu poole pöörduda korduvalt (1 kord iga üksiku SQL lause saatmiseks üle võrgu).

2. Taolisi protseduure hoitakse serveris kompileeritud kujul, mistõttu väheneb protseduuri käivitamisele kulutatav aeg. Sarnase protseduuri hoidmine rakenduses (näiteks ekraanivormide kirjelduses) eeldaks pikema protsessi läbimist enne käivitamist. Selleks tuleb saata käsk serverile (üle võrgu), milline peab mooduli kompileerima ja mällu laadima. Kunikompileeritud versioon mälus püsib, väheneb järgmistele käivitamistele kulutatav aeg. Kui see korra mälust maha loetakse, tuleb kogu protsess otsast peale läbida.
3. Kui serveri protseduur on korra kettalt mällu loetud ning seejärel olnud pidevas kasutuses (pole uuesti mälust maha loetud), siis väheneb järgmiste pöördumiste käivitusaeg, kuna jäävad ära ketta poole pöördumised (pöördutakse juba mälus oleva protseduuri poole).
4. Kui 1 kasutajaprotsess on taolise protseduuri (funktsiooni) poole korra pöördunud, siis väheneb kõigil järgnevatel pöördumistel suvalise kasutaja poolt käivitamisel selle protseduuri käivitamiseks kulutatav aeg oluliselt.

Viimane on seotud samade ühiskasutusega PL/SQL mälupiirkondade (mälupiirkonnad, mida võivad kasutada kõik Oracle'I protsessid) kasutamisega PL/SQL käskude poolt. Taoliste mälupiirkondade kasutamine annab võidu töökiiruses kahel põhjusel:

1) Juba mällu loetud käskude korral jääb ära analüüsi ja väärtustamise faas (parsing - hõlmab väärtustamist, sõltuvuste analüüsi andmesõnastiku põhjal, privileegide kontrollimist, käivitusplaani valimist, mällu laadimist, …), mis võtab aega palju rohkem kui käivitamine ise.
2) Võimaldab kokku hoida mäluressursse (iga protsessi jaoks pole vaja kinni panna uut mälupiirkonda sarnase operatsiooni teostamiseks) ning kasutada kokku hoitud mälu muude samaaegsete tegevuste sooritamiseks.

Ühiskasutusega mälupiirkondade (SQL ja PL/SQL) maksimaalne ärakasutamine eeldab teatud standarditest kinnipidamist. Mõned põhireeglid on siin järgmised:
1) Panna paika suur- ja väiketähtede, tühikute kasutamise standardid objektide nimedes, väljakutsetes.
2) Panna paika objektide paigutamise reeglid nii, et erinevate kasutajate poolt käivitatud käsud pöörduksid täpselt sama objekti poole (kui sama süntaksiga käsud pöörduvad sama nimega objekti poole, millised paiknevad erinevates kasutajaskeemides, ei loeta neid käske identseteks).

Selleks oleks vaja:

- Luua pseudokasutaja
- Paigutada kõik objektid pseudokasutaja kasutajaskeemi ning anda ülejäänud kasutajatele vastavad õigused nende objektidega opereerimiseks. Teiste kasutajate kasutajaskeemides vastavaid objekte üldse mitte luua.
- Luua public sünonüümid kõigi pseudokasutaja objektide peale nii, et suvalise kasutaja poolt antud käsk kujul SELECT * FROM tabel leiaks ühe ja sama tabeli samas kasutajaskeemis (vastasel juhul peaksid kõik pöördumise rakendustest olema realiseeritud kujul SELECT * FROM kasutajaskeem.tabel).

3) Kasutada samasid muutujanimesid.
4) Konstantide asemel kasutada võimalusel muutujaid.

Lisaks töökiiruse optimeerimise võimalusele on serveri protseduuridel /funktsioonidel muidki eeliseid:

1. Sarnase koodi koondamine ühte protseduuri lihtsustab täienduste ja paranduste sisseviimist (muutused sisse viia ainult ühes kohas).
2. Andmesõnastik garanteerib taoliste protseduuride (funktsioonide) kooskõlalisuse teiste andmebaasi objektidega igal ajahetkel.

Lisaks eelpool vaadeldule pakub lisavõimalusi veel funktsionaalselt kokku kuuluvate serveri protseduuride ja funktsioonide koondamine moodulipakettidesse, mis

- Lihtsustab õiguste andmist vastavate moodulite kasutamiseks - anda täitmise õigus kogu paketi peale mingile rollile või kasutajale selle asemel, et anda iga funktsiooni ja protseduuri peale eraldi sama õigust.
- On võimalik parandada rakenduse jõudlust läbi selle, et:
  - Kõik paketis sisalduvad moodulid loetakse korraga mällu, kui pöördutakse 1 poole neist (ning seetõttu väheneb moodulite laadimiseks vajalike pöördumiste arv)
  - Paketis sisalduva mooduli definitsiooni muutmine (seni kuni ei muudeta paketi päises sisalduvat vastava mooduli spetsifikatsiooni) ei sunni Oracle'it uuesti kompileerima seda moodulit välja kutsuvaid alamprogramme (erinevalt paketiväliste moodulite muutmisest).