LATVIJAS UNIVERSITĀTE

DARJA SOLODOVŅIKOVA

UZ DATU NOLIKTAVAS SHĒMAS EVOLŪCIJU ORIENTĒTS VAICĀJUMU DEFINĒŠANAS UN ATTĒLOŠANAS RĪKS

Promocijas darbs
datorzinātņu doktora (Dr. sc. comp.) zinātniskā grāda iegūšanai

Nozare: datorzinātnes
Apakšnozare: datu apstrādes sistēmas un datortīkli

Zinātniskā vadītāja:
Dr. sc. comp.,
LAILA NIEDRĪTE

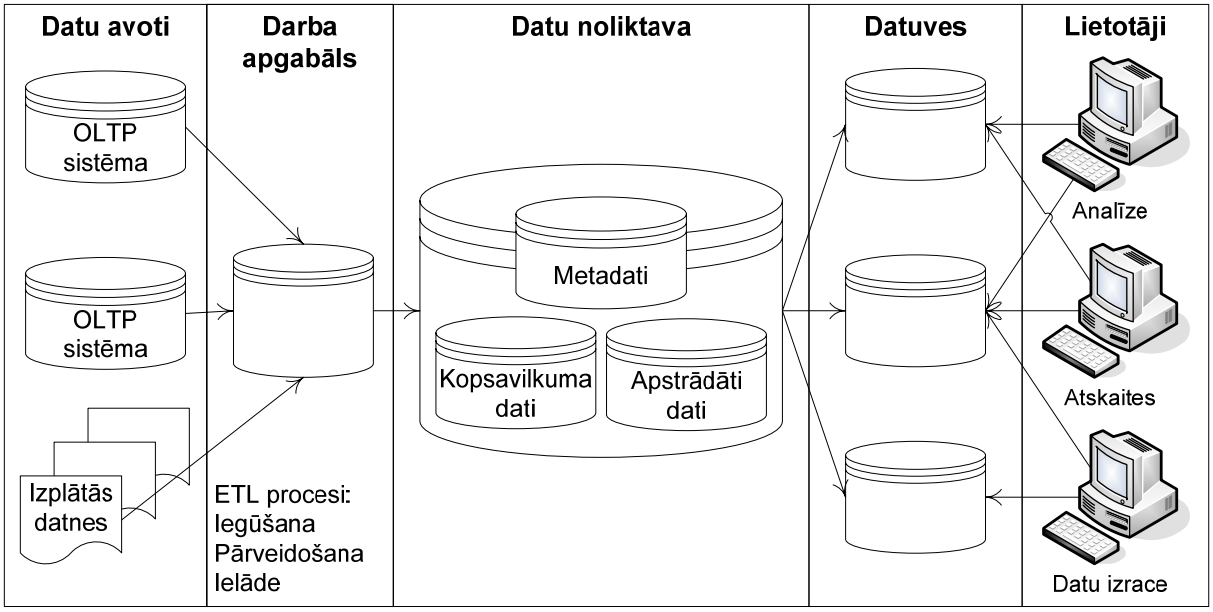
R ī g a - 2010

SATURS

1. IEVADS

Datu noliktava ir datu bāze, kas ir paredzēta vaicājumiem un analīzei, nevis transakciju apstrādei. Tā parasti satur no vairākiem avotiem atvasinātu vēsturisku informāciju. Datu noliktavas atdala analītisko darbu no transakciju apstrādes darba.

Attēlā 1 ir redzama tipiska datu noliktavas arhitektūra. Arhitektūras centrā ir datu noliktavas datu bāze, kas satur metadatus, apstrādātus avotu sistēmu datus, kuri ir pārveidoti, tā lai labāk atbilstu datu noliktavas darījumprasībām, kā arī kopsavilkumu datus.



Att. 1. Datu noliktavas arhitektūra

Papildus datu noliktavas datu bāzei, tipiska datu noliktavas arhitektūra iekļauj darba apgabalu, kas tiek izmantots avotu datu iegūšanai un pārveidošanai, tā lai tie atbilstu datu noliktavas projektējumam un ielādei datu noliktavā. Ar datu iegūšanu, pārveidošanu un ielādi nodarbojas datu iegūšanas, pārveidošanas un ielādes procesi jeb ETL procesi (no angliskā Extract, Transform un Load).

Datu noliktavas datu avoti var būt tiešsaistes transakciju apstrādes sistēmas (OLTP), mantotas sistēmas vai arī vienkāršas izplātās datnes.

Datuves ir datu noliktavas datu apakškopas, kas izstrādātas noteiktai biznesa jomai un ar tām strādā lietotāji, kuri izmanto tiešsaistes analītiskās apstrādes (OLAP) un datu izraces iespējas, kā arī klientu puses datu analīzes rīkus.

Galvenais datu noliktavas mērķis ir lietotājam visērtākajā veidā sniegt precīzu un vēsturiski pareizu informāciju, lai atbalstītu darījumprocesu analīzi un lēmumu pieņemšanu. No vienas puses tas nozīmē, ka datu noliktavai ir jāatspoguļo visas izmaiņas, kas notiek analizējamajā darījumprocesā, un, no otras puses, lietotājam ir jāsaņem atbildes uz viņa jautājumiem pēc iespējas ātrāk un vienkāršāk.

Datu noliktavu var uzskatīt par sistēmu, kur tiek glabāta integrēta, vēsturiska un apkopota informācija no vairākiem datu avotiem. Datu noliktavas ir ļoti atkarīgas gan no analizējamā procesa, gan no datu avotiem. Datu avoti parasti ir atsevišķas neatkarīgas

sistēmas, kur katrai ir savs datu modelis, sava saskarne, vaicājumu iespējas. Šīs sistēmas var mainīties ļoti bieži, t.i. var mainīties šo sistēmu dati, datu modeļi, saskarne, kā arī paši datu avoti var kļūt nepieejami vai arī var parādīties jauni datu avoti. Uzņēmuma programmatūra var mainīties kopā ar uzņēmuma mērķiem un stratēģijām. Jaunas avota programmatūras versijas prasa izmaiņas datu noliktavā. Implementācijas risinājumu izvēle var mainīties un attīstīties, lai apmierinātu darījumprasības un uzņēmuma mērķus.

Ārpus šīm izmaiņām datu avotu sistēmās, bieži mainās arī klientu darījumprasības datu noliktavas programmatūrai. Vairākumu no prasībām definē vadītāji un analītiķi, kas lieto datu noliktavas sistēmu kā atbalstu lēmumu pieņemšanai. Vadītāju un analītiķu darba raksturs ir tāds, ka viņu vajadzības bieži mainās un nesasniedz gala stāvokli, t.i. viņu prasības ir dinamiskas un subjektīvas. Divi iemesli šai dinamiskajai uzvedībai [Bla00] ir:

- Interaktīva daudz-dimensiju analīzes tehnoloģija ir jaunums daudziem darbiniekiem. Tas nozīmē, ka vadītājiem un analītiķiem nav iespējams skaidri formulēt viņu prasības iepriekš.
- Darījumprocesi, kuros piedalās analītiķi, bieži mainās. Šīs izmaiņas darījumprocesos tiek atspoguļotas datu noliktavas programmatūras prasībās. Analītiķi prasa ne tikai ātrāku atbildes laiku (kas var tikt sasniegts ar jaunas un ātrākas aparatūras pasūtīšanu), bet arī vēlas plašāku informāciju, piemēram, pieeju datiem, kas pašlaik nav pieejami datu noliktavā, vai augstākas kvalitātes datus, piemēram, vaicājuma rezultātus ar mazāk nepareizajām vērtībām [Qui99].

Izmaiņas datu avotos vai darījumprasībās var padarīt kļūdainus esošās datu noliktavas shēmas, datu iegūšanas, pārveidošanas un ielādes (ETL) procesus un atskaites, kas definētas no izveidotās datu noliktavas shēmas. Tāpēc visas izmaiņas ir nepieciešams atbilstoši apstrādāt.

Saistībā ar datu noliktavas attīstību var definēt četras problēmas: uzturēšana, pielāgošana, adaptācija un uzturēšana pēc adaptācijas [RLN97]. Par uzturēšanu sauc datu noliktavas atjaunināšanu saskaņā ar izmaiņām avota datos. Par pielāgošanu sauc datu noliktavas satura uzturēšanu pēc izmaiņām datu noliktavas shēmā, kuras izraisa mainītās datu noliktavas darījumprasības. Par adaptāciju sauc datu noliktavas pārveidošanu, lai tā atbilstu izmaiņām datu avotu shēmās. Uzturēšana pēc adaptācijas ir datu noliktavas datu atjaunināšana atbilstoši jaunajai datu noliktavas shēmai.

Šo problēmu triviāls risinājums ir jebkuru izmaiņu gadījumā datu noliktavu projektēt no jauna, bet tas prasa daudz laika un resursu. Ir situācijas, kad eksistējošo datu noliktavas shēmu, datu iegūšanas, pārveidošanas un ielādes procesus un atskaites var automātiski pielāgot izmaiņām bez izstrādātāja dalības, bet esošas datu noliktavas shēmas pielāgošana var izraisīt datu vēstures zudumu. Lai atspoguļotu datu noliktavas evolūciju, ir iespējams realizēt datu noliktavas shēmas versiju pieeju, kad katram mainītam elementam veido jauno versiju.

1.1. Pētījuma motivācija un aktualitāte

Datu noliktavas evolūcijas problēmas ir daudzos projektos, tai skaitā arī projektā, kura izstrādē ir piedalījusies promocijas darba autore. 2004. gadā tika izstrādāta e-studiju datuve,

kas apkopoja datus par kursu pārvaldes sistēmas WebCT izmantošanu. Datuve saturēja trīs zvaigznes shēmas, kur tika uzkrāti dati par e-kursu uzbūvi, lietotājiem un lietotāju aktivitātēm. Datu noliktava tika realizēta Oracle datu bāzē, atskaitēm tika lietots OLAP rīks Oracle Discoverer.

Šajā projektā datuves uzturēšanas laikā nācās saskarties ar datu noliktavas attīstības problēmām. Tika veiktas pieprasītās izmaiņas datu noliktavas programmatūras darījumprasībās, kas noteica datuves shēmas attīstību. Datuve tika papildināta ar vēl vienu zvaigznes shēmu, tika mainīta detalizācijas pakāpe citai zvaigznes shēmai, tika pievienoti jauni elementi datuves shēmai, bet citi no tās tika izmesti. Notika arī dažas izmaiņas datu avotos. Sākumā mainījās WebCT iekšējās datu bāzes struktūra. Notika izmaiņas arī citu datu avotu datu struktūrās, piemēram, Latvijas Universitātes Informatīvajā Sistēmā. Bet 2007. gadā notika pāreja uz citu kursu pārvaldības sistēmu, kas nozīmē, ka viens no datu avotiem kļuva nepieejams, bet tā vietā bija jāizmanto jauns datu avots. Lai risinātu izmaiņu radītās problēmas, datu noliktavas un ETL procesu adaptāciju bija jāveic manuāli, kas prasīja daudz laika un resursu. Papildus tika veidotas atšķirīgas datu noliktavas shēmas versijas un parādījās nepieciešamība nodrošināt lietotājiem atskaišu ģenerēšanu un izpildīšanu no vairākām shēmas versijām, ko nespēja nodrošināt iepriekš lietotais OLAP rīks.

Šobrīd datu noliktavu uzturēšana tiek atbalstīta gan komerciālā, gan atvērtā koda datu noliktavu programmatūrā. Šim nolūkam eksistē vairāki ETL rīki, piemēram, Oracle Warehouse Builder, Informatica PowerCenter, IBM Information Server (Datastage), Pentaho Data Integration, Clover ETL un citi. Komerciālās sistēmas datu noliktavu izstrādei un OLAP rīki neatbalsta ne datu avotu un datu noliktavas shēmas automatizētu izmaiņu apstrādi, ne vienlaicīgu darbu ar vairākām shēmas versijām.

Pēdējo gadu laikā konferencēs par datu noliktavu un datu bāzu pētījumiem, piemēram, DOLAP, DaWaK, CAISE, ADBIS, datu noliktavu uzturēšana un evolūcija ir iekļautas starp interesējošām tēmām. Piemēram, vienā no pēdējiem pārskata rakstiem par datu noliktavu turpmākajiem pētījumiem [RAL+06] par vienu no aktuālākām datu noliktavu projektēšanas problēmām atzīst shēmas evolūciju. Raksta autori uzskata, ka datu noliktavas shēmas evolūcijas problēmas ir tikai daļēji izpētītas un neeksistē komerciālie rīki vai pārstrukturēšanas metodes, kas ir paredzēti šādu problēmu risināšanai. Par galvenajiem pētniecības uzdevumiem šajā jomā ir atzīta efektīvu versiju un datu migrācijas mehānismu nodrošināšana, kas spēj atbalstīt ekspromtvaicājumus, kas aptver vairākas versijas, kā arī ETL procesu pielāgošana datu avota shēmu izmaiņām.

Datu noliktavas evolūcijas problēmas tiek risinātas arī zinātniskajās publikācijās. Piemēram, daži autori apskata izmaiņas datu noliktavas darījumprasībās un mēģina formāli specificēt evolūcijas operācijas un to sekas [HMV99], [MV00], [Bla00], [KPR04]. Citi autori piedāvā metodes kā risināt evolūcijas problēmas, kas ir saistītas ar izmaiņām datu avotos [Bel02], [RKZ00], piemēram, izteikt datu noliktavas shēmu kā transformāciju kopumu no avota shēmām un transformācijas informāciju izmantot, lai adaptētu datu noliktavas shēmu pēc datu avotu shēmas izmaiņām [Mar00], [MP03], [VMP03]. Un pēdējais, manuprāt, visperspektīvākais virziens ir daudzversiju datu noliktavas [BMB+03], [SNP05], [GLR+06], [MW04], [WB05], tas nozīmē, ka pēc izmaiņām datu noliktavas shēmā tiek veidotas vairākas

shēmas versijas, kas atspoguļo datu noliktavas darījumprasības, kas bija spēkā noteiktajā laika periodā, un šo laika periodu sauc par versijas derīguma periodu.

2009. gadā ADBIS (Advances in Databases and Information Systems) konferences ietvaros tika organizēts pirmais seminārs, kas tika veltīts datu noliktavas evolūcijas problēmām (Managing Evolution of Data Warehouses), kur īpaši tika runāts par daudzversiju pieeju, kas apliecina, ka izvēlētā pieeja ir perspektīva. Seminārā ar referātu piedalījās arī promocijas darba autore.

1.2. Promocijas darba mērķis un uzdevumi

Promocijas darba mērķis ir izstrādāt daudzversiju datu noliktavas koncepciju un atskaišu definēšanas un attēlošanas rīku, kas spēj izpildīt atskaites uz vairākām datu noliktavas shēmas versijām.

Lai sasniegtu mērķi, ir jāatrisina šādi uzdevumi:

1) Izpētīt problēmas, kas saistītas ar datu noliktavas attīstību, un eksistējošas pieejas šo problēmu risināšanai.
2) Piedāvāt datu noliktavas arhitektūru, kas ļautu risināt datu noliktavas evolūcijas problēmas, kas ietvērtu risinājumus datu avotu struktūras izmaiņu automātiskai noteikšanai, datu noliktavas shēmas mainīšanai un ETL procesu automatizētai adaptācijai saskaņā ar notikušajām izmaiņām.
3) Izstrādāt daudzversiju datu noliktavas atbalstu, t.i. risinājumu shēmas versiju fiziskajai glabāšanai relāciju datu bāzē, formālo daudzversiju datu noliktavas modeli un metadatus, kas apraksta datu noliktavas shēmas versijas, kā arī fizisko datu noliktavas struktūru un datu noliktavas datu semantiku.
4) Izstrādāt algoritmus datu noliktavas shēmas izmaiņu apstrādei.
5) Izveidot daudzversiju datu noliktavas atskaišu rīku, kas, balstoties uz metadatiem, ļauj definēt, izpildīt un adaptēt atskaites pēc dažāda veida izmaiņām datu noliktavas shēmā.

1.3. Promocijas darba galvenie rezultāti

Galvenie promocijas darba rezultāti ir sekojoši:

- Darbā tika izanalizētas un klasificētas datu noliktavas evolūcijas problēmas. Tika izdalītas problēmu klases, kas ir datu noliktavas uzturēšana, pielāgošana, adaptācija un uzturēšana pēc adaptācijas. Visu problēmu klasēm tika izpētīti esošie risinājumi, tai skaitā arī datu noliktavas shēmas versiju pieeja. Tika konstatētas priekšrocības un trūkumi esošajos risinājumos, kas ir ņemts vērā, izstrādājot jauno pieeju daudzversiju datu noliktavai.
- Piedāvāta datu noliktavas evolūcijas arhitektūra, kas ir spējīga automātiski noteikt izmaiņas datu avotos, adaptēt datu noliktavas shēmu un ETL procesus saskaņā ar administratora lēmumu, apstrādāt tiešas izmaiņas datu noliktavas shēmā, atbalstīt datu noliktavas shēmas versijas gan izstrādes vidē, gan lietotāju vidē atskaitēs.

- Piedāvāti un izstrādāti datu noliktavas shēmas versiju metadati, kas balstās uz atzītu metadatu standartu, un shēmu versiju glabāšanas risinājums relāciju datu bāzē.
- Izstrādāts atskaišu definēšanas un izpildīšanas rīks, kas balstīts uz piedāvāto datu noliktavas shēmas versiju metadatu modeli, un kas ļauj lietotājam pašam viegli un ātri izveidot ekspromt-atskaites uz vairākām datu noliktavas shēmas versijām un analizēt datus atskaitē, attēlojot tos tabulās vai grafikos un veicot OLAP operācijas.
- Izstrādāts SQL vaicājumu konstruēšanas algoritms, kas, balstoties uz lietotāja definētas atskaites specifikāciju metadatos, ģenerē vaicājumus vienai vai vairākām datu noliktavas shēmas versijām saskaņā ar lietotāja izvēli.
- Izstrādātā pieeja tika aprobēta e-studiju datuves shēmas versijām, novērtēta un salīdzināta ar tradicionālo pieeju ar komerciālo atskaišu rīku, kas neatbalsta shēmas versijas. Datu noliktavas shēmas metadatu pilnība tika novērtēta ar testa piemēru kopu, kas pārklāj dažādus datu noliktavas izmaiņu gadījumus.

1.4. Rezultātu publikācijas un prezentācijas konferencēs

Promocijas darbs balstās uz pētījumiem, kuru rezultāti ir atspoguļoti publikācijās:

- Publikācijas par e-studiju datu noliktavu un datu noliktavas shēmas adaptāciju saskaņā ar izmaiņām datu avotos:

[SN05] Solodovņikova D., Niedrīte L. 'Using Data Warehouse Resources for Assessment of E-Learning Influence on University Processes'. *Proceedings of the 9th East-European Conference on Advances in Databases and Information Systems, Tallinn, Estonia*, 2005.

[SN06] Solodovņikova D., Niedrīte L. 'Data Warehouse Adaptation after Changes in Source Schemata'. *Proceedings of the 7th International Baltic Conference on Databases and Information Systems, Vilnius, Lithuania,* 2006.

- Publikācija par datu noliktavas evolūcijas arhitektūru:

[Sol07] Solodovņikova D. 'Data Warehouse Evolution Framework'. *The Fourth Spring Young Researchers Colloquium on Databases and Information Systems, SYRCoDIS'2007, Moscow, Russia,* 2007.

- Publikācijas par datu noliktavas shēmas versijām, metadatiem un izmaiņām:

[Sol08a] Solodovņikova D. 'The Formal Model for Multiversion Data Warehouse Evolution'. *Postconference proceedings of the 8th International Baltic Conference on Databases and Information Systems, Tallinn, Estonia, Frontiers in Artificial Intelligence and Applications by IOS Press*, 2008.

[Sol08b] Solodovņikova D. 'Metadata to Support Data Warehouse Evolution'. *Proceedings of the 17th International Conference on Information Systems Development by Springer, Paphos, Cyprus,* 2008.

- Publikācijas par atskaišu definēšanas un izpildīšanas rīku:

[Sol08c] Solodovņikova D. 'Building Queries on Multiple Versions of Data Warehouse'. *Proceedings of the 8th International Baltic Conference on Databases and Information Systems, Tallinn, Estonia*, 2008.

[Sol09] Solodovnikova D. 'An Approach to Supporting Data Warehouse Schema Versions: Overview and Case Study'. *Proceedings of the 13th International Conference on Advances in Databases and Information Systems, Riga, Latvia*, 2009.

[SN10] Solodovnikova D., Niedrite L. 'Evolution-Oriented User-Centric Data Warehouse'. *Proceedings of the 19th International Conference on Information Systems Development by Springer, Prague, Czech Republic,* 2010 (tiks publicēts).

Promocijas darba rezultāti prezentēti 7 starptautiskajās konferencēs:

- ADBIS 2005. gadā [SN05] un 2009. gadā [Sol09],
- Baltic DB&IS 2006. gadā [SN06] un 2008. gadā [Sol08c],
- SYRCoDIS 2007. gadā [Sol07],
- ISD 2008. gadā [Sol08b] un 2010. gadā [SN10];

un 4 vietējās konferencēs:

- 65. LU zinātniskā konference, Informācijas tehnoloģiju sekcija, 2007. gada februāris, prezentācija: Solodovņikova, D. 'Datu noliktavas evolūcijas problēmu atbalsts atskaišu rīkos',
- 66. LU zinātniskā konference, Informācijas tehnoloģiju sekcija, 2008. gada februāris, prezentācija: Solodovņikova, D. 'Datu noliktavas evolūcijas atbalsts metadatos',
- 67. LU zinātniskā konference, Informācijas tehnoloģiju sekcija, 2009. gada februāris, prezentācija: Solodovņikova D. 'Datu noliktavu evolūcijas problēmas',
- 68. LU zinātniskā konference, Informācijas tehnoloģiju sekcija, 2010. gada februāris, prezentācija: Solodovņikova D. 'Vaicājumu definēšanas un attēlošanas rīks datu noliktavai ar vairākām shēmas versijām'.

1.5. Promocijas darba struktūra

Promocijas darbam ir 177 lappuse, 66 attēls un 13 tabulas. Promocijas darbs ir sakārtots 10 nodaļās. Turpmākajām promocijas darba nodaļām ir sekojoša struktūra.

2. nodaļā tiek definēti datu noliktavu pamatelementi, tiek aprakstītas datu noliktavas reprezentācijas un OLAP operācijas, ko bieži pielieto datu noliktavas datu analīzē.

3. nodaļā ir klasificētas datu noliktavas evolūcijas problēmas, kas ir datu noliktavas uzturēšana, pielāgošana, adaptācija un uzturēšana pēc adaptācijas, apskatīti šo problēmu iespējamie risinājumi datu noliktavām, kas ir definētas kā materializētu skatu kopas uz datu avotiem un kā daudzdimensiju modeļi.

4. nodaļā ir aprakstīta e-studiju datuve, tās modelis, vēsture, tiek ilustrētas evolūcijas problēmas šīs datuves attīstības laikā un izmantotie risinājumi.

5. nodaļā tiek apskatīta datu noliktavas arhitektūra, kas tika projektēta, lai risinātu dažāda veida datu noliktavas evolūcijas problēmas. Tiek aprakstīti arhitektūras komponenti un to darbības principi.

6. nodaļā tiek definēts daudzversiju datu noliktavas formālais modelis un tiek dots datu noliktavas repozitorija shēmas un atskaišu metadatu modeļu apraksts. Šie metadati tika izstrādāti saskaņā ar datu noliktavas formālo modeli. Aprakstītie metadati ir datu noliktavas evolūcijas arhitektūras sastāvdaļa un vairāku arhitektūras komponentu darbība balstās tieši uz šiem metadatiem. Promocijas darba ietvaros izstrādātais rīks definē un izpilda datu noliktavas atskaites, balstoties uz datu noliktavas shēmas un atskaišu metadatiem. Tiek apskatīti metadatu modeļi, kas apraksta datu noliktavas shēmu fiziskajā, loģiskajā un semantiskajā līmenī, kā arī atskaites uz datu noliktavas shēmām.

7. nodaļā formāli tiek aprakstītas procedūras, kas tiek izpildītas dažādu izmaiņu gadījumos. Šīs procedūras veic izmaiņas 6. nodaļā definētā datu noliktavas formālā modeļa instancēs. Tiek aprakstītas arī izmaiņas, kas piedāvātajā pieejā ir jāveic datu noliktavas shēmas metadatos datu noliktavas evolūcijas gadījumā. Šo izmaiņu rezultātā tiek veidota jauna datu noliktavas shēmas versija.

8. nodaļā tiek izklāstīts vaicājumu ģenerēšanas algoritms, kas būvē SQL ekspromt-vaicājumus uz vienu vai vairākām datu noliktavas shēmas versijām, balstoties uz shēmas metadatiem un atskaišu specifikāciju. Šis algoritms tiek izmantots vaicājumu definēšanas un attēlošanas rīkā.

9. nodaļā tiek aprakstītas atskaišu rīka tehniskās realizācijas nianses, prasības atskaišu definēšanai un attēlošanai, rīka arhitektūra un izmantotās tehnoloģijas un rīki, atskaišu noformējuma metadati.

10. nodaļā tiek raksturots piedāvātās pieejas izvērtējums, baltoties uz e-studiju datuves projektu un piedāvātās pieejas pielietojuma salīdzinājums ar pieeju vienā no komerciāliem atskaišu rīkiem, kas neatbalsta datu noliktavas shēmas versijas. Tiek detalizēti aprakstītas izmaiņas datu bāzē, darbības, kas tika veiktas, lai atspoguļotu e-studiju datuves aktivitātes zvaigznes shēmas versijas metadatos, vaicājumu ģenerēšanas process uz divām shēmas versijām. Tiek aprakstīts arī piedāvātas pieejas pielietojums testa piemēros no zinātniskiem rakstiem par datu noliktavas evolūciju, lai izvērtētu pieejas korektumu un datu noliktavas shēmas metadatu pilnību.

Darba nobeigumā tiek apkopoti darba galvenie rezultāti, ieguldījums un secinājumi par izstrādāto pieeju, tiek aplūki tālāko iespējamo pētījumu virzieni un plānotie papildinājumi piedāvātajai pieejai.

2. DATU NOLIKTAVU PAMATELEMENTI

2.1. Nodaļas nolūks

Šīs nodaļās nolūks ir definēt datu noliktavas shēmu un citus turpmāk promocijas darbā izmantotus jēdzienus. Šajā nodaļā tiek aprakstītas divas populārās datu noliktavas reprezentācijas, t.i. daudzdimensiju shēma un materializētu skatu kopums, kā arī datu noliktavas realizācija relāciju datubāzē un tiešsaistes analītiskas apstrādes operācijas, kuras tiek izmantotas, lai darbotos ar datu noliktavas datiem.

2.2. Datu noliktavas daudzdimensiju shēma

2.2.1. Daudzdimensiju shēmas elementi

Lai modelētu datu noliktavu parasti izmanto *daudzdimensiju shēmu*. Daudzdimensiju shēmā galvenais objekts ir *mērījums*, kas satur skaitliskus un parasti aditīvus datus, kas raksturo noteiktu *faktu* [CD97], [MW04]. Faktu piemēri ir pārdošana, piegāde, studentu reģistrācija uz kursiem, u.c. Mērījumu piemēri ir ienākumi, budžets, apgrozījums, transakciju skaits, studentu skaits, u.c.

Katrs mērījums ir atkarīgs no vairākām *dimensijām*, kas raksturo mērījuma kontekstu [CD97]. Dimensiju piemēri ir laiks, preces, pasūtītāji, piegādātāji, studenti, studiju programmas, studiju kursi, u.c. Vairākas dimensijas kopā viennozīmīgi definē mērījuma vērtību. Tas nozīmē, ka daudzdimensiju skatījumā mērījums ir vērtība dimensiju telpā. Katru dimensiju raksturo vairāki *atribūti*, kas parasti satur aprakstošu un teksta informāciju. Atribūtu piemēri ir preces nosaukums, kategorija, pasūtītāja vārds, uzvārds, piegādātāja firmas nosaukums, adrese, studenta vārds, uzvārds, studiju gads, u.c.

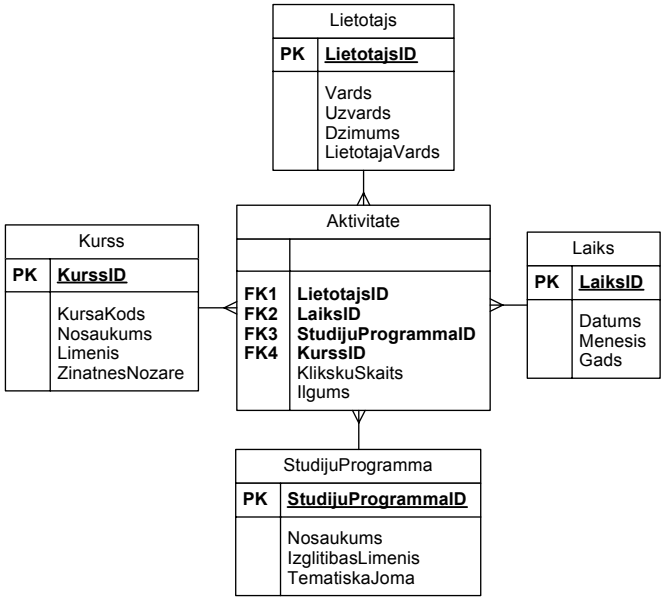
Dimensijas satur arī *hierarhijas*, kas tiek izmantotas datu analīzē. Hierarhijas sastāv no sakārtotiem dimensiju *līmeņiem*, kas savukārt tiek būvēti no dimensijas atribūtiem [Ora03]. Hierarhiju līmeņi definē agregācijas pakāpi, kas ir svarīga analīzes veikšanā. Hierarhijā, katrs līmenis ir loģiski saistīts ar līmeni virs un zem tā. Datu vērtības zemākajā līmenī ir apkopotas datu vērtībās augstākajā līmenī. Piemēram, preču dimensijā ir hierarhijas attiecības starp preces nosaukumu un kategoriju, laika dimensijā ir hierarhijas attiecības starp gadu, mēnesi un dienu.

2.2.2. Daudzdimensiju shēmas realizācijas relāciju datu bāzē

Relāciju datu bāzēs datu noliktavas daudzdimensiju shēmas realizācijai parasti izmanto *zvaigznes shēmu*. Zvaigznes shēma sastāv no vienas *faktu tabulas*, kas satur mērījumus, un vairākām *dimensiju tabulām*, kas attēlo katru dimensiju [CD97]. Katrs ieraksts faktu tabulā satur skaitliskas mērījumu vērtības un norādes uz dimensiju tabulām, kas parasti ir realizētas kā ārējas atslēgas un visu ārējo atslēgu kolonnas parasti veido faktu tabulas

primāro atslēgu. Norādes nodrošina mērījumu koordinātes dimensiju telpā. Katra dimensiju tabula satur kolonnas, kas atbilst dimensijas atribūtiem.

Zvaigznes shēmas piemērs ir redzams attēlā 2. Šajā zvaigznes shēmā ir faktu tabula *Aktivitate*, kas satur mērījumus: *KlikskuSkaits* un *Ilgums*, un četras dimensijas: *Lietotajs, Laiks, Kurss* un *StudijuProgramma*.

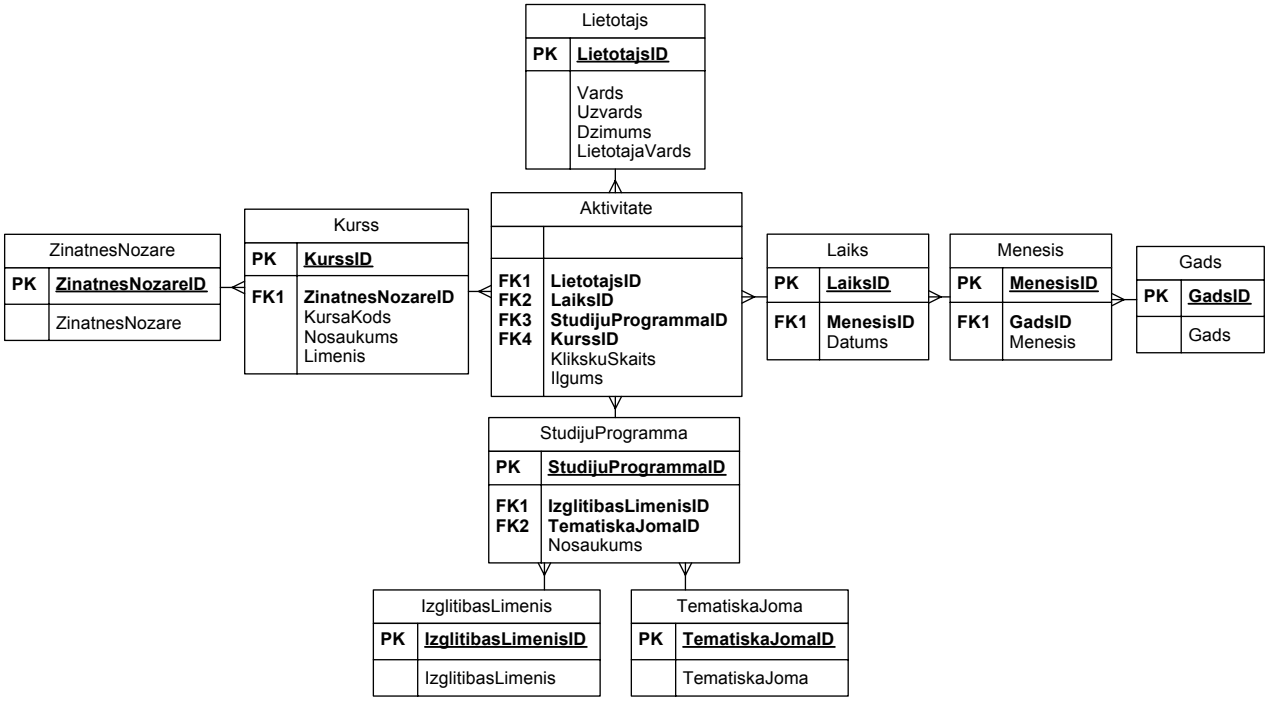


Att. 2. Zvaigznes shēma

Zvaigznes shēmas neatbalsta dimensiju hierarhijas tiešā veidā. *Sniegpārslas shēmas* papildina zvaigznes shēmas ar dimensiju hierarhiju atbalstu, kas ir sasniegts ar dimensiju tabulu normalizāciju. Sniegpārslas shēmas dimensijas ir vieglāk atjaunot, bet zvaigznes shēmas denormalizētās dimensiju tabulas ir vienkāršākas un vairāk piemērotas datu analīzē [CD97]. Zvaigznes shēmām ir labāka veiktspēja, jo vaicājumi uz tās ir vienkāršāki, jo tiek savienots mazāks tabulu skaits, un to izpildīšanas laiks ir ātrāks, visa informācija par dimensijas vērtību tiek glabāta vienā tabulas ierakstā.

Sniegpārslas shēmas piemērs ir redzams attēlā 3. Šajā piemērā dimensiju tabulas *Laiks, StudijuProgramma* un *Kurss* ir normalizētas un ir skaidri redzamas šo dimensiju hierarhijas. Piemēram, dimensiju tabulā *Laiks* ir hierarhija *Gads->Menesis->Datums*. Dimensiju tabulā *StudijuProgramma* ir divas paralēlas hierarhijas: *IzglitibasLimenis-> Nosaukums* un *TematiskaJoma->Nosaukums*.

Promocijas darbā ir izmantotas datu noliktavas zvaigznes shēmas daudzdimensiju datu glabāšanai relāciju datu bāzē. Bet lai varētu attēlot hierarhiju līmeņus shēmu diagrammās, tās tika veidotas kā sniegpārslas shēmu diagrammas.



Att. 3. Sniegpārslas shēma

2.3. Datu noliktava kā materializēto skatu kopa

Vairākos literatūras avotos, tai skaitā arī par datu noliktavas uzturēšanu un datu avotu evolūciju [AAS+97], [GM95], [ZGH+95], datu noliktavu bieži reprezentē kā integrētu materializēto skatu kopumu uz avota sistēmām. Avota sistēmas šajā pieejā parasti ir savā starpā neatkarīgas relāciju datu bāzes sistēmas. Skats ir atvasināta relācija, kas ir definēta citu datu avotu (bāzes) relāciju izteiksmē [GM95]. Skats definē funkciju no bāzes tabulu kopas uz atvasinātu tabulu. Parastiem skatiem šī funkcija tiek izrēķināta katru reizi, kad skatu izmanto. Skatu var materializēt, saglabājot skata kortežus datubāzē. Salīdzinājumā ar skatu, pieeja pie materializēta skata ir daudz ātrāka, jo nav nepieciešams katru reizi rēķināt skatu no bāzes datiem.

Datu noliktavās materializētos skatus izmanto, lai savāktu informāciju no vairākām citām avotu datu bāzēm. Datu noliktavas lietotāju vaicājumi tiek izpildīti uz materializētiem skatiem bez pieejas avotu datu bāzēm. Tiek uzskatīts, ka materializēti skati satur iepriekš aprēķinātus datu noliktavas vaicājumu rezultātus, kas tiek glabāti fiziski datu noliktavas datu bāzē. Tas nozīmē, ka materializētā skata dati kļūst neaktuāli pēc noteikta laika, kad notiek izmaiņas avotu datu bāzēs, tāpēc materializēto skatu datus inkrementāli atjauno.

Ļoti bieži datu noliktavu uzskata par *atlases-projekcijas-savienojuma* (select-project-join vai SPJ) materializētu skatu kopumu [QGM+96]. Tie ir skati, kas satur vienu projekciju, aiz kuras seko viena atlase un tad viens bāzes relāciju Dekarta reizinājums. Tas nozīmē, ka SQL valodā skats tiek definēts ar SELECT ... FROM ... WHERE teikumu.

Papildus atlases-projekcijas-savienojuma skatiem datu noliktavas materializētus skatus var definēt arī ar sarežģītākām konstrukcijām. Piemēram, *agregācijas skati* papildus SELECT, FROM un WHERE daļām satur arī grupēšanas kolonnas (GROUP BY), agregācijas funkcijas SELECT daļā un nosacījumus ar agregācijas funkcijām HAVING daļā [GMR+01]].

Arī skats var būt definēts kā vairāku apakšvaicājumu apvienojums (UNION) vai starpība (EXCEPT).

2.4. Datu noliktavas datu analīze – OLAP operācijas

Lietotāji analizē datu noliktavas informāciju, izmantojot OLAP rīkus. Lai atbalstītu daudzdimensiju datu analīzi, OLAP rīkos ir realizētas OLAP operācijas. Šajā apakšnodaļā tiek apskatītas populārākas OLAP operācijas, kas tiek atbalstītas promocijas darbā piedāvātajā risinājumā.

Agregācija (vai konsolidācija, vai roll-up) ir mērījuma apkopotu datu iegūšana no detalizētākiem mērījuma datiem [JLV+03]. Agregācija sastāv no divām daļām. Pirmkārt, tiek izmantotas datu attiecības (saskaņā ar hierarhiju līmeņiem) dimensijā, pēc kuras lietotājs vēlas apkopot datus. Otrkārt, mērījuma vērtības tiek apkopotas, izmantojot kādu agregācijas funkciju, saskaņā ar izvēlēto hierarhijas līmeni ar mazāku detalizācijas pakāpi. Ar agregācijas operāciju lietotājs var iegūt apkopotus mērījuma datus, mainot detalizācijas pakāpi no zemākas uz augstāku saskaņā ar kādas hierarhijas līmeņiem. Piemēram, mērījuma *KlikskuSkaits* datus par dienu var apkopot datos par mēnesi, bet mēneša datus savukārt var apkopot līdz gada datiem (attēls 2.).

Detalizācija (vai drill-down) ir agregācijas pretēja operācija. Ar šo operāciju iegūst vairāk detalizētus mērījuma datus [JLV+03]. Detalizācijas „ceļš" tiek definēts ar dimensiju hierarhijām. Ar detalizācijas operāciju lietotājs iegūst detalizētākus mērījuma datus, mainot detalizācijas pakāpi no augstāka uz zemāku līmeni.

Sagriešana (vai slicing) ir mērījuma datu iegūšana, kas atbilst noteiktam nosacījumam uz izvēlētas dimensijas [JLV+03]. Ar sagriešanu lietotājs izvēlas kādu noteiktu vērtību vienai dimensijai un iegūst mērījumu vērtības, kas atbilst tikai šai fiksētai dimensijas vērtībai. Piemēram, var izvēlēties aprēķināt lietošanas laiku konkrētam studiju kursam (attēls 2.), izmantojot nosacījumu *Kurss.Nosaukums='Datu noliktavas'*. Atšķir arī atsevišķu operāciju – *griešanu apakškubos* (vai dicing), kad tiek fiksētas vairāku dimensiju vērtības.

Atlase (vai selection, vai screening, vai filtrēšana) ir operācija, kad datiem tiek uzstādīts nosacījums, lai ierobežotu attēlotu datu apjomu [JLV+03]. Piemēram, var atlasīt tikai 10 aktīvākos lietotājus pēc kopēja klikšķu skaita vai atlasīt kursus, kurus neizmanto katru dienu (attēls 2.).

Rotēšana (vai pivoting) maina dimensiju orientāciju atskaitē. Piemēram, OLAP rīkos datu noliktavas datus bieži attēlo ar matricveida atskaitēm, kur dažu dimensiju atribūtu vērtības ir atspoguļotas kā kolonnu galvenes un dažu citu dimensiju atribūtu vērtības ir atspoguļotas kā rindu galvenes, bet mērījuma vērtības ir attēlotas iegūtās tabulas šūnās. Šāda veida atskaitē ar rotēšanas operāciju lietotājs var mainīt kolonnas un rindas vietām vai var pārvietot vienu vai vairākus dimensiju atribūtus no rindas uz kolonnu vai otrādi.

OLAP rīkos lietotāji izmanto arī citas operācijas, piemēram, *kārtošanu* (vai ranking), izrēķinātu atribūtu definēšanu, u.c. [CD97]. Papildus OLAP operāciju atbalstam eksistē daži rīki, kas palīdz lietotājam būvēt *ekspromtvaicājumus* (ad-hoc queries), kas tiek konstruēti uz datu noliktavas shēmas datiem, un kuru rezultāti ir nepieciešami lietotājam analīzei. Bieži grafiskajos OLAP lietotāju rīkos datu noliktavas dati tiek attēloti kā daudzdimensiju datu

kubi, kuru šķautnes atspoguļo datu noliktavas shēmas dimensijas, bet daudzdimensiju apakškubi satur lietotāju interesējošo mērījumu vērtības, kas atbilst konkrētajām dimensijām [HRU96]. Lietotāji analizē divu dimensiju, trīs dimensiju vai vēl lielāku skaitu dimensiju datu kuba apakškubus, lai iegūtu interesējošo informāciju.

3. DATU NOLIKTAVAS EVOLŪCIJAS PROBLĒMAS

3.1. Nodaļas nolūks

Šajā nodaļā ir apskatītas problēmas, kas saistītas ar datu noliktavas attīstību, un kas var padarīt esošas datu noliktavas shēmas, datu atjaunināšanas procesus un atskaites uz datu noliktavas shēmu kļūdainus. Tiek dota dažādu datu noliktavas evolūcijas problēmu klasifikācija un raksturojums un analizētas eksistējošas pieejas šo problēmu risināšanai datu noliktavās, kas definētas kā materializētu skatu kopums uz datu avotiem vai kā daudzdimensiju datu modeļi.

3.2. Problēmu klasifikācija

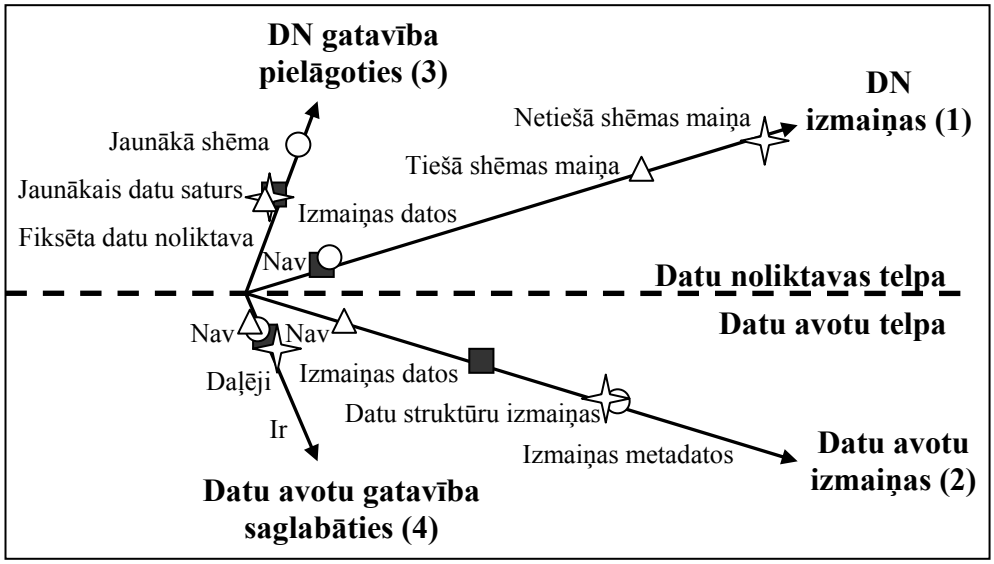
Rakstos par datu noliktavas evolūciju datu noliktavu bieži reprezentē kā materializēto skatu kopumu uz avota sistēmām. Bet tas ne vienmēr ir iespējams, jo datu noliktavai ir sarežģīta arhitektūra, kas sastāv no vairākiem komponentiem.

Materializēto skatu izmantošana datu noliktavu komplicētajā arhitektūra, kur katrs no komponentiem laika gaitā var mainīties, var izraisīt vairākas problēmas, piemēram, problēmas, kas ir saistītas ar skatu *uzturēšanu*, *pielāgošanu* un *adaptāciju*. Šīs problēmas var paplašināt, jo tās paliek spēkā arī daudzdimensiju datu noliktavas reprezentācijā. Tādēļ šajā nodaļā tiks apskatīti arī tie risinājumi, kas piedāvāti datu noliktavu materializēto skatu reprezentācijai.

Materializēto skatu evolūcijas problēmu klasifikācija, kas pamatojas uz izmaiņu veidiem, kas var notikt ar skatu un datu avotiem, un vēlamo skata pielāgošanas līmeni, tika piedāvāta rakstā [RLN97]. Ņemot par pamatu šo klasifikāciju, var sakārtot arī problēmas, kas ir saistītas ar daudzdimensiju modeļa datu noliktavas attīstību. Šī pielāgotā datu noliktavas evolūcijas problēmu klasifikācija ir redzama attēlā 4. Datu noliktavas evolūcijas problēmas ir definētas ar koordinātēm četrās asīs:

- *Ass „Datu avotu izmaiņas"*: Ir triju veidu izmaiņas, kas var notikt datu avotos. Tās ir (1) *izmaiņas metadatos*, piemēram, informācijas pievienošana par to, ka kāda relācija vienā avotā ir saistīta ar citu relāciju no cita avota, (2) *datu struktūru izmaiņas*, piemēram, atribūta vai relācijas pievienošana vai dzēšana, un (3) *izmaiņas datos*, piemēram, jauna kortеža pievienošana relācijai.
- *Ass „Datu avotu gatavība saglabāties"*: Šī dimensija attiecas uz datu avotu gatavību nodrošināt informāciju, kas ir nepieciešama datu noliktavai un ETL procesiem, kas izmanto šos datu avotus. Iespējamās šīs dimensijas vērtības atbilst datu avotu gatavībai nodrošināt visas, dažas vai nekādas spējas (kas, piemēram, ir vaicājumu saskarne, shēma vai datu saturs), kas ir nepieciešamas datu noliktavai. Piemēram, ja datu avots var turpināt piegādāt datus tādā veidā, kā tas jau ir izmantots datu noliktavā, tad nav nepieciešamības neko mainīt datu noliktavas shēmā vai ETL procesos.

- *Ass „Datu noliktavas izmaiņas"*: Vērtība *izmaiņas datos* attiecas uz problēmu, kad mainās dati datu avotos un kā atspoguļot šīs izmaiņas datu noliktavā, nerēķinot visu datu noliktavas saturu no sākuma. Vērtība *tiešā shēmas maiņa* attiecas uz gadījumu, kad pats datu noliktavas izstrādātājs maina datu noliktavas shēmu, bet *netiešā definīcijas maiņa* nozīmē, ka datu noliktavas shēma mainās, reaģējot uz kaut kādām izmaiņām datu avotos.
- *Ass „Datu noliktavas gatavība pielāgoties"*: Šī dimensija apraksta vēlamo elastīguma pakāpi datu noliktavas pielāgošanas izmaiņām kontekstā, t.i. vai datu noliktavas dati vai shēma jāpielāgo pēc izmaiņām datu avotos vai pašā datu noliktavā. Šī dimensija norāda, cik nepieciešami ir dažādi datu noliktavas komponenti (piemēram, dati, shēma, ETL procesi), lai datu noliktavu varētu arī turpmāk izmantot. *Fiksēta datu noliktava* nozīmē, ka datu noliktavā nevar tikt apstrādātas nekādas izmaiņas ar datiem vai shēmu. *Jaunākais datu saturs* nozīmē, ka datu noliktavas lietotāji vēlas, lai tā vienmēr saturētu jaunākos datus pēc jebkurām izmaiņām avotos vai pašā datu noliktavā. *Jaunākā shēma* nozīmē, ka datu noliktavas lietotāji vēlas, lai datu noliktavas shēma būtu pielāgojama izmaiņām datu avotu datu struktūrās.



Datu noliktavas uzturēšana

Datu noliktavas pielāgošana

Datu noliktavas adaptācija

Datu noliktavas uzturēšana pēc adaptācijas

Att. 4. Datu noliktavas evolūcijas problēmu klasifikācija

Saskaņā ar iepriekš aprakstīto klasifikāciju, ir definētas četras datu noliktavas evolūcijas problēmas, proti, uzturēšana, pielāgošana, adaptācija un uzturēšana pēc adaptācijas. Katrai datu noliktavas evolūcijas problēmai atbilst noteiktas koordinātes iepriekš aprakstītajās asīs. Katra problēma ar atbilstošām koordinātēm, kā arī iespējamie risinājumi, ir aplūkoti tālāk.

3.3. Datu noliktavu uzturēšanas problēma

Datu noliktava glabā integrēto informāciju no vairākiem datu avotiem, kas var būt heterogēni, dalīti un autonomi. Kad mainās dati kaut vienā no avotiem, datus datu noliktavā vajag atbilstoši atjaunināt. Datu noliktavas atjaunināšanas procesu, reaģējot uz izmaiņām avotu datos, sauc par datu noliktavas *uzturēšanu*. Datu noliktavas uzturēšanas procesu raksturo sekojošas koordinātes (att. 4.): (1) datu noliktavai nav izmaiņu pirms uzturēšanas procesa izpildes, (2) datu avotos mainās dati, (3) datu noliktavas lietotāji vēlas, lai datu noliktavā vienmēr būtu pieejama jaunākā informācija, un (4) datu avotos nesaglabājas vecais datu stāvoklis.

Tālākajās apakšnodaļās tiek aprakstītas pieejas datu noliktavas uzturēšanas problēmai datu noliktavās. Tiek apskatīti abi datu noliktavas reprezentācijas gadījumi: gan kā materializēto skatu kopums uz avotiem, gan kā daudzdimensiju datu modeļi.

3.3.1. Materializēto skatu uzturēšana

Literatūrā ir studēta uzturēšanas problēma materializētiem skatiem, t.i. materializēta skata uzturēšana datu noliktavā. Materializēta skata uzturēšanas procesu var veikt, vienkārši izrēķinot pilno skatu, katru reizi, kad notiek izmaiņas avota datos [GM95]. Šādus skata uzturēšanas algoritmus sauc par skata pārrēķināšanu.

Dažreiz pilna materializēta skata izrēķināšana ir izšķērdīga. Bieži izdevīgāk rēķināt tikai izmaiņas skatā un atbilstoši rediģēt skatu. Algoritmus, kas izrēķina izmaiņas materializētā skatā, reaģējot uz izmaiņām bāzes relācijās, sauc par materializētā skata inkrementālas uzturēšanas algoritmiem.

Arī skata uzturēšanas algoritmus var iedalīt tūlītējas un atliktas atjaunināšanas algoritmos. Algoritmi, kas pieturas pie tūlītējas atjaunināšanas idejas, atjaunina skatu uzreiz, kad mainās dati bāzes relācijā avotā. Bet ar atliktu atjaunināšanas pieeju datu noliktavas skati tiek atjaunināti kopīgi noteiktā laikā, un izmaiņas tiek izplatītas kā pakete. Atlikta atjaunošana var izpildīties periodiski vai pēc pieprasījuma, kad lietotājs izpilda vaicājumu skatam.

Saistībā ar skata uzturēšanu datu noliktavā eksistē divas problēmas: (a) izmaiņu pārnešana no datu avota uz datu noliktavu un (b) skata atjaunināšana saskaņā ar šīm izmaiņām [SMR+98]. Lai saglabātu datu atbilstību datu noliktavā, piedāvā veikt sekojošas darbības:

- Noskaidrot izmaiņas, kas notika datu avotos un pārnest šo izmaiņu rezultātus uz datu noliktavu.
- Izrēķināt nepieciešamās darbības, kas jāveic, lai atjauninātu skatu saskaņā ar šīm izmaiņām datu avotos.
- Izplatīt šo atjaunināšanu uz datu noliktavu tādā secībā, kā šīs izmaiņas notika datu avotā.

Skatu paš-uzturamība

Skatu sauc par *paš-uzturamu*, ja tā atjaunināšanai ir nepieciešami tikai pats materializētais skats un izmaiņas, kas notika ar relāciju, kas izmantota skata definīcijā [GM95]. Bet vairākos gadījumos paš-uzturēšana nav iespējama, jo skata atjaunināšanai var būt nepieciešama arī papildus informācija par citām relācijām, kas definē skatu, kas var

atrasties vienā vai vairākos datu avotos. Tādēļ, ka vairākumu gadījumos datu avoti ir atsevišķas sistēmas, kas atdalītas no datu noliktavas, skatu atjaunināšana var prasīt kādu noteiktu laiku, kurā var notikt jaunas izmaiņas ar datu avotu. Arī nejaušas izmaiņas datu avotos var radīt nesaskaņas datu noliktavā. Daži datu avoti var neatbalstīt pilnu datu bāzes funkcionalitāti, un pieprasījumi uz šādiem datu avotiem, kas ir vajadzīgi, lai aprēķinātu skata atjaunināšanu, var būt grūti un pat neiespējami. Šo šķēršļu dēļ, materializētam skatam jābūt uzturamam bez piekļuves pie bāzes relācijām.

Ir vairākas pieejas šī skatu paš-uzturēšanas nodrošināšanai [GM95]. Viena no tām rekomendē nokopēt visus bāzes datus datu noliktavā, lai materializēto skatu atjaunināšana būtu iespējama lokāli datu noliktavā. Bet šai pieejai ir vairāki trūkumi. Jo vairāk un vairāk datu tiek pievienoti datu noliktavai, tas palielina apjomu un izraisa informācijas dublēšanu, kas atkal var novest pie nesaskaņām. Šī pieeja neņem vērā iespēju, ka bāzes relāciju korteži var jau būt pašā materializētajā skatā. Šajā gadījumā skata paš-uzturēšanai būs nepieciešami tikai pats skats, bāzes relācijas izmaiņas un kaut kāda bāzes relāciju apakškopa nevis visas relācijas.

Cita pieeja ir aprakstīta rakstā [QGM+96], kur ir piedāvāts algoritms, kas dotam skatam V nosaka palīgskatu kopu A, kas, apstrādājot kopā skatu V un kopu A, padara tos par paš-uzturamiem. Rakstā [Huy97] apskata materializēta skata uzturēšanu, kas ņem vērā arī citus skatus, kas ir materializēti datubāzē. Abās pieejās mēģina atrast minimālo skatu kombināciju datu apjoma un rēķināšanas izmaksu izteiksmē.

Ne vienmēr ir izdevīgi nodrošināt skatu paš-uzturēšanu datu noliktavās [Wid95]. Ja izmaksas paš-uzturēšanas nodrošināšanai ir lielākas par izmaksām, kas ir saistītas ar piekļuvi pie datu avotiem skatu atjaunināšanas izrēķināšanai, izdevīgāk ir atļauties piekļuvi pie avotiem.

Atjaunināšanas filtrēšana

Skaidrs, ka izmaiņas, kas notiek datu noliktavas datu avotos, jāatspoguļo datu noliktavā. Dažas no šīm izmaiņām izraisa skatu atjaunināšanu, ko vajag izplatīt datu noliktavā, bet dažas izmaiņas neietekmē skatus datu noliktavā [SMR+98]. Ja datu avotos būtu iespējams noteikt, ka dotās izmaiņas neizraisīs skatu atjaunināšanu, šīs izmaiņas nevajadzētu izplatīt datu noliktavā. Visas iespējamas izmaiņas, kas neietekmē skatus, tiek atfiltrētas datu avotos, un tikai izmaiņas, kas rezultātā izraisa skatu atjaunināšanu tiek izplatītas datu noliktavā. Atjaunināšanas filtrēšana samazina uzturēšanas transakciju apjomu datu noliktavā, kas savukārt samazina laiku, kas ir nepieciešams, lai padarītu datu noliktavu saskanīgu ar datu avotiem. Lai panāktu atjaunināšanas filtrēšanu, vajag padarīt datu avotus un procedūras, kas veic atjaunināšanu, efektīvākus. Piemēram, ir dažādas metodes, kas atļauj pārbaudīt dalītus integritātes ierobežojumus datu avotu pusē, kad tur notiek izmaiņas [Wid95]. Lai šīs metodes varētu pielietot, lai noskaidrotu izmaiņas, kas ietekmē datu noliktavu, datu avotiem tādā gadījumā jāvar sazināties ar datu noliktavu. Tas izraisa jaunas problēmas, piemēram, ja notiek izmaiņas kāda datu avota vai datu noliktavas shēmā, tad visām dalībpusēm jābūt informētām par šīm izmaiņām, lai tās varētu modificēt ierobežojumu kopu saskaņā ar izmaiņām. Šajā gadījumā materializēto skatu uzturēšanas stratēģijas būtu atkarīgas no ierobežojumu kopas, un

jebkuras izmaiņas šajā kopā rezultātā ģenerētu izmaiņas eksistējušā materializētā skata uzturēšanas procedūrā.

Laiksakritības vadība

Datu noliktavā var parādīties nesaskaņas, jo izmaiņas datu avotos ir nejaušas un dinamiskas [ZGH+95]. Kad datu noliktava saņem informāciju par izmaiņām datu avotā, var gadīties, ka ir vajadzīga papildus informācija skata atjaunināšanai. Tādēļ datu noliktava uzdod vaicājumu pēc papildus informācijas datu avotam. Šie vaicājumi var izpildīties datu avotos vēlāk, nekā vēl jaunas izmaiņas avota datos, tādēļ avota stāvoklis var būt jau cits. Šis process var novest pie nepareizi izrēķinātiem skatiem, t.i. pie anomālijām.

Eksistē vairāki algoritmi, kas ļauj izvairīties no anomālijām, piemēram, pilna skata izrēķināšana, kad notiek izmaiņas, vai datu avotu datu kopēšana datu noliktavā. Pirmais variants prasa daudz laika un resursu, bet otrais prasa daudz resursu, jo vajag glabāt visu avota relāciju datus un atjaunināt tos saskaņā ar izmaiņām avotos. Citi algoritmi tika izstrādāti [ZGH+95], [ZGW96], [ZGW98], [AAS+97], kam ir labākas veiktspējas rezultāti, nekā pilnas pārrēķināšanas un datu kopēšanas algoritmiem. Atkarībā no veida, kā atjaunināšana tiek izplatīta skatā datu noliktavā, skatu uzturēšanas algoritmus var klasificēt pēc īpašībām, kas tiem piemīt [ZGH+95] (tālākajā aprakstā skats vai datu avots maina savu stāvokli pēc katras atjaunināšanas vai izmaiņas):

- *Konverģence* nozīmē, ka pēc pēdējās atjaunināšanas operācijas un kad visas darbības ir beigušās, skats atbilst relācijām datu avotos.
- *Vājš nepretrunīgums* nozīmē, ka katrs skata stāvoklis atbilst kaut kādam derīgam avotu relāciju stāvoklim.
- *Stiprs nepretrunīgums* nozīmē, ka katrs skata stāvoklis atbilst kaut kādam derīgam avota relāciju stāvoklim, pie tam stāvokļu kārtība ir līdzīga avotu stāvokļu kārtībai un beigās datu noliktava atbilst avotu stāvoklim.
- *Pilnība* nozīmē, ka katrs datu avota stāvoklis ir atspoguļots kā atsevišķs datu noliktavas stāvoklis un stāvokļu kārtība datu noliktavā atbilst stāvokļu kārtībai avotos.

Rakstā [ZGH+95] tiek piedāvāts algoritms ECA, kas nodrošina stipra nepretrunīguma īpašību. Algoritma galvenā ideja ir pievienot papildus informāciju vaicājumam, ko datu noliktava uzdod datu avotam, kompensācijas vaicājumu, lai izvairītos no anomālijām, kas var rasties laiksakritīgas atjaunināšanas dēļ. Kompensācijas vaicājums pārveido vaicājumu, ko datu noliktava sūta datu avotam, tā, lai tas atlasītu precīzi to informāciju, kas bija datu avotā pirms laiksakritīgas avota atjaunināšanas. Šī algoritma trūkumi ir tas, ka tas darbojas uz viena skata un viena datu avota (bet uz vairākām relācijām), skatu definīcija ir ierobežota uz Select-Project-Join skatiem, atjaunināšanas operācijas ir apstrādātas pa vienai, nevis kā operāciju kopas.

Rakstā [ZGW98] ir piedāvāta Strobe algoritmu kopa, kas nodrošina laiksakritīgas atjaunināšanas uzturēšanu materializētiem Select-Project-Join skatiem ar vairākiem datu avotiem. Šie algoritmi ir pielietojami dažādos transakciju scenārijos. Rakstā definē trīs transakciju scenārijus:

- *Vienas atjaunināšanas transakcija* nozīmē, ka katra atjaunināšana veido vienu transakciju un katra šī transakcija tiek nosūtīta uz datu noliktavu.
- *Avota-lokāla transakcija* nozīmē, ka atjaunināšanas operāciju virkne tiek uzskatīta par vienu transakciju, ko nosūta datu noliktavai.
- *Globāla transakcija* nozīmē, ka atjaunināšanas operācijas ar vairākiem avotiem notiek vienas globālas transakcijas ietvaros, un šīs transakcijas rezultāti tiek sūtīti datu noliktavai.

Sākumā rakstā [ZGW98] ir aprakstīti algoritmi visiem scenārijiem, kas nodrošina stipra nepretrunīguma īpašību, bet pēc tam tiek piedāvāts algoritms C-Strobe, kas nodrošina pilnības īpašību. Strobe algoritma ideja ir sekojoša. Kad datu noliktava saņem informāciju par atjaunināšanu datu avotos, tā nepielieto atjaunināšanu uzreiz, bet ģenerē darbību sarakstu, ko vajag izpildīt ar skatu. Datu noliktava izpilda šīs darbības tikai tad, kad vairs nav neapstrādātu vaicājumu un visas saņemtās atjaunināšanas tika apstrādātas. Šeit dzēšanas atjaunināšanas tiek izpildītas uzreiz datu noliktavā, bet iespraušanas atjaunināšanas gadījumā iesaistītajiem datu avotiem tiek sūtīti kompensācijas vaicājumi (līdzīgi ECA algoritmam), bet laiksakritības vadība ir nodrošināta ar to, ka algoritms izmet visus dublējamos kortežus. Tas ir iespējams, jo skati šeit satur bāzes relāciju primārās atslēgas. Šo algoritmu problēma ir tā, ka ja avotos notiek nepārtrauktas atjaunināšanas, tad algoritms var nekad nebeigties. Šī problēma ir atrisināta C-Strobe algoritmā, kur laiksakritīgu atjaunināšanu rezultāti, kas notiek laikā, kad datu noliktavas kompensācijas vaicājums izpildās, tiek izņemti, t.i. tiek sasniegts datu avotu stāvoklis pirms laiksakritīgas atjaunināšanas.

Rakstā [AAS+97] ir piedāvāts SWEEP algoritms, kas nodrošina stipra nepretrunīguma īpašību. Tas izmanto līdzīgu pieeju, kā C-Strobe algoritms, tikai SWEEP algoritmam ir mazāks vaicājumu skaits, ko datu noliktava sūta datu avotiem. Tas ir panākts ar to, ka vairāki kompensācijas vaicājumi vienam datu avotam ir grupēti, t.i. vienam atjauninājumam vienam datu avotam kompensāciju vaicājumu skaits var būt ne vairāk, ka datu avotu skaits.

Laicīgu materializētu skatu uzturēšana

Darbā [Yan01] autori apskata *laicīgu* (temporal) materializēto skatu uzturēšanas problēmu. Darbā atzīmē, ka datu noliktavas lietotāji bieži ir interesēti ne tikai tekošā informācijā, bet arī analizē vēsturi. Laicīgi materializēti skati satur vēsturiskus datu avotu stāvokļus, t.i. katrs laicīgā skata ieraksts ir papildināts ar laikspiedolu, kas apzīmē laika brīdi, kad šīs ieraksts bija datu avotos. Darbā [Yan01] arī ir definēta vaicājumu valoda šāda tipa skatiem, kas atbalsta starpības, apvienojuma, projekcijas, atlases un savienojuma operācijas.

Laicīgu materializētu skatu atjaunināšanas algoritms, saņemot no datu avota izmaiņas, pārveido šīs izmaiņas skata atjaunināšanas operācijā, papildinot tās ar laikspiedoliem, un izplata šīs izmaiņas datu noliktavas skatos. Laicīgus skatus var būt nepieciešams atjaunināt arī, kad datu avotos nenotiek nekādas izmaiņas, bet kad vienkārši mainās kāds ar laiku saistīts nosacījums. Piemēram, ja viens skats satur cilvēkus, kas ir vecāki par 20 gadiem, tad šo skatu vajadzēs papildināt ar jauniem ierakstiem, kas atbilst cilvēkiem, kas sasniedz 20 gadu vecumu. Šāda tipa materializētus skatus atjauno tikai tad, kad lietotājs izpilda vaicājumu uz šo skatu.

Rakstā arī ir aplūkota skatu paš-uzturamības problēma dažāda veida laicīgiem materializētiem skatiem. Ir aprakstītas metodes palīgskatu noteikšanai Select-Project-Join skatiem. Palīgskati ir skati, kas satur papildus informāciju (kas nav iekļauta pašas datu noliktavas materializētajos skatos), kas ir vajadzīga, lai padarītu datu noliktavas skatus par paš-uzturamiem.

3.3.2. Daudzdimensiju datu noliktavas uzturēšana

Algoritmi materializēto skatu atjaunošanai ir pielietojami arī datu noliktavas kontekstā, kad datu noliktavu modelē ar daudzdimensiju pieeju. Tos var pielietot, piemēram, uzskatot katru datu noliktavas objektu (faktu vai dimensiju tabulu) par materializēto skatu uz avota datiem.

Tomēr, šie visi algoritmi ir stipri ierobežoti. Tie tiešā veidā darbojas pamatā tikai ar relāciju datu bāzes datu avotiem. Turklāt datu atjaunināšanas process datu noliktavās sastāv no vairākiem soļiem, piemēram, datu iegūšanas, pārveidošanas, integrēšanas un attīrīšanas, jaunu datu atvasināšanas, vēstures būvēšanas un datu ielādes, bet materializētu skatu uzturēšana atspoguļo tikai datu ielādes daļu.

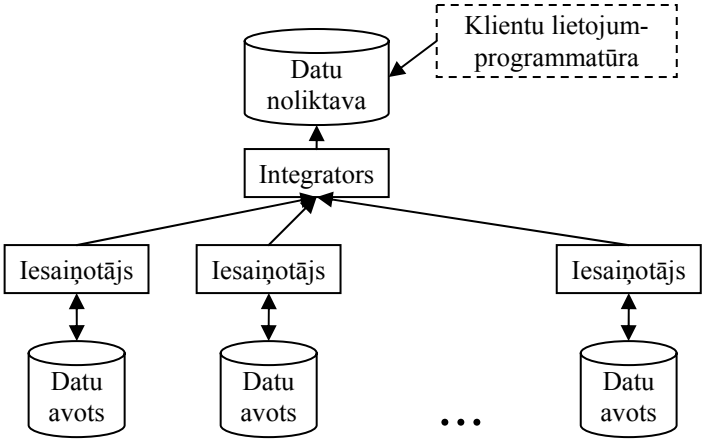
Daži raksti ir veltīti vispārīgam datu noliktavas atjaunināšanas procesam. Piemēram, rakstā [BFM99] atjaunināšanu atspoguļo kā darbplūsmas procesu. Šīs process sastāv no neatkarīgu aktivitāšu kopas (iegūšana, attīrīšana, integrācija, utt.), kuras var organizēt dažādos veidos. Procesu sadala četros soļos. *Sagatavošanas* solis no katra datu avota iegūst datus, kas atbilst izmaiņām, kas ir notikušas kopš pēdējas datu iegūšanas, un attīra šos datus un, ja nepieciešams, arhivē pirms integrācijas. *Integrācijas* solis saskaņo avotu izmaiņas un pievieno tās datu noliktavai. *Agregācijas* solis pārrēķina inkrementāli kopsavilkumus, lietojot šīs izmaiņas. *Pielāgošanas* solis izplata mainītos datus datuvēs. Tiek izdalīti daži notikumu tipi, kas var izraisīt un sinhronizēt atjaunināšanas aktivitātes. Tie varētu būt pārejas notikumi, beigu notikumi vai lietotāja definēti notikumi. Atbilstošus notikumus un aktivitātes izvēlas lietotājs atkarībā no situācijas. Datu avotiem var būt dažādi stāvokļi, kas nosaka, vai notikumu izraisītas aktivitātes izpildās uzreiz, vai gaida. Šie stāvokļi ir aktivitāšu nosacījumi. Arī atjaunināšanas procesu var modelēt nedaudz citādi, uzskatot datu izmaiņas par notikumiem. Atjaunināšanas procesam ir izpildītāji (aģenti): tie var būt cilvēki, kas definē prasības, ierobežojumus un stratēģijas, kā arī datori, kas izpilda aktivitātes.

Rakstā [VGD99] definē un izpilda datu noliktavas atjaunināšanas procesu, pamatojoties uz specifikāciju, kas glabājas objektorientētā metadatu repozitorijā. Pārsvarā šajā repozitorijā glabājas informācija par datu noliktavas shēmu, kā, piemēram, faktu vai dimensiju tabulu atribūti un to atvasinājumi, un tās avotiem, kā, piemērām, avotu atribūti. Starp atribūtiem datu avotos un datu noliktavas atribūtiem eksistē trīs veidu transformācijas. *Struktūras kartēšana* definē, kādi datu avotu atribūti tiek izmantoti datu noliktavas atribūtu atjaunināšanai, piemēram, 1:1, kad atbilstība ir tieša, 1:n, kad no viena avota atribūta iegūst n datu noliktavas atribūtus, un n:1, kad vairāki datu avotu atribūti ir apvienoti vienā datu noliktavas atribūtā. *Vertikāla kartēšana* definē vairāku datu avotu atribūtu atbilstību datu noliktavas atribūtiem. Tiek izmantotas apvienošanas, šķēluma, starpības, filtrēšanas un prioritātes noteikšanas operācijas. *Attīrīšana* definē izmantotās datu attīrīšanas metodes.

Tālākajās apakšnodaļās tiek apskatītas metodes dažu specifisku datu noliktavas uzturēšanas problēmu risināšanai, ko var pielietot kopā ar kādu vispārīgu datu noliktavas uzturēšanas risinājumu.

Uzturēšanas starpnieki

Vairākums materializēto skatu uzturēšanas algoritmu izmanto modeli, kur datu avoti paziņo datu noliktavai par notikušajām izmaiņām, bet realitātē tas bieži nav iespējams, jo avoti ir autonomi un atdalīti no datu noliktavas, tiem var būt dažāda aparatūra, operētājsistēmas, datu bāzes pārvaldības sistēmas, shēmas, u tml. Tāpēc bieži izmanto tā sauktos *starpniekus (mediators)* (att. 5.)*: iesaiņotājus* (wrappers), kas nodrošina saskarni starp datu noliktavu un datu avotiem, kā arī atbild par datu transformāciju pirms datu integrācijas, un *integratorus*, kas veic datu integrāciju. Iesaiņotāju funkcijas iekļauj datu savākšanu no datu avotiem, datu attīrīšanu, pārveidošanu līdz vajadzīgai detalizācijai un formatēšanu. Integratori atbild par nesaskaņu likvidēšanu, kad ieraksti vienā datu avotā neatbilst visiem tiem pašiem datiem citā datu avotā, agregāciju līdz vajadzīgai detalizācijas pakāpei, dublējamas informācijas likvidēšanu utt. [Wie98]. Datu noliktavās procesus, kurus veic starpnieki, sauc par datu iegūšanas, pārveidošanas un ielādes procesiem vai ETL procesiem.



Att. 5. Datu noliktavas arhitektūra ar starpniekiem

Sakarā ar iesaiņotājiem, datu noliktavā rodas jaunas problēmas, piemēram, kā iegūt izmaiņas datu avotos, kā šīs izmaiņas var izplatīt datu noliktavā, kā integrēt informāciju no vairākiem avotiem un nodrošināt jaunākas informācijas piegādi datu noliktavai. Šīs problēmas, kā arī starpnieku pielietošana, ir īpaši aktuālas vidēs, kur datu avoti ir nestrukturēti vai tikai daļēji strukturēti. Vairāki pētījumi ir veltīti šo problēmu risinājumiem. Piemēram, [CAW98] piedāvā modeli datu izmaiņu reprezentācijai daļēji strukturētos datos un valodu šo izmaiņu un datu vaicājumiem.

[BCV99] piedāvā pieeju vairāku, heterogēnu informācijas avotu, kas satur strukturētus un daļēji strukturētus datus, integrācijai, kas balstās uz starpnieku pielietošanu. Šie starpnieki darbojas uz informācijas avotu metadatu pamata un satur sekojošus arhitektūras elementus: vispārīgs objektorientēts datu modelis, kas apraksta avota sistēmas integrācijas procesu, viens vai vairāki iesaiņotāji, integrators un vaicājumu apstrādes komponents.

Rakstā [CAM01] izpēta izmaiņu atrašanas problēmu XML dokumentos [W3Cb], kurus arī uzskata par daļēji strukturētu formātu. Autori piedāvā algoritmu, kas atrod starpības starp diviem XML dokumentiem un aprēķina deltu, kas attēlo šīs starpības. Delta ir definēta kā sekojošo elementāru operāciju kopa: apakškoku dzēšana, apakškoku iespraušana, teksta mezgla vai atribūta vērtības maiņa, mezgla vai apakškoka daļas pārnešana. Piedāvātais algoritms mēģina atrast (lielus) apakškokus, kas palika nemainīgi starp jauno un veco XML dokumenta versiju, tad algoritms cenšas saskaņot atlikušo apakškoku priekšteču un pēcteču mezglus. Šīs algoritms arī ņem vērā identifikācijas atribūtus, lai saskaņotu elementus. Apskatītais algoritms praksē tiek izmantots XML datu noliktavas projekta ietvaros.

Tīmekļa datu avoti

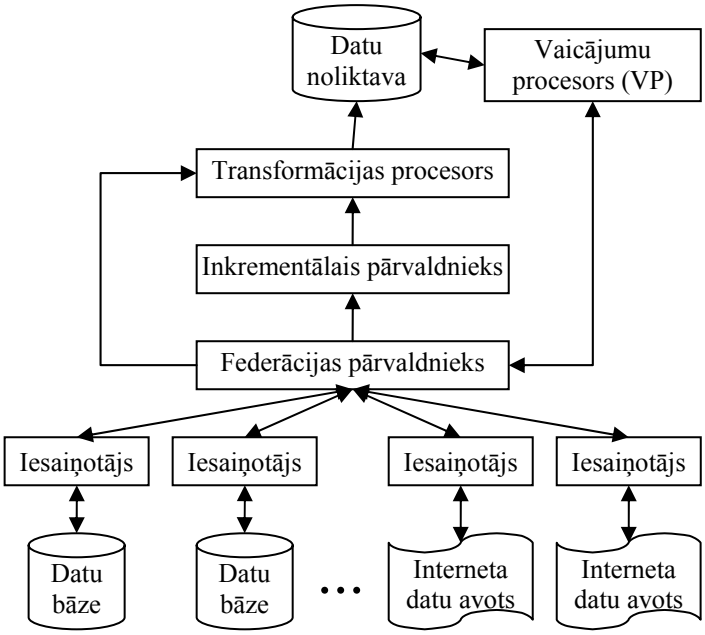
Mūsdienās arvien vairāk datu noliktavu izmanto informāciju no datu avotiem, kas atrodas tīmeklī. Interneta datu avotu specifika ir tāda, ka to formāti nav strukturēti un šie avoti ir ārkārtīgi mainīgi. Šeit datu iegūšanas problēma ir ļoti aktuāla, jo pirmkārt šiem datu avotiem parasti neeksistē standarta saskarnes vaicājumiem, otrkārt vaicājuma rezultāti nav strukturēti, treškārt ne vienmēr ir pieejama informācija par datu avota struktūru.

Sakarā ar šīm problēmām, var izdalīt dažādas pieejas to risināšanai. Piemēram, vairāki pētījumi ir veltīti vaicājumu valodai, kas var iegūt informāciju no interneta avotiem, piemēram, HTML formāta dokumentiem, un pārstrukturēt šo informāciju vajadzīgā veidā [KS95], [LSS96], [AMM98], [ML01]. Datu iegūšanai no XML formāta datu avotiem var veikt ar XQuery vaicājumu valodas palīdzību [W3Cc]. Citi pētījumi ir veltīti iesaiņotāju ģenerācijai interneta datu avotiem [GRV+98], [CMM01].

Metodes, kas ir aprakstītas šajos pētījumos, ir vispārīgas un tos var drīzāk pielietot informācijas meklēšanai internetā, nevis datu noliktavas kontekstā. Tie visi neņem vērā datu noliktavas specifiku, piemēram, tādu īpašību, ka dati tiks integrēti ar datiem no citiem datu avotiem.

Raksts [Zhu99] piedāvā vienotu datu noliktavas ietvaru, kas atbalsta vaicājumus uz interneta datu avotiem un citiem datu noliktavas avotiem. Vienkāršots šis ietvars ir attēlots attēlā 6. Šajā darbā ir izmantota hibrīda pieeja, kad bieži izmantotie dati tiek iegūti no interneta un ielādēti datu noliktavā, bet sistēma arī ir spējīga dot atbildes uz vaicājumiem, kas izmanto interneta datus, kas nav materializēti datu noliktavā. Interneta datu reprezentācijai tiek izmantots MIX modelis (Metadata based Integration model for data X-change), kas satur gan datu aprakstus, gan pašus datus. Aprakstītās datu noliktavas struktūras pamatā ir federācijas pārvaldnieks (federation manager), kas pārvalda vairākus iesaiņotājus. Iesaiņotāji nodrošina saskarni ar datu avotiem, iegūst datus no datu avotiem un pārveido MIX modelī. Arī datu noliktavas struktūrā (skat. attēlu 6.) ir transformācijas procesors, kas pārveido datus, kas tiek iegūti no MIX modeļa (datu transformācijas ir aprakstītas [ZB01]), un ielādē šos datus noliktavā. Datu atjaunināšanu datu avotos izseko inkrementālais pārvaldnieks (incremental manager), kas salīdzina datus, ko iegūst un integrē federācijas pārvaldnieks, ar datiem no MIX modeļa un aprēķina izmaiņas, kuras tiek padotas transformācijas procesoram. Vaicājumu interfeisu nodrošina vaicājumu procesors, kurš lieto datus no datu noliktavas un arī var iegūt datus no datu avotiem, kas nav materializēti datu noliktavā, caur federācijas pārvaldnieku.

Visos šajos pētījumos ir apskatīti tikai gadījumi, kad nepieciešamā informācija atrodas tīmekļa lappusēs HTML vai XML formātā. Bet interneta datu noliktavas gadījumā dati varētu tikt iegūti arī no tīmekļa servera žurnāla failiem, piemēram, klikšķu plūsmas analīzei [KM00]. Žurnāla failus automātiski ģenerē visi populārie tīmekļa serveri. Žurnāla fails satur pa vienam ierakstam katrai HTTP transakcijai, kas nodrošina piemērotu detalizācijas pakāpi lietotāju aktivitātes analīzei. Parasti žurnāla faili atbilst CLF (Common Log File) formātam [W3Ca], t.i. katrs žurnāla faila ieraksts satur lietotāja IP adresi (vai domēna nosaukumu), lietotāja identifikatoru (parasti netiek izmantots), lietotāja vārdu (kura arī var nebūt), datumu un laiku, kad notika aktivitāte, HTTP pieprasījumu, kas sastāv no HTTP metodes, vienotā resursu identifikatora (URI) un HTTP protokola versijas), statusa kodu, kas rāda vai aktivitāte izpildījās veiksmīgi, pārsūtīto baitu skaitu un laiku sekundēs, kurā HTTP pieprasījums tika apstrādāts.



Att. 6. Datu noliktavas struktūra [Zhu99]

Galvenais žurnāla failu trūkums ir tas, ka tie nav domāti datu ielādei datu noliktavās. Galvenokārt tie ir vajadzīgi tīmekļa servera darbības novērošanai un atkļūdošanai un lietotāju darbību analīzei. Žurnāla failu izmantošana kā datu avots datu noliktavai izraisa vairākas problēmas, piemēram, sesiju un lietotāju identifikācija, datu integrācija ar citiem datu avotiem, dinamiskā tīmekļa lappušu ģenerēšana, dublējamās informācijas esamība, u.c. [LB03]. Šīs problēmas vajag risināt kā datu noliktavas ETL procesu apakšuzdevumus.

3.3.3. Datu noliktavas uzturēšanas pieeju analīze

Datu noliktavas uzturēšanas problēma ir sen zināma un pētīta problēma, un eksistē daudz metožu tās risināšanai gan materializētiem skatiem, gan datu noliktavām, kas ir būvētas ar daudzdimensiju shēmām. Materializēto skatu uzturēšanas pieejas satur ne tikai risinājumus pašu skatu datu atjaunināšanai saskaņā ar izmaiņām bāzes relācijās, bet arī risinājumus, kas spēj noteikt, vai datu avotā notikušas izmaiņas ietekmē skatu, kas padara skatu par paš-

uzturamo, kas atbalsta vienlaicīgu materializēta skata atjaunināšanu un avota relāciju datu atjaunināšanu, un risinājumus laicīgu materializētu skatu uzturēšanai. Parasti materializēta skata uzturēšanas process sastāv no datu avotu izmaiņu, kas ietekmē skatu, identificēšanas, datu starpību starp skata saturu un datu avota saturu noteikšanas un skata datu atjaunināšanas operāciju izpildīšanas.

Daudzdimensiju datu noliktavas uzturēšana parasti tiek veikta ar datu iegūšanas, pārveidošanas un ielādes (ETL) procesiem. Vienkāršākajā situācijā šie procesi var būt realizēti kā datu ielādes procedūras, kas atjaunina datus katrā datu noliktavas tabulā. Bet, lai atvieglotu ETL procesu izstrādi un mainīšanu, daži risinājumi izmanto metadatus, kas apraksta gan datu noliktavas un datu avotu shēmu, gan ETL procesa daļas kā iepriekš definētu veidu transformācijas. Šajā apakšnodaļā 3.3. apskatītie risinājumi daudzdimensiju shēmām ir paredzēti vairāku specifisku problēmu risināšanai, piemēram, saziņai ar datu avotiem un datu iegūšanai no datu avotiem, ar ko nodarbojas uzturēšanas starpnieki.

3.4. Datu noliktavas pielāgošanas problēma

Kad datu noliktavas jau ir izstrādāta un ir izveidoti atbilstoši uzturēšanas procesi, lietotāji izmanto datu noliktavas datu analīzē. Bet lietotāju darījumprasības var mainīties laikā, tāpēc kopā ar tām var arī mainīties datu noliktavas shēma, jo bieži ir grūti iepriekš noteikt datu noliktavas shēmas, kas apmierinātu visu datu noliktavas lietotāju darījumprasības. Šeit rodas svarīgs jautājums, kā uzturēt datu noliktavu, kad mainās tās shēma, vai var izvairīties no pilnas datu noliktavas datu pārrēķināšanas, jo tā var prasīt daudz laika un resursu. Šo problēmu sauc par datu noliktavas *pielāgošanas* problēmu. Datu noliktavas pielāgošanas problēma atšķiras no uzturēšanas problēmas ar to, ka to raksturo sekojošas koordinātes (att. 4.): (1) šo procesu izraisa tiešā datu noliktavas shēmas maiņa, un (2) datu avotos nenotiek izmaiņas (Izmaiņas datu avotu datos principā neietekmē pielāgošanas problēmu, jo tiek risinātas ar uzturēšanu). Pārējas koordinātes ir vienādas ar uzturēšanas problēmas koordinātēm: (3) datu noliktavas lietotāji vēlas, lai datu noliktavā vienmēr būtu pieejama jaunākā informācija, un (4) datu avotos nesaglabājas vecais datu stāvoklis.

Tālākajās apakšnodaļās tiek aprakstītas pieejas datu noliktavas pielāgošanas problēmas risināšanai datu noliktavās. Tiek apskatīti abi datu noliktavas reprezentācijas gadījumi: gan kā materializēto skatu kopums uz avotiem, gan kā daudzdimensiju datu modeļi.

3.4.1. Materializēto skatu pielāgošana

Rakstos [MD96], [Moh97], [GMR95] un [GMR+01] apskata problēmu, kā nodrošināt jaunāku informāciju skatā, kad mainās skata definīcija, t.i. reaģējot uz skata pārdefinēšanu. Skata pārdefinēšanas procesu var uzskatīt par primitīvu izmaiņu virkni [GMR95]. Pielāgošana ir izteikta kā papildus vaicājums uz veco skatu vai veco skatu un bāzes relācijām, ko vajag izpildīt, lai pielāgotu skatu atbilstoši pārdefinēšanai.

Iespējamās primitīvās izmaiņas, kas var būt skata definīcijā, ir atribūta pievienošana vai dzēšana, agregācijas funkcijas pievienošana vai dzēšana, DISTINCT operatora pievienošana vai dzēšana, UNION vai EXCEPT operanda pievienošana vai dzēšana,

predikāta pievienošana, dzēšana vai modificēšana dažādās vaicājuma daļās (piemēram, SELECT daļā vai WHERE daļā).

Primitīvas izmaiņas var notikt jebkurā materializētā skata definīcijas daļā. Rakstos [GMR95] un [GMR+01] tiek apskatītas visas iespējamās izmaiņas dažāda veida materializēto skatu definīcijas daļās. Katrai primitīvai izmaiņai katrā materializētā skata definīcijas daļā tika definētas nepieciešamās darbības, kas jāveic, lai pielāgotu materializētā skata datus jaunajai definīcijai. Tika aprakstīti arī izdarītie pieņēmumi un tika identificēta visa nepieciešamā informācija, lai varētu veikt pielāgošanas procesu, piemēram, pielāgot materializēto skatu dažreiz var palīdzēt tāda informācija kā bāzes relāciju atslēgas, atribūti, kas ir iekļauti izvēles nosacījumos, katra korteža atbilstošo avota kortežu skaits, agregācijas funkcijas papildus tām, kas ir iekļautas materializētā skata definīcijā, vai identifikatori, kas norāda uz apakšvaicājumu UNION materializētajā skatā, no kurienes tika iegūts katrs kortežs. Rakstos arī identificētas izmaiņas materializēto skatu definīcijā, kuras nevar pielāgot, piemēram, kad palielinās materializētā skata detalizācijas pakāpe, to nevar pielāgot, ja jauna kolonna, kas palielina detalizāciju, nav funkcionāli izsakāma no citiem atribūtiem. Agregācijas skatiem ir aprakstīts, kā var izrēķināt agregācijas funkcijas mainītai materializētā skata definīcijai, lietojot šo funkciju rezultātus materializētā skata definīcijai, kas eksistēja pirms izmaiņas, un kad tas nav iespējams. Rakstos arī atzīmēts, ka vairākas primitīvas izmaiņas, kas seko viena aiz otras, var apvienot vienā pielāgošanas vaicājumā vai atjaunošanas darbībā, ka nav nepieciešams rēķināt starprezultātus pēc katras izmaiņas.

Rakstos [MD96] un [Moh97] ir apskatīta materializētu skatu pielāgošanas problēma datu noliktavās. Rakstā [MD96] Select-Project-Join materializētu skatu pielāgošana ir apskatīta gadījumos, kad notiek izmaiņas skata SELECT, FROM un WHERE daļās. Šie algoritmi pārsvarā neatšķiras no algoritmiem [GMR95] un [GMR+01], bet dažās situācijas tie tiek paplašināti, piemēram, gadījumiem, kad skats nesatur bāzes relāciju atslēgas. Šo algoritmu galvenā ideja ir pievienot 'savienojumu-skaita' atribūtu bāzes relācijām un 'iegūšanas-skaita' atribūtu katram skatam. 'Savienojumu-skaits' apzīmē, cik dažādos veidos relācijas kortežs ir savienojams ar citu relāciju kortežiem. Šo atribūtu piedāvāts glabāt visiem relāciju savienojumiem, kas var piedalīties skatu definīcijās. 'Iegūšanas-skaits' apzīmē, cik dažādos veidos var iegūt skata kortežu no bāzes relāciju kortežiem.

Rakstā [Moh97] pielāgošanas iespējas ir paplašinātas, papildus vienkāršiem SELECT-FROM-WHERE materializētiem skatiem, aplūkojot arī vispārīgākus materializētus skatus, kas ir definēti ar relācijas algebras savienojuma, apvienošanas, starpības, projekcijas un atlases operatoriem. Šeit materializētu skatu reprezentē kā izteiksmes koku. Izteiksmes koks ir binārs koks, kur lapas attēlo bāzes relācijas, kas ir izmantotas skata definīcijā, un ne-lapas mezgli satur binārus relācijas algebras operatorus, piemēram, savienojumu, apvienošanu un starpību. Katram ne-lapas mezglam ir divi bērni – binära relāciju algebras operatora operandi. Unāras operācijas, tādas kā atlase un projekcija, ir attēlotas kā šķautnes. Papildus izmaiņām skatu definīcijās, kas ir aprakstītas rakstā [MD96], šeit vēl ir aplūkotas tādas izmaiņas, kā binära mezgla pievienošana vai dzēšana, binära operatora pievienošana vai dzēšana. Šeit izdarītie pieņēmumi un papildus informācija, kas tiek glabāta, lai varētu skatu pielāgot, ir līdzīga kā [MD96].

Pieeju, kas aprakstīta šajos rakstos [MD96] un [Moh97], ne vienmēr var pielietot datu noliktavas vidē, jo ne vienmēr ir iespējams modificēt datu avotus, pievienot tiem jaunus atribūtus. Arī rakstos nav apskatītas visas iespējamās izmaiņas materializētu skatu definīcijā, tādas kā bāzes tabulu kolonnu pievienošana vai dzēšana no SELECT teikuma (detalizācijas pakāpes maiņa), DISTINCT operatora pievienošana vai dzēšana u.c.

3.4.2. Daudzdimensiju datu noliktavas pielāgošana

Tāpat kā ar uzturēšanas problēmu risināšanas metodēm, iepriekš aprakstītās metodes materializēto skatu pielāgošanai var izmantot arī vienkāršākām datu noliktavām, kas ir realizētas kā daudzdimensiju shēmas. Tomēr šīs metodes neņem vērā datu noliktavas daudzdimensiju modeļa aspektus, kas ierobežo atbalstītas izmaiņas datu noliktavā. Piemēram, nav apskatītas hierarhijas un hierarhiju līmeņi, kuros arī var notikt izmaiņas, kas ietekmē hierarhiju struktūru. Sekojošās apakšnodaļās ir aplūkotas dažādas pieejas datu noliktavas daudzdimensiju shēmas izmaiņu apstrādei, t.i. daudzdimensiju datu noliktavas pielāgošanas problēmu risināšanai.

Satura izmaiņas dimensijās

Datu noliktavu jomā parasti ir pieņemts, ka fakti ir dinamiski, bet dimensijas ir relatīvi statiskas, taču praksē tas ne vienmēr ir pareizs apgalvojums, jo izmaiņas dimensijās var notikt daudzu iemeslu dēļ, piemēram, prasību maiņas dēļ [HMV99]. Satura izmaiņas dimensijās var ietekmēt dimensijas hierarhiju līmeņu struktūru, tāpēc šīs izmaiņas var uzskatīt par pielāgošanas problēmām.

Dimensiju satura atjaunināšanas risinājumi *lēni mainīgām dimensijām* ir aprakstīti [KR02]. Kad mainās kaut kāda dimensiju tabulas atribūta vērtība ir trīs iespējas, kā šo izmaiņu veikt datu noliktavā. Pirmais risinājums ir vienkārši pārrakstīt jaunu atribūta vērtību pa virsu vecai vērtībai. Bet šajā gadījumā vēsturiska informācija par šī atribūta vērtību nesaglabāsies, tādēļ var izmantot otro risinājumu, kad veido jaunu dimensiju tabulas ierakstu ar jaunu atribūta vērtību. Faktiem, kas attiecas uz laiku, kad atribūtam bija vecā vērtība, piekārto veco dimensijas ierakstu, bet faktiem, kas attiecas uz laiku pēc atribūta maiņas, piešķir jauno dimensijas ierakstu. Bet šajā gadījumā nav iespējams analizēt jaunos faktus pret veco dimensijas atribūta vērtību. Lai risinātu šo problēmu, var izmantot trešo risinājuma variantu un nepievienot jauno dimensijas ierakstu, bet pievienot jaunu kolonnu tabulai, lai saglabātu veco atribūta vērtību. Lēni mainīgām dimensijām atkarībā no situācijas var izvēlēties vienu no trijiem iepriekšminētiem risinājumiem vai hibrīdu pieeju. Ātri mainīgus dimensiju atribūtus var atdalīt no lēni mainīgiem atribūtiem un apvienot atsevišķā dimensiju tabulā.

Iepriekš aprakstītie risinājumi dimensiju izmaiņām ir ierobežoti uz gadījumu, kad mainās vienas dimensijas atribūts, bet reāli var būt situācijas, kad viens dimensijas loceklis (instance vai ieraksts dimensiju tabulā) tiek sadalīts divos, vai divi atsevišķi locekļi tiek apvienoti. Rakstā [EKM01] piedāvā *laicīgas* (temporal) datu noliktavas izmantošanu šai situācijai. Tas nozīmē, ka katram dimensijas loceklim dimensiju tabulās tiek glabātas *versijas*, kur katrai versijai ir definēts noteikts laika intervāls, kad tā bija „spēkā". Šādas datu noliktavas metadatos tiek glabātas saistības starp viena dimensijas locekļa versijām un

transformācijas funkcijas, kas definē kādā veidā jāpārveido ar šo versiju saistītos mērījumus, ja vaicājums uz šiem mērījumiem ietver periodu, kad bija spēkā vairākas versijas. Piemēram, mērījumus par vairākiem dimensiju ierakstiem var summēt, vai viena dimensiju ieraksta mērījumus var sadalīt atbilstoši noteiktam procentu sadalījumam. Rakstā [EKM01] piedāvāta datu noliktavas arhitektūra, kas atbalsta laicīgu datu noliktavu. Kad lietotājs izpilda vaicājumu uz datu noliktavas datiem, kas ietver periodu, kad bija spēkā vairākas versijas, tad lietotājam tiek paziņots, ka viņš var izmantot vairākas dimensijas locekļa versijas, un pats lietotājs izvēlas viņam interesanto versiju. Tad speciālais transformators pārveido vaicājuma rezultātus ar transformācijas funkcijām un atgriež rezultātu.

Strukturālas izmaiņas dimensijās

Strukturālas un satura izmaiņas dimensijās ir aplūkotas [HMV99]. Šeit tiek definēta daudzdimensiju shēma un tās instance, kas sastāv no dimensiju un faktu tabulu instancēm. Daudzdimensiju shēmas definīcija atbilst 2. nodaļā aprakstītai definīcijai. Dimensijas instance tiek definēta ar vienu konkrētu daudzdimensiju shēmu un *roll-up funkciju* kopu, kur katra funkcija nosaka noteiktas hierarhijas līmeņu saistības, t.i. līmeņu hierarhijas. Šajā rakstā tiek definēta faktu tabulas shēma, kas sastāv no dimensiju līmeņiem un no mērījuma. Faktu tabulas instance tiek definēta ar funkciju, kas nosaka mērījuma vērtību, kas atbilst dotai dimensiju līmeņu vērtību kopai.

[HMV99] definē piecus dimensiju strukturālo izmaiņu operatorus: vispārināšana, specializācija, dimensijas hierarhijas līmeņu sasaistīšana, līmeņu saistību dzēšana un līmeņa dzēšana; un divus dimensiju instanču izmaiņu operatorus: instances pievienošana un dzēšana. Šo operatoru ietekme ir aprakstīta tālāk.

Vispārināšanas operators izveido jaunu līmeni dimensijā, un kādam eksistējošam dimensijas līmenim ir roll-up tipa attiecība līdz šīm līmenim, t.i. jaunais līmenis ir iesprausts visaugstāk līmeņu hierarhijā. Pie tam ir definēta arī funkcija, kas nosaka šo divu līmeņu vērtību saistības, t.i. roll-up funkcija no zemāka līmeņa līdz augstākam līmenim. Šīs izmaiņas gadījumā mainās dimensijas līmeņu kopa (tiek pievienots jauns līmenis) un hierarhiju struktūra, kā arī mainās dimensijas instances roll-up funkcija (tiek pievienota funkcionāla atkarība līdz jaunam līmenim).

Specializācijas operators arī izveido jaunu līmeni dimensijā, un šim līmenim ir roll-up tipa attiecība līdz zemākajam dimensijas līmenim, t.i. tas kļūst par jauno zemāko līmeni visās dimensijas līmeņu hierarhijās. Funkcija, kas nosaka šo divu līmeņu vērtību saistības, arī ir definēta šim operatoram. Šis operators pievieno jauno līmeni dimensijas līmeņu kopā un hierarhijas struktūrā, kā arī papildina dimensijas instances roll-up funkciju ar atkarībām no jauna līmeņa.

Līmeņu sasaistīšanas operators definē roll-up funkciju starp diviem neatkarīgiem līmeņiem (l_a un l_b) dimensijā. Divi līmeņi ir neatkarīgi, ja nav roll-up funkcijas no viena līmeņa līdz otram. Šī operatora pielietošanas rezultātā dimensija tiek papildināta ar funkcionālo atkarību starp šiem diviem līmeņiem, t.i. līmenis l_a atradīsies uzreiz zem līmeņa l_b līmeņu hierarhijā. Bet no līmeņu hierarhiju struktūras tiek izdzēstas visas liekas roll-up funkcijas saistības. Piemēram, ja eksistē kāds līmenis l, kas līmeņu hierarhijā atrodas uzreiz

zem l_b un vienlaikus eksistē līmeņu „ceļš" no līmeņa l uz līmeni l_a, tad tiek izdzēsta roll-up funkcija no līmeņa l uz līmeni l_b.

Līmeņu saistību dzēšanas operators izdzēš roll-up funkciju no līmeņa l_a uz līmeni l_b. Pēc šī operatora izpildīšanas vajag nodrošināt, lai visiem līmeņiem, kas hierarhijā atrodas zem l_a eksistētu roll-up tipa attiecība līdz visiem līmeņiem, līdz kuriem tiem bija roll-up attiecība pirms roll-up funkcijas dzēšanas no l_a uz l_b, piemēram, ja eksistēja roll-up funkcija no l_b uz kādu līmeni l_c, tad jānodrošina, lai no l_a varētu nokļūt līdz l_c līmeņu hierarhijā.

Līmeņa dzēšanas operators izdzēš līmeni un ar to saistītas roll-up funkcijas. Zemāko dimensijas līmeni drīkst izdzēst tikai tādā gadījumā, kad tam ir tikai viens augstāks līmenis, ar ko zemākais līmenis ir saistīts ar roll-up funkciju. Līdzīgi kā līmeņa saistību dzēšanas operatoram, līmeņa dzēšanas operatora pielietošanas gadījumā jānodrošina, lai saglabātos hierarhija no jebkura līmeņa l_1, kas līmeņu hierarhijā atrodas uzreiz zem izdzēsta līmeņa, līdz jebkuram līmenim l_2, kas līmeņu hierarhijā atrodas uzreiz virs izdzēsta līmeņa, t.i. ja vidus līmeņa dzēšanas gadījumā šīs saistības netiek saglabātas, tad dimensiju shēmai un instancēm jāpieliek roll-up funkcija no l_1 uz l_2.

Instances pievienošanas operators iesprauž jauno elementu noteiktā līmenī l_a. Relāciju datu bāzes realizācijā tas nozīmē jauna ieraksta iespraušanu dimensiju tabulā. Šajā gadījumā jādefinē jauna elementa saistības (roll-up funkcijas) ar elementiem visiem līmeņiem, kas atrodas uzreiz virs l_a līmeņu hierarhijā. Šis operators nemaina dimensiju, tikai instances.

Instances dzēšanas operators izdzēš vienu elementu noteiktā līmenī. Relāciju datu bāzes realizācijā tas nozīmē ieraksta izdzēšanu no dimensiju tabulas. Šīs operators ir definēts tikai gadījumā, kad nav neviena elementa zemākos līmeņos, kuriem eksistē roll-up tipa attiecība līdz izdzēstajam elementam, t.i. izdzēstais elements neatrodas virs kāda saistīta elementa.

Ar mainītu dimensiju saistītu faktu tabulu ietekmē tikai daži no iepriekš aprakstītajiem operatoriem. Zemāka līmeņa dzēšanas gadījumā faktu tabulas mērījumi tiek apkopoti, bet cita līmeņa dzēšanas gadījumā, šo līmeni vienkārši izdzēš no visām ar mainīto dimensiju saistītām faktu tabulām. Specializācijas operācijas gadījumā nevar vispārīgā gadījumā nodefinēt izmaiņas faktu tabulā, bet ja zināma mērījumu sadalīšana pa jauna līmeņa elementiem, tad var izrēķināt detalizētākus mērījumus. Instances pievienošanas gadījumā jādefinē faktu tabulas mērījumu vērtības, kas atbilst jaunai instancei, bet instances dzēšanas gadījumā mērījumi, kas ir saistīti ar izdzēstu elementu, kļūst nedefinēti, šajā gadījumā var vai nu izdzēst šos mērījumus no faktu tabulas, vai nu piesaistīt tos citiem dimensijas elementiem.

Rakstā [VMR+02] ir aplūkota iepriekš aprakstīto dimensiju izmaiņu atbalsta realizācija, kas balstās uz relāciju tiešsaistes analītiskās apstrādes serveri un speciālu valodu daudzdimensiju izmaiņu specifikācijai.

Rakstā [BFB08] autori piedāvā lietotāju-orientētu pieeju, kas atbalsta datu noliktavas shēmas izmaiņas. Šī pieeja atļauj lietotājiem definēt jaunus hierarhijas līmeņus un analizēt datus dažādās jaunajās detalizācijas pakāpēs. Pieeja sastāv no četrām fāzēm: (1) lietotāju zināšanu iegūšana, (2) zināšanu integrācija, (3) datu noliktavas shēmas atjaunošana, (4) tiešsaistes analīze.

Autoru pieejas galvenā doma ir piesaistīt lietotājus datu noliktavas shēmas evolūcijai, lai izveidotu jaunas detalizācijas pakāpes atbilstoši lietotāju zināšanām. Pieejas pirmajā solī

dati par jauniem dimensijas hierarhiju līmeņiem tiek iegūti no lietotājiem kā „Agregācijas Datu Zināšanas" („Aggregation Data Knowledge" – ADK), kas ir likums formā „if-then", kas definē kādas zemāka līmeņa atribūtu vērtības atbilst jaunajām augstākā līmeņa atribūtu vērtībām, t.i. lietotāji sagādā jaunā austākā līmeņa vērtības.

Otrajā solī ADK likumi tiek pārveidoti kartējuma tabulās. Katram likumam tiek veidota jauna tabula ar nosacījumiem ar atribūtiem un atbilstošām vērtībām jaunajam līmenim. Katram atribūtam tiek meklēts nosacījums, ko atribūts apmierina, un šim atribūtam tiek piekārtota atbilstoša jauna līmeņa vērtība no kartējuma tabulas.

Trešajā solī evolūcija tiek implementēta datu noliktavā. Ir atbalstītas tikai divu veidu izmaiņas: jaunā līmeņa iespraušana hierarhijas „galā" vai starp diviem līmeņiem.

Visa pieeja ir balstīta uz R-DW (formālu likumu bāzētu datu noliktavas) modeli, kas sastāv no fiksētas un mainīgas daļas. Fiksētā daļa apraksta faktu tabulas saistības ar zemākajiem dimensiju līmeņiem. Mainīgā daļa apraksta dimensiju hierarhiju detalizācijas līmeņus un ar tiem saistītos likumus. Šajā modelī ir arī atbalstītas līmeņu versijas, kad vienu un to pašu līmeni veido dažādi lietotāji ar dažādiem nosacījumiem, bet nav atbalstītas versijas, kas atspoguļo noteiktas prasības konkrētajā laika periodā. Arī modelim ir divas īpašības, kas attiecas uz likumiem: nosacījumi likuma if-daļā definē neatkārtojošas vērtības (t.i. viena zemākā līmeņa atribūta vērtība nevar apmierināt vairākus nosacījumus), un nosacījumi definē visas līmeņa vērtības (t.i. jauno līmeni definē visiem dimensijas locekļiem).

Fiziski jaunā līmeņa izveidošana tiek realizēta kā jaunas tabulas izveidošana un sasaistīšana ar zemāka un augstāka līmeņa tabulām, t.i. datu bāzē tiek veidotas atsevišķas tabulas katram dimensijas līmenim.

Šajā apakšnodaļā aprakstīti risinājumi ir izmantojami dažādām izmaiņām daudzdimensiju shēmas dimensiju struktūrā, bet tajos netiek apskatītas izmaiņas faktos, mērījumos un netiek atbalstīta jaunas dimensijas pievienošana shēmai, kas arī var notikt datu noliktavā. Aprakstītie risinājumi neatbalsta datu noliktavas shēmas versijas, kas atspoguļo datu noliktavas prasības noteiktā laikā.

Nākošajā apakšnodaļā ir aprakstītas pieejas izmaiņu apstrādei dažādās daudzdimensiju shēmas daļās, tai skaitā arī faktos un mērījumos.

Izmaiņas daudzdimensiju shēmās

Rakstā [BSH99] ir piedāvātas daudzdimensiju shēmas un tās instances formālas definīcijas, kas ir ērtas izmaiņu definēšanai. Daudzdimensiju shēmas definīcija būtiski neatšķīrās no šajā darbā izmantotās definīcijas, kas dota 2. nodaļā.

Daudzdimensiju shēmas instance ir definēta ar funkciju kopu (viena funkcija katrai klasifikācijai), kur katra funkcija definē klasifikācijas attiecības starp divu dimensiju līmeņu elementiem (instancēm). Šīs funkcijas sasaista divu dimensiju līmeņu instances. Arī katrai daudzdimensiju shēmas instancei ir jābūt funkcijai, kas sasaista zemāko dimensiju līmeņu instances ar mērījumiem – faktu atribūtiem, un katram aprakstošam dimensijas atribūtam jābūt funkcijai, kas piešķir šī atribūta vērtību katram dimensijas loceklim (instancei).

[Bla00] arī izmanto iepriekš aprakstītas definīcijas. Tikai šajā darbā [Bla00] ir ieviesti arī daudzdimensiju shēmas integritātes ierobežojumi: katram faktam jābūt sasaistītam ar

vismaz vienu dimensijas līmeni, katram dimensiju līmenim jābūt sasaistītam ar vismaz vienu faktu vai jābūt kādas klasifikācijas hierarhijas sastāvdaļai un katras hierarhijas zemākajam līmenim jābūt sasaistītam ar faktu, katram atribūtam jābūt piešķirtam vai nu kādam faktam vai nu dimensiju līmenim.

[BSH99] un [Bla00] ir definētas arī vairākas primitīvas evolūcijas operācijas, kas darbojas ar dimensiju līmeņiem, atribūtiem, hierarhiskām attiecībām starp līmeņiem (rakstā tās tiek sauktas par klasifikācijas attiecībām), un faktiem: līmeņa iespraušana un dzēšana, atribūta iespraušana, dzēšana, piesaistīšana dimensiju līmenim, atvienošana no dimensiju līmeņa, piesaistīšanas faktam un atvienošana no fakta, hierarhiskās attiecības iespraušana un dzēšana, fakta iespraušana, dimensiju līmeņa pievienošana faktam un dzēšana no fakta. Katra primitīva evolūcija, kas tiks aprakstīta tālāk, var izraisīt shēmas nesaskaņas, tādēļ pēc vairāku primitīvo evolūciju virknes, evolūcijas sesijas, tiek pārbaudīti integritātes ierobežojumi.

Katrai operācijai tiek norādītas nepieciešamās pielāgošanas darbības pašai shēmai un shēmas instancēm. Piemēram, hierarhiskās attiecības iespraušanas operācija definē hierarhisku attiecību starp diviem eksistējošiem dimensiju līmeņiem, kas var būt izolēti vai saistīti ar citām attiecībām. Šīs operācijas rezultātā ir iegūta jauna shēma, kur divi dotie dimensiju līmeņi ir sasaistīti, t.i. eksistē hierarhiska attiecība no viena uz otro. Shēmas instancēs ir definēta jauna hierarhiska attiecība no viena dimensiju līmeņa locekļiem uz otra dimensiju līmeņa locekļiem. Dimensiju līmeņa pievienošana faktam pievieno eksistējošo dimensijas līmeni eksistējošam faktam, tādēļ fakta dimensiju skaits palielinās par vienu (palielinās detalizācijas pakāpe). Šīs operācijas izpildīšanai ir nepieciešama funkcija, kas definē, kā jaunas fakta vērtības var tikt izrēķinātas, balstoties uz palielinātu dimensiju līmeņu kopu un veco fakta vērtību. Šeit var būt vairāki risinājumi, kas ir atkarīgi no situācijas, piemēram, ja iepriekš bija tikai viena instance, kas apzīmēta ar pievienoto dimensijas līmeni, tad jauna fakta vērtība, kas atbilst šai instancei, vienkārši ir vienāda ar vecā fakta vērtību. Šīs operācijas rezultātā ir iegūta jauna shēma, kura ir papildināta ar eksistējoša fakta saistību ar eksistējošo dimensijas līmeni. Shēmas instancēs mainās faktu koordinātes (dimensiju līmeņi) un vērtības, kas ir izrēķinātas ar atbilstošo funkciju.

Darbā [Bla00] iepriekš aprakstītā daudzdimensiju shēma ir realizēta loģiskajā līmenī ar relāciju datu bāzes shēmu – zvaigznes shēmu. Tātad, fakti ir attēloti kā faktu tabulas, mērījumi kļūst faktu tabulas kolonnas, dimensijas kļūst par vienu vai vairākām dimensiju tabulām, dimensiju atribūti – par dimensiju tabulu kolonnām, dimensiju tabulas ir saistītas ar faktu tabulu kā dimensijas ir saistītas ar faktiem. Datu noliktavas zvaigznes shēma neatbalsta visus daudzdimensiju shēmas komponentus, piemēram, hierarhijas, tādēļ šī papildus informācija tiek glabāta metadatos.

Relāciju datu bāzes shēmās arī ir evolūcijas operācijas, kas var tikt izteiktas ar daudzdimensiju shēmas evolūcijas operāciju palīdzību: mērījuma kolonnas iespraušana un dzēšana, atribūta kolonnas iespraušana un dzēšana, faktu tabulas kopā ar dimensiju tabulu iespraušana, faktu tabulas iespraušana un dzēšana, dimensiju tabulas iespraušana, dimensiju līmeņa iespraušana, hierarhiskas attiecības starp dimensijas hierarhijas līmeņiem iespraušana un dzēšana, dimensiju līmeņa dzēšana, dimensijas tabulas dzēšana.

Katra šī operācija maina zvaigznes shēmu un metadatus. Piemēram, relāciju datu bāzes shēmas hierarhiskas attiecības starp dimensijas līmeņiem iespraušanas operācija

iespraužjauno hierarhisku attiecību starp diviem eksistējošiem dimensiju līmeņiem. Tā ir vienāda ar hierarhiskas attiecības iespraušanas operāciju daudzdimensiju shēmā. Ja abi līmeņi jau atrodas vienā dimensijā, dimensijas instances nemainās, pretējā gadījumā jāpieliek jauna kolonna dimensiju tabulai un jāaizpilda tās vērtības ar atbilstošu funkciju, kas nosaka hierarhiju. Relāciju datu bāzes shēmas dimensijas tabulas iespraušanas operācija iespraužjauno saistību starp eksistējošo faktu un dimensijas tabulu. Tā atbilst dimensiju līmeņa pievienošanas faktam operācijai daudzdimensiju shēmā. Šajā gadījumā palielinās faktu tabulas detalizācijas pakāpe un eksistējošie mērījumi jāpielāgo ar speciālu transformācijas funkciju.

Iepriekš aprakstītās evolūcijas operācijas atļauj pārnest evolūcijas operācijas no daudzdimensiju shēmas konceptuālā līmeņa loģiskajā relāciju datu bāzes shēmas līmenī.

Rakstā [KPR04] ir apskatītas astoņas evolūcijas operācijas, kas ietekmē datu noliktavas shēmu. Operācijas ir apakškopa no iepriekš aprakstītajām operācijām no darba [Bla00]. Bet katrai operācijai tiek dota formālā semantika ne tikai zvaigznes, bet arī sniegpārslas datu noliktavas shēmām. Katram shēmas veidam un katrai izmaiņai tiek definētas nepieciešamās izmaiņas shēmā, datu mainīšanas iespējas un izmaiņu ietekme uz vaicājumiem, kas satur mainīto elementu un kas iepriekš tika izmantoti, lai analizētu datus datu noliktavā.

Rakstā [BD09] un promocijas darbā [Ban07] tiek definēts vispārināts formālais datu noliktavas modelis un šī modeļa ierobežojumi (konstreinti). Modelis tiek definēts, izmantojot ULD modeli (autori: Bowers un Delcambre), kur tiek definēti četri modeļa līmeni: metadatu līmenis, meta-shēmas līmenis, shēmas līmenis un datu līmenis. Katrs līmenis ir iepriekšēja līmeņa instance. Līmeņa konstrukciju definēšanai izmanto kolekcijas (kopas), kortežus un atomārus datus.

Modeļa ierobežojumi tiek sadalīti divās grupās: pievienošanas (addition) ierobežojumi tiek definēti, lai nodrošinātu modeļa korektumu, kad modelim tiek pievienots fakts, dimensija, līmenis, hierarhija vai kubs, un dzēšanas (deletion) ierobežojumi tiek definēti, lai nodrošinātu modeļa korektumu, kad no modeļa tiek izdzēsts fakts, dimensija, līmenis, hierarhija vai kubs. Ierobežojumu piemēri: faktam/dimensijai jābūt pievienotam shēmai; mērījumam jāpieder faktam; līmenis varētu tikt pievienots tikai eksistējošai dimensijai; hierarhijai jābūt pievienotai dimensijai; u.c.

Rakstā tiek definētas evolūcijas operācijas, kas maina dimensijas, faktus un kubus. Darba teorētiska daļa ir ļoti līdzīga Blaschka darbam [Bla00]. Autori apskata savas piedāvātās pieejas implementāciju, t.i. vispārināta modeļa un ierobežojumu implementāciju relāciju datu bāzē.

Arī vispārinātajā modelī autori definē „papildus" konstrukcijas, tādas, kā vairākas (multiple) hierarhijas (būtībā vairāki ceļi no viena dimensijas līmeņa līdz otram), nepārklājošas (non-covering) hierarhijas (kad dažām instancēm kādam no līmeņiem ir roll-up attiecība nevis līdz tuvākajam līmenim, bet citam, izlaižot tuvāko līmeni), non-onto hierarhijas (kad augstāka līmeņa instancei nav zemāka līmeņa instances), nestingras hierarhijas (kad attiecība starp augstāko līmeni un zemāko līmeni ir daudz-pret-daudz).

Tiek apskatītas arī vēl papildus evolūcijas operācijas, kas padara parasto hierarhiju par paralēlu (multiple) un otrādi, operācijas, kas pievieno un izdzēš nepārklājošo hierarhiju

dimensijai, operācijas, kas pievieno un izdzēš non-onto hierarhiju dimensijai, operācijas, kas pievieno un izdzēš nestingru hierarhiju dimensijai.

Rakstā arī ir aprakstīts, kā autori implementē agregācijas (sum, min, max, count, average) un OLAP (group by, cube) operatorus, ņemot vērā īpašas dimensijas un datu noliktavas evolūciju. Lai labāk saprastu agregāciju, grafiskai agregācijas operatoru attēlošanai izmanto daudzdimensiju režģi [HRU96], kur virsotnes atspoguļo visus iespējamus group-by operatorus dimensiju līmeņiem, šķautnes savieno vienu virsotni ar otru, kas nozīmē, ka vienam no līmeņiem ir roll-up attiecība līdz otram līmenim. Autori papildina daudzdimensiju režģi, lai atbalstītu nepārklājošas, non-onto un nestingras hierarhijas. Daudzdimensiju režģis ir arī formāli definēts vispārinātajā modelī. Darbā arī piedāvā instanču hierarhiju režģi, kas ir līdzīgs daudzdimensiju režģim, ko arī definē vispārinātajā modelī.

Tālāk autori rāda, kā agregācija tiek atbalstīta vispārinātajā modelī, kā agregācija tiek izpildīta datu noliktavā, kas atbalsta papildus konstrukcijas. Tad tiek apskatīti evolūcijas operatori, kas adaptē vispārināto modeli. Operatoriem tiek definēta to ietekme uz daudzdimensiju režģi, instanču hierarhiju režģi un procedūras soļi, kas adaptē vispārināto modeli pēc dažādām izmaiņām.

3.4.3. Datu noliktavas pielāgošanas pieeju analīze

Šajā 3.4. apakšnodaļā tika aprakstītas pieejas, kas ir izmantojamas dažādām izmaiņām materializētos skatus un daudzdimensiju shēmā. Materializētu skatu gadījumā pielāgošanas problēmas risināšanai ir apskatītas pieejas, kas papildina skata definīciju ar papildus konstrukcijām, lai skata pārdefinēšanas gadījumā, izmantojot šo papildus informāciju, var izrēķināt vērtības jaunām vai mainītām skata kolonnām vai rindām, nepārrēķinot visu skatu no sākuma. Šīs pieejas var izmantot tikai gadījumos, kad datu noliktavas tabulas ir definētas kā materializēti skati uz datu avotu tabulām, t.i. kad datu avoti ir relāciju datu bāzes.

Daudzdimensiju datu noliktavas gadījumā šajā nodaļā aprakstītas metodes definē operācijas, kas maina datu noliktavas shēmu, un šo operāciju sekas. Katrai no metodēm ir savi trūkumi un priekšrocības. Piemēram, dažas metodes [HMV99], [VMR+02] aplūko tikai izmaiņas dimensiju struktūrā un nevar tikt izmantotas faktu pielāgošanas gadījumos. Metodes arī var izmantot kombinēti, lai varētu pārklāt lielāku datu noliktavas shēmas izmaiņu skaitu dažāda veida datu noliktavas shēmas reprezentācijas relāciju datubāzē. Piemēram, darbā [Ban07] nav atbalstītas tādas izmaiņas, kad faktu tabulai pievieno mērījumu vai izdzēš to, bet darbā [Bla00] piedāvātajā metodē nav atbalstītas speciālas hierarhijas, piemēram, non-onto hierarhijas, nestingras hierarhija, u.c. Bet rakstā [KPR04] papildus vēl tiek apskatīta dažādu evolūcijas operāciju ietekme uz sniegpārslas datu noliktavas shēmu. Aprakstītajās pieejās nav atbalstīti dimensijas atribūti, kas satur tikai aprakstošu informāciju un nedefinē nekādu hierarhijas līmeni, t.i. nepiedalās nevienā hierarhijā (netieši tos var nodefinēt kā hierarhijas, kas sastāv tikai no viena līmeņa). Arī ne visās pieejās ir apskatīts jautājums par evolūcijas operāciju realizāciju datu līmenī un kā ir jāpielāgo atskaites, kas tika izpildītas uz mainīto datu noliktavas shēmu. Arī šīs apakšnodaļas risinājumos nav atbalstītas shēmas versijas un izmantotie repozitoriji neatbilst nekādam standartam (piemēram, CWM).

Šie risinājumi apskata tikai problēmas, saistītas ar izmaiņām datu noliktavā, ko izraisa datu noliktavas darījumprasību attīstība, bet nerisina problēmas, saistītas ar datu noliktavas adaptāciju pēc izmaiņām datu avotos.

3.5. Datu noliktavas adaptācija un uzturēšana pēc adaptācijas

Saistībā ar datu avota shēmas maiņu, rodas problēmas, kā šo shēmas evolūciju izplatīt datu noliktavā un kā adaptēt datu noliktavas shēmu, datus un iegūšanas, transformācijas un ielādes (ETL) procesus, atbilstoši jaunai datu avotu shēmai. Šīs problēmas sauc, respektīvi, par datu noliktavas *adaptāciju* un *uzturēšanu pēc adaptācijas*. Datu noliktavas adaptācijas problēmu raksturo sekojošas koordinātes (att. 4.): (1) datu noliktavai nav tiešo izmaiņu apriori, (2) datu avotos notiek datu struktūru izmaiņas, kas izraisa datu noliktavas shēmas maiņu, (3) datu noliktavas lietotāji vēlas, lai datu noliktavai vienmēr būtu jaunākā shēma, un (4) datu avotos nesaglabājas iepriekš pieejamās spējas (avotā mainījās gan shēma, gan dati). Datu noliktavas uzturēšanas pēc adaptācijas problēmu raksturo sekojošas koordinātes (att. 4.): (1) datu noliktavas shēma tika adaptēta, t.i. tā tika mainīta adaptācijas procesā, (2) datu avoti mainīja savas datu struktūras, (3) datu noliktavas lietotājs vēlas, lai datu noliktavā vienmēr būtu pieejama jaunākā informācija, un (4) datu avotos nesaglabājas ne vecais datu stāvoklis, ne shēma.

Rakstā [RKZ00] apskata iemeslus datu noliktavas shēmas evolūcijai pēc datu avotu izmaiņām. Shēmas un saskarnes izmaiņas notiek diezgan bieži, jo ne tikai ir grūti iepriekš noteikt avotu datu bāzes shēmu pirmajā izstrādes etapā, bet arī avotu sistēmu darījumprasības var mainīties laika gaitā. Viens no iemesliem datu noliktavas shēmas evolūcijai var būt situācija, kad semantiski līdzīga informācija var tikt glabāta dažādā veidā. Piemēram, datu avotā datiem var atbilst rinda tabulā, bet datu noliktavā - kolonna, tādēļ, kad parādās jauns ieraksts datu avotā, datu noliktavā jāpieliek klāt jauna kolonna tabulai. Līdzīga problēma rodas, kad tiek integrēta dažādi modelēta vai dažādu formātu informācija, piemēram, relāciju datu bāzes tabulu dati un XML formāta dati. Izmaiņas datu avotos var izplatīt dažādos veidos. To var, piemēram, izdarīt starpnieku līmenī, vai datu noliktavas līmenī.

Datu noliktavas adaptācija un uzturēšana pēc adaptācijas ir cieši saistītas, tāpēc tās ir apvienotas vienā nodaļā. Tālākajās apakšnodaļās tiek aprakstītas pieejas datu noliktavas adaptācijas problēmas risināšanai datu noliktavās. Tiek apskatīti abi datu noliktavas reprezentācijas gadījumi: gan kā materializēto skatu kopums uz avotiem, gan kā daudzdimensiju datu modeļi.

3.5.1. Materializēto skatu adaptācija un uzturēšana pēc adaptācijas

Vairāki pētījumi ir veltīti materializētu skatu adaptācijas [LNR97] un tālākas uzturēšanas problēmu risināšanai. Lai risinātu šīs problēmas jāatbild uz vairākiem jautājumiem:

- Kā var noteikt, kādus adaptācijas scenārijus var izvēlēties?
- Kāda papildus informācija ir nepieciešama, lai varētu veikt adaptāciju?
- Kādā veidā var novērtēt skata jaunās (iegūtas pēc adaptācijas) definīcijas pareizību?
- Kādu adaptācijas procesā iegūto skata definīciju izvēlēties?

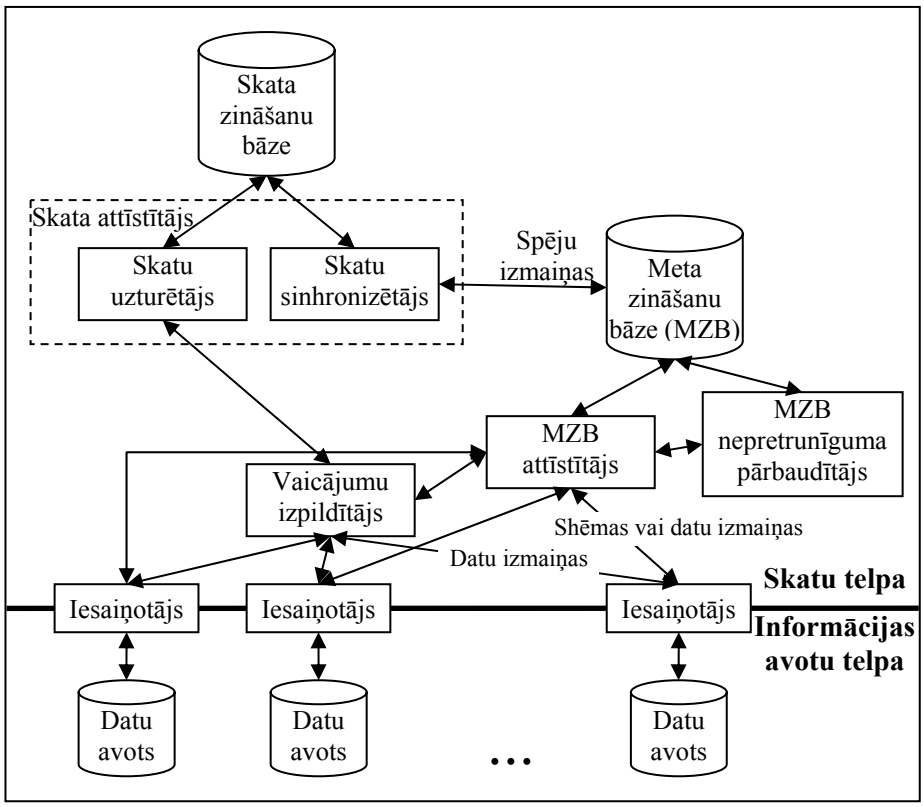
- Kādu stratēģiju izvēlēties, lai pielāgotu skata saturu pēc adaptācijas?

Sekojošas pieejas materializētu skatu adaptācijas un uzturēšanas pēc adaptācijas atbild uz šiem jautājumiem.

Materializētu skatu adaptācija

Materializētu skatu adaptācijas risinājums ir piedāvāts rakstā [LNR97]. Šeit ir aprakstīta attīstošu skatu vides (Evolvable View Environment) EVE arhitektūra (att. 7.), kas atbalsta skata definīciju adaptēšanu, reaģējot uz izmaiņām datu avotos, un tālāko uzturēšanu pēc adaptācijas. EVE pārraksta materializēta skata definīciju, aizvietojot trūkstošus atribūtus ar piemērotiem komponentiem no citiem datu avotiem, kur tas ir iespējams.

EVE sastāv no divām daļām: skatu telpas un informācijas avotu telpas. Informācijas avotu telpa satur vairākus datu avotus, ar kuriem saziņa notiek caur iesaiņotājiem, kas iegūst gan datu avotu informācijas saturu, gan metadatus, piemēram, shēmas izmaiņas, veiktspējas datus, saistības ar citiem datu avotiem. Kad jauns datu avots pievienojas sistēmai, tas sniedz ziņas par savām iespējām, datu modeli un datu saturu meta zināšanu bāzē (MZB). Kad kādā datu avotā notiek izmaiņas, atbilstošais iesaiņotājs paziņo šīs izmaiņas MZB attīstītājam, kas veic atbilstošas izmaiņas MZB. Skata uzturētājs izplata datu satura izmaiņas, t.i. veic skata uzturēšanu, un maina skata saturu pēc adaptācijas. Skata sinhronizētājs veic skata adaptāciju saskaņā ar lietotāja uzstādītām opcijām. MZB nepretrunīguma pārbaudītājs pārbauda un kļūdas gadījumā izlabo ierobežojumus, kas ir uzstādīti MZB datu avotiem.



Att. 7. EVE arhitektūra [LNR97]

EVE sistēmā materializēti skati ir definēti ar papildinātu SQL valodu – attīstāmo-SQL (Evolvable-SQL) vai E-SQL. Šajā valodā papildus parastai SELECT-FROM-WHERE skata definīcijai ir ieviesti parametri, kas nosaka kā skata definīcija var attīstīties, mainoties datu avotu shēmai vai datiem. Katram skata komponentam (atribūtam SELECT sadaļā, relācijai FROM sadaļā un nosacījumam WHERE sadaļā) ir piekārtotas divu parametru vērtībās. *Neobligātuma* parametrs nosaka, vai komponents ir nepieciešams, vai tas var tikt izmests, ja tam nevar atrast aizvietošanu. *Aizvietojamības* parametrs specificē, vai komponents var vai nevar tikt aizvietots skata adaptācijas procesā. Arī materializēta skata definīcijā var norādīt kādā veidā var mainīties skata datu saturs adaptācijas procesā (skata satura saglabāšanas parametrs), t.i. var uzstādīt, ka mainīta skata datu saturam jābūt vienādam ar oriģināla skata datu saturu, vai tām jābūt oriģināla skata datu satura apakškopai, vai tām jāsatur oriģināla skata saturu, bet var saturēt arī citus datus, vai ir atļauts, ka mainīts skats satur dažus datus no oriģināla skata un vēl papildus datus (aptuveni atbilst).

Datu avotu metadatu glabāšanai MZB tiek izmantots modelis informācijas avotu aprakstam (Model for Information Source Description) – MISD modelis. Šīs modelis atļauj definēt datu satura, tipa integritātes, kārtības integritātes, savienojuma un daļējas/pilnas informācijas ierobežojumus. Datu satura ierobežojumi definē pieejamās katra datu avota relācijas (datu satura nosaukums un relācijas vārds). Tipa integritātes ierobežojumi definē pieļaujamo vērtību apgabalu (domēnu) katram atribūtam. Kārtības integritātes ierobežojumi definē nosacījumus relācijas kortežiem (piemēram, atribūts1<atribūts2, u tml.). Savienojuma ierobežojumi definē nosacījumus, pēc kādiem var savienot divas relācijas. Daļējas/pilnas informācijas ierobežojumi starp divām relācijām nosaka, ka pirmās relācijas fragments (kolonnas vai rindas) satur vai ir vienāds ar otrās relācijas fragmentu (kolonnas vai rindas).

EVE sistēma atbalsta sekojošas izmaiņas datu avotos: atribūta dzēšana, pievienošana un nosaukuma maiņa, relācijas dzēšana, pievienošana un nosaukuma maiņa. Atribūta vai relācijas pievienošanas izmaiņas neietekmē skata definīciju, tāpēc tās tikai tiek izplatītas MZB. Atribūta vai relācijas nosaukuma maiņas gadījumā vienkārši tiek mainīts nosaukums visos ietekmētos komponentos. Citu izmaiņu gadījumā sākumā tiek noteikti visi materializēti skati, ko šīs izmaiņas ietekmē. Tad katram ietekmētam skata komponentam tiek noteikts, vai to var aizvietot un vai tas ir obligāts. Gadījumā, kad komponents ir obligāts un to nevar aizvietot, adaptācijas algoritmi paziņo neveiksmi. Citā gadījumā algoritmi meklē aizvietojumu komponentam MZB, kas atbilst noteiktiem nosacījumiem. Atribūtam šie nosacījumi ir sekojoši: datu tipa atbilstība starp veco un jauno atribūtu, savienojuma ierobežojuma eksistence starp vecā atribūta relāciju un jaunā atribūta relāciju, datu satura atbilstība starp veco un jauno atribūtu. Relācijai šie nosacījumi ir sekojoši: obligātu un aizvietojamu vecas relācijas atribūtu aizvietojumu eksistence jaunā relācijā, datu tipa atbilstība starp skatā izmantotajiem vecas un jaunas relācijas atribūtiem, datu satura atbilstība starp veco un jauno relāciju. Kad atribūtu vai relāciju aizvietojums ir atrasts, algoritms aizvieto visu skatu definīcijās šo atribūtu vai relāciju ar aizvietojumu un rediģē atbilstoši nosacījumus WHERE daļā.

Materializētu skatu uzturēšana pēc adaptācijas

Kad materializēts skats tika adaptēts, t.i. tā definīcija mainījās, reaģējot uz datu avotu shēmas vai datu izmaiņām, šī skata datus vajag atbilstoši modificēt. Šo problēmu var reducēt uz skata pielāgošanas problēmu, pārveidojot adaptācijas operācijas primitīvajās skata modificēšanas operācijās [NR98]. Tomēr metodes pielāgošanas problēmas risināšanai ne vienmēr efektīvi strādā adaptācijas gadījumā, jo katras primitīvas operācijas starprezultāti varētu būt ļoti lieli, turklāt pielāgošanas algoritmi uzskata, ka datu avotu pusē nenotiek izmaiņas, t.i. visas pieejamās datu avotu struktūras paliek nemainīgas.

Rakstā [NR98] piedāvā skata uzturēšanas algoritmus, kas varētu tikt izmantoti pēc materializēta skata adaptācijas. Šie algoritmi izmanto informāciju par iesaistītiem datu avotiem un adaptācijas procesu, kas tiek veikts EVE sistēmā. Skatu uzturēšanai pēc adaptācijas tiek piedāvātas četras stratēģijas:

- skata pilna pārrēķināšana;
- SYNCMAB stratēģija, kad izmanto daļējas/pilnas informācijas ierobežojumus starp iedzēsto relāciju un tās aizvietotāju un pašu izdzēsto relāciju, t.i. šī stratēģija strādā pirms izmaiņām datu avotos;
- SYNCMAA stratēģija, kad pielieto speciālas metodes, nelietojot informāciju par izdzēsto relāciju, t.i. šī stratēģija strādā pēc izmaiņām datu avotos;
- adaptācijas stratēģijas, kas tiek izmantotas skata satura pielāgošanai pēc skata definīcijas maiņai.

SYNCMAB stratēģijas pielietojums ir atkarīgs no pārdefinētā materializētā skata satura attiecības pret oriģinālā materializētā skata saturu, šo attiecību definē skata satura saglabāšanas parametrs [NR98]. Kombinējot šo skata satura saglabāšanas parametru un daļējas/pilnas informācijas ierobežojumus starp izdzēsto un aizvietotāja relāciju, var noteikt, ka: (1) pārveidotā skata datu saturs ir vienāds ar oriģinālā skata datu saturu; (2) tas ir oriģinālā skata datu satura apakškopa; (3) tas satur oriģinālā skata saturu, bet var saturēt arī citus datus; vai (4) pārveidots skats satur dažus datus no oriģinālā skata un vēl papildus datus (aptuveni atbilst). Atkārībā no attiecībām starp pārveidotā skata saturu un oriģinālā skata saturu tiek iegūts jaunā pārveidotā skata saturs, kura izrēķināšanai ir izmantots oriģinālā skata saturs.

(1) Ja pārveidotā skata saturs ir vienāds ar oriģinālā skata saturu, tad vienkārši jāņem oriģinālā skata atribūtu projekcija uz aizvietotājatribūtiem, t.i. pārdefinētajā skatā tiek ievietoti dati no oriģinālā skata aizvietotājatribūtiem un dati no atribūtiem, ko nevar aizvietot, netiek ievietoti.

(2) Ja pārveidotā skata saturs ir oriģinālā skata satura apakškopa, tad jāaprēķina starpība starp izdzēsto relāciju un aizvietotāja relāciju, un šī starpība jāizņem no oriģinālā skata atribūtu projekcijas uz aizvietotājatribūtiem, t.i. pārdefinētajā skatā tiek ievietota oriģinālā skata datu apakškopa.

(3) Ja pārveidots skats satur datus no oriģināla skata un vēl citus datus, tad vajag oriģināla skata atribūtu projekcijai uz aizvietotājatribūtiem pievienot datu starpību starp aizvietotāja relāciju un izdzēsto relāciju.

(4) Ja datu saturs pārveidotajam skatam aptuveni atbilst datu saturam oriģinālajam skatam, tad vajag izņemt no oriģinālā skata atribūtu projekcijas uz

aizvietotājatribūtiem datu starpību starp izdzēsto relāciju un aizvietotāja relāciju un pievienot datu starpību starp aizvietotāja relāciju un izdzēsto relāciju.

SYNCMAA stratēģijai sākumā arī vajag noteikt attiecību starp pārdefinētā skata saturu un oriģinālā skata saturu, kombinējot skata satura saglabāšanas parametru un daļējas/pilnas informācijas ierobežojumus starp izdzēsto un aizvietotāja relāciju, līdzīgi SYNCMAB stratēģijai. SYNCMAA stratēģijai tiek izdarīts pieņēmums, ka visi atribūti, kas piedalās skata definīcijā WHERE sadaļā, arī parādās SELECT sarakstā.

(1) Ja pārveidotā skata saturs ir vienāds ar oriģinālā skata saturu, tad jārīkojas līdzīgi SYNCMAB stratēģijai gadījumā (1).
(2) Ja pārveidotā skata saturs ir oriģinālā skata satura apakškopa, tad rēķina izdzēstās relācijas kortežus, kas ir oriģinālajā skatā, bet neparādās aizvietotāja relācijā. Šo var izdarīt saskaņā ar pieņēmumu par atribūtu esamību skata definīcijas SELECT sarakstā. Šos iegūtos kortežus izdzēš no oriģinālā skata atribūtu projekcijas uz aizvietotājatribūtiem.
(3) Ja pārveidotais skats satur datus no oriģinālā skata un vēl citus datus, tad vajag oriģinālā skata atribūtu projekcijai uz aizvietotājatribūtiem pievienot datu starpību starp aizvietotāja relāciju un oriģinālā skata izdzēstas relācijas atribūtu datiem.
(4) Ja datu saturs pārveidotajam skatam aptuveni atbilst datu saturam oriģinālajā skatā, tad vajag izņemt no oriģinālā skata atribūtu projekcijas uz aizvietotājatribūtiem datu starpību starp oriģinālā skata izdzēstās relācijas atribūtu datiem un aizvietotāja relāciju un pievienot datu starpību starp aizvietotāja relāciju un oriģinālā skata izdzēstās relācijas atribūtu datiem.

Laiksakritības vadība

Iepriekš aprakstītie algoritmi materializētu skatu adaptācijai ir izmantojami tikai vidēs, kur nenotiek vienlaicīgas shēmas izmaiņas vairākos datu avotos un datu izmaiņas. Bet datu noliktavas integrē autonomus datu avotus, kur shēmas izmaiņas var notikt neatkarīgi, tādēļ šīs aspekts arī jāņem vērā.

Laiksakritības vadība kļūst nepieciešama, kad viena datu avota shēmas vai datu izmaiņu apstrādes (skata uzturēšanas vai adaptācijas) laikā notiek arī shēmas vai datu izmaiņas citā vai tajā pašā datu avotā.

Rakstā [ZR99] apskata problēmu, kad viena datu avota shēmas maiņas apstrādes laikā notiek arī datu izmaiņas citos datu avotos. Šajā rakstā tika izdarīts pieņēmums, ka datu avoti ir *daļēji kooperatīvi*, t.i. avotu datu izmaiņu gadījumā datu avoti sākumā veic izmaiņas un tad informē par tām datu noliktavu, avota shēmas izmaiņu gadījumā datu avots sākumā informē datu noliktavu par paredzētajām izmaiņām savā shēmā, gaida, kamēr šīs izmaiņas tiks apstrādātas datu noliktavā, un tikai tad veic pašas shēmas izmaiņas. Saskaņā ar šo pieņēmumu laiksakritības dēļ var rasties divas problēmas: (1) kad notiek vienlaicīgas izmaiņas datu avota datu saturā, kamēr datu noliktavas pusē tiek apstrādātas datu avotu datu izmaiņas (sk. nod. 3.3.1.) un (2) kad notiek vienlaicīgas izmaiņas datu avota shēmā un cita datu avota datos.

Rakstā piedāvātais otrās problēmas risinājums ir sekojošs. Skatu uzturēšanas un adaptācijas rīki, kas darbojas aprakstītajā sistēmā, saņem notikušās izmaiņas no datu avotiem

un veic attiecīgu adaptēšanas procesu, sūtot papildus informācijas vaicājumus datu avotiem caur *vaicājumu pārvaldnieku*, kas savukārt atpazīst laiksakritības problēmas tāpēc, ka pirmkārt visas neapstrādātas izmaiņas datu avotos (izmaiņas datos vai datu shēmās) tiek sakrātas *atjaunināšanas ziņu rindā*, un otrkārt visām pēc kārtas notikušam atjaunināšanām tiek piešķirts unikāls identifikators. Tāpēc, ja atjaunināšanas ziņu rindā ir datu atjaunināšanas ziņas ar identifikatoru, kas ir lielāks par apstrādājamo datu shēmas atjaunināšanu, tad ir notikusi laiksakritības kļūda, un vaicājuma pārvaldnieks saprot, ka vaicājuma rezultāts saturēs laiksakritīgas datu atjaunināšanas rezultātus, t.i. papildus vai trūkstošus kortežus.

Ja laiksakritības kļūda ir notikusi kāda atribūta vai relācijas pārsaukšanas dēļ, tad vaicājumu pārvaldnieks vienkārši maina atbilstoša komponenta nosaukumus vaicājumā. Ja laiksakritības kļūda ir notikusi datu atjaunināšanas dēļ, vaicājumu pārvaldnieks paziņo par šo problēmu *lokālas kompensācijas* rīkam, t.i. vaicājumu pārvaldnieks sūta kļūdaino vaicājumu ar tā rezultātiem un laiksakritīgas datu atjaunināšanas rezultātus. Lokālas kompensācijas rīks izrēķina papildus vai trūkstošos kortežus, ko vajag pielikt klāt vai izņemt no apstrādāta vaicājuma rezultāta. Katrai ietekmētai relācijai, kas piedalās vaicājuma definīcijā un kas tika atjaunināta, tiek izveidota laicīga tabula ar tādu pašu struktūru. Katra šī tabula ir aizpildīta ar kortežiem, kas tika izdzēsti vai pievienoti laiksakritīgas atjaunināšanas rezultātā, pieliekot katram kortežam atbilstoši mīnus vai pluss zīmi. Apstrādātā vaicājumā aizvieto ietekmēto relāciju nosaukumus ar atbilstošo laicīgo tabulu nosaukumiem un izpilda vaicājumu. Rezultātā tiks iegūti korteži, kurus jāpieliek klāt apstrādāta vaicājuma rezultātam (korteži ar mīnuss zīmi), korteži, kurus jāizņem no apstrādāta vaicājuma rezultāta (korteži ar pluss zīmi). Šo operāciju rezultātā iegūtie vaicājuma rezultāti tiek atgriezti vaicājumu pārvaldniekam, kas tālāk padod tos skatu uzturēšanas vai adaptācijas rīkiem.

Šīs metodes priekšrocība ir tā, ka sistēma nesūta papildus vaicājumus datu avotiem laiksakritības problēmas rašanās dēļ un veic visas kompensācijas darbības lokāli, tāpēc tas nerada jaunas laiksakritības problēmas. Iepriekš aprakstītai sistēmas arhitektūrai ir liels trūkums, jo tā bloķē datu avotu, kamēr datu avotu shēmas izmaiņas tiek izplatītas skatos. Bet šo bloķēšanu ne vienmēr var realizēt datu avotos, jo tie var būt neatkarīgi no datu noliktavas.

Rakstā [ZR00] apskata sistēmas arhitektūru, kas atbalsta laiksakritīgas shēmas un datu atjaunināšanas bez papildus nosacījumiem datu avotiem. Šīs sistēmas arhitektūra ir līdzīga [ZR99], tikai šeit datu avota shēmas izmaiņas ir iegūtas, kad skatu atjaunināšanas rīks nevar izpildīt vaicājumu uz datu avotu shēmas maiņas dēļ. Šajā gadījumā skatu atjaunināšanas rīks, kas nodrošina jaunāko datu saturu skatā, vai skatu pielāgošanas rīks, kas adaptē skata datu saturu atbilstoši jaunai skata definīcijai, paziņo par kļūdu adaptācijas rīkam, kas adaptē skatu definīcijas saskaņā ar jauno datu avotu shēmu. Ja skata definīcijas adaptācijas laikā datu avotā, kam tika mainīta shēma, notiek arī datu izmaiņas, tad skatu atjaunināšanas rīks sūta visas atjaunināšanas operācijas, kas notika datu avotos, skatu pielāgošanas rīkam, kas pēc adaptācijas pielāgo skata datu saturu.

Rakstos [LCR02] un [CZR06] iepriekš aprakstīto sistēmu uzlabo, lai varētu apstrādāt arī laiksakritīgas datu avotu shēmas evolūcijas, t.i. tādas shēmas izmaiņas operācijas, ka otra operācija notiek pirmās operācijas apstrādes laikā, tāpēc skatu atjaunināšanas vaicājumi, kas tiek sūtīti datu avotiem pirmās operācijas apstrādes ietvaros, netiek izpildīti kļūdas dēļ. Aprakstīta algoritma galvenā ideja ir apvienot datu vai shēmas konfliktējošas izmaiņas un

apstrādāt tās kopā kā paketi. Šīs konfliktējošās izmaiņas sākumā tiek sadalītas grupās, kur katrā grupā ir viena datu avota izmaiņas. Tālāk katras grupas izmaiņas sadala datu atjaunināšanas izmaiņās un shēmas izmaiņās. Tālāk izmaiņas iepriekš apstrādā. Pirmkārt viena datu avota shēmas izmaiņas dažreiz var kombinēt, piemēram, ja atribūta nosaukums mainās no A uz B un tad no B uz C, tad var apskatīt tikai vienu izmaiņu – nosaukuma maiņa no A uz C. Otrkārt dažas datu atjaunināšanas izmaiņas var kļūt nesaskanīgas shēmas maiņas dēļ. Šādas izmaiņas arī vajag pārveidot, piemēram, ja kādā relācijā sākumā tiek iesprausts kortežs *[1,2]*, tad tiek izdzēsts pirmais atribūts, un tad tiek iesprausts kortežs *[5],* pirmo un trešo datu izmaiņas var apvienot, ņemot vērā otru shēmas izmaiņu, kā kortežu *[2]* un *[5]* iespraušana. Lai datu izmaiņas varētu kombinēt, rakstā piedāvā atstāt tikai datu izmaiņas atribūtos, kas ir kopīgi mainītai relācijai pirms shēmas izmaiņām un pēc visām shēmas izmaiņām. Rezultātā tiek atstātas datu izmaiņas, kas notika relācijas atribūtos pirms shēmas izmaiņām, izņemot tos atribūtos, kuri tika izmesti shēmas izmaiņas rezultātā. Ja tiek izmesta kāda relācija, tad visas šīs relācijas datu atjaunināšanas netiek izplatītas. Kad visas izmaiņas ir iepriekš apstrādātas, tad sākumā pārdefinē skata definīciju saskaņā ar šīm izmaiņām katram datu avotam, un tad pielāgo skata datu saturu sekojoši. Ja datu avotā notika tikai datu izmaiņas, tad tās ir apstrādātas līdzīgi [ZR00]. Ja datu avotā tika izdzēsta relācija, tad nekas nav jādara, jo datu atjaunināšana šajā relācijā jau neko nemaina. Ja datu avotā tika izdzēsts atribūts, tad arī nekas nav jādara, jo šīs izmaiņas netiešā veidā tiks izplatītas, pielāgojot skata saturu pēc pārdefinēšanas.

3.5.2. Daudzdimensiju datu noliktavas adaptācija un uzturēšana pēc adaptācijas

Iepriekš aprakstītās metodes materializēto skatu adaptācijai un tālākai uzturēšanai var izmantot arī datu noliktavām, kas ir realizētas ar zvaigznes shēmām relāciju datubāzē. Piemēram, arī zvaigznes shēmas elementiem var definēt parametrus, kas definē, vai elementi ir nepieciešami un vai tie ir aizvietojami [LNR97]; datu avotiem var uzstādīt dažādus ierobežojumus; arī daudzdimensiju shēmās var izmantot idejas par avota atribūtu un relāciju aizvietošanu ar citiem atribūtiem vai relācijām. Tomēr šīs metodes neņem vērā dažus datu noliktavas aspektus, tādus, kā iespējamās izmaiņas ETL procesos. Turklāt iepriekš aprakstītajos risinājumos uzskata, ja datu avotos pievienojas jaunas relācijas vai atribūti, tad nav jāmaina datu noliktava, bet īstenībā šādas izmaiņas datu avotos var izraisīt datu noliktavas shēmas paplašinājumu, jo iespējams parādās jauna noderīga informācija, ko agrāk neņēma vērā.

Vispār datu noliktavas datu avotu shēmu maiņas var apstrādāt dažādi, ne tikai mainot pašas datu noliktavas shēmu, bet arī var neko nemainīt, mainīt datu noliktavas ETL procesus vai modificēt datu noliktavas datu saturu [CMM+03]. Šīs datu noliktavas adaptācijas iespējas arī var izmantot, lai apstrādātu datu avotu shēmas izmaiņas. Sekojošās nodaļās ir aplūkoti iespējami algoritmi, kas risina adaptācijas problēmas.

Datu noliktavas shēmas adaptācija

Datu noliktavas datu avoti attīstās, tāpēc var mainīties datu noliktavas datu avotu shēmas vai datu avoti var kļūt nepieejami vai jauni datu avoti var parādīties. Tas var izraisīt nesaskaņas datu noliktavā. Triviālais risinājums šai problēmai ir manuāli pārveidot datu

noliktavas shēmu. Tas nozīmē, ka datu noliktavas izstrādes procesu jāveic no sākuma, izpētot prasības, izdarot projektējuma lēmumus.

Darbā [Mar00] piedāvā metodes, kas ļauj izvairīties no datu noliktavas izstrādes no sākuma. Šīs metodes balstās uz datu avotu shēmu transformāciju ar *transformācijas primitīvām*, kas veido saistības starp datu noliktavas un datu avotu shēmām un tiek izmantotas ETL procesu ģenerācijai. Lai nodrošinātu datu noliktavas shēmas nepretrunīgumu, tiek definēti datu noliktavas shēmas *invarianti*, kas būtībā ir ierobežojumi, piemēram, references integritātes ierobežojumi, t.i. katrai ārējai atslēgai jābūt atbilstošai primārai atslēgai relācijās, ar kuriem tā saistās.

Darbā [Mar00] izmanto sekojošas transformācijas primitīvas: identitāte (ģenerē noteiktas relācijas kopiju), datu filtrs, laika elementa pievienošana, primārās atslēgas ģenerēšana, ārējas atslēgas atjaunināšana, DD-pievienošana (pievieno relācijai atribūtu, atvasinātu no citām relācijām), atribūta pievienošana, hierarhijas roll-up operācija (datu apkopošana pēc viena relācijas atribūta), agregācijas ģenerēšana (atribūtu izņemšana no faktu tabulas, lai apkopotu mērījumus), datu masīva ģenerēšana (veido relāciju, kuras kolonnas ir citas relācijas noteikta atribūta vērtības), dalīšana (sadala relāciju divās sadaļās), hierarhijas ģenerēšana, minidimensijas izdalīšana (izņem atribūtu kopu no vienas relācijas un veido otro no tiem), jaunais dimensiju krustojums (dažādu ar faktu tabulu saistītu dimensiju komplektu veidošana, t.i. kādu dimensiju pievienošana vai atvienošana, lai iegūtu mērījuma datus dažādos griezumos).

Katrai iepriekš minētai transformācijas primitīvai ir dota specifikācija, kas satur nepieciešamo avota shēmu un citus argumentus, rezultātā iegūto shēmu un darbības, kas jāveic, lai izpildītu primitīvu, kas vairumā gadījumu ir SQL vaicājums.

Kad datu noliktavas projektētājs būvē datu noliktavas shēmu, viņš specificē to ar transformācijas primitīvām no datu avotu shēmām. Kad mainās datu avotu shēma, informāciju šajā transformācijas specifikācijā izmanto, lai adaptētu datu noliktavas shēmu un ETL procesus. Atbalstītās izmaiņas datu avotu shēmās ir atribūta pārsaukšana, pievienošana un dzēšana, relācijas atslēgas maiņa, relācijas pārsaukšana, pievienošana un dzēšana.

Datu avotu shēmas maiņas gadījumā jāveic vairākas darbības, lai noskaidrotu datu noliktavas izmaiņas. Katram datu noliktavas elementam (atribūtam) aprēķina detalizētu transformācijas specifikāciju, kas sastāv no pamata operācijām, tādām kā relācijas pievienošana/dzēšana, atribūtu kopas pievienošana/dzēšana/kopēšana, primāras vai ārējas atslēgas pievienošana/dzēšana. No šīs specifikācijas noskaidro, no kuriem datu avota shēmas elementiem ir atkarīgs datu noliktavas elements un kādā veidā (piemēram, datu noliktavas atribūts tiek iegūts no vairākiem datu avotu atribūtiem ar funkciju). Iespējamās atkarības ir sekojošas: nav atkarības, nokopēts, izmantots aprēķināšanā, nepieciešams aprēķināšanai.

Tālāk jāveic adaptācijas darbības. Ja datu avotā tiek pārsaukts atribūts, datu noliktavas shēma nav jāmaina, jāmaina tikai datu iegūšanas process.

Ja datu avotā tiek pievienots jauns atribūts, datu noliktavas izstrādātājam jālemj, (1) vai šis jaunais atribūts ir noderīgs datu noliktavai, (2) kādā veidā to vajag pievienot datu noliktavas shēmai, (3) vai jaunās datu noliktavas struktūras aizvieto kādas esošas struktūras. Ja pirmā jautājuma atbilde ir 'Nē', tad datu noliktava nemainās. Citā gadījumā izstrādātājam

jāspecificē ar transformācijas primitīvām, kā var iegūt jaunās datu noliktavas struktūras. Ja trešā jautājuma atbilde ir 'Jā', liekās datu noliktavas struktūras tiek izmestas.

Ja datu avotā tiek izdzēsts atribūts, datu noliktavas izmaiņas ir atkarīgas no atkarībām, kas eksistē starp datu noliktavas elementiem un izdzēsto datu avotu atribūtu. (1) Ja atribūtam ir kopija datu noliktavā, tad jāizņem atbilstošie datu noliktavas elementi no shēmas, vai jāpielīdzina tos null vērtībai, un jāmaina transformācijas specifikācija. (2) Ja atribūts ir izmantots atvasināta datu noliktavas atribūta izrēķināšanā, tad ir divas iespējamās darbības. Pirmkārt, var izdzēst atvasinātu atribūtu no datu noliktavas. Otrkārt, var mainīt izrēķināšanas funkciju. Pirmajā gadījumā tiek mainīta datu noliktavas shēma un transformācijas specifikācija, otrajā gadījumā – tikai transformācijas specifikācija. (3) Ja atribūts ir nepieciešams kāda datu noliktavas atribūta izrēķināšanai, tad tas atribūts jāizņem no datu noliktavas shēmas, jo 'nepieciešams' nozīmē, ka šis atribūts darbojas kā savienojuma atribūts starp avota un datu noliktavas relācijām.

Ja datu avotā mainās primārās atslēgas atribūts no 'veca' atribūta uz 'jaunu' atribūtu, tad datu noliktavas izmaiņas ir atkarīgas no atkarībām, kas eksistē starp datu noliktavas elementiem un mainīto atslēgu. (1) Ja kāds datu noliktavas atribūts ir 'veca' datu avota atslēgas atribūta kopija, tad ir divas iespējas. (i) Ja šis datu noliktavas atribūts ir datu noliktavas relācijas primārā atslēga, datu noliktavas shēma jāadaptē, mainot primāro atslēgu, lai tā atbilstu 'jaunai' datu avota atslēgai, ja jaunas avota atslēgas atribūta nav datu noliktavā, tas jāpievieno. (ii) Ja 'vecs' avota atslēgas atribūts ir nokopēts datu noliktavas relācijas ārējā atslēgā, tad datu noliktavas atribūts, kas atbilst 'vecam' avota atslēgas atribūtam, jāmaina uz citu datu noliktavas atribūtu, kas atbilst 'jaunam' avota atslēgas atribūtam, un transformācijas specifikācijā jānorāda jauns datu iegūšanas veids. (2) Ja 'vecs' avota atslēgas atribūts ir izmantots datu noliktavas atribūtu izrēķināšanā, tad nekas nav jādara, jo izmaiņas neietekmēs datu noliktavu. (3) Ja 'vecs' avota atslēgas atribūts ir nepieciešams datu noliktavas atribūta izrēķināšanai, tad var vai nu izdzēst šo atvasināto datu noliktavas atribūtu, vai aizvietot transformācijas specifikācijā nepieciešamo atribūtu ar 'jauno' avota atslēgas atribūtu, ņemot vērā atbilstošas savienojuma datu noliktavas relācijas atslēgas maiņu.

Pēc datu noliktavas transformācijas pēc datu avotu izmaiņām, jāpārbauda ierobežojumu (rakstā 'invariantu') izpildīšanās. Nepieciešamības gadījumā, jāveic darbības ierobežojumu izpildīšanai, piemēram, ja no relācijas izņem avota relācijas atslēgu, tad jāizdzēš arī visi citi atribūti, kas ir nokopēti no avota relācijas.

Kad jauna datu noliktavas shēma ir iegūta, esošie dati jāpārveido atbilstoši jaunai struktūrai. To var izdarīt ar *instances pārvēršanas funkcijām*, kas ir vaicājumi, kas jāizpilda uz veciem datu noliktavas datiem. Dažos gadījumos šīs funkcijas nevar noteikt automātiski, tāpēc izstrādātājam jānodrošina nepieciešamā informācija. Instances pārvēršanas funkcijas ir atkarīgas no veiktām datu noliktavas shēmas izmaiņām.

Rakstā [MP03] tiek piedāvāta līdzīga pieeja datu integrācijai. Šīs pieejas pamatā ir shēmu transformācijas primitīvas, kas definē kādā veidā globālā shēma tiek iegūta no lokālām shēmām. Datu noliktavu kontekstā par globālo shēmu var uzskatīt datu noliktavas shēmu, un par lokālām shēmām var uzskatīt datu avotu shēmas. No šīm transformācijām ir iespējams izsecināt, kā lokālās shēmas varētu būt iegūtas no globālās shēmas. Tiek apskatīta globālās un

arī lokālo shēmu evolūcija. Shēmu adaptācijai, līdzīgi kā arī iepriekš aprakstītajā pieejā, tiek izmantota informācija par transformācijām.

ETL procesu adaptācija

Dati no datu avotiem tiek iegūti, pārveidoti un ielādēti datu noliktavā ar ETL procesiem. Tāpēc viens no risinājumiem datu noliktavas adaptācijai ir pārveidot ETL procesus, kurus ietekmējušas datu avotu shēmas izmaiņas.

ETL procesi ir atbildīgi arī par datu avotu shēmu izmaiņu atpazīšanu [CMM+03]. To var izdarīt, piemēram, analizējot žurnāla failus, ja datu avotam ir šīs iespējas, veicot 'aptaujas', t.i. izpildot speciālus vaicājumus datu avotiem, lai noskaidrotu izmaiņas, vai izstrādāt speciālas programmas, kas izmanto citus līdzekļus. Piemēram, datu bāzes pārvaldības sistēmu datu avotos var izstrādāt trigerus, kas automātiski paziņo par datu avotā notikušajām izmaiņām, kad ir izpildītas kādas datu definēšanas operācijas, piemēram, tabulas izveidošana, kolonnas dzēšana utt.

Kad izmaiņas ir atpazītas, ETL procesus vajag atbilstoši adaptēt, lai saglabāt tā spējas izpildīt vaicājumus uz datu avotiem, pārveidot vaicājumu rezultātus un atjaunināt datu noliktavu, veikt datu attīrīšanu un transformāciju.

Rakstā [VMP03] apskata kartējumu (mapping) adaptāciju pēc datu avotu shēmas maiņas. Šeit kartējums definē kādā veidā mērķa shēma (datu noliktavas shēma) ir iegūta no avota shēmas. Ir apskatīti kartējumi starp relāciju un XML shēmām. Tiek piedāvāts algoritms, kas atpazīst kartējumus, ko ietekmē izmaiņas avotā, un ģenerē kartējumu pārrakstīšanas variantus, kas ir saskanīgi ar iesaistītām shēmām. Apstrādājamās izmaiņas avota shēmās ir sadalītas trīs grupās: ierobežojumu izmaiņas (jauna savienojuma ierobežojuma pievienošana vai dzēšana), shēmas mazināšana vai paplašināšana (piemēram, atribūta pievienošana relāciju shēmai vai elementa dzēšana no XML shēmas) un shēmas pārstrukturēšana (shēmas elementu pārsaukšana, kopēšana vai pārvietošana).

Dažreiz nav nepieciešams mainīt visu ETL procesu, jo izmaiņas datu avotu shēmā var ietekmēt tikai vienu šī procesa daļu, piemēram, datu iegūšanu (kad jāadaptē iesaiņotāju), datu pārveidošanu vai ielādi (kad jāadaptē integratorus).

3.5.3. Datu noliktavas adaptācijas un uzturēšanas pēc adaptācijas pieeju analīze

Šajā 3.5. apakšnodaļā tika aprakstītas pieejas datu noliktavas adaptācijai un turpmākai uzturēšanai pēc izmaiņām datu avotos materializētu skatu un daudzdimensiju datu noliktavu realizācijām. Materializētu skatu gadījumā šīs problēmas pēc būtības nozīmē, kā pārrakstīt skata definīcijas vaicājumu, kuru ietekmē datu avota izmaiņas, un kā atjaunināt skata datus pēc izmaiņām, lai nepārrēķinātu visu skatu. Ar materializētu skatu adaptācijas problēmām datu noliktavās pārsvarā nodarbojās viens autoru kolektīvs, kuru risinājumi tika apskatīti šajā apakšnodaļā. Tāpat kā datu noliktavas pielāgošanas problēmu risināšanu materializētiem skatiem (apakšnodaļa 3.4.1.), arī šajā apakšnodaļā aprakstītās pieejas var izmantot tikai gadījumos, kad datu noliktavas tabulas ir definētas kā materializēti skati uz datu avotu tabulām, t.i. kad datu avoti ir relāciju datu bāzes. Dažas idejas materializētu skatu adaptācijai var pielietot arī daudzdimensiju datu noliktavām, piemēram, izmaiņu noteikšanu datu avotos ar iesaiņotāju palīdzību.

Daudzdimensiju datu noliktavu adaptācijai ir piedāvāti risinājumi, kas izmanto informāciju par ETL procesiem, lai atbilstoši avotu izmaiņām adaptētu datu noliktavas shēmu. Šajos risinājumos ir atbalstītas vairākas iespējamās izmaiņas, kas var notikt datu avotā. Katrai izmaiņai ir labi nodefinēts process, kas jāveic automātiski vai automatizēti, lai realizētu izmaiņu. Bet šie risinājumi ir vairāk vērsti uz datu integrācijas problēmu risināšanu un neņem vērā daudzdimensiju shēmas elementu atšķirības, piemēram, adaptācijas process varētu būt dažāds atkarībā no tā, vai ir jāadaptē dimensija vai faktu tabula.

Citi risinājumi adaptē tikai ETL procesus un nemaina datu noliktavas shēmu. ETL procesu adaptācijas risinājumi arī nav ideāli, jo jaunu datu struktūru parādīšanās gadījumā (piemēram, jaunas tabulas vai tabulas kolonnas izveidošana) tie nemaina datu noliktavu, jo ETL procesi var turpināt strādāt bez kļūdām. Bet šī jaunā informācija varētu būt noderīga datu noliktavai, un tādēļ tā būtu jāpievieno eksistējošai datu noliktavas shēmai.

Risinājumi daudzdimensiju datu noliktavas adaptācijai apskata tikai problēmas, kas saistītas ar datu noliktavas adaptāciju pēc izmaiņām datu avotos, bet neapraksta kādas izmaiņas ir jāveic eksistējošos datu noliktavas datos.

3.6. Datu noliktavas versiju pieeja

Iepriekš aprakstītas pieejas datu noliktavas shēmas evolūcijas apstrādei atbalsta tikai vienu datu noliktavas shēmu un tās instanci. Kad izmaiņas ir izplatītas datu noliktavas shēmā, visi dati šajā shēmā tiek pārveidoti, lai atbilstu jaunai struktūrai. Tas var izraisīt vēstures zudumu, jo var pazust iepriekš pieejami datu noliktavas dati.

Vairākos rakstos piedāvā datu bāzes shēmas versiju pieeju shēmas evolūcijas problēmas atrisināšanai. Shēmu versiju izmantošana (versioning) nozīmē, ka datu bāzes sistēma nodrošina pieeju pie visiem datiem, gan pagātnes, gan nākotnes, caur lietotāja definētām versiju saskarnēm [Rod95]. Šīs pieejas praktiski atbalsta, ka pēc izmaiņām datu noliktavas shēmā tiek veidotas shēmas versijas, kas atspoguļo darījumprasības, kas bija spēkā noteiktajā laika periodā, un šo laika periodu sauc par versijas derīguma periodu. Literatūrā datu noliktavas shēmas versiju pieejas ir izmantotas tikai daudzversiju datu noliktavas evolūcijas problēmu risināšanai.

3.6.1. Daudzdimensiju datu noliktavas shēmas versijas

Atšķir divu veidu versiju pieejas [Rod95]. *Daļēja shēmas versiju izmantošana* nozīmē, ka lietotājs var veikt izmaiņas tikai vienai shēmas versijai (parasti pēdējai). *Pilna shēmas versiju izmantošana* nozīmē, ka lietotājs var modificēt visas shēmas versijas.

Rakstā [EKM02] apraksta metadatu modeli, kas atbalsta versiju izmantošanu datu noliktavām. Šis modelis atbalsta ne tikai zvaigznes shēmas, bet arī vairāku zvaigznes shēmu apvienojumus – zvaigznājus. Saskaņā ar šo modeli, pamatā datu noliktavas shēma sastāv no faktu versijām (rakstā tiek lietots kuba versijas jēdziens), dimensijām, dimensiju līmeņiem (rakstā tiek lietots kategorijas jēdziens) un dimensiju locekļiem, kas ir konkrētas dimensijas līmeņu atribūtu vērtības. Katram shēmas elementam ir laika intervāls, kad tas ir spēkā.

Dimensijas līmeņi un dimensiju locekļi ir sakārtoti hierarhijās. Tādēļ, ka hierarhijas var mainīties laikā, hierarhijām arī eksistē versijas metadatos, kā arī attiecībām starp dimensijām un līmeņiem, un līmeņiem un dimensiju locekļiem.

Katrai fakta versijai var būt iepriekšējā un nākošā versija, un katra versija ir spēkā noteiktā laika periodā. Datus no vienas fakta versijas var pārveidot datos citā (iepriekšējā vai nākošā) versijā ar speciālu transformācijas funkciju. Šī funkcija pārveido faktu tabulas šūnu vērtības no precīzi vienas versijas uz nākošo vai iepriekšējo versiju. Turklāt var eksistēt vairākas transformācijas funkcijas, kas pārveido datus no vienas versijas uz nākošo vai iepriekšējo. Transformācijas funkcijas tiek glabātas matricas veidā. Lai pārveidotu datus no vienas fakta versijas citā, kas nav iepriekšējā vai nākošā, tad vajag sareizināt visas transformācijas matricas, kas eksistē starp visām pēc kārtas esošām faktu versijām.

Iepriekš aprakstīto datu noliktavas modeļa integritāti nodrošina ierobežojumu kopa, piemēram, dimensija var tikt piesaistīta faktu tabulai tikai tajā laika periodā, kas ir abu tabulu laika intervālu kopīgs apakšintervāls, līdzīgi arī ar citām saistībām.

Arī šajā modelī katram dimensijas loceklim ir aprēķināta *stabilu intervālu kopa*, kas uzlabo vaicājumu veiktspēju. Šis intervāls nosaka laika periodu, kad nebija tādu citu komponentu modifikāciju, kas ietekmē vaicājumu rezultātus ar doto dimensijas locekli. Un tas nozīmē, ka dati, kas tiek iegūti ar vaicājumiem uz stabilu intervālu, nav jāpārveido. Zemāka līmeņa dimensiju locekļiem hierarhijā šīs intervāls vienkārši ir vienāds ar laika periodu, kad šīs loceklis ir spēkā. Citiem locekļiem, kam eksistē zemāka līmeņa locekļi, t.i. 'bērnu' locekļi, šie intervāli ir iegūti no laika periodiem, kad bija spēkā locekļa 'bērnu' locekļi un hierarhijas attiecība starp dimensiju līmeņiem, kam atbilst šie locekļi.

Iepriekš minētajā rakstā ir aplūkots tikai laicīgas datu noliktavas metadatu modelis, bet nav aprakstīts kādā veidā var izpildīt vaicājumus uz šādu datu noliktavu, kas aptver vairākas shēmu versijas. Vaicājumu izpilde daudzversiju datu noliktavām ir aplūkota rakstā [MW04]. Šī raksta vaicājumu izpildes pieeja ir balstīta uz metadatu pārvaldības risinājumu, kas tiek piedāvāts rakstā [WB05].

Šeit [WB05] daudzversiju datu noliktava sastāv no shēmas versijām un šo versiju instancēm. Tiek atšķirtas divu veidu versijas. *Reālas* versijas ir izveidotas, lai atspoguļotu izmaiņas reālā biznesa vidē. *Alternatīvas* versijas ir veidotas simulācijas nolūkiem. Vairākas alternatīvas versijas var tikt izveidotas no vienas reālas versijas. Katrai versijai ir noteikts laika intervāls, kad tā ir spēkā.

Tiek apskatītas divu veidu izmaiņas: 12 *shēmas izmaiņas operācijas*, piemēram, jaunas dimensiju tabulas izveide, un 3 *dimensiju struktūras izmaiņas operācijas*, piemēram, vairāku dimensijas instanču apvienošana. Shēmas izmaiņu operācijas ir vienādas ar operācijām, kas ir definētas darbā [Bla00], kas tika aprakstītas apakšnodaļā 3.4.2., bet rakstā [WB05] šo operāciju rezultātā tiek veidota jauna datu noliktavas shēmas versija, nevis tiek mainīta esošā shēma. Dažas shēmas izmaiņas operācijas, piemēram, jauna atribūta pievienošana dimensijai, neizraisa izmaiņas lietotāju vaicājumos vai datu zudumu, tādēļ šīs izmaiņas var tikt veiktas, neizveidojot jaunu versiju (atkarībā no administratora izvēles).

Vaicājumi uz šāda veida datu noliktavu tiek izpildīti trijos etapos: (1) vaicājums tiek sadalīts daļējos vaicājumus, kur katrs vaicājums izpildāms uz vienas versijas; (2) katrs daļējs vaicājums tiek izpildīts uz atbilstošas versijas; (3) katra daļēja vaicājuma rezultāti tiek

piedāvāti lietotājam atsevišķi kopā ar versijas informāciju un metadatu informāciju. Ja lietotājs izvēlas apkopot rezultātus no visām versijām, tad tas tiek veikts ar administratora definētām transformācijas funkcijām.

Rakstā [WB05] ir arī apskatīts jautājums par izmaiņu noteikšanu datu avotos un to apstrādi un datu noliktavas shēmas adaptāciju šīm izmaiņām.

Darbā [SNP05] tiek dota definīcija daudzdimensiju modelim, kas atbalsta versijas. Šī definīcija ir ļoti līdzīga tai, kas dota rakstā [Bla00], bet atšķiras tikai ar to, ka atbalsta shēmu versijas. Šajā darbā ir arī formalizētas versiju evolūcijas operācijas, kas rezultātā maina datu noliktavas shēmas elementus. Šīs versiju evolūcijas operācijas ir arī līdzīgas operācijām, aprakstītām rakstā [Bla00], kā arī formālas modifikācijas datu noliktavas shēmā, kas notiek evolūcijas operāciju rezultātā, tiek definētas līdzīgi rakstam [Bla00] ar vienu atšķirību, kas ir jaunas versijas izveidošana modifikācijas rezultātā. Vairākas versiju evolūcijas operācijas ir apvienotas versiju funkcijā, kas vai nu veido jaunu versiju vai maina esošu datu noliktavas shēmu.

Cita pieeja shēmu versiju izmantošanai ir aplūkota [GLR+04] un [GLR+06]. Šeit kopā ar shēmas versijām arī tiek glabātas *palielinātas* (augmented) shēmas. Kad notiek kaut kādas izmaiņas shēmā, tad vispirms tiek iegūta jauna shēmas versija un visām iepriekšējām versijām tiek aprēķinātas palielinātas shēmas, kuras tiek aizpildītas ar datiem. Pēc būtības palielināta shēma ir iegūta no „parastas shēmas", pievienojot visus jaunos atribūtus un funkcionālās atkarības (hierarhijas) starp tiem. Vispār administrators var izvēlēties izmaiņas, kas tiks izplatītas palielinātā shēmā, tādēļ panāk efektīvāku datu apjoma pārvaldi.

Šajā rakstā ir definētas izmaiņas: atribūta pievienošana vai dzēšana, funkcionālas atkarības pievienošana vai dzēšana. Katru izmaiņu izplata dažādi, lai iegūtu palielinātas versijas ar veciem datiem. Kad tiek pievienots jauns mērījuma atribūts, administrators sagādā vērtības šīm mērījumam, parasti, izrēķinot tās no eksistējošiem mērījumiem. Kad tiek pievienota jauna dimensija, administrators var veikt detalizēšanu, kad no apkopotiem datiem iespējams iegūt detalizētākus datus. Kad tiek pievienots atvasināts mērījums, tā vērtības tiek aprēķinātas no citu mērījumu vērtībām. Ja tiek pievienots aprakstošs atribūts, tad administratoram jāsagādā vērtības šim jaunajam atribūtam tādā veidā, lai visas funkcionālas atkarības, kur šis atribūts piedalās, saglabātos. Kad tiek pievienota jauna funkcionāla atkarība, tad vienkārši nepieciešams pārbaudīt, vai tā izpildās.

Kad tiek izpildīts vaicājums, kas ietver vairākas versijas, tad sistēma var rīkoties divos veidos. Tā var atrast maksimālu shēmas elementu kopu, kas paliek visās vaicājumā izmantotās versijās, un izpildīt vaicājumu uz tiem. Vai tā var izmantot palielinātas shēmas, lai iegūtu lielāku šo elementu kopu. Tādā veidā iegūta elementu kopa arī definē iespējamos apkopošanas un detalizēšanas virzienus.

3.6.2. Daudzdimensiju datu noliktavas shēmas versiju pieeju analīze

Daudzversiju datu noliktavas ir diezgan jauns virziens datu noliktavas evolūcijas problēmu risināšanai. Datu noliktavas versiju pieeju var izmantot gan datu noliktavas adaptācijas, gan pielāgošanas problēmu risināšanai. Vairāku shēmas versiju izmantošanas gadījumā netiek pazaudēta ne tikai datu, bet arī shēmas izmaiņu vēsture, tāpēc versiju pieejas šķiet piemērotākas evolūcijas atbalstam. Bet vairāku shēmu veidošana un uzturēšana prasa

lielākas izmaksas un izraisa papildus uzdevumus, tādus kā versijas veidošanas nepieciešamības noteikšana, datu noliktavas elementu izvēle, kam tiek veidotas versijas, vairāku versiju fiziskā glabāšana un uzturēšana, operāciju noteikšana, kuru rezultātā tiek veidotas jaunas shēmas versijas, esošo datu noliktavas datu transformēšana no vienas versijas uz citu, un galvenais uzdevums vaicājumu izpildīšana uz daudzversiju datu noliktavām. Šajā apakšnodaļā aprakstītas pieejas atrisina tikai daļu no šiem uzdevumiem. Šīm pieejām ir neatrisinātas tādas problēmas kā, piemēram, daudzversiju datu noliktavas shēmu glabāšana relāciju datu bāzē, lietotāju atskaišu definēšana un vaicājumu konstruēšana un eksistējošo atskaišu adaptēšana, lai atspoguļotu vairāku versiju esamību. Kaut gan vairākums pieeju ir balstītas uz metadatiem, kas apraksta datu noliktavas versijas, nevienā no pieejām nav izmantoti metadati, kas atbilst kādam metadatu standartam, piemēram, Common Warehouse Metamodel (CWM) standartam [OMG02], kas sagādā grūtības risinājumu integrācijai gan ar esošām datu noliktavas sistēmām, gan salīdzināšanai ar citu autoru pieejām.

3.7. Nodaļas secinājumi

Šajā nodaļā 3. tika aprakstīti esošie risinājumi datu noliktavas evolūcijas problēmām. Aprakstītajiem risinājumiem ir gan savas priekšrocības, gan trūkumi, kas ir analizēti atsevišķi katras evolūcijas problēmas risinājumiem apakšnodaļās 3.3.3., 3.4.3., 3.5.3. un 3.6.2.

Datu noliktavas uzturēšanas problēma ir labi izpētīta un eksistē gan vairākas metodes, gan arī komerciāli rīki datu iegūšanas, pārveidošanas un ielādes procesu definēšanai un izpildei. Saistībā ar uzturēšanas problēmu zinātniskajā literatūrā risina tādus uzdevumus, kā nestrukturētu datu avotu izmantošana datu noliktavai, izmaiņu izsekošana datu avotos u.c.

Datu noliktavas pielāgošanas problēmu risinājumi parasti iekļauj datu noliktavas shēmas formālo definīciju, kas dažādiem risinājumiem var būt dažāda, un datu noliktavas shēmas izmaiņu izplatīšanu šajā formālajā definīcijā. Ne visos pielāgošanas problēmas risinājumos tiek apskatītas datu modifikācijas, kas jāveic pēc datu noliktavas shēmas pielāgošanas. Nav arī risinājumu datu noliktavas atskaišu pielāgošanai pēc datu noliktavas shēmas izmaiņām.

Datu noliktavas adaptācijas problēmu risinājumi ir balstīti uz datu iegūšanas, pārveidošanas un ielādes procesu specifikāciju, kas tiek izmantota izmaiņu gadījumā datu avotos, lai noskaidrotu nepieciešamās modifikācijas, kas jāveic datu noliktavas shēmā. Šajos risinājumos netiek ņemti vērā daudzdimensiju datu noliktavas elementi, piemēram, dimensijas, faktu tabulas, hierarhijas. Nav apskatītas arī izmaiņas, kas jāveic datu noliktavas datos pēc adaptācijas, gadījumā, kad datu noliktava ir reprezentēta ar daudzdimensiju modeli.

Kopumā visi risinājumi apskata tikai viena veida evolūcijas problēmas, piemēram, uzturēšanu, pielāgošanu vai adaptāciju. Bet datu noliktavai ir jāspēj pielāgoties visām problēmām, t.i. visas evolūcijas problēmas ir jārisina kopā. Dažādu piedāvāto risinājumu vienlaicīga izmantošana ir apgrūtināta, jo tām ir dažādas datu struktūras un izmantotās metodes.

Vairāki risinājumi uzskata datu noliktavu par materializēto skatu kopu uz avota datiem. Bet bieži izstrādājot datu noliktavu, galvenās pūles tiek veltītas sarežģītiem datu iegūšanas, pārveidošanas un ielādes (ETL) procesiem. Materializēto skatu risinājumi neņem

vērā ETL procesus un nepiedāvā ETL procesu adaptācijas metodes. Tas nozīmē, ka materializēto skatu risinājumus ir iespējams izmantot tikai ļoti ierobežotai vienkāršāku datu noliktavu kopai. Vairākums rakstu par materializētu skatu uzturēšanu, pielāgošanu un adaptāciju tika publicēti diezgan sen (pirms 2000. gada), tas liecina par to, ka šī pieeja tagad nav pārāk aktuālā.

Apakšnodaļās 3.4. un 3.5. apskatīti risinājumi, kas adaptē datu noliktavas shēmu saskaņā ar notikušām izmaiņām darījumprasībās un datu avotos, var izraisīt vēstures zudumus un neatspoguļo datu noliktavas evolūciju, jo šajos risinājumos vienmēr pastāv tikai viena aktuāla datu noliktavas versija. Bet datu noliktavām ir jāuztur vēsturiski korekta informācija, tādēļ ir korekti izmantot daudzversiju datu noliktavu pieeju. Tāpēc šajā darbā ir izvēlēta daudzversiju pieeja datu noliktavas evolūcijas atbalstam. Šī pieeja ir jaunākā, salīdzinot ar citām literatūrā aprakstītajām pieejām datu noliktavu jomā, jo pirmā publikācija par shēmas versiju izmantošanu datu noliktavās parādījās tikai 2002. gadā. Daudzversiju pieeja datu noliktavu evolūcijas problēmu risināšanai joprojām tiek attīstīta, piemēram, 2009. gadā notika pirmais seminārs par datu noliktavas evolūcijas tēmu ADBIS konferences ietvaros. Seminārā par vienu no aktuālākiem izpētes objektiem datu noliktavas evolūcijas jomā tika atzīta daudzversiju pieeja.

Apakšnodaļā 3.6. tika aprakstīti esošie risinājumi datu noliktavas evolūcijas problēmām ar daudzversiju pieeju. Šiem risinājumiem ir daži neatrisināti uzdevumi, kas tika apskatīti promocijas darba ietvaros. Piemēram, netiek piedāvātas metodes daudzversiju datu noliktavas realizācijai relāciju datu bāzē. Nav izdomāts arī algoritms lietotāju ekspromtvaicājumu konstruēšanai un izpildīšanai uz vairākām datu noliktavas shēmas versijām, kā arī eksistējošo atskaišu adaptēšanai pēc jauno shēmas versiju izveidošanas. Esošie risinājumi daudzversiju datu noliktavām parasti ir balstīti uz metadatiem, kas apraksta dažādus atsevišķus datu noliktavas shēmas aspektus. Šie izmantotie metadati ir dažādi dažādās pieejās un neatbilst nekādam standartam, kas nozīmē, ka tie nav izmantojami kombinācijā ar citām pieejām un ar eksistējošiem rīkiem.

4. E-STUDIJU DATUVES ATTĪSTĪBA

4.1. Nodaļas nolūks

Šīs nodaļas nolūks ir aprakstīt datuvi, kas tālākās nodaļās tiek izmantota piemēros un kuras attīstības problēmas motivēja darba autori pievērsties datu noliktavas evolūcijas problēmām. Latvijas Universitātes datu noliktavā tika izmantota arhitektūra ar darba apgabalu un datuvēm. Viena no Latvijas Universitātes datu noliktavas datuvēm ir e-studiju datuve, kam ir veltīta šī nodaļa. Šī datuve tika pakāpeniski izstrādāta no 2004. gada. E-studiju datuve visvairāk mainījās, salīdzinājumā ar pārējām Latvijas Universitātes datuvēm. Šīs datuves lietošanas laikā notika vairākas izmaiņas gan datu avotos, gan darījumprasībās, t.i. tā attīstījās. Šīs datuves, kas tiek aprakstīta tālāk, evolūcija ilustrē iepriekšējās nodaļās aprakstītas problēmas.

4.2. E-studiju datuves raksturojums

4.2.1. E-studiju datuves mērķis un vēsture

E-studiju datuve tika izstrādāta kā Latvijas Universitātes datu noliktavas daļa E-universitātes projekta ietvaros. E-universitātes projekts pastāvēja no 2002. līdz 2008. gadam un viens no e-universitātes projekta mērķiem bija nodrošināt studiju procesa atbalstu ar atbilstošu kursu pārvaldības sistēmu un kursu materiāliem. Līdz 2007. gada beigām kursu organizēšanai tika izmantota sistēma WebCT un kopš 2008. gada tiek lietota sistēma Moodle. WebCT un Moodle ir kursu vadības sistēmas, kas tika izstrādātas augstākās izglītības iestādēm e-studiju organizēšanai. Kursus, kuri ir organizēti šajās sistēmās un tiek pasniegti caur Internetu, sauc par e-kursiem.

E-studiju datuves galvenais mērķis ir nodrošināt analīzi e-studiju vides ietekmei uz dažādiem universitātes procesiem. Galvenie analīzes mērķi bija:

- Sistēmas lietošanas vērtēšana;
- Pasniedzēju aktivitātes vērtēšana;
- Komponentu lietošanas vērtēšana;
- Studentu e-kursu izmantošana.

E-studiju datuves primārie lietotāji ir e-studiju administrēšanas darbinieki. Datuves realizācijai ir izvēlēta Oracle vide, jo Latvijas Universitātes datu noliktava ir izveidota Oracle vidē, bet e-studiju datuve ir tās sastāvdaļa.

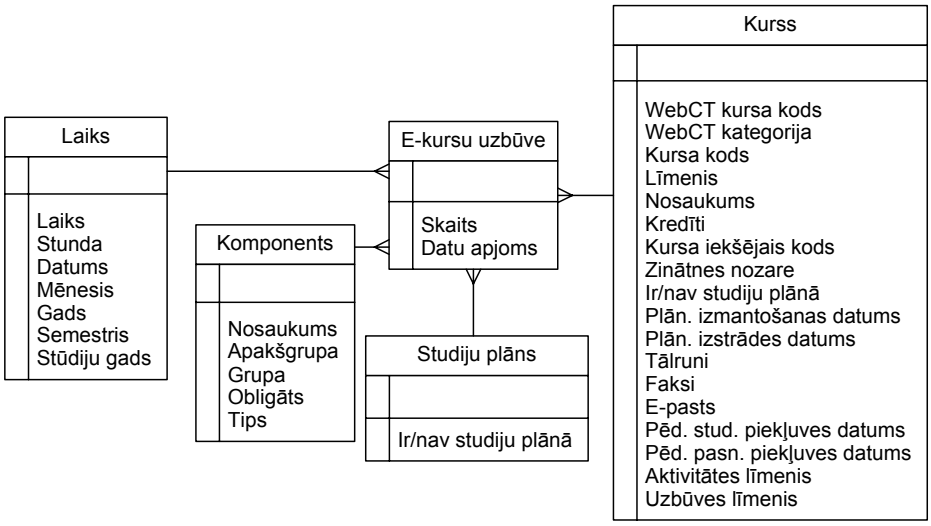
E-studiju datuves darījumprasības ir kursu vadības sistēmu attīstības un izmantošanas datu apkopošana un pieejamība analīzei. Attīstības un izmantošanas dati iekļauj e-kursu uzbūves datus, e-kursu lietošanas statistiku un studentu individuālo statistiku. Datuve dod iespēju analizēt sistēmas lietošanu – vai studenti un pasniedzēji vispār lieto kursu vadības sistēmu, cik daudz un cik regulāri viņi to dara, ar mērķi noskaidrot tālāku e-kursu izstrādes nepieciešamību, arī lai laicīgi noskaidrotu problēmas neaktīvas izmantošanas gadījumā. Var arī vērtēt pasniedzēju aktivitāti, lai noskaidrotu pasniedzēju slodzes pieaugumu, komponentu lietošanu – kurus komponentus izmanto vairāk, vai kursu pārvaldības sistēmas specifiskos

komponentus vai tikai kursu materiālu piegādes iespējas. Nepieciešams arī sekot studentu e-kursu izmantošanai – kā tā ietekmē studentu rezultātus (atzīmes). Studentus var sadalīt grupās pēc sekmēm, analizēt kuras grupas studenti izmanto e-kursus vairāk, kurus komponentus.

Atskaitēm uz e-studiju datuves shēmām tiek lietots atskaišu rīks Oracle Discoverer, kas atļauj definēt un izpildīt ekspromt-atskaites uz datu noliktavas shēmu. Lai to izdarītu, datu noliktavas administratoram ir jādefinē Oracle Discoverer rīkā datu noliktavas shēma metadatu repozitorijā. Datu noliktavas tabulas tiek aprakstītas kā mapes, kas satur elementus, kas atbilst tabulu kolonnām. Papildus tiek definētas saistības starp tabulām atbilstoši ārējo atslēgu saistībām. Lai definētu atskaiti, lietotājam ir jāizvēlas elementi no mapēm, jādefinē atskaišu nosacījumus un kalkulācijas. Balstoties uz metadatiem repozitorijā un atskaites definīciju, Oracle Discoverer ģenerē SQL vaicājumu, kuru rezultāts tiek attēlots lietotājam. E-studiju datuvei tika izveidotas ap 100 atskaites, kas attēlo nepieciešamos datus par e-kursu uzbūvi, lietošanu un lietotāju aktivitāti.

4.2.2. E-studiju datuves modelis

E-studiju datuves modelis [SN05] ir vairākas daudzdimensiju zvaigznes shēmas, kas ir apvienotas ar kopīgām dimensijām. Datuves zvaigznes shēmas attēlo informāciju par e-kursu uzbūvi un izmantošanu. Tās sastāv no trim faktu tabulām E-kursu uzbūve, E-kursu lietošana un Aktivitāte, kas satur mērījumus. Pārējas tabulas ir dimensijas, kas satur aprakstošus datus.



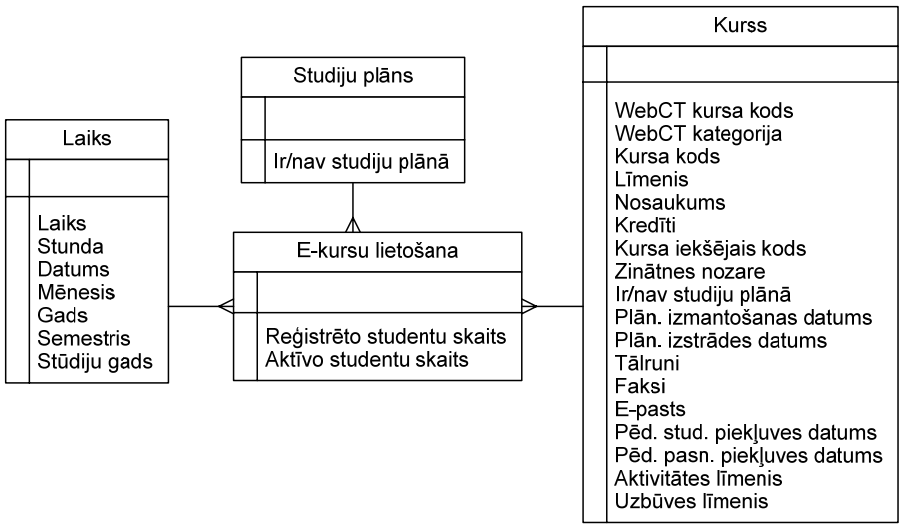
Att. 8. E-kursu uzbūves zvaigznes shēma

Attēlā 8. ir dota e-kursu uzbūves zvaigznes shēma – faktu tabula ar tai atbilstošajām dimensijām. Faktu tabulā *E-kursu uzbūve* tiek uzkrāti dati par e-kursa uzbūvi, t.i. e-kursa komponentu *skaits* un *datu apjoms* komponentam e-kursā uz noteikto laika brīdi. Komponents ir atsevišķs kursu vadības sistēmas rīks, piemēram, pasts, kalendārs, diskusijas u tml. Šī faktu tabula izmanto sekojošas dimensijas:

- *Laiks* ir standarta dimensija datu noliktavām. Tā definē laika brīdi, kurā dažādi faktu tabulu mērījumi bija spēkā. Laika dimensija satur atribūtus *Laiks, Stunda, Datums, Mēnesis* un *Gads*, kas glabā datumu un laiku precizitātē līdz sekundei. Papildus šiem

atribūtiem tika iekļauti arī atribūti *Semestris* un *Studiju gads*, kas glabā akadēmiskā kalendāra semestri (rudens vai pavasara semestris) un studiju gadu (piemēram, 2009./2010. studiju gads).

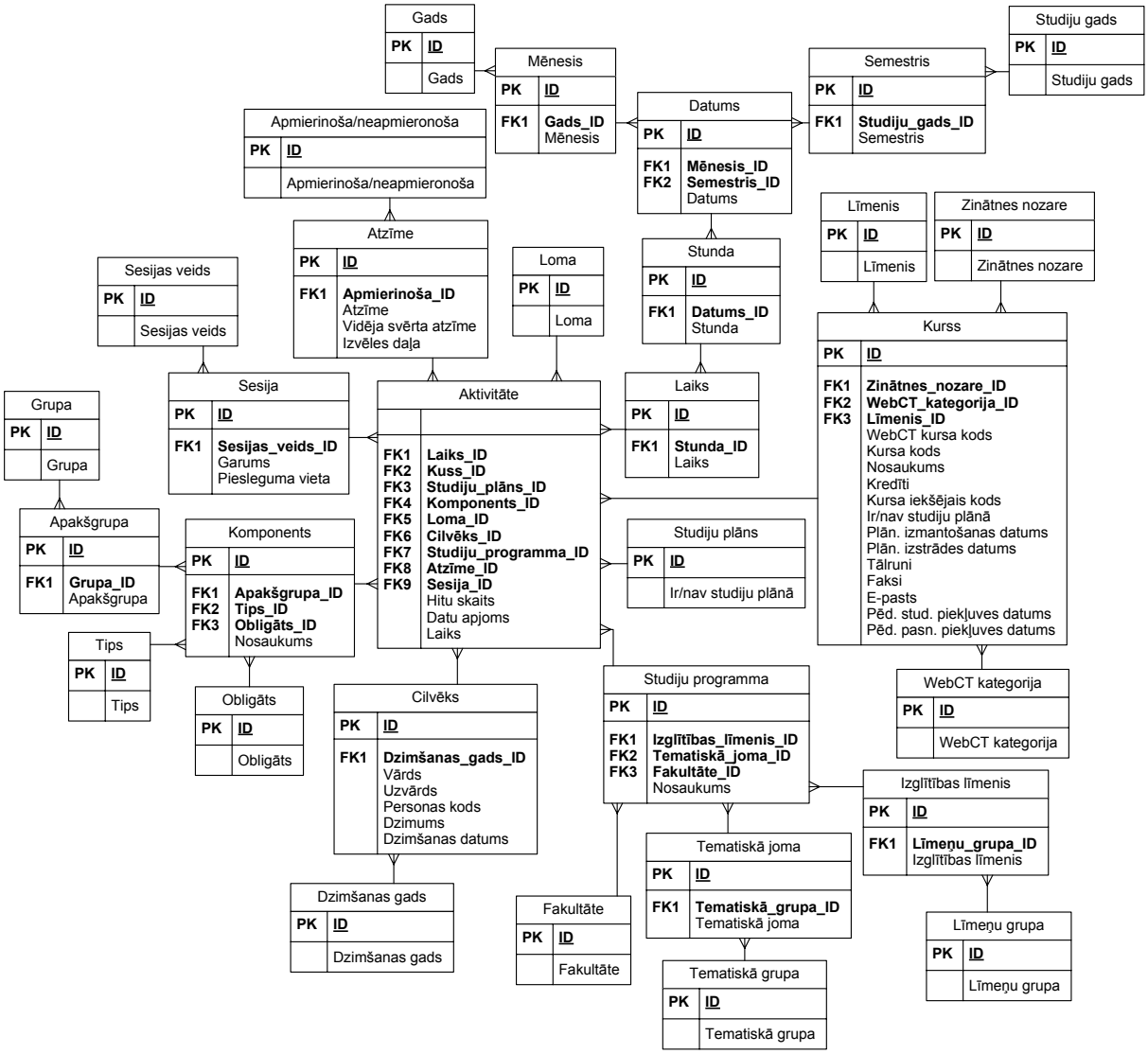
- *Komponents* satur informāciju par visiem kursu vadības sistēmas komponentiem. Atribūts *Obligāts* nosaka, vai komponents ir obligāts e-kursu uzbūvei (prasība e-kursu izstrādei E-Universitātes projektā). *Tips* norāda, vai komponents ir statisks vai dinamisks. Komponents ir statisks, ja tas ir izveidots un netiek mainīts kursa izmantošanas laikā, vai tiek mainīts ļoti reti. *Dinamisks* komponents var tikt mainīts vai izveidots kursa laikā. Komponentu sadalījums pa *grupām* un *apakšgrupām* tika noteikts saskaņā ar kursu vadības sistēmu komponentu klasifikāciju [Web05].
- Dati par visiem kursiem tiek glabāti dimensiju tabulā *Kurss*. Šeit *Līmenis* norāda, kurā studiju gadā ir paredzēts lasīt kursu (bakalaura, maģistra, doktorantūras un profesionālām programmām). *Kredīti* ir kredītpunktu skaits par kursu. *Ir/nav studiju plānā* – vai kurss tiek pasniegts tekošā semestrī. Tika definēti divi *Aktivitātes līmeņi*: 'Aktīvs' un 'Pasīvs'. 'Aktīvi' kursi ir tādi, kur aktīvo studentu skaits (kas vismaz vienu reizi ir pieslēgušies kursam) uz doto laika brīdi ir lielāks par 5% no reģistrēto studentu skaita. 'Pasīvi' kursi ir visi pārējie kursi. *Uzbūves līmeņi* arī ir divi: 'Mācību materiālu piegāde' un 'Uzlabots'. 'Mācību materiālu piegāde' tiek piešķirts kursiem, kas satur tikai mācību materiālus, t.i. lieto kursu vadības sistēmu kā parastu interneta lappusi, un nesatur papildus rīkus, kas ir speciāli e-apmācības videi, tādus kā testus, paš-testus, uzdevumus. Pārējiem kursiem, kas izmanto kursu vadības sistēmas papildiespējas, ir piešķirts līmenis 'Uzlabots'. Atribūtu *Aktivitātes līmenis* un *Uzbūves līmenis* vērtības tiek aprēķinātas datuves atjaunināšanas laikā un var mainīties laika gaitā.
- Dimensiju tabula *Studiju plāns* satur tikai vienu atribūtu *Ir/nav studiju plānā*, kas norāda, ka e-kurss tiek pasniegts noteiktā semestrī. Šīm atribūtam ir 2 vērtības: pasniegts / netiek pasniegts.



Att. 9. Lietošanas zvaigznes shēma

Otrā zvaigznes shēma datu noliktavā ir attēlā 9. redzamā *Lietošanas* zvaigznes shēma. Faktu tabulā *E-kursu lietošana* ir *reģistrēto* un *aktīvo studentu skaits* uz doto laika brīdi.

Reģistrētie studenti ir tie, kas ir reģistrēti e-kursam e-studiju vidē kā lietotāji ar lomu students, *aktīvie studenti* it tie reģistrētie studenti, kas ir kaut reizi ir pieslēgušies kursam. Šī zvaigznes shēma izmanto dimensijas: *E-kurss, Laiks, Studiju plāns,* kas jau iepriekš ir aprakstītas.



Att. 10. Aktivitātes zvaigznes shēma

Aktivitātes zvaigznes shēmā (att. 10.) ir savākta informācija par studentu aktivitātēm e-studiju vidē e-kursu apguves laikā. Šeit faktu tabulā *Aktivitāte* tiek glabāta izmantošanas informācija: *hitu skaits*, *datu apjoms* un *laika periods*, kurā lietotājs ir strādājis ar kursa komponentu. Dimensijas *Laiks, E-kurss, Komponents* un *Studiju plāns* jau tika aprakstītas iepriekš. Šajā zvaigznes shēmā parādās jaunas dimensijas:

- Dimensijā *Cilvēks* ir dati par visiem e-kursu lietotājiem.
- Tabulā *Atzīme* ir studentu sekmes e-kursos un vidēja atzīme par visiem studenta apgūtajiem kursiem.
- Dimensiju tabulas *Loma* atribūts *Loma* norāda lietotāja lomu e-kursā. Sistēmā ir četras lietotāju lomas: *students, dizaineris, pasniedzēja palīgs (teaching assistant), viesis*.
- Studentu studiju programmām ir dimensiju tabula *Studiju programma*.

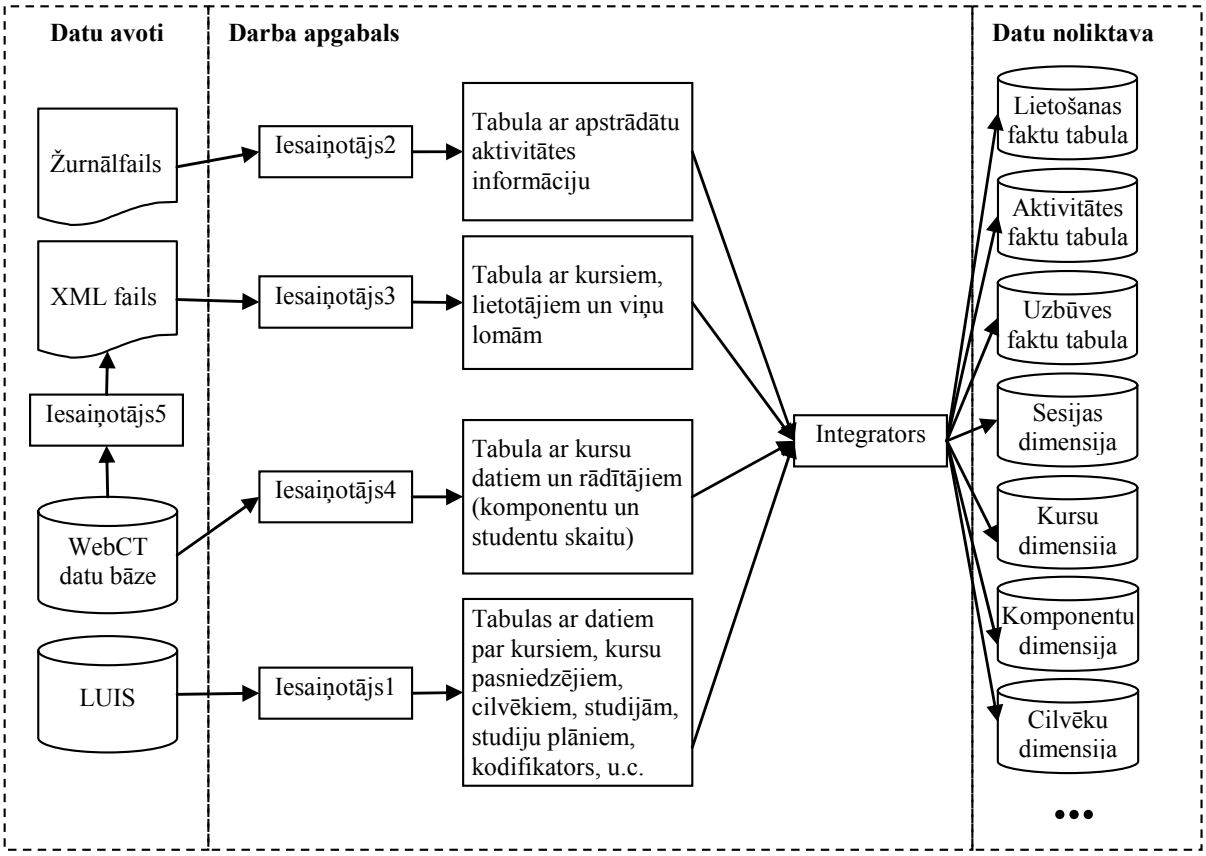
- Vienu pieslēgšanos kursu vadības sistēmai raksturo ieraksts dimensiju tabulā *Sesija.* Atribūta *Sesijas veids* klasificēšana tika izveidota pēc sekojoša principa: Īsa (0-1 minūte); Vidēja (1-10 minūtes); Gara (10-60 minūtes); Ļoti gara (>60 minūtes). *Pieslēguma vieta* (Latvijas Universitātes tīkls vai cita vieta) ir definēta ar IP adresi.

4.3. E-studiju datuves attīstība

4.3.1. E-studiju datuves uzturēšanas problēmas

Lai risinātu E-studiju datuves uzturēšanas problēmu, tika izstrādāti datu iegūšanas, pārveidošanas un ielādes procesi – ETL procesi. Šie procesi tika izpildīti vienu reizi nedēļā un tie atjauninăja datuvi ar datiem, kas apraksta notikumus iepriekšējās nedēļas laikā, t.i. datuves uzturēšanai tika izmantoti atliktas atjaunināšanas algoritmi.

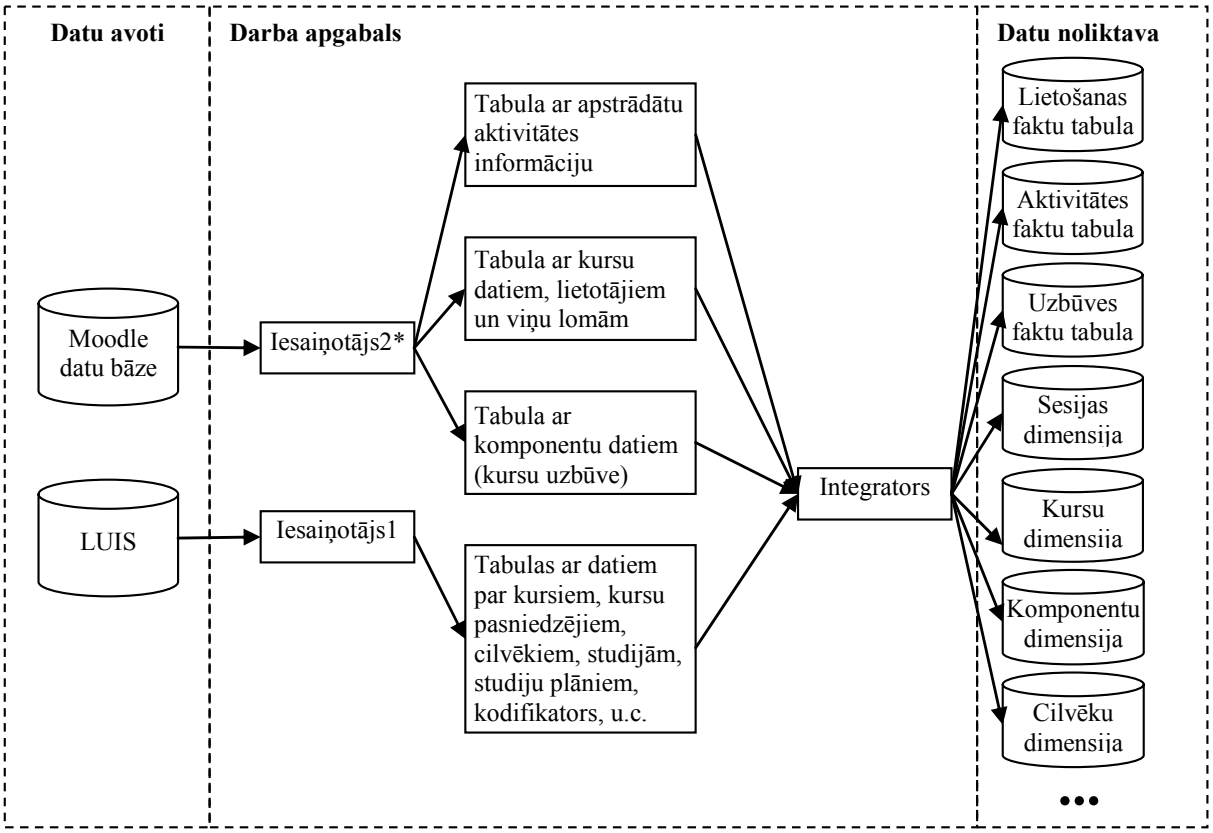
Datuves uzturēšanas laikā tika izmantoti sekojoši datu avoti: Latvijas Universitātes Informatīvā Sistēma (LUIS), WebCT tīmekļa servera žurnāla faili, WebCT iekšējā datu bāze, kuru dati tika izmantoti XML formātā, Moodle iekšējā datu bāze.



Att. 11. Kopējais uzturēšanas process no WebCT datu avotiem

Datuves uzturēšanai vēsturiski tika izmantoti divi procesi. Viens process tika lietots, lai uzturētu e-studiju datuvi, kad Latvijas Universitātē tika lietota kursu vadības sistēma WebCT. Attēlā 11. ir attēlots kopējais datu iegūšanas, pārveidošanas un ielādes process no WebCT datu avotiem un LUIS relāciju datu bāzes tabulām. Šajā procesā tika izmantoti 5 iesaiņotāji, kas ieguva datus no datu avotiem, veica sākotnējo apstrādi un pārnesa datus uz

darba apgabalu, un viens integrators, kas apstrādāja avota datus, attīrīja tos, pārveidoja citā formātā, apvienoja, piekārtoja nepieciešamos dimensiju vai faktu tabulu identifikatorus un ielādēja datus datuvē.

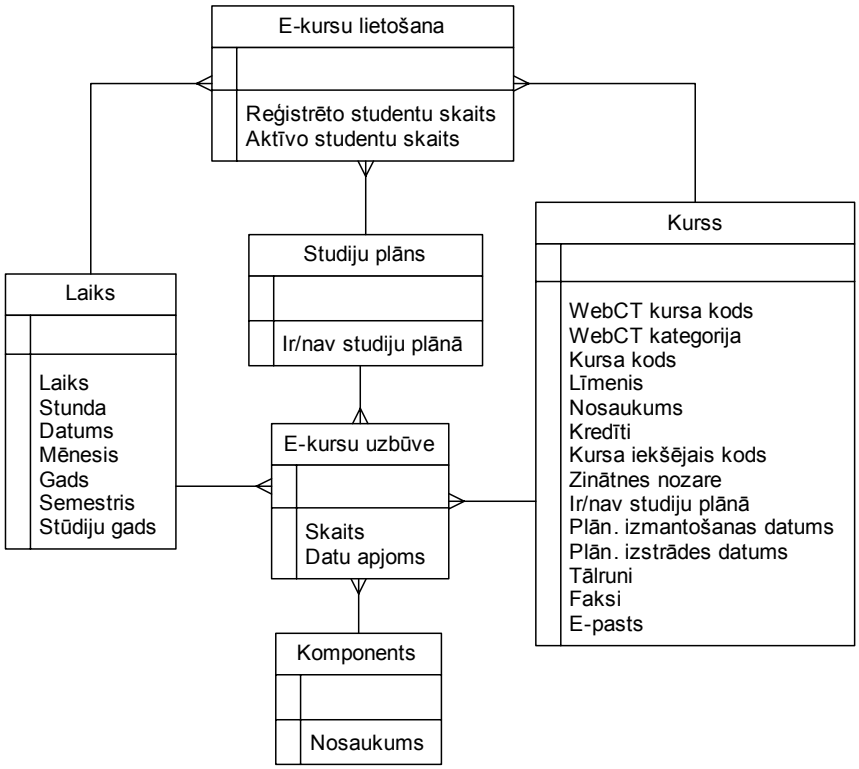


Att. 12. Kopējais uzturēšanas process no Moodle datu avotiem

Pēc pāriešanas uz Moodle kursu vadības sistēmu ETL process tika pārveidots un pašlaik tiek izmantots otrais datuves uzturēšanas process (att. 12.), kur WebCT datu avotu vietā izmanto Moodle iekšējās datu bāzes failus. Otrais process atšķiras ar to, ka tas nesatur iesaiņotājus 2-5, jo visi nepieciešami dati, kas tika iegūti no dažādiem WebCT datu avotiem, tiek pārnesti no viena datu avota – Moodle iekšējas datu bāzes ar vienu iesaiņotāju 2*.

4.3.2. E-studiju datuves pielāgošanas problēmas.

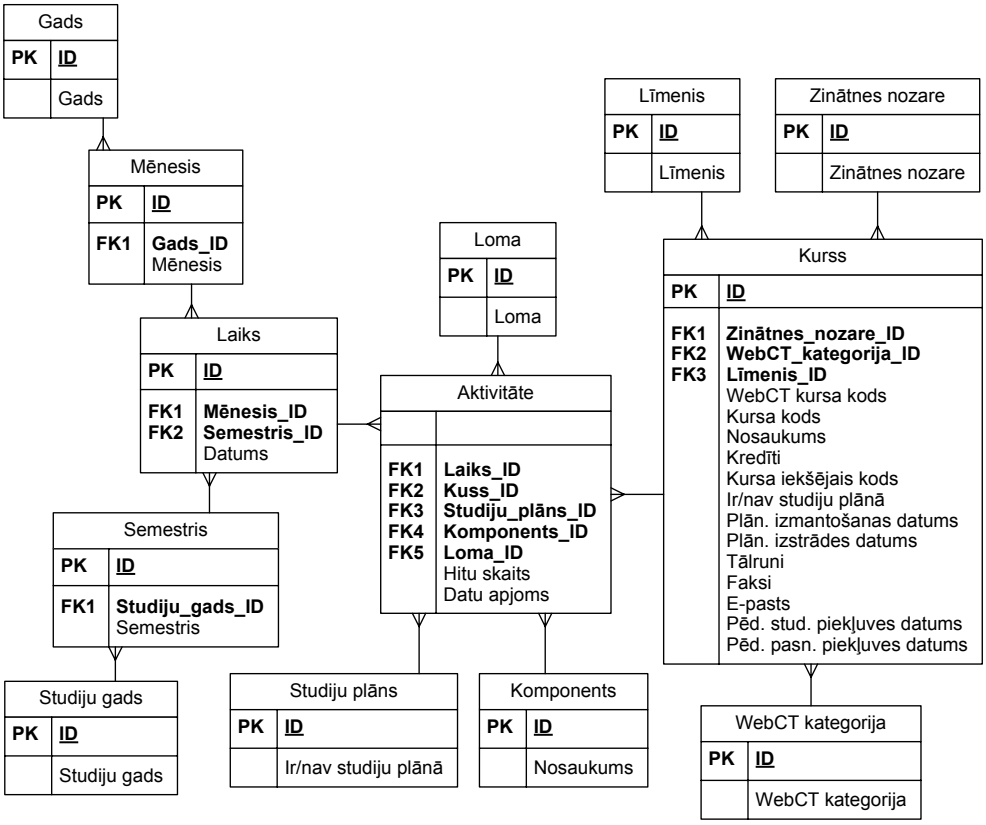
Lietošanas laikā E-studiju datuve attīstījās jauno darījumprasību dēļ. Sākotnējās darījumprasības tika izanalizētas un konceptuāla un loģiska projektējuma rezultātā tika iegūtas divas zvaigznes shēmas: uzbūves zvaigznes shēma un lietošanas zvaigznes shēma (attēls 13.). Saskaņā ar sākotnējām darījumprasībām lietotāji vēlējās analizēt e-kursu uzbūves datus un apkopotu e-kursu lietošanas statistiku, kas saturēja tikai kopējo reģistrēto un aktīvo studentu skaitu kursā. Saskaņā ar sākotnējo projektējumu tika izstrādāta datuves pirmā versija, t.i. tika izanalizēti pieejamie datu avoti un noprogrammēti ETL procesi.



Att. 13. Pirmā datuves versija

Kad lietotāji sāka izmantot datuves pirmo versiju, izrādījās, ka tajā pieejamā informācija nav pietiekama, tāpēc tika nolemts papildināt darījumprasības. Jaunās prasības, kas tika izvirzītas, ir: sagādāt analīzei e-kursu izmantošanas datus, kas satur informāciju par studentu un dizaineru e-kursu lietošanu – aktivitāti. Jaunās prasības tika izanalizētas, jauna aktivitātes zvaigznes shēma (attēls 14.) tika noprojektēta un tika realizēta otrā datuves versija. Šī shēma saturēja aktivitātes faktu tabulu ar diviem mērījumiem: kopējais datu apjoms un hitu skaits (klikšķu skaits), ko veica lietotāji ar noteiktu lomu ar komponentiem vienā kursā vienas nedēļas laikā. Tā deva iespēju analizēt mērījumus pēc sešām dimensijām: laiks, komponents, studiju plāns, kurss, loma. Kursu dimensijai tika pielikti divi jauni aprakstoši atribūti: *Pēdējais studentu piekļuves datums* un *pēdējais dizaineru piekļuves datums*. Kursu dimensijas tabulas adaptācija notika kopā ar aktivitātes faktu tabulas pirmreizējo ielādi, jo jauno atribūtu vērtības tika atvasinātas no atbilstošo mērījumu vērtībām.

Projektējuma laikā tika izpētīti pieejamie datu avoti un tika izvēlēti žurnāla faili datu iegūšanai par aktivitāti. Jaunās zvaigznes shēmas pirmreizējai ielādei tika izmantoti dati no WebCT žurnāla failu arhīva, t.i. datuve tika adaptēta, izmantojot papildus informāciju par pagātnes aktivitāti. Datuves ETL process tika papildināts ar jaunām procedūrām aktivitātes datu ielādei.



Att. 14. Aktivitātes zvaigznes shēma otrajā datuves versijā

Kad lietotāji sāka izmantot otro datuves versiju, izrādījās, ka esošās struktūras neapmierina vēlamo detalizācijas pakāpi. Tādēļ tika sastādīta jauna darījumprasību specifikācija, kur bija norādīts, ka lietotāju aktivitāti kursu vadības sistēmā vajag analizēt katram lietotājam un katrai sesijai. Atkal tika veikta šo prasību analīze un jauns projektējums tika sastādīts.

Saskaņā ar projektējumu aktivitātes zvaigznes shēma attīstījās un tika sasniegta tās pēdējā versija (att. 10.). Šī pēdējā versija ir aprakstīta nodaļā 4.2.2. Tā atšķiras no iepriekšējās versijas ar to, ka tika pievienots jauns mērījums, kas satur kopējo laiku sekundēs, kad lietotājs ir nodarbojies ar komponentu kursā. Tika pievienotas arī jaunas dimensijas: cilvēks, studiju programma, atzīme, sesija.

Laika dimensijai palielinājās jauni atribūti: precīzs laiks un stunda. Šie atribūti arī tika apvienoti hierarhijā. Šo atribūtu vērtības tika aizpildītas automātiski. Komponentu dimensijai arī tika pievienoti jauni klasifikācijas atribūti, kas veido jauno komponentu hierarhiju. To vērtības tika aizpildītas manuāli saskaņā ar WebCT komponentu klasifikāciju.

Cilvēku un studiju programmu dimensijas jau eksistēja Latvijas Universitātes datu noliktavā un tie jau bija aizpildītas ar datiem. Sesiju un atzīmju dimensiju nebija, tāpēc tās nācās aizpildīt ar datiem. Diemžēl eksistējošo faktu tabulu pielāgot neizdevās, jo šajā situācijā nebija transformācijas funkciju, kas varēja detalizētākam faktu tabulas ierakstam dabūt datus no apkopotas vērtības. Saskaņā ar pēdējās datuves versijas projektējumu tika mainīti un papildināti ETL procesi.

4.3.3. E-studiju datuves adaptācijas problēmas

E-studiju datuves dzīves cikla laikā attīstījās ne tikai pati datuve, bet arī notika vairākas izmaiņas datu avotos. Pirmkārt, mainījās WebCT iekšējas datu bāzes struktūra. Lai risinātu šo problēmu, nācās pārprogrammēt iesaiņotāju 4 (att. 11.), kas iegūst datus par e-kursu uzbūvi, jo notikušās izmaiņas ietekmēja tikai to.

Otrkārt, notika izmaiņas LUISa datu struktūrās. Šīs izmaiņas ietekmēja datu atjaunināšanu par kursiem un studiju plāniem. Lai risinātu šīs problēmas, tika adaptēts ETL process, kas nosaka, vai e-kurss ir vai nav studiju plānā. ETL procesa adaptāciju veica izstrādātājs, programmējot PL/SQL procedūras.

Treškārt 2007. gada beigās – 2008. gada sākumā notika pāriešana uz Moodle kursu vadības sistēmu un rezultātā mainījās viens no e-studiju datuves datu avotiem no WebCT uz Moodle. Lai adaptētu e-studiju datuvi, nācās pārprogrammēt ETL procesu daļu, kas ieguva datus no kursu vadības sistēmas datu avotiem. Šīs adaptācijas rezultātā tika izveidots jaunais iesaiņotājs 2* (att. 12.), kas iegūst datus no Moodle iekšējas datu bāzes un ielādē tos datu bāzes tabulās darba apgabalā. Iesaiņotāji 2-5 (att. 11.) vairs netiek lietoti, jo kursu vadības sistēma WebCT vairs netiek lietota.

4.4. Nodaļas secinājumi

Šajā nodaļā tika aprakstītas problēmas, kas saistītas ar E-studiju datuves atjaunināšanu, darījumprasību evolūciju un datu avotu izmaiņām. Šīs izmaiņas ietekmēja ETL procesus un arī datu noliktavas shēmu. Lai risinātu šīs problēmas datu noliktavas administratoram nācās manuāli pārveidot datu noliktavas projektējumu, veidot jauno datu noliktavas shēmas versiju un ETL procesus.

Šo pārveidojumu rezultātā tika iegūtas divas aktivitātes zvaigznes shēmas versijas, kur pirmā versija vairs netiek atjaunota, bet satur vēsturiskos datus par kursu vadības sistēmas lietošanu no 2004. gada septembra. Bet otrā aktivitātes zvaigznes shēmas versija, kas tika izveidota 2005. gada sākumā, ir aktuālā un tiek atjaunināta ar ETL procesiem. Notikušas izmaiņas datuvē ietekmēja ne tikai datuves shēmu un ETL procesus, bet arī atskaites uz datuves shēmām. Lietotais atskaišu rīks Oracle Discoverer neatbalsta vienlaicīgu darbu ar vairākām datu noliktavas shēmas versijām.

Visas šīs attīstības problēmas prasīja daudz laika un resursu, tāpēc tika nolemts automatizēt izmaiņu apstrādes procesu un izstrādāt datu noliktavas atskaišu rīku, kas būs spējīgs attēlot datus no vairākām shēmas versijām vienā atskaitē. Sekojošās nodaļās ir aprakstīta piedāvātā pieeja datu noliktavas attīstības problēmu risināšanai un izstrādātais atskaišu rīks.

E-studiju datuves izstāde un uzturēšanas problēmu risinājumi tika aprakstīti un publicēti rakstā:

Solodovņikova D., Niedrīte L. 'Using Data Warehouse Resources for Assessment of E-Learning Influence on University Processes'. *Proceedings of the 9th East-European Conference on Advances in Databases and Information Systems, Tallinn, Estonia*, 2005.

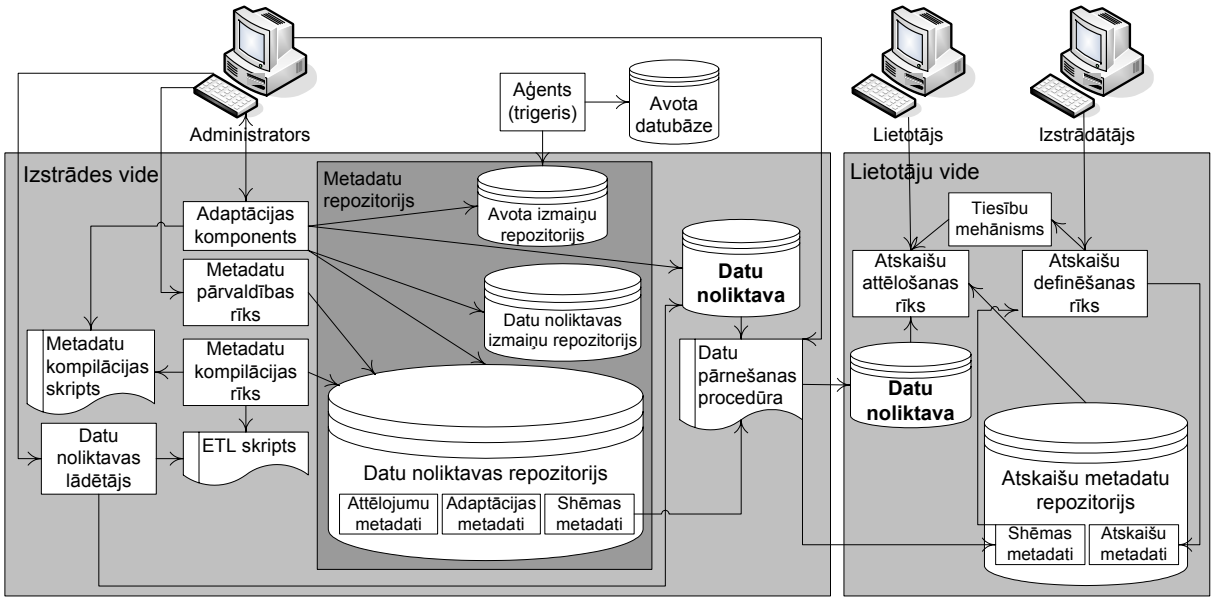
5. DATU NOLIKTAVAS EVOLŪCIJAS ARHITEKTŪRA

5.1. Nodaļas nolūks

Šajā nodaļā ir aprakstīta datu noliktavas evolūcijas arhitektūra, kas spēj apstrādāt iepriekš minētās datu noliktavas evolūcijas problēmas. Piedāvātā arhitektūra spēj apstrādāt gan avotu izmaiņas, gan izmaiņas datu noliktavas shēmā, ko izraisa darījumprasību attīstība. Šī darba ietvaros piedāvātais atskaišu definēšanas un attēlošanas rīks ir datu noliktavas evolūcijas arhitektūras sastāvdaļa. Šīs arhitektūras idejas tika realizētas prototipā, kas ir aprakstīts šajā nodaļā, bet atskaišu definēšanas un attēlošanas rīks tika tālāk attīstīts šī darba ietvaros. Tādēļ, ka izstrādātā rīka darbība ir lielā mērā balstīta uz datu noliktavas metadatiem, tad šajā nodaļā tiek dots ieskats par to, kā rodas un tiek mainīti rīka darbībai nepieciešamie metadati.

5.2. Datu noliktavas evolūcijas arhitektūras komponenti

Lai atbalstītu iepriekš minētas datu noliktavas evolūcijas problēmas: uzturēšanu, pielāgošanu un adaptāciju, tiek piedāvāta datu noliktavas arhitektūra, kas ir redzama 15. attēlā.



Att. 15. Datu noliktavas evolūcijas arhitektūra

Šī arhitektūra sastāv no *izstrādes vides* un *lietotāju vides*. Izstrādes vidē atrodas datu noliktavas metadatu repozitorijs un citi komponenti, kas tiks apskatīti tālāk, notiek datu ieguves, pārveidošanas un ielādes procesi un izmaiņu apstrāde. Lietotāju vidē datu noliktavas izstrādātāji un lietotāji definē un izpilda atskaites uz vienu vai vairākām datu noliktavas shēmas versijām.

5.2.1. Izstrādes vide

Izstrādes vides komponenti

Galvenais izstrādes vides komponents ir *adaptācijas komponents*, kas apstrādā izmaiņas avota modelī, kas ir tabulu pārsaukšana, dzēšana, pievienošana, atribūtu pārsaukšana, dzēšana un datu tipa maiņa, identificē potenciālas izmaiņas datu noliktavas shēmā un jaunu versiju izveidošanas iespējas, adaptē datu noliktavu vai izveido jaunu versiju saskaņā ar administratora lēmumu, izveido nepieciešamos shēmas versiju metadatus un atbilstoši pielāgo ETL procesus.

ETL procesu specifikācijai un datu noliktavas shēmas definēšanai ir paredzēts *metadatu pārvaldības rīks*, kas satur grafisku klienta rīku, ar ko datu noliktavas administrators vai izstrādātājs modelē shēmu un ETL procesus.

ETL procesu skriptu ģenerācijai ir paredzēts *metadatu kompilācijas rīks*, kas izmanto attēlojumu metadatus no metadatu repozitorija. ETL procesu izpildei tiek izmantots *datu noliktavas lādētājs*, kurš izpilda noģenerētos ETL skriptus. *Datu pārnešanas procedūra* pārnes datu noliktavas datus un shēmas metadatus, kas ir nepieciešami atskaitēm, no izstrādes vides uz lietotāju vidi, uz *atskaišu metadatu repozitoriju.*

Metadatu repozitorijs

Lai realizētu datu noliktavas evolūcijas arhitektūras funkcionalitāti, tiek izmantoti dati no *metadatu repozitorija*. Metadatu repozitorijam ir trīs daļas, kas ir paredzētas dažāda veida metadatu glabāšanai.

Datu noliktavas repozitorijā tiek uzkrāti *shēmas metadati*, kas ir dati par datu noliktavas shēmas versijām, kas ir nepieciešami atskaišu definēšanai un izpildīšanai, tai skaitā arī saites starp dažādām versijām. Datu noliktavas repozitorijā tiek glabāti *attēlojumu metadati*, kas definē ETL procesu loģiku. Attēlojumu metadatos ir aprakstīti avota datu objekti (piemēram, tabulas un kolonnas relāciju datu bāzē vai faili), kas ir vajadzīgi ETL procesiem. Papildus datu noliktavas repozitorijs satur vēl *adaptācijas metadatus*, kur adaptācijas komponents uzkrāj informāciju, kuru izmanto datu noliktavas adaptācijai pēc izmaiņām datu avotos. Datu noliktavas repozitorija shēmas un attēlojumu metadatus uztur metadatu pārvaldības rīks. Detalizēts datu noliktavas repozitorija shēmas metadatu apraksts ir atrodams nodaļā 6.3. Adaptācijas metadati ir sīkāk aprakstīti nodaļā 5.3.1.

Metadatu repozitorijs satur arī *datu noliktavas izmaiņu repozitoriju*, kurš uzkrāj datu noliktavas shēmas potenciālās adaptācijas vai versiju iespējas, no kurām administrators izvēlas piemērotākos, kas tiek tālāk realizētas.

Datu noliktavas datu avotos darbojas *aģenti*, kas izseko izmaiņas avota struktūrās un uzkrāj šīs izmaiņas *avotu izmaiņu repozitorijā*. Datu noliktavas izmaiņu repozitorijs un avota izmaiņu repozitorijs tiek aprakstīts nodaļā 5.3.1.

5.2.2. Attēlojumu metadati

ETL procesu specifikācija tiek glabāta attēlojumu metadatos datu noliktavas repozitorijā izstrādes vidē. Attēlojumi definē ETL procesu loģiku, t.i. kādā veidā dati tiek iegūti no avota tabulām, pārveidoti un ielādēti mērķa datu noliktavas tabulās.

Attēlojums būtībā sastāv no vairākiem attēlojuma komponentiem, kas veic dažādas datu transformācijas un kas ir savienoti savā starpā. Parasti komponentiem ir ieejas parametri, kurus komponents pārveido par izejas parametriem, kas tiek izmantoti citos komponentos kā ieejas parametri. Parametri komponentā ir apvienoti vienā vai vairākās grupās. Grupas veids nosaka tās parametru veidus, t.i. grupa var būt ieejas, izejas vai ieejas/izejas, kad vieni un tie paši parametri ir izmantoti komponenta ieejā un izejā, t.i. netiek transformēti.

Eksistē dažādu veidu komponenti. *Tabulas* komponents attēlo avota vai datu noliktavas tabulu. Šim komponentam ir viena ieejas/izejas grupa ar parametriem, kas attēlo tabulas kolonnas.

Lai attēlotu secības objektus ir izmantots *Secības* komponents, ko parasti lieto tabulas unikālo atslēgu ģenerēšanai, jo tas ģenerē unikālas vērtības, kas iet pēc kārtas. Šim komponentam ir izejas grupa ar diviem parametriem: viens parametrs (NEXTVAL) palielina secību un atgriež nākošo vērtību, bet otrais parametrs (CURRVAL) atgriež secības pašreizējo vērtību.

Savienojuma komponents tiek izmantots, lai apvienotu datus no diviem vai vairākiem komponentiem un izmantotu tos datu ielādei vienā komponentā. Šim komponentam ir vairākas ieejas grupas ar parametriem, kas katra atbilst vienai parametru kopai, kas tiek iegūta no viena komponenta, t.i. katram komponentam, kas sniedz datus Savienojuma komponentam, tiek veidota atsevišķa ieejas grupa. Komponentam ir viena izejas grupa ar parametriem, kas atbilst visiem ieejas parametriem. Savienojuma komponentam ir savienojuma nosacījums, kas ir izteiksme, līdzīga SQL vaicājuma WHERE daļai, kas norāda kādā veidā ieraksti vienā ieejas parametru kopā atbilst ierakstiem citā ieejas parametru kopā.

Lai izmantotu datu ieguvei avota datu apakškopu ar noteiktu nosacījumu, izmanto *Filtra* komponentu. Šim komponentam ir viena ieejas/izejas grupa ar parametriem, kas tiek iegūti no avota komponenta un tiek tālāk ielādēti mērķa komponentā. Filtra komponentam ir filtrēšanas nosacījums, kas arī ir izteiksme līdzīga SQL vaicājuma WHERE daļai, kas norāda, kādi ieraksti tiek atgriezti pēc filtrēšanas.

Ir arī *Transformācijas* komponents, kas pārveido kolonnas vērtību ar SQL funkciju, saglabājot ieejas ierakstu kardinalitāti. Transformācijas komponents atbilst funkcijai vai procedūrai, kas ir standarta SQL funkcija vai procedūra vai mērķa sistēmā nokompilēta lietotāja procedūra vai funkcija, kas tiek glabāta attēlojumu metadatos. Ieejas un izejas parametri Transformācijas komponentam atbilst ieejas un izejas parametriem atbilstošai repozitorija funkcijai vai procedūrai. Ja Transformācijas komponents atbilst funkcijai, tad funkcijas rezultāts ir pievienots komponentam kā izejas parametrs.

Lai dabūtu izrēķināmas kolonnas vērtības, tiek lietots *Izteiksmes* komponents, kas atļauj rakstīt SQL izteiksmes, kas definē ne-procedūras algoritmus vienam izejas komponenta

parametram. Šim komponentam var būt vairāki ieejas parametri, kas piedalās izteiksmes tekstā, kas arī var saturēt mainīgo vārdus un funkciju izsaukumus.

Konstantes komponents atļauj definēt konstantes. Konstantes tiek inicializētas attēlojuma izpildes sākumā. Konstantes vērtības var tikt izmantotas jebkurā vietā attēlojumā. Konstantes komponentam ir viena izejas grupa, kas var saturēt vienu vai vairākus parametrus, kam ir izteiksmes īpašība, kas satur konstantes vērtību, kas var tikt izrēķināta ar SQL izteiksmi.

5.2.3. Lietotāju vide

Lietotāju vidē papildus datu noliktavai atrodas arī *atskaišu metadatu repozitorijs*, kas satur datu noliktavas *shēmas metadatus*, kas tiek pārnesti no izstrādes vides attēlojumu repozitorija, un *atskaišu metadatus*, kas apraksta datu noliktavas shēmas objektus un atskaišu specifikāciju un kurus izveido datu noliktavas izstrādātājs ar grafisko *atskaišu definēšanas rīku*.

Datu noliktavas lietotāji atskaitēm izmanto *atskaišu attēlošanas rīku*. Atskaišu attēlošanas rīks ļauj definēt ekspromtvaicājumus, attēlot atskaites tabulu veidā un grafiski un analizēt datus atskaitēs, izmantojot hierarhijas un citas OLAP iespējas. Izmantojot saites, kas tiek veidotas starp datu noliktavas shēmas versijām metadatu repozitorijā, atskaišu attēlošanas rīks var izpildīt vaicājumus uz datiem, kas atbilst vairākām datu noliktavas shēmas versijām vai tikai uz vienu versiju atkarībā no lietotāja izvēles. Vairāku versiju gadījumā lietotājs pats var izvēlēties, saskaņā ar kuru versiju tiks attēloti ekspromtvaicājuma rezultāti, ja tas ir iespējams. Atskaišu attēlošanas rīka detalizēts darbības apraksts tiek dots nodaļā 9.

Arī lietotāju vidē ir realizēts *tiesību mehānisms*, kas īstenībā ir metadati, kas definē, kuras atskaites var izmantot noteikts lietotājs un kuriem datu noliktavas datiem lietotājam ir pieeja. Šos metadatus izmanto atskaišu attēlošanas rīks. Tiesību mehānismu uzstāda izstrādātājs ar atskaišu definēšanas un attēlošanas rīkiem.

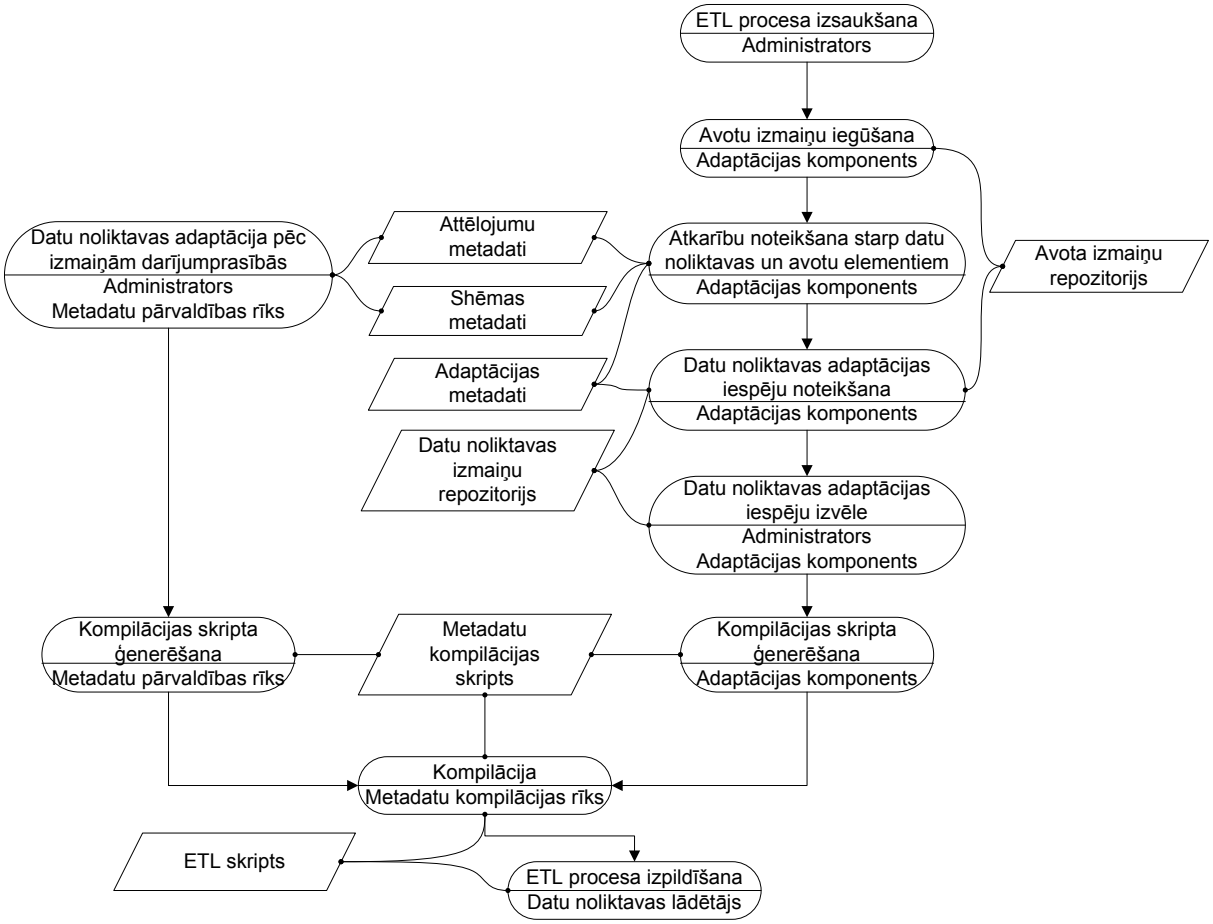
5.3. Datu noliktavas evolūcijas arhitektūras darbība

5.3.1. Datu noliktavas adaptācija pēc izmaiņām darījumprasībās

Ja datu noliktavas shēmu maina pats administrators darījumprasību izmaiņu dēļ, tad visas izmaiņas tiek veiktas ar metadatu pārvaldības rīku, kurš ļauj izveidot jauno datu noliktavas versiju vai labot veco. Saskaņā ar jauno datu noliktavas shēmas versiju automātiski tiek pielāgoti arī ETL procesu metadati attēlojumu metadatos. Detalizētas procedūras, kā tiek mainīti datu noliktavas shēmas metadati un kādas izmaiņas tiek veiktas datubāzē shēmas modifikācijas rezultātā, tiek aprakstītas nodaļā 7.

5.3.2. Datu noliktavas adaptācija pēc izmaiņām datu avotos

Avota izmaiņu apstrāde notiek pirms ETL procesu izpildīšanas izstrādes vidē, ko izsauc datu noliktavas administrators. Attēlā 16. ir dots datu noliktavas adaptācijas procesa modelis.



Att. 16. Datu noliktavas adaptācijas process

Sākumā adaptācijas komponents analizē izmaiņas avotu izmaiņu repozitorijā un nosaka, kādas izmaiņas ietekmē datu noliktavas shēmu un ETL procesus.

Šim nolūkam adaptācijas komponents izmanto datus no datu noliktavas repozitorija adaptācijas metadatiem par datu noliktavas elementu atkarību no avota elementiem. Ja dati no viena attēlojuma komponenta parametra ir izmantoti otra attēlojuma komponenta parametra izrēķināšanai vai ir vienkārši nokopēti otrajā parametrā, tad otrais parametrs ir *atkarīgs* no pirmā.

Adaptācijas metadati

Atkarību glabāšanai tiek izmantoti adaptācijas metadati, kas glabā atkarības starp datu noliktavas elementiem un datu avotu elementiem. Šajā nodaļā tiks apskatīta metadatu repozitorija adaptācijas daļa, kas ir tieši izmantota izmaiņu apstrādē. Adaptācijas metadati sastāv no divām tabulām ATKARIBUTABULA un DWH_ATKARIBAS.

Tabulā ATKARIBUTABULA glabā informāciju par visām atkarībām starp attēlojumu komponentu parametriem. Šī tabula satur identifikatorus no attēlojumu metadatu tabulām datu noliktavas repozitorijā. Tabulas kolonnas ir attēlotas tabulā 1.

Tabula 1. Tabulas ATKARIBUTABULA kolonnas

Kolonna	Apraksts
MAP_ID	Attēlojuma identifikators no attēlojumu metadatu tabulas
MAP_ COMPONENT_ID	Komponenta identifikators no komponentu metadatu tabulas
PARAMETER_ID	Komponenta parametra identifikators no parametru metadatu tabulas.
ATK_MAP_ COMPONENT_ID	Identifikators komponentam, no kura parametrs ir atkarīgs, no komponentu metadatu tabulas
ATK_ PARAMETER_ID	Identifikators parametram, no kura parametrs ir atkarīgs, no parametru metadatu tabulas
ATK_VEIDS	Komponenta tips, no kura parametrs ir atkarīgs
ATK_DATA_ ENTITY_ID	Avota tabulas identifikators no tabulu metadatu tabulas, ja parametrs ir atkarīgs pa tiešo no avota tabulas atribūta; vai secības identifikators no secību metadatu tabulas, ja parametrs ir atkarīgs pa tiešo no secības komponenta
ATK_DATA_ ITEM_ID	Avota tabulas kolonnas identifikators no kolonnu metadatu tabulas, ja parametrs ir atkarīgs pa tiešo no avota tabulas atribūta
ATK_IZTEIKSME	Atkarības izteiksme

Šo tabulu ATKARIBUTABULA aizpilda adaptācijas komponents pirms avota izmaiņu apstrādes, dati tiek iegūti no attēlojumu metadatiem. Atkarību informāciju izrēķina katram attēlojumu komponenta parametram, kas ir atkarīgs no cita parametra. Katram parametram vispirms tiek noskaidrots cita komponenta parametrs, no kura tas ir atkarīgs, tad, atkarībā no komponenta tipa, no kura parametrs ir atkarīgs, izrēķina atkarības veidu un izteiksmi. Atkarības izteiksme ir SQL vaicājuma WHERE nosacījuma daļa, ja komponents ir Savienojums; funkcija, ja komponents ir Izteiksme, vai Transformācija vai vienkārši konstantes vērtība, ja komponents ir Konstante.

Tālāk šie dati par parametru atkarībām ir izmantoti, lai noskaidrotu, kādi datu noliktavas modeļa elementi ir atkarīgi no datu avotu modeļa elementa, ar kuru notika izmaiņa. Pirms dati no avota modeļa elementa tiek ielādēti datu noliktavā, šim elementam pielieto vairākas transformācijas, t.i. elements tiek transformēts vairākos attēlojuma komponentos. Šie visi komponenti ir atkarīgi viens no otra, tāpēc no datiem par parametru un komponentu atkarībām var izsekot, kuri datu noliktavas modeļa elementi ir atkarīgi no mainīta avota modeļa elementa.

Atkarības starp datu noliktavas atribūtiem un datu avotu atribūtiem tiek glabātas tabulā DWH_ATKARIBAS, kuras kolonnas ir aprakstītas tabulā 2.

Tabula 2. Tabulas DWH_ATKARIBAS kolonnas

Kolonna	Apraksts
ATKARIBA_ID	Atkarības unikālais identifikators, primārā atslēga
DWH_TABULA	Datu noliktavas tabulas nosaukums
DWH_KOLONNA	Datu noliktavas tabulas kolonnas nosaukums
AVOTA_TABULA	Avota tabulas nosaukums
AVOTA_KOLONNA	Avota tabulas kolonnas nosaukums
ATK_VEIDS	Atkarības veids
AVOTA_ID	Avota shēmas identifikators no shēmu metadatu tabulas
MERKA_ID	Datu noliktavas (mērķa) shēmas identifikators no shēmu metadatu tabulas
DATUMS	Laiks, kad atkarība bija spēkā
DERIGS	Šis lauks satur vērtību 'Y', ja atkarība ir spēkā, vai vērtību 'N', ja atkarība vairs nav spēkā. Šo kolonnu maina uz 'N' visiem avota tabulas ierakstiem pirms kārtējas parametru atkarību apstrādes.

Atkarības starp datu noliktavas atribūtiem un datu avotu atribūtiem tiek iegūti apstrādājot tabulas ATKARIBUTABULA datus un datu noliktavas shēmas metadatus ar rekursīvo procedūru, kas katram avota atribūtam iegūst visus parametrus, kas ir no tā atkarīgi. Tad katram iegūtam parametram izvēlas visus parametrus, kas ir no tā atkarīgi, un tā tālāk, kamēr tiek sasniegts attēlojuma komponenta parametrs, kas attēlo datu noliktavas tabulas atribūtu. Šajā procesā tiek iegūts „*atkarības ceļš*" no avota atribūta līdz datu noliktavas atribūtam.

Vispār var eksistēt vairāki atkarības ceļi katram datu noliktavas atribūtam no dažādiem avota atribūtiem. Katram ceļam apstrādes laikā tiek definēts atkarības veids, kas eksistē starp datu noliktavas atribūtiem un datu avota atribūtiem.

Sekojoši *atkarības veidi* var eksistēt starp datu noliktavas elementiem un datu avota elementiem:

- *Nokopēts* – tas nozīmē, ka datu noliktavas atribūts ir vienāds ar avota atribūtu. Šis atkarības veids parādās situācijās, kad datu noliktavas tabulas atribūts tiek iegūts pa tiešo no datu avota tabulas atribūta vai no vairāku avota tabulu savienojuma. Reāli tas nozīmē, ka visā atkarības ceļā datu avota atribūtam tiek pielietoti Tabulas, Savienojuma vai Filtra komponenti, t.i. tabulā ATKARIBUTABULA kolonna ATK_VEIDS ir 'Join', 'Filter' vai 'Table'.
- *Izmantots aprēķināšanā* – tas nozīmē, ka datu noliktavas atribūts ir izrēķināts no avota atribūta. Šis atkarības veids parādās situācijās, kad avota atribūts piedalās kā arguments funkcijā, kuras rezultātā iegūti dati ir ielādēti datu noliktavas atribūtā vai atribūtos. Reāli tas nozīmē, ka atkarības ceļā datu avota atribūtam tiek pielietots Transformācijas vai Izteiksmes komponenti un atribūts ir izmantots atkarības izteiksmē, t.i. eksistē ieraksts tabulā ATKARIBUTABULA, kur ATK_VEIDS ir 'Transformation' vai 'Expression' un ATK_IZTEIKSME satur šo avota atribūtu.
- *Izmantots filtrā* – tas nozīmē, ka datu avota atribūts ir izmantots Filtra komponentā nosacījuma daļā. Reāli tas nozīmē, ka atkarības ceļā datu avota atribūtam tiek

pielietots Filtra komponents un atribūts ir izmantots filtrēšanas izteiksmē, t.i. ATK_VEIDS ir 'Filter FExpression' un ATK_IZTEIKSME satur šo atribūtu.

- *Izmantots savienojumā* – tas nozīmē, ka datu avota atribūts ir izmantots savienojuma WHERE daļā. Reāli tas nozīmē, ka atkarības ceļā datu avota atribūtam tiek pielietots Savienojuma komponents un atribūts ir izmantots savienojuma izteiksmē, t.i. ATK_VEIDS ir 'Join WHERE' un ATK_IZTEIKSME satur šo atribūtu.

Parasti attēlojumā viens avota atribūts var tikt izmantots vairākos attēlojuma komponentos, tāpēc tam var būt vairāki atkarības veidi. Tāpēc atkarību apstrādes laikā tabulas DWH_ATKARIBAS kolonnā ATK_VEIDS tiek sākumā sakrāti visi atkarības veidi, kas eksistē tabulā ATKARIBUTABULA, kas apstrādes beigās tiek pārveidoti, kombinēti un saglabāti tabulā DWH_ATKARIBAS.

Avota izmaiņu metadati

Katrā datu avotā ir aģents, kas nodarbojas ar izmaiņu noteikšanu un metadatu uzkrāšanu par notikušajām izmaiņām. Izmaiņu metadati tiek uzkrāti Avota izmaiņu repozirorijā, kas sastāv no tabulas AVOTU_IZMAINAS, kuras kolonnas ir aprakstītas tabulā 3. Tabula AVOTU_IZMAINAS, kur tiek uzkrātas izmaiņas no visiem datu avotiem, tiek papildināta pirms katras adaptāciju apstrādes, bet katrā datu avotā arī eksistē līdzīga tabula ar izmaiņām, kas notika tikai noteiktajā datu avotā, bet tā nesatur kolonnas OWNER un AVOTA_ID.

Tabula 3. Tabulas AVOTU_IZMAINAS kolonnas

Kolonna	Apraksts
OWNER	Avota shēmas nosaukums, kur notika izmaiņa
AVOTA_ID	Avota shēmas identifikators no shēmu metadatu tabulas
IZMAINA_ID	Izmaiņas identifikators
LAIKS	Laiks, kad notika izmaiņa
OBJEKTA_TIPS	Mainītā objekta tips
OBJEKTA_NOSAUKUMS	Mainītā objekta nosaukums
TABULAS_NOSAUKUMS	Ja tika mainīts avota tabulas atribūts, tad šajā laukā glabā tabulas nosaukumu, kas satur atribūtu, ar kuru notika izmaiņa.
DARBIBA	Izmaiņas veids: CREATE, DROP, MODIFY, RENAME
VECS_OBJEKTA_NOSAUKUMS	Vecais mainītā objekta nosaukums
VECS_DATU_TIPS	Mainītās kolonnas datu tips
VECS_DATU_GARUMS	Mainītās kolonnas garums
VECA_DATU_PRECIZITATE	Mainītās kolonnas precizitāte
VECA_DATU_SKALA	Mainītās kolonnas skala
VECS_IR_NAV_NULL	Šis lauks norāda, vai kolonna var saturēt NULL vērtības

TEKSTS	SQL vaicājuma teksts, kas izraisīja izmaiņas
APSTRADATS	Šis lauks satur vērtību 'Y', ja datu noliktava jau tika adaptēta pēc izmaiņas, vai vērtību 'N', ja šī izmaiņa vēl jāapstrādā.

Izmaiņas, kuras ir nepieciešams apstrādāt un kuras tiek uzkrātas avota izmaiņu repozitorijā, ietver avota tabulas pievienošanu, dzēšanu un nosaukuma maiņu, avota tabulas kolonnas pievienošanu, dzēšanu, nosaukuma un datu tipa maiņu. Tālāk apskatīsim kā šīs izmaiņas ir aprakstītas metadatos.

Tabulas AVOTU_IZMAINAS kolonnu vērtības ir atkarīgas no izmaiņas veida.

- *Avota tabulas pievienošana* – Avota tabulas pievienošanai atbilst darbība 'CREATE' un objekta tips 'TABLE'. Ja datu avotā tiek izveidota jauna tabula, tad metadatos saglabā jaunas tabulas nosaukumu kolonnā OBJEKTA_NOSAUKUMS.
- *Avota tabulas dzēšana* – Šim izmaiņas veidam izveido vairākus ierakstus tabulā AVOTU_IZMAINAS: viens ieraksts par tabulas dzēšanu un pa vienam ierakstam katrai dzēstas tabulas kolonnai līdzīgi kā kolonnas dzēšanas izmaiņas veidam. Avota tabulas dzēšanai atbilst darbība 'DROP' un objekta tips 'TABLE'. Tabulas nosaukumu ieraksta laukā OBJEKTA_NOSAUKUMS. Ja iepriekš ir bijusi izmaiņa par tādas pašas relācijas pievienošanu, bet izmaiņa par relācijas pievienošanu vēl netika apstrādāta, tad tā tiek dzēsta no tabulas AVOTU_IZMAINAS un izmaiņa par tabulas dzēšanu netiek saglabāta. Ja izmaiņa par izdzēstas tabulas kādas kolonnas izveidošanu, kas notikusi iepriekš, vēl netika apstrādāta, tad tā tiek dzēsta un izmaiņa par šīs kolonnas dzēšanu netiek izveidota. Ja pirms dzēšanas ar tabulas kolonnām notika vēl citu veidu izmaiņas, tad no avota izmaiņu tabulas AVOTU_IZMAINAS tiek izdzēsti ieraksti, kas atbilst neapstrādātām šīs tabulas vai tās kolonnu izmaiņām. Tas ir izdarīts tāpēc, ka šīs izmaiņas anulē viena otru, tāpēc datu noliktavas modelis nav jāadaptē pēc šīm izmaiņām.
- *Avota tabulas nosaukuma maiņa* – Avota tabulas nosaukuma maiņai atbilst darbība 'RENAME' un objekta tips 'TABLE'. Tabulas jauno nosaukumu ieraksta laukā OBJEKTA_NOSAUKUMS, bet veco saglabā laukā VECS_OBJEKTA_NOSAUKUMS.
- *Avota kolonnas pievienošana* – Avota kolonnas pievienošanai tabulai atbilst darbība 'ADD' un objekta tips 'COLUMN'. Jaunas kolonnas nosaukumu saglabā kolonnā OBJEKTA_NOSAUKUMS un datu tipu saglabā kolonnās VECS_DATU_TIPS, VECS_DATU_GARUMS, VECA_DATU_PRECIZITATE, VECA_DATU_SKALA, VECS_IR_NAV_NULL.
- *Avota kolonnas dzēšana* – Avota kolonnas dzēšanai atbilst darbība 'DROP' un objekta tips 'COLUMN'. Dzēstas kolonnas nosaukumu saglabā kolonnā OBJEKTA_NOSAUKUMS un datu tipu saglabā kolonnās VECS_DATU_TIPS, VECS_DATU_GARUMS, VECA_DATU_PRECIZITATE, VECA_DATU_SKALA, VECS_IR_NAV_NULL. Ja pirms dzēšanas kolonna tika mainīta, tad no avota izmaiņu tabulas AVOTU_IZMAINAS tiek izdzēsti ieraksti, kas atbilst neapstrādātām šīs kolonnas izmaiņām. Ja izmaiņa par kolonnas pievienošanu vai tabulas izveidošanu vēl netika apstrādāta, tad tā tiek dzēsta no tabulas AVOTU_IZMAINAS, un izmaiņa par kolonnas dzēšanu netiek saglabāta.

- *Avota kolonnas nosaukuma maiņa* – Avota kolonnas nosaukuma maiņai atbilst darbība 'RENAME' un objekta tips 'COLUMN'. Kolonnas jauno nosaukumu ieraksta laukā OBJEKTA_NOSAUKUMS, bet veco saglabā laukā VECS_OBJEKTA_NOSAUKUMS.
- *Avota kolonnas datu tipa maiņa* – Avota kolonnas datu tipa maiņai atbilst darbība 'MODIFY' un objekta tips 'COLUMN'. Kolonnas veco datu tipu ieraksta laukos VECS_DATU_TIPS, VECS_DATU_GARUMS, VECA_DATU_PRECIZITATE, VECA_DATU_SKALA, VECS_IR_NAV_NULL.

Datu noliktavas adaptācijas iespējas

Tālāk tiks aplūkots, kādā veidā var adaptēt datu noliktavas modeli un ETL procesus pēc izmaiņām datu avotos. Sākumā jāanalizē notikušās izmaiņas. Lai noteiktu, vai ir jāveic datu noliktavas modeļa adaptācija un vai ir jāpielāgo ETL procesi, izmanto atkarības, kas tiek uzkrātas metadatu repozitorija adaptācijas metadatos. Ja datu noliktavas modeļa elementi nav atkarīgi no mainīta avota elementa, tad datu noliktava un ETL procesi nav jāadaptē. Katrai avota izmaiņai, no kuras ir atkarīgi datu noliktavas elementi, adaptācijas komponents iegūst iespējamos risinājumus, kas veido jaunu datu noliktavas versiju vai adaptē datu noliktavas shēmu un ETL procesus.

Dažām avota shēmas izmaiņām ir iespējami vairāki risinājumi, kā var adaptēt datu noliktavas shēmu vai izveidot jaunu shēmas versiju. Tālāk tiks aplūkots, kādi ir iespējamie risinājumi dažādu izmaiņu gadījumos, kad datu noliktavas elementi ir atkarīgi no mainītā avota elementa.

Jaunas tabulas pievienošana

Ja datu avotā tiek pievienota jauna tabula, tad datu noliktavas shēmu un ETL procesus nevar adaptēt automātiski. Tāpēc šī izmaiņa jāpaziņo datu noliktavas administratoram, bet nav jāapstrādā automātiski. Ja administrators izlemj, ka datu noliktavas shēma vai ETL procesi ir jāmaina, lai izmantotu jauno tabulu, tad adaptācijas komponents izveido attēlojumu metadatus, kas apraksta jauno avota tabulu. Tad administrators pats pielāgo datu noliktavas shēmu vai veido jaunu shēmas versiju ar metadatu pārvaldības rīku datu bāzē un datu noliktavas shēmas metadatos, un modelē ETL procesa attēlojumus attēlojumu metadatos arī ar metadatu pārvaldības rīku.

Tabulas dzēšana

Ja datu avotā tika dzēsta tabula, tad šo izmaiņu apstrādā kā vairāku kolonnu dzēšanu no tabulas un beigās automātiski izdzēš no datu noliktavas repozitorija attēlojumu metadatiem datus, kas atbilst izdzēstai avota tabulai.

Tabulas pārsaukšana

Ja avota tabulu pārsauc, tad datu noliktavas shēma nav jāadaptē un ir jāmaina tikai ETL procesi automātiski bez datu noliktavas administratora dalības. Adaptācijas komponents maina avota tabulas nosaukumu attēlojumu metadatos.

Jaunas kolonnas pievienošana

Jaunas avota kolonnas pievienošana ir līdzīga jaunas tabulas pievienošanai. Šo izmaiņu nevar apstrādāt automātiski, arī nevar piedāvāt adaptācijas alternatīvas, tāpēc jaunas datu noliktavas shēmas versijas veidošanu vai izmaiņas datu noliktavas shēmā un ETL procesu metadatos veic datu noliktavas administrators pats. Adaptācijas komponents tikai

adaptē repozitorijā avota tabulas metadatus: izveido jaunās kolonnas aprakstu attēlojumu metadatos.

Kolonnas dzēšana

Ja tiek izdzēsta avota tabulas kolonna, tad vispirms izdzēstās kolonnas avota tabulas metadatus repozitorijā attiecīgi jāsaskaņo ar reālo situāciju. Visas nepieciešamās darbības veic Adaptācijas komponents automātiski. Avota kolonnas dzēšanas izmaiņai ir vairāki alternatīvi adaptāciju veidi.

Risinājums 1. Ja kāds datu noliktavas shēmas elements (mērījums vai atribūts) ir atkarīgs no izdzēstās kolonnas un atkarības veids ir 'Kopija', tad viens risinājums būtu izveidot jaunu datu noliktavas shēmas versiju bez atkarīgā shēmas elementa un izdzēst visus attēlojumu komponentu parametrus, kas ir atkarīgi no avota kolonnas. Šajā gadījumā atkarīgais datu noliktavas shēmas elements netiek izdzēsts no datu noliktavas tabulas datubāzē, bet vairāk netiek aizpildīts ar ETL procesiem. Lai to panāktu, tiek adaptēti ETL procesi, jo tiek dzēsti ieraksti, kas apraksta parametrus, kas parādās atkarības ceļā no izdzēstās avota kolonnas uz datu noliktavas shēmas elementu. Ja datu noliktavas shēmas elementam ir NOT NULL ierobežojums, tad tas jānoņem gan no tabulas metadatiem, gan no datu bāzes. Iepriekš minētās darbības veic adaptācijas komponents automātiski.

Risinājums 2. Vēl viens risinājums būtu aizvietot avota kolonnu ar kādu citu kolonnu un turpmāk ielādēt datu noliktavas shēmas elementu ar datiem no alternatīvas avota kolonnas. Šo risinājumu var realizēt pus-automātiski, ja alternatīvā kolonna atrodas tabulā, kur bija arī izdzēstā kolonna. Šajā gadījumā administratoram tikai jāizvēlas alternatīvā kolonna, bet pārējo darbu veic adaptācijas komponents automātiski.

Risinājums 3. Vēl viens risinājums būtu izrēķināt avota kolonnu no citām kolonnām un turpmāk ielādēt datu noliktavas shēmas elementu ar datiem no alternatīvām kolonnām. Šo risinājumu arī var realizēt pus-automātiski, ja alternatīvās kolonnas atrodas tabulā, kur bija arī izdzēstā kolonna. Šajā gadījumā administratoram tikai jāizvēlas alternatīvās kolonnas un jānorāda aprēķināšanas izteiksme, bet pārējo darbu veic adaptācijas komponents automātiski.

Ja izdzēstā kolonna ir izmantota kāda datu noliktavas shēmas elementa aprēķināšanai, tad iespējamie risinājumi ir līdzīgi situācijai, kad atkarīgais datu noliktavas shēmas elements ir nokopēts no avota kolonnas.

Ja izdzēstā kolonna ir izmantota filtrā, tad filtrēšanas nosacījuma dzēšana izraisa lieku ierakstu iegūšanu no datu avota. Šajā gadījumā administratoram tiek piedāvāts (1) izmest filtrēšanas nosacījuma daļu, kur piedalās izdzēstais atribūts un/vai (2) papildināt filtrēšanas nosacījumu.

Risinājums 4. Pirmajā gadījumā adaptācijas komponents no attēlojuma Filtra komponenta filtrēšanas nosacījuma izteiksmes automātiski izņem daļu, kas satur izdzēsto kolonnu.

Risinājums 5. Otrajā gadījumā izdara to pašu, tikai pēc tam vēl papildina šo nosacījumu ar virkni, ko sastāda administrators.

Ja izdzēstā kolonna ir izmantota savienojumā, tad tās dzēšana no WHERE nosacījuma var ietekmēt ETL procesu, jo izdzēstā kolonna bija izmantota, lai savienotu divus komponentus. Šīs kolonnas izmešana no attēlojuma var izraisīt kļūdainu datu ielādi datu noliktavā. Šai izmaiņai eksistē pus-automātiskais risinājums, kad administrators izvēlas

alternatīvo kolonnu vai funkciju no alternatīvām kolonnām un konstantēm, ar ko izdzēsto kolonnu var aizvietot. Šī gadījuma apstrāde ir līdzīga risinājumam 2. vai 3.

Risinājums 6. Cits risinājums var būt, kad administrators norāda alternatīvu savienojuma veidu katram Savienojuma komponentam, kurā savienojuma izteiksmē bija lietota izdzēstā kolonna. Šajā gadījumā adaptācijas komponents no attēlojuma Savienojuma komponenta savienojuma nosacījuma izteiksmes automātiski izņem daļu, kas satur izdzēsto kolonnu, tad papildina šo nosacījumu ar virkni, ko sastāda administrators.

Kolonnas pārsaukšana

Ja tika mainīts avota kolonnas nosaukums, tad datu noliktavas shēma nav jāadaptē un ir jāmaina tikai ETL procesi automātiski bez datu noliktavas administratora dalības. Attēlojumu metadatos vienkārši maina kolonnas nosaukumu.

Kolonnas datu tipa maiņa

Ja tika mainīts avota kolonnas datu tips, tad ir vairāki iespējami risinājumi. Visiem risinājumiem Adaptācijas komponents sākumā automātiski maina attēlojumu metadatus, kas apraksta avota tabulas kolonnas.

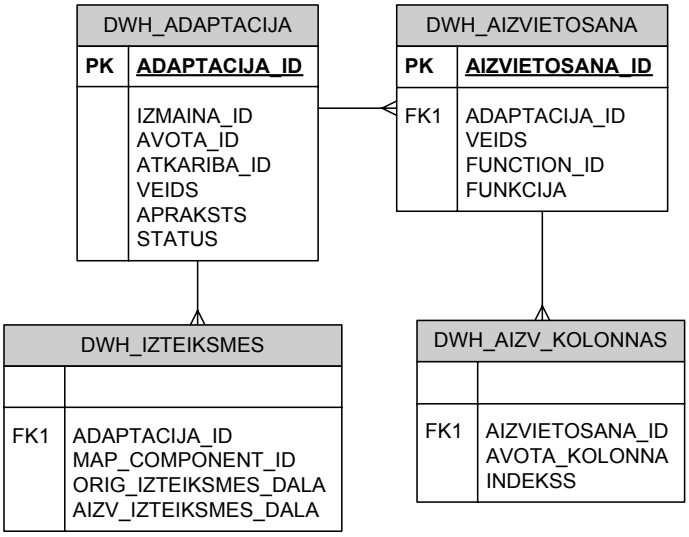
Risinājums 1. Ja kāds datu noliktavas shēmas elements ir atkarīgs no mainītās avota kolonnas un atkarības veids ir 'Kopija', tad šim datu noliktavas shēmas elementam attiecīgi arī jāmaina datu tips. Šeit administratoram ir iespējas izvēlēties veidot jaunu datu noliktavas shēmas versiju ar mainītu elementu vai veikt elementa datu tipa maiņu eksistējošā shēmā. Adaptācijas komponents automātiski maina atbilstošus attēlojumu metadatus un veic izmaiņas arī datu bāzē, administratoram tikai jāizvēlas šis adaptācijas veids.

Cits risinājums, kas ir derīgs visiem atkarības veidiem, ir norādīt alternatīvu tās pašas avota tabulas kolonnu ar saskanīgu datu tipu, kas tiks izmantota mainītās kolonnas vietā, vai funkciju no alternatīvām kolonnām. Šī gadījuma apstrāde ir līdzīga risinājumam 2. vai 3. avota kolonnas dzēšanas izmaiņai.

Risinājums 2. Vēl viens risinājums, kas arī ir derīgs visiem atkarības veidiem ir konvertēšanas funkcijas pievienošana, kas pārveido avota kolonnu par vērtību ar veco datu tipu. Šo risinājumu var pielietot, ja mainījās pats datu tips, nevis datu skala, precizitāte u tml. Šajā gadījumā administratoram jānorāda konvertēšanas funkcija, bet atbilstošus attēlojumu metadatus maina Adaptācijas komponents.

Datu noliktavas izmaiņu repozitorijs

Datu noliktavas izmaiņu repozitorijā tiek glabāta informācija par adaptācijas un versiju izveidošanas risinājumiem katrai izmaiņai, kas notika datu avotā. Lai analizētu izmaiņas, dati par notikušām avota izmaiņām ir nokopēti adaptācijas metadatos tabulā AVOTU_IZMAINAS no katra datu avota. Pēc tam datu avotos notikušās izmaiņas tiek analizētas un katrai izmaiņai tiek noteikti iespējami risinājumi. Alternatīvās adaptācijas iespējas tiek saglabātas datu noliktavas izmaiņu repozitorijā, kas ir attēlots attēlā 17.



Att. 17. Datu noliktavas izmaiņu repozitorijs

Šeit visas adaptācijas alternatīvas tiek glabātas tabulā DWH_ADAPTACIJA, kuras kolonnu apraksti ir redzami tabulā 4. Datu bāzes tabulās DWH_AIZVIETOSANA, kuras kolonnas ir aprakstītas tabulā 5., un DWH_AIZV_KOLONNAS, kuras kolonnu apraksti ir doti tabulā 6., glabā informāciju par kolonnām, kas aizvieto izdzēstās avota kolonnas, un funkcijām, ja kolonna tiek izrēķināta no alternatīvām kolonnām. Bet datu bāzes tabula DWH_IZTEIKSMES, kura ir aprakstīta tabulā 7., satur datus par filtrēšanas un savienojuma izteiksmju papildināšanu.

Tabula 4. Tabulas DWH_ADAPTACIJA kolonnas

Kolonna	**Apraksts**
ADAPTACIJA_ID	Adaptācijas unikāls identifikators
IZMAINA_ID	Izmaiņas identifikators no tabulas AVOTU_IZMAINAS
AVOTA_ID	Avota identifikators, kurā notika izmaiņa
ATKARIBA_ID	Atkarības identifikators no tabulas DWH_ATKARIBAS, kurai var pielietot adaptāciju
VEIDS	Adaptācijas veids, kas var saturēt arī jaunas versijas izveidošanas informāciju, piemēram, kolonnas aizvietošana, kolonnas dzēšana jaunajā versijā, datu tipa konvertācija u.c.
APRAKSTS	Noģenerēts teksts par notikušo izmaiņu un piedāvāto adaptāciju, ko rāda datu noliktavas administratoram, lai viņš varētu izvēlēties piemērotāko adaptāciju
STATUS	Adaptācijas statusam ir sekojošas vērtības: 1. 'Priekšlikums', ja datu noliktavas administrators vēl nav izlēmis par šo izmaiņu 2. 'Akceptēts', ja administrators ir izlēmis realizēt šo adaptāciju 3. 'Neakceptēts', ja adaptāciju nerealizēs 4. 'Izpildīts', ja adaptācija tika realizēta

Katrai neapstrādātai izmaiņai no tabulas AVOTU_IZMAINAS Adaptācijas komponents iegūst atkarīgos datu noliktavas shēmas elementus un katram elementam nosaka visas iespējamās adaptācijas alternatīvas un katrai alternatīvai izveido ierakstu tabulā DWH_ADAPTACIJA. Ja ar vienu un to pašu avota tabulu notika vairākas neapstrādātas izmaiņas, tad tabulā DWH_ADAPTACIJA tiek izveidoti ieraksti par pirmo izmaiņu, kas notika agrāk. Kad šī izmaiņa tiks apstrādāta, Adaptācijas komponents izveido ierakstus par nākošo izmaiņu utt. Ja datu noliktavas shēmas elements ir atkarīgs no mainītas kolonnas vai tabulas un atkarības veids sastāv no vairāku atkarības veidu kombinācijas, tad adaptācijas variantus izvēlas atbilstoši katram atkarības veidam kombinācijā. Tādā veidā tiek iegūti kombinētie adaptācijas veidi.

Kad visas izmaiņas ir izanalizētas un ir atrastas adaptācijas alternatīvas, t.i. tabula DWH_ADAPTACIJA ir aizpildīta, tad administratoram tiek paziņots par neapstrādātām notikušām izmaiņām datu avotā, datu noliktavas shēmas elementiem, kas ir atkarīgi no mainītām avota tabulām un tabulu kolonnām, un adaptācijas variantiem, ko var veikt, lai šīs izmaiņas apstrādātu. Administrators izvēlas, kādas adaptācijas jāveic, vai izvēlas neveikt automātisku adaptāciju, un maina datu noliktavas shēmu un ETL procesu specifikāciju pats ar Metadatu pārvaldības rīku.

Ja administrators tomēr izvēlas veikt noteiktu automātisku adaptāciju, tad atkarībā no izvēlētā adaptācijas veida, Adaptācijas komponents aizpilda pārējās datu noliktavas izmaiņu repozitorija tabulas.

Ja izvēlētais adaptācijas veids ir aizvietot ar citu kolonnu, tad adaptācijas komponents pieprasa administratoram izvēlēties mainītās tabulas citu kolonnu, kas aizvietos izdzēsto. Tabulā DWH_AIZVIETOSANA (Tabula 5.) izveido ierakstu par aizvietošanu, bet kolonnu sarakstu, kas aizvieto izdzēsto, ieraksta tabulā DWH_AIZV_KOLONNAS (Tabula 6.).

Tabula 5. Tabulas DWH_AIZVIETOSANA kolonnas

Kolonna	**Apraksts**
AIZVIETOSANA_ID	Aizvietošanas ieraksta unikāls identifikators
ADAPTACIJA_ID	Adaptācijas identifikators, uz ko attiecas aizvietošana
VEIDS	Šī kolonna norāda aizvietošanas veidu: 1. 'Viena kolonna', ja izdzēsta kolonna tiek aizvietota ar vienu citu avota tabulas kolonnu; 2. 'Izrēķina no kolonnām', ja izdzēsta kolonna tiek izrēķināta no citām avota tabulas kolonnām.
FUNCTION_ID	Ja izdzēsta kolonna tiks aizvietota ar vienu citu kolonnu, tad šī kolonna ir tukša. Citā gadījumā šī kolonna satur funkcijas identifikatoru.
FUNKCIJA	Ja izdzēsta kolonna tiks aizvietota ar vienu citu kolonnu, tad šī kolonna ir tukša. Citā gadījumā šī kolonna satur funkcijas nosaukumu, kur argumentu vietā ir kolonnas indeksi no tabulas DWH_AIZV_KOLONNAS atbilstošiem ierakstiem vai konstantes.

Ja izdzēstu kolonnu aizvieto tikai viena cita avota tabulas kolonna, tad tiek izveidots viens ieraksts tabulā DWH_AIZV_KOLONNAS. Ja izdzēsta kolonna tiks izrēķināta no citām avota tabulas kolonnām ar funkciju, tad katrai kolonnai, kas piedalās funkcijas izteiksmē tiek izveidots atsevišķs ieraksts tabulā DWH_AIZV_KOLONNAS, kur kolonnā AVOTA_KOLONNA ieraksta atbilstošas kolonnas nosaukumu, bet kolonnā INDEKSS atbilstošas kolonnas kārtas numuru funkcijas izteiksmē.

Tabula 6. Tabulas DWH_AIZV_KOLONNAS kolonnas

Kolonna	Apraksts
AIZVIETOSANA_ID	Aizvietošanas identifikators, uz ko attiecas kolonna
AVOTA_KOLONNA	Avota kolonnas nosaukums
INDEKSS	Ja izdzēsta kolonna ir izrēķināta no citām avota tabulas kolonnām, tad šī kolonna satur kolonnas (kas aizvieto) kārtas numuru citu funkcijas argumentu starpā funkcijas izteiksmē.

Ja administrators izvēlas papildināt filtrēšanas vai savienojuma nosacījumu, tad Adaptācijas komponents pieprasa administratoram papildinājumus visiem filtrēšanas vai savienojuma nosacījumiem, kuros piedalās izdzēsta kolonna. Katrai šādai nosacījuma izteiksmei tiek veidots ieraksts tabulā DWH_IZTEIKSMES (Tabula 7.), kur ieraksta Filtra vai Savienojuma komponenta identifikatoru, oriģinālās izteiksmes daļu, kur piedalījās parametrs, kas attēloja izdzēsto kolonnu vai bija nokopēts no cita parametra, kas attēloja izdzēsto kolonnu, un izteiksmes daļu, kas aizvietos oriģinālās izteiksmes daļu. Ja administrators izvēlas izdzēst nosacījuma daļu, tad kolonnā AIZV_IZTEIKSMES_DALA neko neieraksta.

Tabula 7. Tabulas DWH_IZTEIKSMES kolonnas

Kolonna	Apraksts
ADAPTACIJA_ID	Adaptācijas identifikators, uz ko attiecas izteiksme
MAP_COMPONENT_ID	Komponenta identifikators, kas satur izteiksmi
ORIG_IZTEIKSMES_DALA	Oriģinālā izteiksmes daļa
AIZV_IZTEIKSMES_DALA	Izteiksmes daļa, kas aizvieto oriģinālu
PARAMETER_ID	Parametra identifikators, kas atbilst izdzēstai kolonnai

Kad administrators izvēlas veikt noteiktas adaptācijas visām neapstrādātām avota izmaiņām, un datu noliktavas izmaiņu repozitorijs ir aizpildīts, un ja administrators ir izlēmis veidot jaunu datu noliktavas versiju, tad adaptācijas komponents maina datu noliktavas repozitorija shēmas metadatus, lai tie saturētu jaunu versiju. Ja administrators izvēlas veikt adaptāciju, neveidojot jaunu versiju, adaptācijas komponentam nav nepieciešams veidot jaunu versiju metadatos un tas atjauno pēdējās shēmas versijas metadatus. Tālāk adaptācijas komponents maina ETL procesu specifikāciju (attēlojumu metadatus). Adaptācijas komponents izveido arī jaunas datu struktūras, lai glabātu jaunas datu noliktavas shēmas versijas datus, vai veic datu noliktavas shēmas adaptāciju pa tiešo datubāzē. Izmaiņas datubāzē tiek veiktas, lai saskaņotu datu noliktavas datu bāzes realizāciju ar shēmas metadatiem.

5.3.3. ETL procesu kompilācija

Adaptācijas vai pielāgošanas beigās (gan izmaiņu gadījumā datu avotos, gan darījumprasībās) mainīto ETL procesu specifikācijai tiek izveidots metadatu kompilācijas skripts, ko izveido vai nu adaptācijas komponents avota izmaiņu gadījumā, vai metadatu pārvaldības rīks, ja tika mainītas datu noliktavas darījumprasības. Kompilācijas skriptu palaiž metadatu kompilācijas rīks, kas ģenerē izpildāmo ETL skriptu, kuru tālāk izpilda datu noliktavas lādētājs. Par veiksmīgu adaptāciju un pielāgošanu tiek paziņots datu noliktavas administratoram, kurš tad var palaist mainītu ETL procesu izpildi.

5.3.4. Atskaišu adaptācija

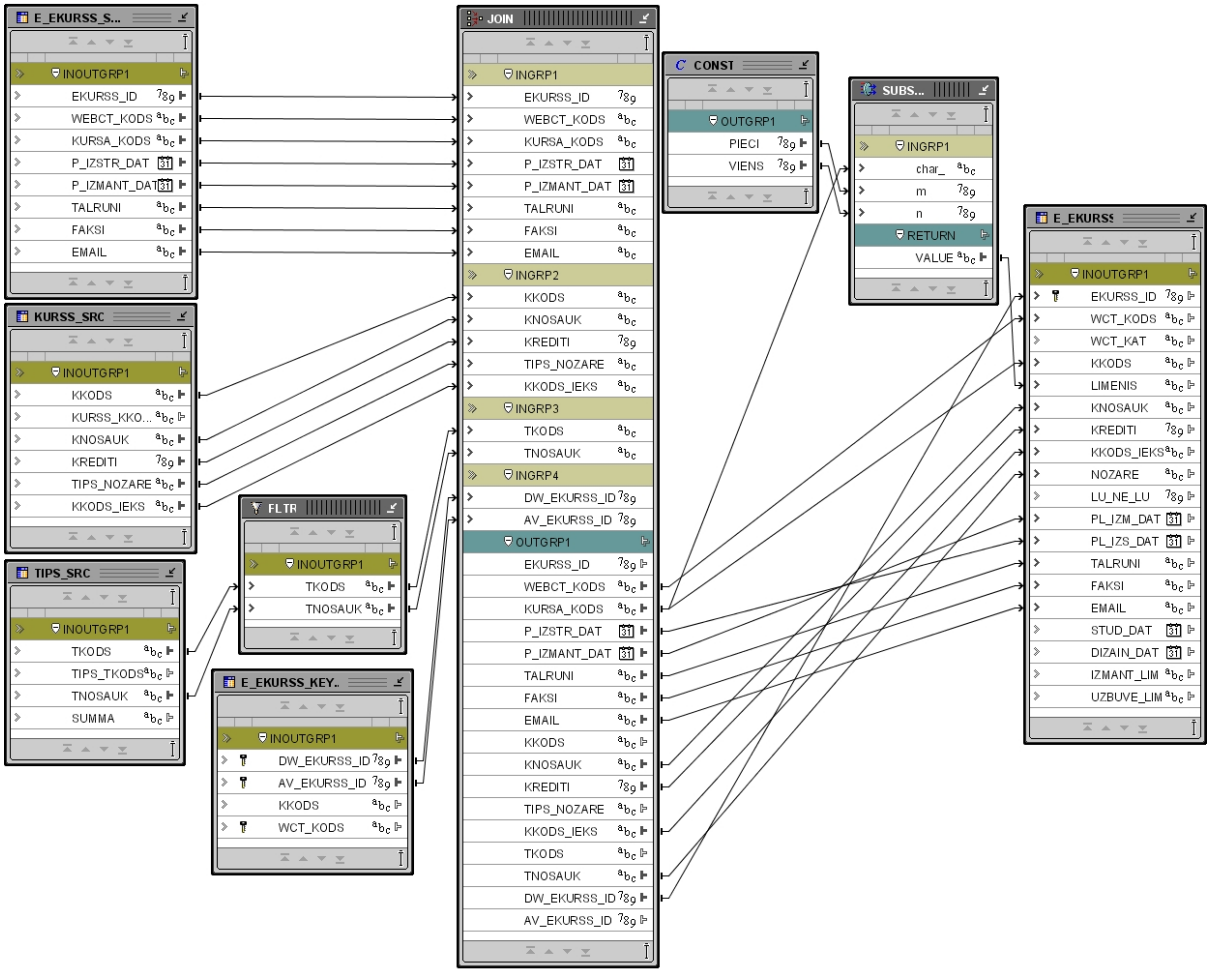
Kad notiek izmaiņas datu noliktavas shēmā, kuras izraisīja datu noliktavas darījumprasību evolūcija vai izmaiņas datu avotu shēmā, lietotāju vidē atskaišu metadatu repozitorija atskaišu un shēmas metadati, kas apraksta datu noliktavu, kļūst neatbilstoši mainītai shēmai vai jaunai shēmas versijai un vaicājumi uz datu noliktavu vairs nevar izpildīties. Tāpēc jaunu datu pārnešanas laikā no izstrādes vides uz lietotāju vidi datu pārnešanas procedūra atjauno arī atskaišu un versiju metadatus, lai tie atbilstu jaunajai datu noliktavas shēmas versijai. Piedāvātajā pieejā datu noliktavas shēmas izmaiņu un shēmas versiju izveidošanas gadījumos lietotāja definētas atskaites turpina izpildīties, pateicoties atskaišu attēlošanas rīka vaicājumu konstruēšanas algoritmam (kas ir aprakstīts nodaļā 8.), kas ir viens no šī darba būtiskiem jauninājumiem.

5.3.5. Prototipa realizācija

Arhitektūras prototipa realizācijā tika izmantotas Oracle Warehouse Builder kā *metadatu pārvaldības rīks* un kā repozitorijs *datu noliktavas shēmas metadatu* glabāšanai.

Oracle Warehouse Builder rīka standarta iespējas tika izmantotas datu noliktavas ETL procesu specifikācijai [Ora03a]. ETL procesa specifikācijas piemērs E-kursu dimensijai e-studiju datuvē ir redzams attēlā 18. Piemēra attēlojumā vispirms dati no avota tabulām un atslēgu kartējuma tabulas tiek savienoti, izmantojot Savienojuma komponentu (JOIN). Pirms tam tabulas TIPS_TMP ieraksti papildus tiek filtrēti (FILTER) no pārējiem tipiem, lai atstātu tikai nozares nosaukumus, ar filtrēšanas nosacījumu. Kursa līmenis tiek iegūts no kursa koda ar funkciju, kas tiek attēlota kā Transformācijas komponents (SUBSTR) ar ieejas parametriem KURSA_KODS un Konstantes komponenta (CONST) parametriem PIECI un VIENS.

Arhitektūras prototipa realizācijā *attēlojumu metadatiem* tika izmantots standarta Oracle Warehouse Builder rīka repozitorijs. Šajā rīkā informāciju par attēlojumiem var iegūt no publiskiem skatiem, kas arī tiek lietoti datu avotu izmaiņu apstrādē un datu noliktavas adaptācijā (skatīt nodaļu 5.3.2.).



Att. 18. Attēlojuma piemērs

Prototipa realizācijā tika izveidoti *adaptācijas metadati, avotu izmaiņu repozitorijs, datu noliktavas izmaiņu repozitorijs* un *adaptācijas komponents* atbilstoši šajā nodaļā 5. aprakstītajam. Prototips tika uzprogrammēts valodā PL/SQL un tam ir interneta saskarne (attēls 19.).

Priekšlikumi adaptācijām

Īpašnieks	Laiks	Darbība	Objekta nosaukums	Tabulas nosaukums	Datu noliktavas tabula	Datu noliktavas kolona	Adaptācijas veids	Apraksts
DISCOVER	11.05.2006 12:27:06	DROP	FAKSI	E_EKURSS_TMP	E_EKURSS	FAKSI	Neaizpildīt tālāk	Tabulas E_EKURSS_TMP kolona FAKSI tika dzēsta. Adaptācija: Neaizpildīt tālāk tabulas E_EKURSS atkarīgu kolonu FAKSI
DISCOVER	11.05.2006 12:27:06	DROP	FAKSI	E_EKURSS_TMP	E_EKURSS	FAKSI	Izdzēst atribūtu	Tabulas E_EKURSS_TMP kolona FAKSI tika dzēsta. Adaptācija: Izdzēst tabulas E_EKURSS atkarīgu kolonu FAKSI
DISCOVER	11.05.2006 12:27:06	DROP	FAKSI	E_EKURSS_TMP	E_EKURSS	FAKSI	Aizvietot ar citu kolonu	Tabulas E_EKURSS_TMP kolona FAKSI tika dzēsta. Adaptācija: Aizvietot izdzēstu kolonu ar citu tabulas E_EKURSS_TMP kolonu
DISCOVER	11.05.2006 12:27:06	DROP	FAKSI	E_EKURSS_TMP	E_EKURSS	FAKSI	Izrēķināt no citām kolonām	Tabulas E_EKURSS_TMP kolona FAKSI tika dzēsta. Adaptācija: Izrēķināt izdzēstu kolonu no citām tabulas E_EKURSS_TMP kolonām
DISCOVER	11.05.2006 12:27:06	DROP	FAKSI	E_EKURSS_TMP	E_EKURSS	FAKSI	Neveikt adaptāciju	Tabulas E_EKURSS_TMP kolona FAKSI tika dzēsta. Adaptācija: neveikt adaptāciju
DISCOVER	11.05.2006 12:28:29	MODIFY	TNOSAUK	STATUSI_TMP	E_EKURSS	NOZARE	Mainīt datu tipu	Tabulas STATUSI_TMP kolonas TNOSAUK datu tips tika mainīts. Mainījās sekojošas īpašības: Datu garums no 742 uz 1500. Adaptācija: Līdzīgi mainīt datu tipu tabulas E_EKURSS atkarīgai kolonai NOZARE
DISCOVER	11.05.2006 12:28:29	MODIFY	TNOSAUK	STATUSI_TMP	E_EKURSS	NOZARE	Aizvietot ar citu kolonu	Tabulas STATUSI_TMP kolonas TNOSAUK datu tips tika mainīts. Adaptācija: Aizvietot mainītu kolonu ar citu tabulas STATUSI_TMP kolonu

Att. 19. Adaptācijas risinājumi

Izstrādātajā prototipā *metadatu kompilācijas rīka* darbības veic Oracle Warehouse Builder sastāvdaļa OMB Plus, kas izpilda Warehouse Builder Oracle MetaBase (OMB) skriptus [Ora03a]. Bet *datu noliktavas lādētājs* tika realizēts kā PL/SQL valodas procedūru komplekts.

Relācijas datu bāzes datu avotiem *aģents*, kas nodarbojas ar izmaiņu savākšanu datu avota tabulās, tika realizēts kā DDL trigeris, kas izseko visas DDL operācijas datu avotā un ieraksta to rezultātus un vēsturi avota izmaiņu repozitorijā. Prototipa realizācija tika izmantoti Oracle datu bāzes pārvaldības sistēmas datu avoti.

Promocijas darba ietvaros tika izstrādāti jaunie datu noliktavas *shēmas metadati*, kas atbalsta datu noliktavas shēmas versijas, un *atskaišu metadati*, kas tiek aprakstīti nodaļā 6.

Tika izveidots arī *atskaišu definēšanas rīks*, ko noprogrammēja (ne promocijas darba autore) atbilstoši promocijas darba autores izstrādātai specifikācijai un metadatiem. Attēlā 20. ir redzams izstrādātā atskaišu definēšanas rīka ievadformas piemērs.

Promocijas darba ietvaros tika izstrādāts *atskaišu attēlošanas rīks*, kura darbības apraksts ir atrodams nodaļā 8. un apakšnodaļā 9.5.

Att. 20. Atskaišu definēšanas rīka ievadformas piemērs

5.4. Secinājumi par datu noliktavas evolūcijas arhitektūru

Datu noliktavas evolūcijas arhitektūra tapa papildinot adaptācijas arhitektūru, kas tika noprojektēta un realizēta iepriekš [SN06]. Adaptācijas arhitektūra varēja automātiski noteikt izmaiņas datu avotu shēmās un adaptēt datu noliktavas shēmu un ETL procesus, saskaņā ar administratora lēmumu.

Atšķirībā no adaptācijas arhitektūras datu noliktavas evolūcijas arhitektūra ir spējīga apstrādāt ne tikai izmaiņas datu avotos, bet arī tiešas izmaiņas datu noliktavas shēmā. Otrā

svarīgākā atšķirība ir tā, ka evolūcijas arhitektūrā ir paredzēts datu noliktavas versiju atbalsts, gan izstrādes vidē, gan lietotāju vidē atskaitēs.

No pārējiem literatūrā sastopamiem datu noliktavas evolūcijas problēmu risinājumiem piedāvāto arhitektūru atšķir tas, ka tā atbalsta uzreiz vairākas evolūcijas problēmas, nevis tikai apakškopu. Tiek atbalstītas gan izmaiņas datu avotos, kas ietekmē datu noliktavas shēmu (datu noliktavas adaptācija), gan izmaiņas datu noliktavas darījumprasībās (datu noliktavas pielāgošana), gan ETL procesu adaptācija un izpildīšana (datu noliktavas uzturēšana). Izvēlētajā pieejā ir atbalstītas vairākas datu noliktavas shēmas versijas un datu noliktavas realizācijai tika izvēlēta daudzdimensiju shēma.

Šīs arhitektūras būtiska daļa, kas tika izstrādāta šī darba ietvaros, ir atskaišu definēšanas un attēlošanas rīks un metadati, kas apraksta datu noliktavas shēmu un atbalsta šīs shēmas vairākas versijas, kā arī definē transformācijas elementiem no vienas shēmas versijas uz citu. Šajā nodaļā aprakstīta arhitektūra definē, kā var notikt izmaiņas datu noliktavas shēmā un kā šīs izmaiņas varētu būt realizētas. Arhitektūras komponenti tika realizēti prototipā, kas daļēji pārbauda arhitektūras idejas. Sekojošajās nodaļās tiks aprakstīti metadati, kas atļauj daudzversiju datu noliktavas izveidi, kā arī atskaišu definēšanas un attēlošanas rīka darbība.

Šajā nodaļā piedāvāta datu noliktavas evolūcijas arhitektūra tika aprakstīta un publicēta rakstā:

Solodovņikova D. 'Data Warehouse Evolution Framework'. *The Fourth Spring Young Researchers Colloquium on Databases and Information Systems, SYRCoDIS'2007, Moscow, Russia,* 2007.

Datu noliktavas adaptācijas risinājums pēc izmaiņām datu avotos ar adaptācijas komponentu un metadatiem, kā arī datu noliktavas adaptācijas arhitektūra, tika aprakstīti un publicēti rakstā:

Solodovņikova D., Niedrīte L. 'Data Warehouse Adaptation after Changes in Source Schemata'. *Proceedings of the 7th International Baltic Conference on Databases and Information Systems, Vilnius, Lithuania,* 2006.

6. DAUDZVERSIJU DATU NOLIKTAVAS FORMĀLĀIS MODELIS UN METADATI

6.1. Nodaļas nolūks

Šajā nodaļa tiek dots formālais modelis daudzversiju datu noliktavai. Formālais modelis apraksta datu noliktavu loģiskajā, fiziskajā un semantiskajā līmenī. Loģiskais līmenis apraksta datu noliktavas shēmas versijas un tas nav atkarīgs no konkrētās implementācijas. Fiziskais līmenis atspoguļo vairāku datu noliktavas shēmas versiju glabāšanu relāciju datu bāzē. Bet semantiskais līmenis apraksta datu noliktavas shēmas elementus biznesa terminoloģijā.

Papildus daudzversiju datu noliktavas formalizācijai, šajā nodaļā ir aprakstīti daudzversiju datu noliktavas shēmas metadati un atskaišu metadati. Šie metadati būtībā ir formālā modeļa realizācija. Šie metadati ir izmantoti datu noliktavas evolūcijas arhitektūrā, kas ir aprakstīta nodaļā 5. Semantiskā, loģiskā un fiziskā līmeņa metadati atbilst datu noliktavas repozitorija shēmas metadatiem. Atskaišu metadati atbilst datu noliktavas evolūcijas arhitektūras atskaišu metadatu repozitorija atskaišu metadatiem. Šajā nodaļā ir apskatīta pieeja vairāku datu noliktavas shēmas versiju glabāšanai relāciju datubāzē. Relāciju datubāzēs nav iebūvētu līdzekļu, lai detalizēti aprakstītu daudzdimensiju modeli. Tāpēc šim nolūkam tiek izmantoti shēmas metadati. Šie metadati apraksta datu noliktavas daudzdimensiju shēmas specifiku (faktu tabulas, dimensijas, hierarhijas, u.c.), shēmas versijas un daudzversiju datu noliktavas glabāšanas veidu relāciju datubāzē. Atskaišu metadati ir cieši saistīti ar shēmas metadatiem un tie apraksta atskaišu definīcijas no datu noliktavas shēmas elementiem.

Par pamatu daudzdimensiju datu noliktavas formālam modelim un arī metadatiem tika izmantots Common Warehouse Metamodel [OMG02], [PCT+03], kas ir datu noliktavas metadatu standarts, ko ir izstrādājusi organizācija Object Management Group, lai vienkāršotu metadatu apmaiņu starp datu noliktavu lietojumprogrammām. CWM satur vairākas pakotnes, kas apraksta dažādus datu noliktavas aspektus. Daudzdimensijas shēmas aprakstīšanai tiek izmantoti elementi, kas ir būtiski datu noliktavas modelim, no sekojošām CWM pakotnēm: OLAP, Relational, Transformation un Business Nomenclature. CWM tika paplašināts, lai nodrošinātu datu noliktavas evolūciju.

Promocijas darba piedāvātajā formālajā modelī un metadatos ir iespējams modelēt datu noliktavas shēmu fiziskajā, loģiskajā un semantiskajā līmenī. Fiziskais modelis balstās uz CWM pakotni Relational, kas apraksta relāciju datus, loģiskais modelis balstās uz CWM pakotni OLAP, kas apraksta datus, kas ir nepieciešami OLAP rīkiem. Šie divi līmeņi ir saistīti caur objektiem, kas ir definēti CWM pakotnē Transformation. Bet semantiskais līmenis ir balstīts uz CWM pakotni Business Nomenclature.

CWM metamodelī datu noliktavas elementi ir modelēti kā klases, bet šīs nodaļas formālajā modelī papildus klašu diagrammām tiek dotas arī formālas definīcijas daudzdimensiju datu noliktavas shēmu versijām, to realizācijai un izmantotiem terminiem un jēdzieniem, lai varētu formāli aprakstīt datu noliktavas shēmas izmaiņas. CWM klašu diagrammas tika papildinātas ar datu struktūrām datu noliktavas shēmas versiju atbalstam

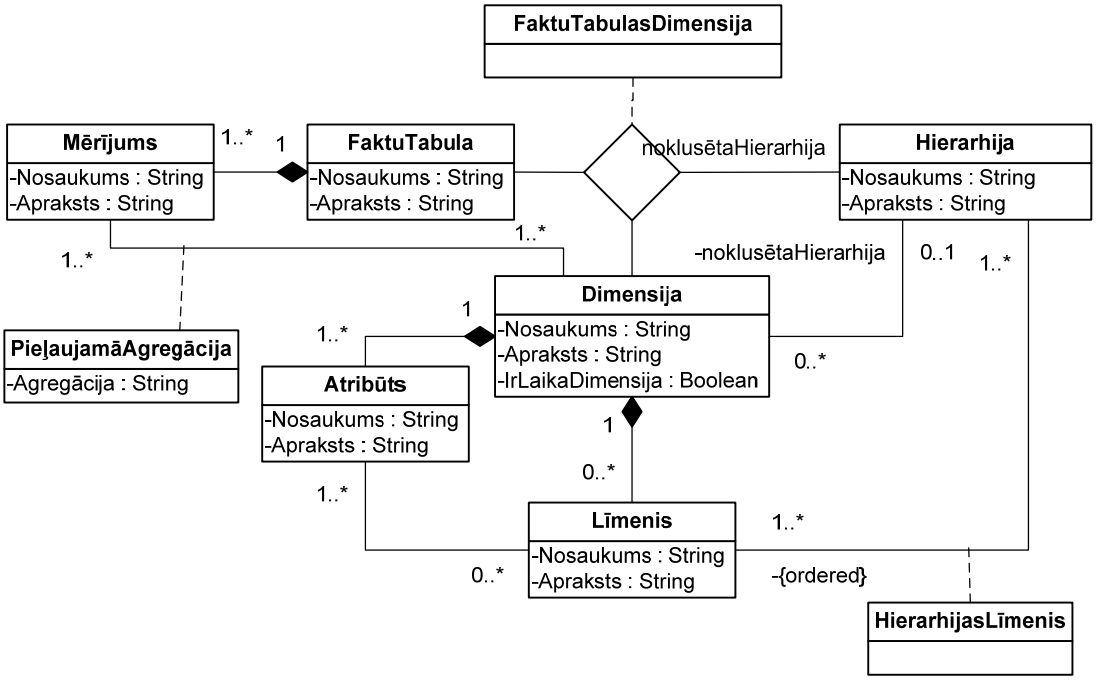
loģiskajā un semantiskajā līmenī, ar lietotāju tiesību definīciju fiziskajā līmenī. Tika izstrādāts arī oriģinālais atskaišu metamodelis.

6.2. Datu noliktavas formālais modelis

6.2.1. Loģiskais līmenis

Loģiskajā līmenī datu noliktava ir shēmu un to versiju kopa (attēli 21., 22.). Šī līmeņa klases balstās uz CWM pakotni OLAP. Katrai datu noliktavas datuvei tiek veidota atsevišķa shēma (klase *Shēma*), kas satur dimensijas (klase *Dimensija*) un faktus (klase *FaktuTabula*), kas ir savā starpā saistīti ar asociāciju *FaktuTabulasDimensija*. Faktiem ir mērījumi (klase *Mērījums*), kas glabā skaitliskas vērtības. Mērījumiem eksistē pieļaujamas agregācijas attiecībā pret dažādām dimensijām, piemēram, SUM, AVG, COUNT u.c. Tās ir realizētas kā asociācijas *PieļaujamāAgregācija* starp klasēm *Dimensija* un *FaktuTabula*. Pieļaujamās agregācijas nav iekļautas CWM un tās tika ieviestas, lai atskaitēs lietotāji varētu pielietot tikai noteiktas agregācijas funkcijas konkrētiem mērījumiem attiecībā pret konkrētām dimensijām, lai neiegūtu nesavietojamus rezultātus, jo daži mērījumi var būt summējami attiecībā pret vienu dimensiju un nav summējami attiecībā pret citu dimensiju.

Atbilstoši CWM, faktam var būt kādas dimensijas noklusēta hierarhija, kas tiek izmantota, lai grupētu faktu tabulas mērījumus attiecībā pret šīs dimensijas līmeņiem. Noklusēta hierarhija ir iekļauta asociācijā *FaktuTabulasDimensija*.

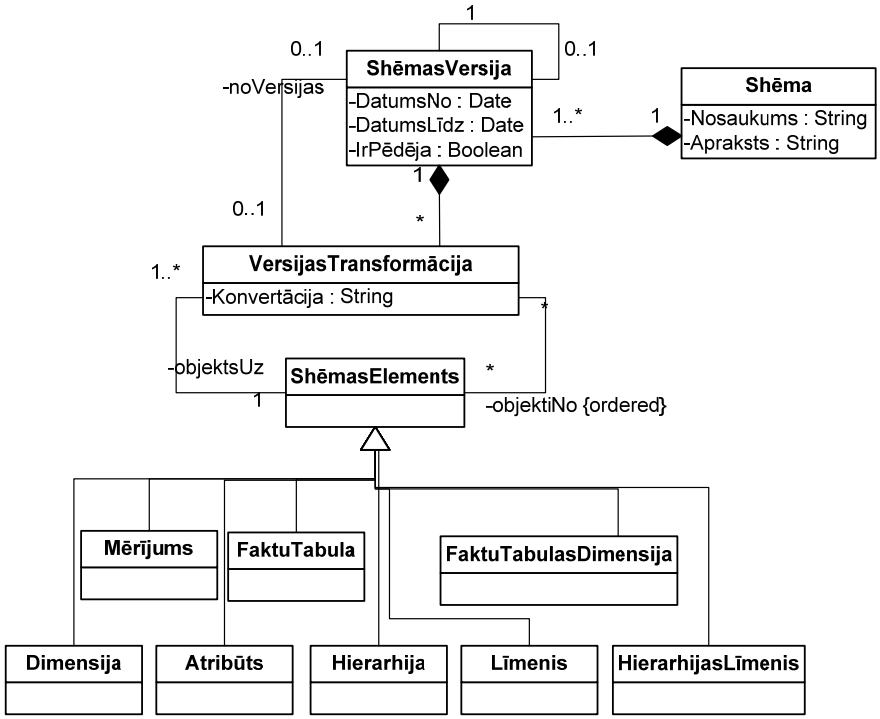


Att. 21. Loģiskā līmeņa formālais modelis

Dimensijas klase satur atribūtu *IrLaikaDimensija*, kas īpaši atzīmē laika dimensiju. Dimensijām var būt vairākas hierarhijas, kas tiek attēlotas kā klase *Hierarhija*. Viena no hierarhijām var būt noklusēta, tā ir atzīmēta ar asociāciju *noklusētaHierarhija*. Hierarhijas apraksta attiecības starp dimensiju līmeņiem, kuriem atbilst klase *Līmenis*. Dimensijas sastāv no atribūtiem, kuriem atbilst klase *Atribūts*. Saistība starp līmeni un atribūtu tiek definēta ar

asociāciju starp klasēm Atribūts un Līmenis. Līmeņi ir piesaistīti hierarhijai ar asociāciju *HierarhijasLīmenis*. Šajā asociācijā līmeņi ir sakārtoti, lai nodrošinātu līmeņu hierarhiju.

Lai būtu iespēja attēlot dažādas datu noliktavas shēmas versijas, CWM tika paplašināts ar klasēm *ShēmasVersija* un *VersijasTransformācija* (attēls 22.). Katrai shēmas versijai atbilst klases *ShēmasVersija* instance. Katrai versijai ir derīguma periods, kas ir definēts atribūtos *DatumsNo* un *DatumsLīdz*. Visu versiju derīguma periodi nepārklājas un iet pēc kārtas. Atribūts *IrPēdējā* norāda, vai šī versija ir tagad spēkā. Katrai versijai, izņemot pirmo, ir norāde uz iepriekšējo versiju, no kuras tā tika iegūta. Šī norāde ir realizēta kā asociācija starp klases *ShēmasVersija* instancēm.



Att. 22. Shēmas versijas loģiskajā līmenī

Piedāvātajā pieejā ir iespējams veidot versijas sekojošiem shēmas elementiem: mērījumiem, atribūtiem, faktu tabulām, dimensijām, faktu tabulu un dimensiju asociācijām, hierarhijām, līmeņiem un asociācijām starp līmeni un hierarhiju. Katras versijas tajā esošie shēmas elementi un elementi, kas neeksistē versijā, bet var tikt iegūti no citiem eksistējošiem shēmas elementiem, tiek piesaistīti klasei *ShēmasVersija* caur klasi *VersijasTransformācija*. *ShēmasVersija* satur vairākas *VersijasTransformācijas*, kas atbilst katram shēmā eksistējošam vai no citiem elementiem izrēķinātam shēmas elementam. Shēmas elementu versijas tiek atbalstītas sekojoši:

- Ja kāds shēmas elements eksistē versijā, tad tam tiek veidota asociācija *objektsUz*, kas savieno elementu ar klases *VersijasTransformācija* instanci, kur atribūts *Konvertācija* ir ar vērtību NULL.
- Bet ja shēmas elements neeksistē versijā, bet var tikt iegūts ar transformācijas funkciju no citiem versijā eksistējošiem shēmas elementiem, tad šis elements ir arī savienots ar klases *VersijasTransformācija* instanci ar asociāciju *objektsUz*, bet *VersijasTransformācija* instancē atribūts *Konvertācija* satur šo transformācijas funkciju un versijā eksistējošie

elementi, kas ir izmantoti transformācijas funkcijā, ir savienoti ar to pašu klases *VersijasTransformācija* instanci ar asociāciju *objektiNo*. Arī šajā gadījumā ir izveidota papildus asociācija ar nosaukumu *noVersijas* starp *VersijasTransformāciju* un to (citu) *ShēmasVersiju*, kurā eksistē izrēķinātais shēmas elements.

Šī klašu diagramma atļauj veidot vairākas versijas transformācijas katram mainītam elementam, t.i. viens un tas pats shēmas elements var būt piesaistīts vairākām shēmas versijām ar dažādām versijas transformācijām. Piedāvātā klašu diagramma atļauj transformēt elementu datus no jebkuras versijas uz jebkuru citu, bet parasti ir izveidotas transformācijas divos virzienos no iepriekšējas shēmas versijas un uz nākošo shēmas versiju.

Tālāk tiek dota loģiskā līmeņa datu noliktavas shēmas formāla definīcija.

Shēma ir kortežs $DWS=<F,D,FDA,V,MAP>$, kur $F=\{fact_1,...,fact_a\}$ ir faktu kopa, $D=\{dim_1,...,dim_b\}$ ir dimensiju kopa, $FDA=\{fda_1,...,fda_c\}$ ir faktu un dimensiju asociāciju kopa, $V=\{v_1,...,v_u\}$ ir shēmas versiju kopa, kur $v_u \in V$ ir pēdēja shēmas versija, kas pašlaik ir derīga, un $MAP=\{map_1,...,map_e\}$ ir atribūtu un mērījumu attēlojumu kopa. Atribūtu un mērījumu attēlojumi ir aprakstīti apakšnodaļā 6.2.3.

Fakta un dimensijas asociācija fda∈FDA ir kortežs $<dim,fact>$, kas savieno dimensiju $dim \in D$ ar faktu $fact \in F$.

Dimensija dim∈D ir kortežs $<ATT,LEV,H,AL,HL>$, kur $ATT=\{att_1,...,att_f\}$ ir *atribūtu* kopa, $LEV=\{lev_1,...,lev_g\}$ ir *līmeņu* kopa, $H=\{h_1,...,h_k\}$ ir *hierarhiju* kopa, $AL=\{al_1,...,al_l\}$ ir atribūtu un līmeņu asociāciju kopa un $HL=\{hl_1,...,hl_m\}$ ir hierarhiju un līmeņu asociāciju kopa.

Atribūta un līmeņa asociācija al∈AL ir kortežs $<att,lev>$, kas savieno atribūtu $att \in ATT$ ar līmeņi $lev \in LEV$.

Hierarhijas un līmeņa asociācija hl∈HL ir kortežs $<lev,h>$, kas savieno līmeņi $lev \in LEV$ ar hierarhiju $h \in H$.

Fakts fact∈F ir kortežs $<MS,MA>$, kur $MS=\{ms_1,...,ms_n\}$ ir *mērījumu* kopa un $MA=\{ma_1,...,ma_t\}$ ir mērījumu pieļaujamo agregāciju kopa.

Mērījuma *pieļaujama agregācija* ma∈MA ir kortežs $<ms,dim,agg>$, kas definē, ka mērījumam *ms* var pielietot agregācijas funkciju $agg \in \{'SUM','AVG','MIN','MAX','COUNT','DETAIL'\}$ attiecībā pret dimensiju *dim*.

Tālāk tiek aprakstīti formālā modeļa objekti, kas atspoguļo vairāku datu noliktavas shēmas versiju būtību (attēls 22.).

Shēmas versija v∈V ir kortežs $<\tau_{from}, \tau_{to}, VT>$, kur τ_{from} un τ_{to} ir laikspiedoli, kas definē shēmas versijas derīguma periodu, $VT=\{vt_1,...,vt_r\}$ ir versiju transformāciju kopa.

Versijas transformācija vt∈VT ir kortežs $<el,conv,E>$, kas nosaka, ka elements $el \in ELEM$ ($ELEM=F \cup D \cup FDA \cup ATT \cup LEV \cup H \cup AL \cup HL \cup MS$) tiek iegūts no elementu kopas $E \subseteq ELEM$ ar konvertācijas funkciju *conv*. Ja elements *el* eksistē versijā (t.i. atbilst kādam fiziskā līmeņa elementam), tad $vt=<el,null,\emptyset>$. Šādu versijas transformāciju sauksim par *tiešo* versijas transformāciju. Caur versijas transformācijām shēmas versijai ir piesaistīti visi elementi, kas eksistē versijā vai kas ir iegūti no citiem elementiem, kas eksistē versijā.

6.2.2. Fiziskais līmenis

Fiziskajā līmenī tiek aprakstīta datu noliktavas relāciju datu bāzes shēma (attēls 23.). un daudzdimensiju shēmas attēlojums (mapping) relāciju datu bāzes datu struktūrās. Šī līmeņa klases diagramma balstās uz CWM pakotnēm Relational un Transformation. Fiziskā līmeņa klašu diagramma būtiski neatšķīrās no CWM pakotnēm Relational un Transformation, vienīgi tā tika papildināta ar klasēm, kas apraksta lietotāju tiesības datu noliktavas shēmas elementiem un datiem.

Fiziskais līmenis nesatur versiju informāciju, jo fiziski datubāzē vienmēr pastāv tikai viena shēmas versija un versijas tiek realizētas loģiskajā līmenī. Piedāvātajā pieejā ir arī iespējams atbalstīt vairākas tabulu fiziskās versijās (t.i. kad izmaiņas gadījumā fiziski tiek veidotas atsevišķas tabulas, kas glabā shēmas elementu versijas), bet nodaļā 7. aprakstītajās izmaiņu apstrādes procedūrās tiek uzskatīts, ka fiziski datubāzē visas mainītās kolonnas un šo kolonnu dati tiek glabāti vienā tabulā, lai vienkāršotu ETL procesu izveidi, mainīšanu un izpildīšanu. Vienas fiziskas shēmas versijas realizācija izvēlēta tāpēc, ka, jo vairāk ir tabulu datubāzē, jo vairāk ETL procesu procedūru ir jāveido. Turklāt, ja vieni un tie paši dati glabājas dažādās tabulās, tad ir grūtāk nodrošināt referenciālo integritāti.

Klase *KolonnuKopa* attēlo relāciju datu veidus. Tā var būt tabula, skats vai SQL vaicājuma rezultāts. Atribūtā *AtjaunošanasDatums* tiek glabāts datums, kad kolonnu kopā pēdējo reiz tika atjaunoti dati. Kolonnu kopa sastāv no kolonnām (klase *Kolonna*), kam ir aprakstīti nosaukums, datu tips (atribūti *DatuTips, Precizitāte, Skala, Garums*) un vai ir atļautas NULL vērtības (atribūts *IrNull*).

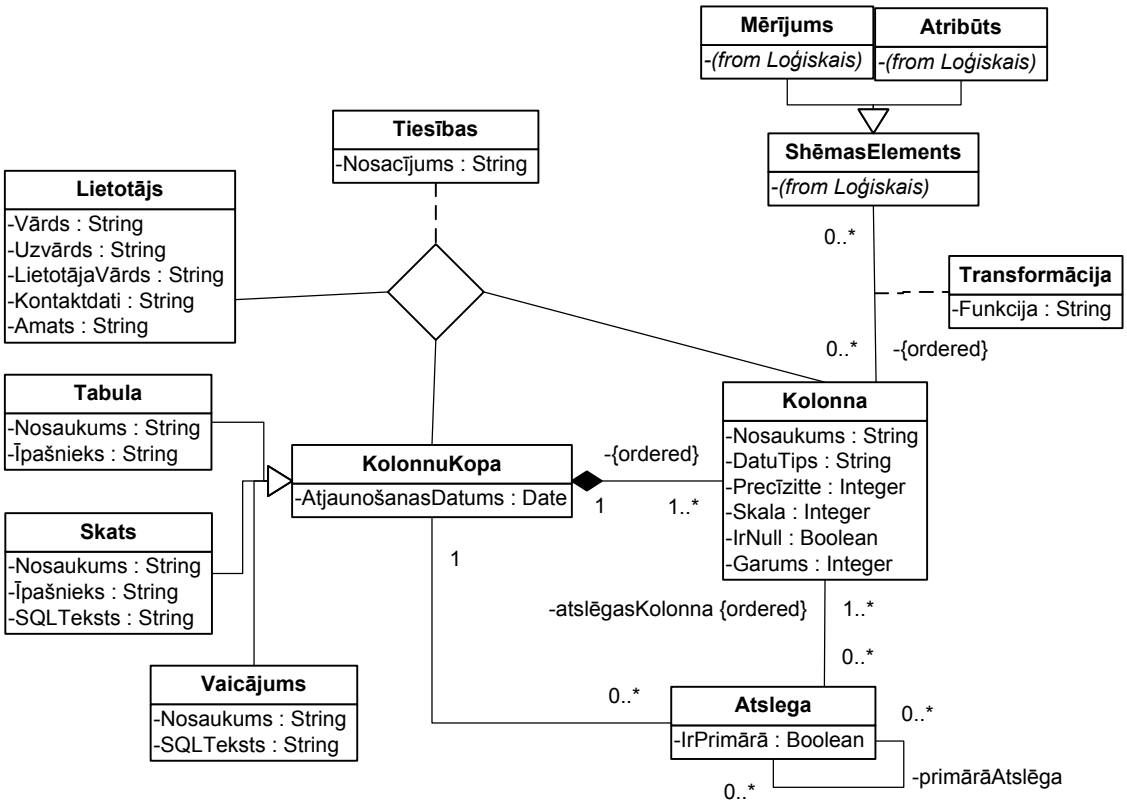
Kolonnu kopai arī var būt viena primārā un viena vai vairākas ārējās atslēgas, kam atbilst klases *Atslēga* instances. Atribūts *IrPrimārā* definē, vai atslēga ir primārā vai ārējā. Klasei *Atslēga* ārējām atslēgām ir izveidota asociācija uz citu klases *Atslēga* instanci, kas apzīmē attiecīgas kolonnu kopas primāro atslēgu. Atslēgu kolonnas ir sasaistītas ar primāro vai ārējo atslēgu caur asociāciju *atslēgasKolonna*.

Asociācija *Transformācija* savieno tabulu, skatu vai vaicājumu kolonnas ar atribūtiem un mērījumiem no loģiskā līmeņa modeļa un definē, kā tiek iegūti atribūtu un mērījumu dati no kolonnām (atbilstoši CWM pakotnei Transformation). Vispārīgajā gadījumā *Transformācijas* definē funkcijas, kas no vienas vai vairākām kolonnām iegūst atribūtu vai mērījumu vērtības.

Fiziskais modelis tika papildināts ar klasi *Lietotājs* un asociāciju *Tiesības*, kas ir nepieciešamas lietotāju tiesību pārvaldei. Ar šo objektu palīdzību var definēt informāciju, kas ir pieejama lietotājiem. Datu noliktavā var glabāties jūtīgi dati, ko nevar izmantot visi lietotāji. Arī dažu datu aplūkošana var būt atļauta vieniem lietotājiem, bet aizliegta citiem, piemēram, departamentu vadītāji var apskatīt tikai sava departamenta datus. Lietotāju tiesības atļauj ne tikai ierobežot datu aplūkošanu saskaņā ar drošības prasībām, bet arī atfiltrēt informāciju, kas lietotāju neinteresē.

Klases *Lietotājs* instances atbilst datu noliktavas lietotājiem, kam ir tiesības uz datu noliktavas datiem. Asociācija *Tiesības* definē, ka noteiktam lietotājam ir tiesībām uz datu noliktavas tabulām, skatiem vai SQL vaicājumu rezultātiem, kolonnām un datiem tajās. Šajā asociācijā atribūts *Nosacījums* satur nosacījumu, kam ir jāizpildās, lai lietotājs varētu apskatīt datus kolonnu kopā vai kolonnā. Ja lietotājam ir tiesības uz visām kolonnām kolonnu kopā,

tad asociācija *Tiesības* tiek veidota kā bināra asociācija starp klases *Lietotājs* instanci un klases *KolonnuKopa* instanci.



Att. 23. Fiziskā līmeņa formālais modelis

Tālāk tiek dota fiziskā līmeņa datu noliktavas relāciju datu bāzes shēmas formāla definīcija.

Relāciju datu bāzes shēma ir kortežs $DBS=<CS>$, kur $CS=\{cs_1,...,cs_s\}$ ir *kolonnu kopu* kopa. Kolonnu kopa atspoguļo datu bāzes tabulu, skatu vai SQL valodas vaicājumu.

Kolonnu kopa $cs \in CS$ ir kortežs $<C,KEY,DOMA>$, kur $C=\{c_1,...,c_y\}$ ir *kolonnu* kopa, $KEY=\{key_1,...,key_m\}$ ir atslēgu kopa un $DOMA=\{doma_1,...,doma_z\}$ ir kolonnu un to vērtību apgabalu (domēnu) asociāciju kopa.

Atslēga $key \in KEY$ ir kortežs $<type,KC,pk>$, kur $type \in \{$'PRIMARY', 'FOREIGN'$\}$, $KC \subset C$ ir *atslēgas kolonnu* kopa. Ja *type='FOREIGN'* (ārējā atslēga), tad $pk \in KEY$' ir primārā atslēga, *KEY'* ir jebkuras kolonnu kopas atslēgu kopa. Ja *type='PRIMARY'* (primārā atslēga) tad *pk* ir *null*.

Kolonnas un vērtību kopas asociācija $doma \in DOMA$ ir kortežs $<c,dom>$, kas sasaista vērtību apgabalu *dom* ar kolonnu $c \in C$.

6.2.3. Attēlojumi

Datu noliktavas mērījumu un atribūtu *attēlojums* $map \in MAP$ ir kortežs $<el,mapfun,EC>$, kur $el \in ATT \cup MS$ ir attēlots elements, *mapfunc* ir funkcija, kas specificē kā elements *el* tiek iegūts no kolonnu kopas $EC \subseteq C$. Šajā darbā tiek apskatīts vienkāršākais

gadījums, kad atribūti un mērījumi tiek iegūti pa tiešo no datu bāzes kolonnām ar attēlojuma funkciju "kopija", t.i. *map=<el,'kopija',{c}>*, kur $c \in C$.

6.2.4. Lietotāju tiesības

Fiziskajā līmenī (attēls 23.) arī ir definētas lietotāju tiesības datu noliktavas elementiem un datiem. *Lietotāju tiesības* ir kortežs *UP=<PRIV>*, kur $PRIV=\{priv_1,...,priv_x\}$ ir atsevišķu tiesību kopa.

Tiesība priv∈PRIV ir kortežs *<usr,cs,c,pcond>*, kur *usr* ir datu noliktavas lietotājs, $cs \in CS$ ir kolonnu kopa, $c \in C$ ir kolonna un *pcond* ir nosacījums, kam jāizpildās, lai lietotājs varētu apskatīt datus kolonnā *c*. Ja lietotājam ir pieejamas visas kolonnu kopas *cs* kolonnas, tad *c* ir *null*.

6.2.5. Atskaišu modelis

Atskaišu modelis apraksta lietotāju definētu atskaišu struktūru (attēls 24.).

CWM satur pakotni Information Visualization, kas apraksta kādā veidā datu noliktavas shēmas elementi tiek attēloti, piemēram, atskaitēs, grafikos, interneta vietnēs u.c. Šis CWM modelis nav pietiekams, jo ir ļoti vispārīgs un precīzi neapraksta atskaites elementus. Tāpēc atskaišu definēšanas un attēlošanas rīkam tika izveidots jauns atskaišu modelis. Tas tika veidots līdzīgi Oracle Discoverer atskaišu struktūrai, jo tas atļauj definēt un izpildīt plašu atskaišu loku.

Atskaites ir grupētas darba burtnīcās (klase *DarbaBurtnīca*) un darba lapās (klase *DarbaLapa*). Katra darba burtnīca satur vienu vai vairākas darba lapas, kur katra lapa atbilst vienai atskaitei. Darba lapas var būt divu veidu: tabulas vai matricveida, ko nosaka atribūts *Tips*.

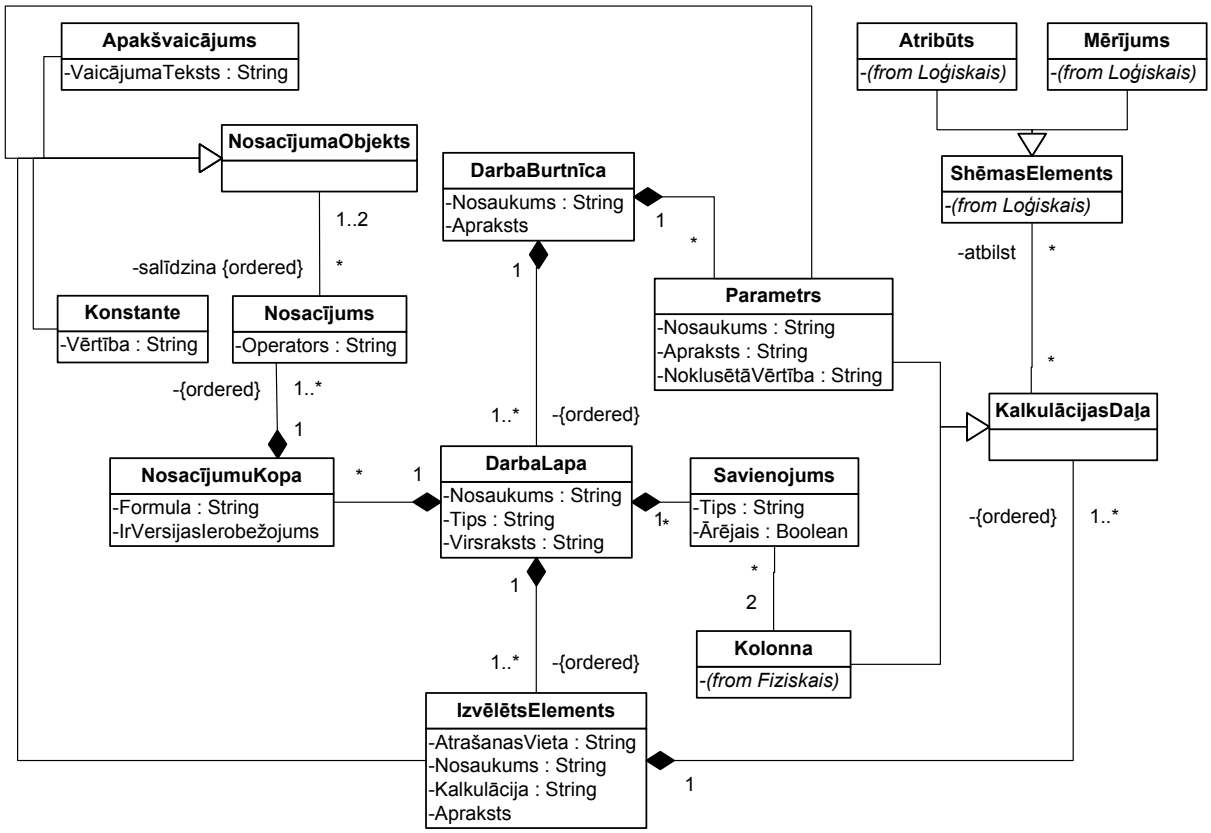
Darba lapas satur izvēlētus elementus (klase *IzvēlētsElements*), kas definē atskaites datus. Izvēlētu elementu dati var atrasties tabulas veida atskaites kolonnās vai matricveida atskaites kolonnu un rindu galvenēs vai šūnās. Izvēlētie elementi var definēt arī lappušu objektus, kas satur izvēlētā elementa vērtības un lietotājs var izvēlēties vienu no vērtībām, lai attēlotu atskaitē tikai šai vērtībai atbilstošus datus. Lappušu objekti tiek izmantoti, lai veiktu sagriešanas (slicing) OLAP operāciju. Izvēlēts elements tiek definēts ar kalkulāciju (atribūts *Kalkulācija*), kas satur formulu, ar ko aprēķina izvēlētā elementa vērtības. Izvēlētā elementa kalkulācijas formulas operandi ir klases *KalkulācijasDaļa* instances, kas var būt vai nu parametri vai kolonnas no fiziskā līmeņa modeļa. Tas nozīmē, ka izvēlēts elements var tikt izrēķināts no vienas vai vairākām kolonnām vai lietotāja ievadītām parametru vērtībām. Ja kalkulācijas daļa ir datu noliktavas shēmas elements (dimensijas atribūts vai faktu tabulas mērījums no loģiskā līmeņa modeļa), tad atbilstoša klases *Atribūts* vai *Mērījums* instance tiek savienota ar klases *KalkulācijasDaļa* instanci ar asociāciju *atbilst*.

Klases *Savienojums* atbilst savienojumiem starp atskaitē izmantotajām kolonnu kopām. Savienojumi definē attiecības starp kolonnām no fiziskā līmeņa modeļa. Savienojumā tiek norādīts tips, kas ir salīdzināšanas operators ('=', '<>', '<' u.c.), kas salīdzina kolonnas. Savienojums var būt ārējais, kas tiek norādīts atribūtā *Ārējais*.

Atskaitei var arī definēt dažādus nosacījumus (klase *Nosacījums*). Katrs nosacījums definē vai nu divu nosacījuma objektu attiecības (t.i. divi nosacījuma objekti tiek salīdzināti ar kādu salīdzināšanas operatoru, piemēram, '=', '>=', 'IN', 'LIKE', u.c.), vai viena nosacījuma objekta esamību ar operatoriem 'IS NULL', 'IS NOT NULL', 'EXISTS' un 'NOT EXISTS'. Nosacījuma objekts var būt vai nu kāds izvēlēts elements, vai konstanta teksta vai skaitliskā vērtība, vai SQL vaicājums (klase *Apakšvaicājums*), vai parametrs. Vairāki nosacījumi ir piesaistīti atskaitei caur klasi *NosacījumuKopa*, kur atribūts *Formula* satur nosacījumu formulu ar AND, OR un NOT loģiskiem operatoriem. Atskaitei varētu būt piesaistītas vairākas nosacījumu kopas, kas nozīmē, ka ir jāizpildās visām nosacījuma formulām, t.i. nosacījumu formulas ir sasaistītas ar AND loģisko operatoru.

Klases *NosacījumuKopa* un *Nosacījums* arī apzīmē versijas laika ierobežojumus (atribūts *IrVersijasIerobežojums*). Šāda veida nosacījumi norāda, ka atskaitei jāizmanto datu noliktavas shēmas versijas, kas bija spēkā tajā laikā, kas definēts ar versijas laika ierobežojumiem. Ja atskaitei netiek norādīts versijas laika ierobežojums, tad atskaitei tiek izmantotas visas shēmas versijas. Versijas laika ierobežojums neaizvieto nosacījumus uz laika dimensiju, kas parasti tiek izmantoti datu noliktavas atskaitēs.

Nosacījumos un izvēlēto elementu kalkulācijās var tikt izmantoti parametri (klase *Parametrs*), kuru vērtības ievada lietotājs, kad izpilda atskaiti.



Att. 24. Atskaišu formālais modelis

Tālāk tiek dota datu noliktavas atskaišu formāla definīcija.

Darba burtnīca ir kortežs $WB=<PARAM,WS>$, kur $PARAM=\{param_1,...,param_w\}$ ir atskaišu parametru kopa un $WS=\{ws_1,..,ws_v\}$ ir darba lapu kopa. *Parametrs* param∈PARAM ir kortežs $<dv>$, kur *dv* ir parametra noklusēta vērtība.

Darba lapa ws∈WS ir kortežs $<title, type, SE, JOIN, CONDSET\}$, kur *title* ir atskaites nosaukums, $type \in \{$'TABLE','CROSSTAB'$\}$ ir atskaites tips (tabulas vai matricveida atskaite), $SE=\{se_1,...,se_p\}$ ir izvēlēto elementu kopa, $JOIN=\{join_1,...,join_q\}$ ir savienojumu kopa un $CONDSET=\{condset_1,...,condset_o\}$ ir *nosacījumu kopu* kopa.

Izvēlēts elements se∈SE ir kortežs $<loc,calcfun,CPART>$, kur $loc \in \{$'COLUMN','ROW','CELL','PAGE ITEM'$\}$ ir izvēlētā elementa *atrašanas vieta* atskaitē, *calcfun* ir izvēlēta elementa *aprēķināšanas funkcija*, un $CPART \subseteq PARAM \cup C$ ir *kalkulācijas daļu* kopa. Kalkulācijas daļa var atbilst mērījumam vai atribūtam.

Savienojums join∈JOIN ir kortežs $<c_1,c_2,jtype,outer>$, kur $c_1 \in C$ un $c_2 \in C$ ir savienotās kolonnas, $jtype \in \{$'=','<','>','<=','>='$\}$ ir savienojuma tips un $outer \in \{$'LEFT','RIGHT','NO'$\}$ ir parametrs, kas definē, vai savienojums ir kreisais ārējais savienojums (vērtība LEFT), labais ārējais savienojums (vērtība RIGHT) vai nav ārējais savienojums (vērtība NO).

Nosacījumu kopa condset∈CONDSET ir kortežs $<formula,isversion,COND>$, kur *formula* ir vairāku nosacījumu kombinācijas formula, *isversion* ir pazīme, vai nosacījumu kopa definē versijas ierobežojumu, un $COND=\{cond_1,...,cond_b\}$ ir *nosacījumu* kopa.

Nosacījums cond∈COND ir kortežs $<op,$ *ind,* $condobj_1,condobj_2>$, kur $op \in \{$'<', '>', '=', '<=', '>=', '<>', *'IS NULL', 'IS NOT NULL',* 'LIKE', *'IN', 'NOT IN', 'EXISTS', 'NOT EXISTS'*$\}$ ir nosacījuma operators; *ind* ir skaitlis, kas definē nosacījuma 'vietu' nosacījumu kopas formulā (saskaņā ar indeksu nosacījumi tiek sakārtoti); $condobj_1 \in CONDOBJ$ un $condobj_2 \in CONDOBJ$ elementi, kas tiek salīdzināti ar nosacījuma operatoru (CONDOBJ=CONST∪SQ∪PARAM∪SE, kur *CONST* ir konstantu vērtību kopa un *SQ* ir apakšvaicājumu kopa).

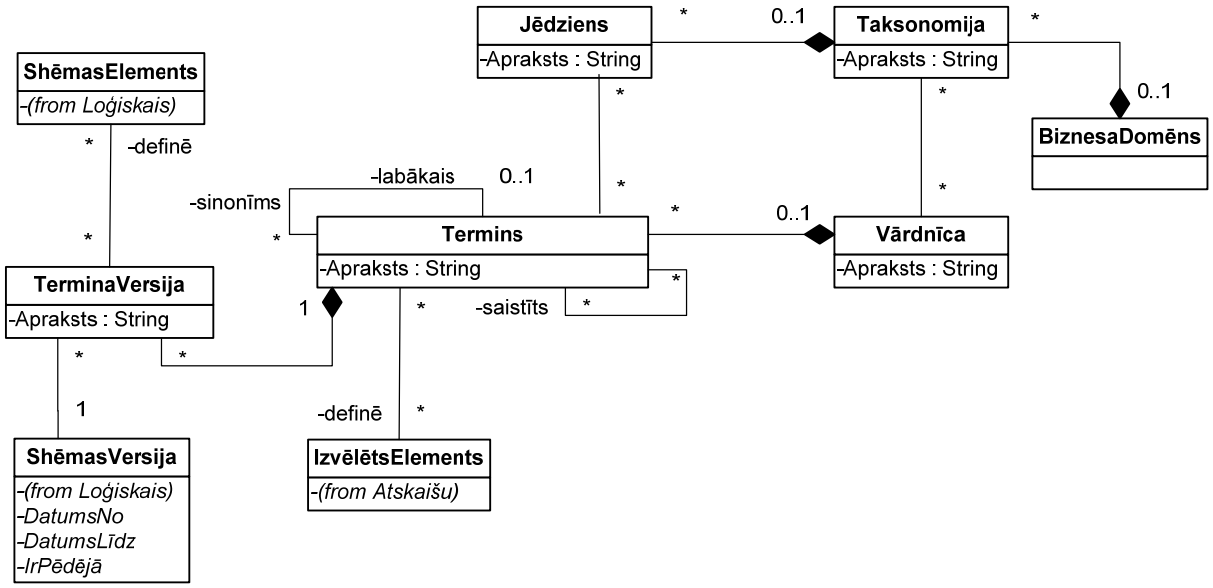
6.2.6 Semantiskais līmenis

Datu noliktavas lietotājiem ir jāsaprot semantika datiem, kas ir attēloti atskaitē no biznesa viedokļa. Viņiem arī ir jāspēj analizēt šos datus, izmantojot visus nepieciešamos līdzekļus, tai skaitā OLAP operācijas "drill-down" un "roll-up", lietojot hierarhijas. Turklāt ir vēlams, lai lietotāji varētu mainīt vai veidot atskaites paši no elementiem, kas viņiem ir pazīstami, lai atskaišu veidošanas process kļūtu vienkāršs. Šajā nolūkā ir nepieciešams aprakstīt katru datu noliktavas modeļa elementu biznesa valodā. Šo aprakstu varētu arī izmantot lietotāji, lai izteiktu savas prasības attiecībā uz informāciju un izmaiņām šajās prasībās, kā arī lai padarītu saprašanos starp lietotājiem un datu noliktavas izstrādātājiem skaidrāku. Datu noliktavas elementu apraksts biznesa valodā tiek atspoguļots semantiskajā līmenī.

CWM satur pakotni Business Nomenclature, ko var izmantot biznesa jēdzienu attēlošanai. Šī pakotne tika paplašināta un tika iegūts semantiskā līmeņa klašu diagramma, kas ir redzama attēlā 25. Galvenās klases, kas ir izmantotas datu noliktavas elementu

aprakstīšanai, ir *Termini* un *Jēdzieni*, kas ir apvienotas, respektīvi, *Vārdnīcās* un *Taksonomijās*. Taksonomijas tiek veidotas konkrētam biznesa domēnam, kas ir nozare vai biznesa joma. Jēdziens ir kāda datu noliktavas elementa vai datu, kas tiek glabāti kādā elementā, semantiskā nozīme vai priekšstats, bet termins ir īpašs vārds vai frāze, ko izmanto lietotāji, kad runā par jēdzienu. Termini definē izvēlētos elementus, kas ir izmantoti atskaitēs. Daži termini varētu būt atzīti par "labākiem", lai aprakstītu jēdzienu. Dažiem terminiem ir sinonīmi. Arī termini var būt saistīti viens ar otru.

CWM modelis tika papildināts ar klasi *TerminaVersija*, kas ir saistīta ar konkrētu shēmas versiju. Termina versija atspoguļo kāda datu noliktavas shēmas elementa nozīmi, kas bija spēkā laika periodā, ko definē atbilstošās shēmas versijas atribūti *DatumsNo* un *DatumsLīdz*. Dažādas terminu versijas var būt izveidotas, kad mainās datu noliktavas darījumprasības. Piemēram, vienam terminam 'Lietošanas laiks' var būt divas versijas 'Lietošanas laika sekundēs' un 'Lietošanas laiks stundās', kas atbilst dažādiem mērījumiem.



Att. 25. Semantiskā līmeņa formālais modelis

Tālāk tiek dota semantiskā līmeņa formāla definīcija.

Biznesa domēns ir kortežs $BD=<TAX,GLOS,TGMAP,CTMAP>$, kur $TAX=\{tax_1,...,tax_c\}$ ir taksonomiju kopa, $GLOS=\{glos_1,...,glos_e\}$ ir vārdnīcu kopa, $TGMAP=\{tgmap_1,..,tgmap_g\}$ ir asociāciju kopa starp taksonomijām un vārdnīcām un $CTMAP=\{ctmap_1,...,ctmap_h\}$ ir asociāciju kopa starp jēdzieniem un terminiem.

Taksonomija $tax \in TAX$ sastāv no *jēdzienu* kopas: $tax=<CONC>$; $CONC=\{conc_1,...,conc_d\}$.

Vārdnīca $glos \in GLOS$ ir kortežs $<TERM,PTMAP,CONTMAP,SEMAP>$, kur $TERM=\{term_1,...,term_f\}$ ir *terminu* kopa, $PTMAP=\{ptmap_1,...,ptmap_i\}$ ir labāko terminu un sinonīmu terminu asociāciju kopa, $CONTMAP=\{contmap_1,...,contmap_j\}$ ir saistītu terminu asociāciju kopa un $SEMAP=\{semap_1,...,semap_k\}$ ir izvēlētu elementu definīciju kopa.

Termins $term \in TERM$ ir kortežs $<termdef,TV>$, kur *termdef* ir termina definīcija (vārds vai frāze, kas apraksta terminu) un $TV=\{tv_1,...,tv_l\}$ ir termina versiju kopa.

Termina versija tv∈TV ir kortežs <*tvdef,E,v*>, kur *tvdef* ir termina versijas definīcija (vārds vai frāze, kas apraksta termina versiju), $E \subseteq FDA \cup ATT \cup LEV \cup H \cup AL \cup HL \cup MS$ ir datu noliktavas shēmas elementu kopa, kuru nozīmi definē termina versija, un $v \in V$ ir datu noliktavas shēmas versija, kurā termina versija bija spēkā. Shēmas elementi un shēmas versijas ir loģiskā līmeņa formālā modeļa elementi.

Asociācija starp labāko terminu un sinonīmu terminu ptmap∈PTMAP ir kortežs <*prefterm,synterm*>, kur $prefterm \in TERM$ ir termins, kas ir atzīts par „labāku" un $synterm \in TERM$ ir „labākā" termina sinonīms.

Saistītu terminu asociācija contmap∈CONTMAP ir kortežs <$term_1$, $term_2$>, kas savieno saistītus terminus $term_1 \in TERM$ un $term_2 \in TERM$.

Izvēlēta elementa definīcija semap∈SEMAP ir kortežs <*term,se*>, kas nosaka, ka atskaites izvēlēts elements $se \in SE$ atbilst terminam $term \in TERM$. Izvēlēts elements ir atskaišu formālā modeļa elements.

Asociācija starp taksonomiju un vārdnīcu tgmap∈TGMAP ir kortežs <*tax,glos*>, kas savieno taksonomiju $tax \in TAX$ un vārdnīcu $glos \in GLOS$, kas ir saistītas.

Asociācija starp jēdzieniem un terminiem ctmap∈CTMAP ir kortežs <conc,term>, kas nosaka, ka termins $term \in TERM$ ir izmantots, lai aprakstītu jēdzienu $conc \in CONC$.

6.3. Datu noliktavas shēmas metadati

6.3.1. Datu noliktavas shēmas metadatu apraksts

Sekojošajās apakšnodaļās tiek dots datu noliktavas metadatu repozitorija shēmas metadatu un atskaišu repozitorija atskaišu metadatu apraksts un fiziskais modelis, atbilstoši formālajam modelim. Datu noliktavas metadatu repozitorija shēmas metadatos ir iespējams modelēt datu noliktavas shēmu fiziskajā, loģiskajā un semantiskajā līmenī. Bet atskaišu metadati satur atskaišu specifikāciju. Sekojošie metadati tika veidoti, lai tajos varētu aprakstīt datu noliktavas shēmu un atskaites atbilstoši formālā modeļa definīcijām.

6.3.2. Loģiskā līmeņa metadati

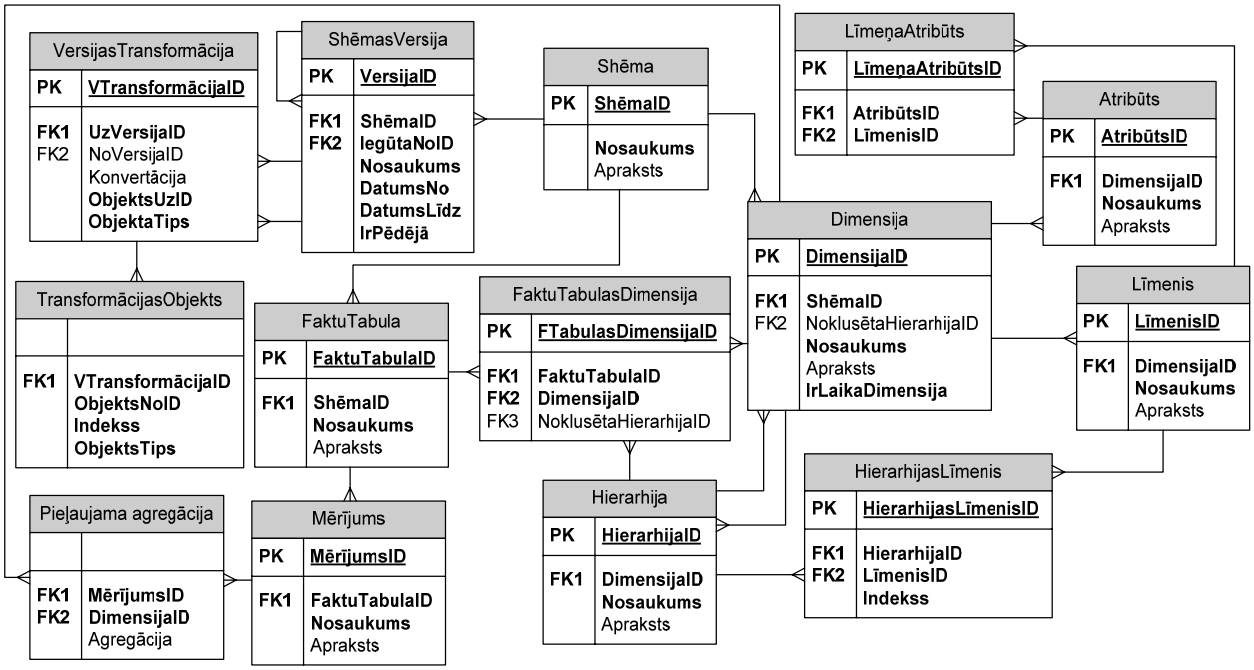
Loģiskā līmeņa shēmas metadati tiek izmantoti, lai aprakstītu datu noliktavas shēmas versijas atbilstoši loģiskā līmeņa formālajam modelim, kas tika dots apakšnodaļā 6.2.1. Fiziski loģiskā līmeņa metadati ir realizēti ar relāciju datu bāzes tabulām. Loģiskā līmeņa metadatu fiziskais modelis ir redzams attēlā 26.

Katrai datu noliktavas datuvei tiek veidots ieraksts tabulā *Shēma*. Šai tabulai tiek piesaistītas visas datuves dimensijas (tabula *Dimensija*) un faktu tabulas (tabula *FaktuTabula*), kas ir savā starpā saistītas ar tabulas *FaktuTabulasDimensija* palīdzību. Faktu tabulu mērījumu metadati tiek glabāti tabulā *Mērījums*. Katram mērījumam tiek glabātas pieļaujamas agregācijas attiecībā pret dažādām dimensijām tabulā *Pieļaujama agregācija*. Faktu tabulām var piesaistīt noklusētas hierarhijas kolonnā *NoklusētaHierarhijaID*.

Dimensijas glabājas tabulā *Dimensija*. Laika dimensijas ir īpaši atzīmētas ar lauku *IrLaikaDimensija*. Dimensiju hierarhiju metadati tiek glabāti tabulā *Hierarhija*. Noklusētā dimensijas hierarhija ir piesaistīta dimensijai caur kolonnu *NoklusētaHierarhijaID*. Dimensiju līmeņi tiek glabāti tabulā *Līmenis*. Dimensijas atribūtu metadati glabājas tabulā *Atribūts*. Saistība starp līmeni un atribūtu tiek glabāta tabulā *LīmeņaAtribūts*. Lai piesaistītu noteiktu līmeni hierarhijai tiek veidots ieraksts tabulā *HierarhijasLīmenis*, kas satur indeksu, kas norāda līmeņa 'vietu' visu līmeņu kokā. Šādā veidā līmeņi ir sakārtoti.

Visas shēmas versijas tiek glabātas tabulā *ShēmasVersija*. Katrai versijai ir periods (kolonnas *DatumsNo* un *DatumsLīdz*), kad tā bija spēkā. Iepriekšējas versijas identifikators tiek glabāts kolonnā *IegūtaNoID*. Pirmajai versijai šajā kolonnā ir savs identifikators. Ja notiek kāda izmaiņa datu noliktavas shēmā, kuras rezultātā tiek izveidota jaunā versija, tad tabulā *ShēmasVersija* parādās jauns ieraksts.

Metadati par versiju transformācijām tiek glabāti tabulā *VersijasTransformācija*. Šajā tabulā lauks *ObjektsUzID* satur elementa identifikatoru, kas vai nu eksistē dotajā versijā vai var būt izrēķināts no citiem shēmas versijā eksistējošiem elementiem. Viens shēmas elements var būt saistīts ar vairākām shēmas versijām caur tabulas *VersijasTransformācija* vairākiem ierakstiem. Tabulas *VersijasTransformācija* kolonnā *ObjektaTips* glabājas shēmas elementa tips, piemēram, mērījums, atribūts u.c. Pārējo kolonnu dati atšķiras shēmas elementiem, kas eksistē versijā, un shēmas elementiem, kas ir izrēķināti no eksistējošiem elementiem.



Att. 26. Loģiskā līmeņa metadati

Ja shēmas elements eksistē shēmas versijā, tad tam tiek veidots ieraksts tabulā *VersijasTransformācija*, kur kolonnas *NoVersijaID* un *Konvertācija* nav aizpildītas, kolonnā *UzVersijaID* ir saglabāts shēmas versijas, kas satur shēmas elementu, identifikators. Eksistējošiem elementiem netiek veidoti nekādi ieraksti tabulā *TransformācijasObjekts*.

Ja shēmas elements neeksistē dotajā versijā V1, bet var tikt izrēķināts no citiem elementiem, kas eksistē dotajā versijā V1, un shēmas elements eksistēja citā shēmas versijā

V2, tad šim elementam tiek veidots ieraksts tabulā *VersijasTransformācija*, kur kolonnā *UzVersijaID* ir saglabāts dotās versijas V1 identifikators, kolonnā *Konvertācija* ir funkcija, kas iegūst neeksistējošo shēmas elementu no citiem shēmas versijā V1 eksistējošiem elementiem. Laukā *NoVersijaID* ir identifikators shēmas versijai, kurā eksistē mainītais shēmas elements, t.i. versijai V2. Tabulā *TransformācijasObjekts* ir dati par citiem shēmas versijā V1 eksistējošiem elementiem, kas ir izmantoti mainītā elementa izrēķināšanai. Šajā tabulā laukā *ObjektsNoID* glabā šo aprēķināšanā izmantoto elementu identifikatorus, laukā *ObjektaTips* glabā elementu tipus (piemēram, atribūts, mērījums, u.c.), un laukā Indekss glabā elementa kārtas numuru konvertācijas funkcijā.

Gadījumam, kad kāds jauns elements parādījās kādā versijā, bet šī elementa vērtības var manuāli specificēt esošiem datiem (piemēram, kad dimensijā parādās jauns atribūts, kura vērtības var aizpildīt eksistējošiem dimensiju tabulas ierakstiem), tad var rasties situācija, kad elements neeksistēja iepriekšējā versijā un tas nevar tikt izrēķināts no citiem elementiem, bet tā vērtības jau ir ievadītas un var tikt izmantotas atskaitēs. Lai atbalstītu arī šādus gadījumus, metadati par šādiem elementiem tiek glabāti līdzīgi mainītiem elementiem, ko var izrēķināt no citiem elementiem, tikai tabulā *TransformācijasObjekts* tiek izveidots ieraksts, kam laukā *ObjektsNoID* tiek glabāts identifikators pašam jaunajam shēmas elementam.

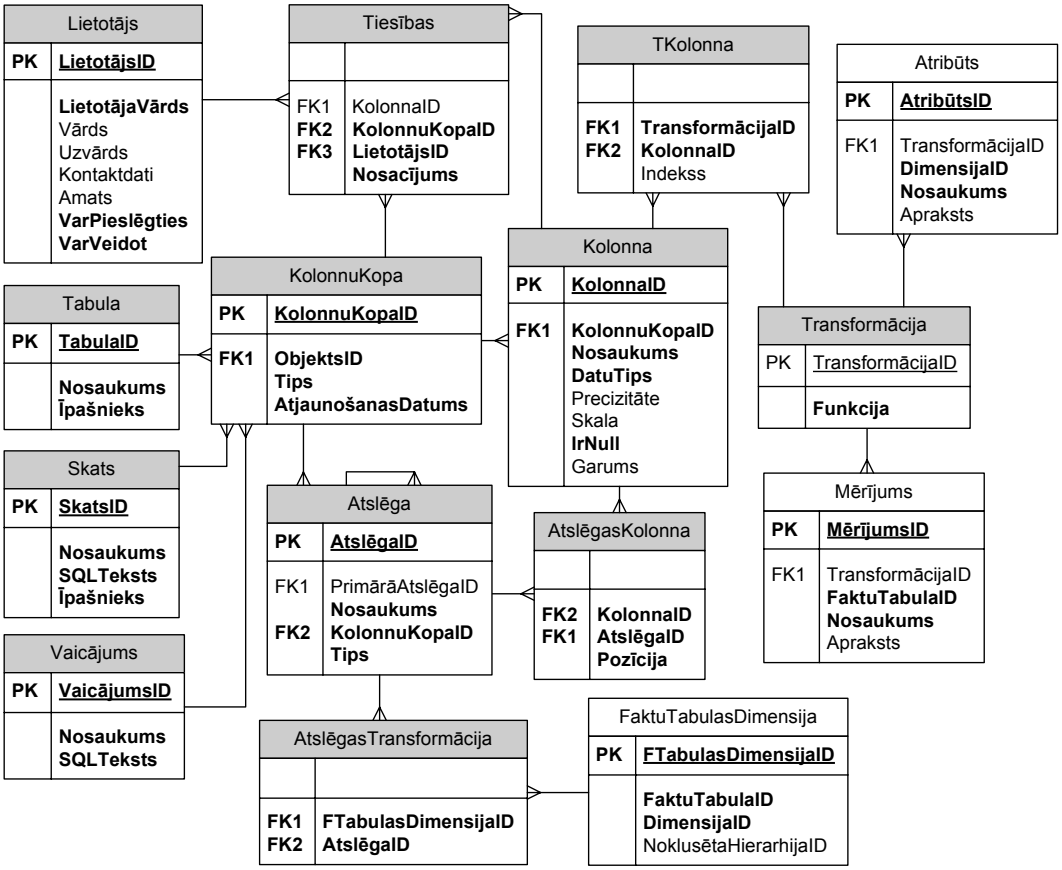
Versijas transformācijas veido datu noliktavas administrators, kad notiek izmaiņas datu noliktavas shēmā un tiek veidota jaunā shēmas versija.

6.3.3. Fiziskā līmeņa metadati un lietotāju tiesības

Fiziskā līmeņa metadati apraksta datu noliktavas relāciju shēmu un daudzdimensiju shēmas attēlojumu relāciju datu bāzes datu struktūrās. Šie metadati tika veidoti saskaņā ar fiziskā līmeņa formālo modeli, kas dots apakšnodaļās 6.2.2.-6.2.4. Fiziskā līmeņa metadati ir realizēti fiziski ar relāciju datu bāzes tabulām. Fiziskā līmeņa metadatu fiziskais modelis ir redzams attēlā 27.

Metadati par tabulām, skatiem un vaicājumiem tiek glabāti tabulās *Tabula, Skats* un *Vaicājums*. Šīs tabulas ir savienotas ar tabulu *KolonnuKopa*, kas glabā datu atjaunošanas datumu un tipu, kas var būt tabula, skats vai vaicājums. Metadati par tabulu, skatu un vaicājumu kolonnām tiek glabāti tabulā *Kolonna*.

Metadati par kolonnu kopas ārējām un primārām atslēgām tiek glabāti tabulā *Atslēga*. Tabulas kolonna *Tips* nosaka, vai atslēga ir primārā vai ārējā. Šajā tabulā ārējām atslēgām tiek glabāts identifikators attiecīgas tabulas primārajai atslēgai kolonnā *PrimārāAtslēgaID*. Atslēgu kolonnas ir savienotas ar primāro vai ārējo atslēgu caur tabulu *AtslēgasKolonna*.



Att. 27. Fiziskā līmeņa metadati

Atslēgas tiek piesaistītas *FaktuTabulasDimensijai* loģiskajā līmenī caur *AtslēgasTransformāciju*. Tas tiek izmantots gadījumos, kad viena fiziskā tabula, kas atbilst vairākām dimensijām, tiek saistīta ar faktu tabulu ar vairākām atslēgām, lai norādītu, kādu atslēgu izmantot ar kādu dimensiju, būvējot SQL vaicājumu. Piemēram, tabula, kas glabā datumus un laikus, var atbilst dimensijām *Laiks_No*, *Laiks_Līdz* vai *Laiks_Uz*, kas ir savienotas ar faktu tabulu ar trijām dažādām atslēgām. Būvējot vaicājumu, šo laika tabulu vajag savienot ar faktu tabulu kā trīs dažādas tabulas, izmantojot tabulu aizstājvārdus (alias).

Metadati par transformācijām, kas definē kādā veidā ir iegūti atribūtu un mērījumu dati, tiek glabāti tabulā *Transformācija*, kas satur kolonnu *Funkcija*, kas definē atribūtu un mērījumu aprēķināšanas formulas.

Pagaidām datu noliktavas shēmas versiju pārvaldei tiek izmantotas tikai atribūti un mērījumi, kas pa tiešo ir iegūti no tabulu kolonnām, t.i. vienam mērījumam vai atribūtam atbilst viena tabulas, skata vai vaicājuma kolonna datubāzē. Nākotnē ir plānots atbalstīt versijas atribūtiem un mērījumiem, kas ir iegūti no tabulu, skatu vai vaicājumu kolonnām ar funkciju.

Tabulas *Lietotājs* un *Tiesības* ir izmantotas lietotāju tiesību metadatu glabāšanai. Tabulā *Lietotājs* glabājas visu lietotāju dati. Tabulā *Tiesības* ir informācija par noteikta lietotāja tiesībām uz datu noliktavas tabulām, skatiem vai SQL vaicājumu rezultātiem (lauks *KolonnuKopaID*), kolonnām (lauks *KolonnaID*) un datiem. Šajā tabulā kolonnā *Nosacījums* tiek ierakstīts nosacījums, kam ir jāizpildās, lai lietotājs varētu apskatīt datus tabulā vai kolonnā. Nosacījums ir izteikts kā SQL vaicājuma WHERE daļa. Ja lietotājam ir tiesības uz

visām kolonnām tabulā, skatā vai vaicājumā, tad tabulā *Tiesības* tiek veidots viens ieraksts, kur lauks *KolonnaID* ir tukšs.

6.3.4. Atskaišu metadati un lietotāju tiesības uz atskaitēm

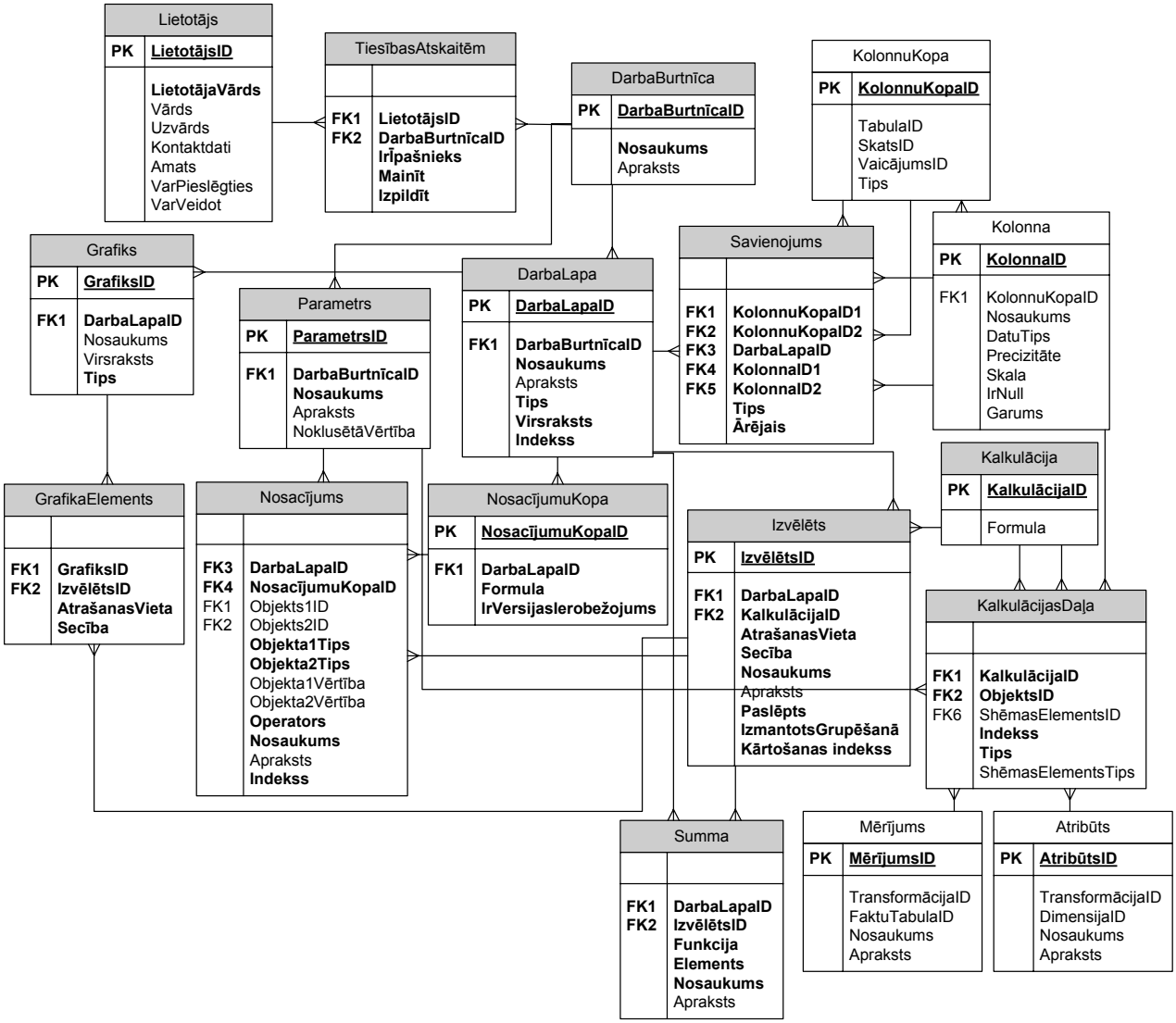
Atskaišu metadati (attēls 28.) apraksta izveidoto atskaišu struktūru. Atskaišu metadatos ir iespējams aprakstīt definētas atskaites atbilstoši formālajam atskaišu modelim, kas tika aprakstīts apakšnodaļā 6.2.5.

Metadati par darba burtnīcām un lapām, kas satur atskaites, tiek glabāti tabulās *DarbaBurtnīca* un *DarbaLapa*. Darba lapas veids (tabulas vai matricveida) tiek glabāts kolonnā *Tips*. Kolonnā *Indekss* tiek saglabāta darba lapas vieta darba burtnīcā attiecībā pret pārējām lapām.

Katra atskaite satur dažādus izvēlētus elementus, kuru informācija tiek glabāta tabulā *Izvēlēts*. Katram izvēlētam elementam kolonnā *AtrašanasVieta* tiek glabāta tā atrašanas vieta atskaitē, piemēram, rinda, kolonna, šūna, lappušu objekts. Kolonnā *Secība* tiek glabāts izvēlētā elementa kārtas numurs citu vienas atskaites izvēlētu elementu kopā, piemēram, rindas numurs no augšas, kolonnas numurs no kreisās puses. Kolonnā *Paslēpts* ir pazīme, vai izvēlēts elements tiek attēlots atskaitē vai vienkārši ir izmantots, bet lietotājam netiek parādīts. Kolonna *IzmantotsGrupēšanā* norāda, vai izvēlēts elements ir izmantots skaitlisko datu grupēšanā, ja tas ir paslēpts. Tas ir nepieciešams, lai izvēlētu elementu varētu izmantot nosacījumos, neizmantojot grupēšanā.

Izvēlēts elements tiek definēts ar kalkulāciju, kas glabājas tabulā *Kalkulācija*. Šeit laukā *Formula* glabājas formula, ar ko aprēķina izvēlēto elementu. Formulā var izmantot jebkuras datubāzē definētas funkcijas un operācijas, tai skaitā summas un citas kopsavilkuma funkcijas, kā arī lietotāju definētas funkcijas. Ja izvēlētais elements atbilst vienkārši tabulas kolonnai vai parametram, tad formula ir tukša. Formulas daļas tiek glabātas tabulā *KalkulācijasDaļa*. Šajā tabulā kolonna *Tips* norāda kalkulācijas daļas tipu, kas var būt kolonna vai parametrs. Pašas kolonnas vai parametra identifikators ir kolonnā *ObjektsID*. Kolonnā *Indekss* norāda kalkulācijas daļas vietu formulā. Ja kalkulācijas daļa atbilst kādam datu noliktavas shēmas elementam (atribūtam vai mērījumam no loģiskā līmeņa metadatiem), tad šis shēmas elements tiek piesaistīts caur kolonnu *ShēmasElementsID* un tā tips (Atribūts vai Mērījums) ir saglabāts kolonnā *ShēmasElementaTips*.

Atskaitē izmantotas kolonnu kopas ir piesaistītas caur tabulu Savienojums, kas papildus vēl norāda kādā veidā kolonnu kopas ir saistītas atskaitē, ar kādām kolonnām. Pēc noklusēšanas tabulas ir saistītas caur primārās un ārējās atslēgas attiecībām. Kolonnā *Tips* norāda salīdzināšanas operatoru, kas ir starp kolonnām: =, >=, <=, >, <, !=. Kolonna *Outer* norāda vai kolonnas ir saistītas ar ārējo savienojumu (vērtības LEFT, RIGHT, NO).



Att. 28. Atskaišu metadati

Tabulā *Nosacījums* tiek glabāti atskaites nosacījumu metadati. Nosacījuma operators tiek saglabāts kolonnā *Operators*. Kolonnās *Objekta1ID* un *Objekta2ID* tiek glabāti salīdzināto nosacījuma objektu identifikatori, ja nosacījuma objekti ir parametri vai izvēlēti elementi. Laukos *Objekta1Tips* un *Objekta2Tips* tiek saglabāts nosacījuma objektu tipi, kas ir parametrs, izvēlēts elements, konstante vai SQL vaicājums. Ja nosacījuma objekts ir konstanta vērtība vai SQL vaicājums, tad laukos *Objekta1Vērtība* un *Objekta2Vērtība* glabā konstantes vērtību vai SQL vaicājuma tekstu. Nosacījumiem eksistē sekojošie ierobežojumi. Nav atļauts veidot vienā atskaitē nosacījumus uz agregātfunkcijām un izvēlētiem elementiem, kas ir paslēpti un netiek izmantoti grupēšanai, jo šāda nosacījumu kombinācija nevar tikt pārveidota par SQL vaicājumu.

Nosacījumi ir savienoti ar darba lapu caur tabulu *NosacījumuKopa*, kur laukā *Formula* norāda nosacījumu formulu. Tabulā *Nosacījums* kolonnā *Indekss* ieraksta nosacījuma vietu kopējā visu nosacījumu formulā.

Tabulās *NosacījumuKopa* un *Nosacījums* arī tiek glabāta informācija par versijas laika ierobežojumu (*IrVersijasIerobežojums*=1). Šāda veida nosacījumi netiek izmantoti atskaitei, lai atfiltrētu atskaites datus, bet tiek lietoti, lai noskaidrotu, kādas versijas bija spēkā

norādītajā laika periodā. Versijas laika ierobežojumos tiek lietots atslēgas vārds *Laiks* kā pirmais nosacījuma objekts. Kā otrais nosacījuma objekts tiek lietota konstanta vērtība, kas apzīmē laiku. Piemēram, nosacījums *Laiks<='01.01.2010'* nozīmē, ka atskaitei jāizmanto datu noliktavas shēmas versijas, kas bija spēkā no 2010. gada sākuma.

Metadati par parametriem, kas ir izmantoti atskaitē glabājas tabulā *Parametrs*. Par parametru tiek glabāts tā nosaukums, apraksts un noklusētā vērtība.

Lai varētu atbalstīt papildus funkcionalitāti atskaišu rīkā (piemēram, lai nodrošinātu datu analīzi grafikos un kopsavilkuma datus), atskaišu metadati satur vairāk informācijas, nekā formālais modelis. Tālāk tiek aprakstīti papildus metadati, kas tika izveidoti.

Skaitliskus atskaites izvēlētos elementus var summēt attiecībā pret kolonnām un rindām (matricveida atskaitēs). Lai to nodrošinātu, katrai summai tiek veidots ieraksts tabulā *Summa*, kam piesaista summētu izvēlētu elementu laukā *IzvēlētsID*, izvēlas kādu funkciju pielietot (Summa, vidēja vērtība, ierakstu skaits, maksimālā vērtība, minimālā vērtība). Kolonnā *Elements* izvēlas, attiecībā pret kādu atskaites daļu rēķināt summu. Ir iespējamie varianti: kolonna vai rinda. Arī summai definē nosaukumu un aprakstu, kas parādīsies atskaitē.

Dažus atskaites izvēlētos elementus var attēlot grafikos. Katram atskaites grafikam tiek izveidots ieraksts tabulā *Grafiks* ar atbilstošu nosaukumu, virsrakstu un noteikto tipu. Grafiku tipi ir līdzīgi Microsoft Excel grafikiem, piemēram, kolonnas (column), joslas (bar), pīrāgs (pie), līnija (line), laukums (area) u.c. Atskaites elementi ir piesaistīti grafikam caur tabulu *GrafikaElements*, kur laukos *AtrašanasVieta* un *Secība* norāda izvēlēta elementa vietu grafikā.

Atskaišu metadati arī satur tabulas *Lietotājs* (kopīgs ar fiziskā līmeņa metadatiem) un *TiesībasAtskaitēm*, kas ir nepieciešami lietotāju tiesību pārvaldībai. Lietotājam var būt dažāda veida tiesības (īpašnieks, mainīt, izpildīt) uz darba burtnīcām. Kad lietotājs izpilda atskaiti, sākumā tiek pārbaudītas lietotāja tiesības uz darba burtnīcu (un darba lapu) un tad tiesības uz atskaites izvēlētu elementu kalkulācijās izmantotām kolonnām un kolonnu kopām, kas šīs kolonnas satur.

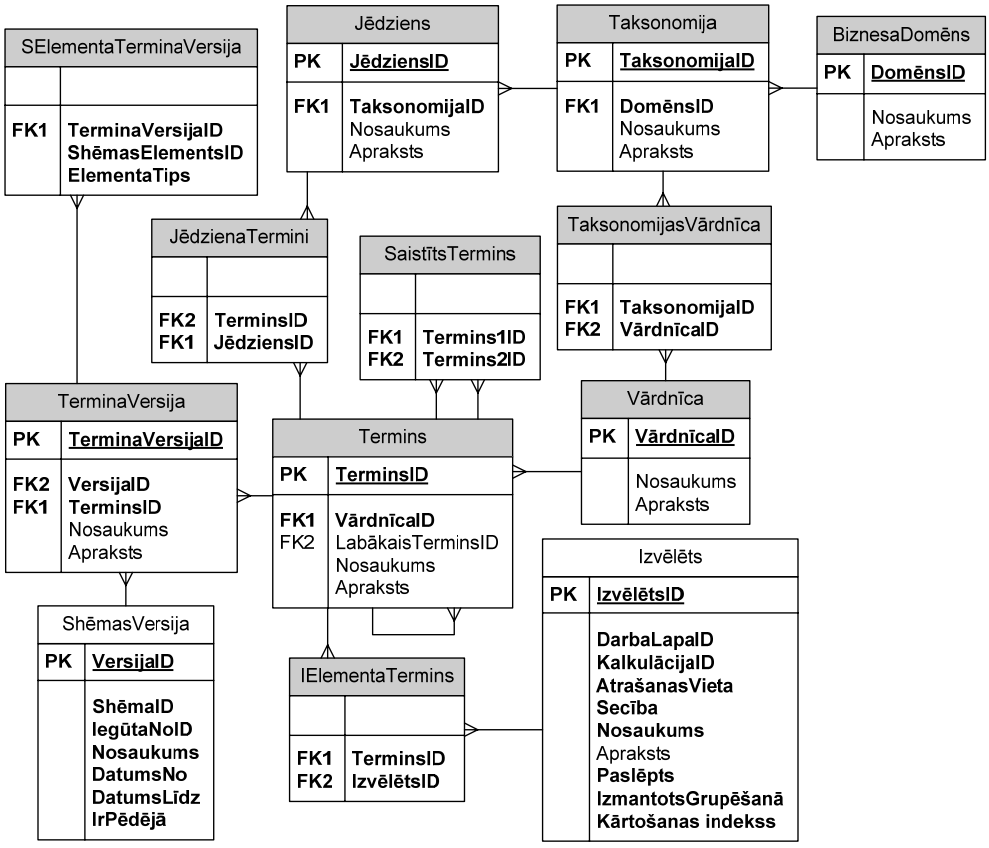
Lietotājam var būt tiesības veidot jaunu darba burtnīcu, kas tiek piereģistrēts tabulā *Lietotājs* kolonnā *VarVeidot*. Kad lietotājs izveido darba burtnīcu, viņam/viņai tiek iedotas īpašnieka tiesības (tabulā *TiesībasAtskaitēm* kolonnā *IrĪpašnieks*), kas ļauj lietotājam izpildīt, mainīt un dzēst darba burtnīcu un tās darba lapas. Ja lietotājam ir tiesības mainīt darba burtnīcu (tabulā *TiesībasAtskaitēm* kolonnā *Mainīt*), tas nozīmē, ka lietotājs drīkst izpildīt darba burtnīcu un veikt jebkuras izmaiņas, bet nedrīkst dzēst darba burtnīcu vai kādu no tās darba lapām. Ja lietotājam ir tikai tiesības izpildīt darba burtnīcu (tabulā *TiesībasAtskaitēm* kolonnā *Izpildīt*), tad viņš/viņa nevar neko darba burtnīcā mainīt vai dzēst, bet var saglabāt to pašu darba burtnīcu kā jaunu un tajā veikt izmaiņas, ja lietotājam ir tiesības veidot jaunu darba burtnīcu (tabulā *Lietotājs* kolonnā *VarVeidot*).

6.3.5. Semantiskie metadati

Datu noliktavas elementu apraksts biznesa valodā tiek glabāts semantiskajos metadatos. Metadati par biznesa domēnu tiek glabāti tabulā *BiznesaDomēns*. Katram biznesa domēnam atbilstošas taksonomijas tiek glabātas tabulā *Taksonomija*. Ar taksonomiju saistītās

vārdnīcas (tabula Vārdnīca) tiek savienotas ar taksonomijām caur tabulu TaksonomijasVārdnīca. Jēdzienu un Terminu definīcijas tiek glabātas tabulās Jēdziens un Termins.

Terminiem glabā 'labākā' termina identifikatoru kolonnā *LabākaisTerminsID*. Arī saistītie termini ir savstarpēji savienoti tabulā *SaistītsTermins*. Atskaišu izvēlētos elementus definējošie termini ir piesaistīti tabulai *Izvēlēts* no atskaišu metadatiem caur tabulu *IElementaTermins*. Termina versiju metadati tiek glabāti tabulā *TerminaVersija*, kas satur atbilstošas shēmas versijas identifikatoru no tabulas *ShēmasVersija* no loģiskā līmeņa metadatiem. Loģiskā līmeņa shēmas elementu terminu versijas ir piesaistītas konkrētiem shēmas elementiem caur tabulu *SElementaTerminaVersija*, kur kolonnā *ElementaTips* tiek glabāts atbilstošā shēmas elementa veids, piemēram, mērījums, atribūts, hierarhija, līmenis u.c.



Att. 29. Semantiskie metadati

6.4. Nodaļas secinājumi

Galvenais šīs nodaļas rezultāts ir formālais modelis, kas formāli apraksta datu noliktavas shēmu loģiskajā, fiziskajā un semantiskajā līmenī, kā arī atskaites uz datu noliktavu. Piedāvātā formālā modeļa izstrādei par pamatu tika izmantots CWM metamodelis, kuru pakotnes tika paplašinātas, lai varētu definēt datu noliktavas shēmas versijas, terminu versijas un atskaites. Loģiskā līmeņa formālais modelis tika papildināts ar jaunām klasēm *ShēmasVersija* un *VersijasTransformācija*, kas papildus datu noliktavas shēmas versijas informācijai apraksta arī kādā veidā datu noliktavas shēmas elementi varētu būt pārveidoti no

vienas shēmas versijas uz citu. Loģiskā līmeņa formālais modelis tika papildināts arī ar informāciju par pieļaujamām agregācijas funkcijām, ko var pielietot mērījumiem attiecībā pret dažādām dimensijām. Šī informācija ir nepieciešama, lai korekti analizētu datus atskaitē un neiegūtu datu semantikai neatbilstošus rezultātus. Fiziskā līmeņa formālais modelis tika papildināts ar lietotāju tiesībām uz datu noliktavas datiem, lai kontrolētu informācijas drošību. Atskaišu formālais modelis ir oriģināls, tā izstrādē netika izmantotas eksistējošas CWM pakotņu klases. Atskaišu formālajā modelī var aprakstīt lielu atskaišu loku, gan tabulas veida, gan matricveida atskaites, var izmantot nosacījumus, kopsavilkumus atskaitēs, parametrizēt atskaites, veidot lietotāja definētus savienojumus starp tabulām. Semantiskā līmeņa formālais modelis tika papildināts ar klasi TerminaVersija, kas atspoguļo datu noliktava shēmas elementu nozīmi konkrētajā laika periodā, kas atbilst datu noliktavas shēmas versijas derīguma periodam.

Papildus katra līmeņa klašu diagrammai, kā arī atskaišu formālajam modelim, tika dota arī datu noliktavas formālā definīcija, kas tālāk promocijas darbā 7. nodaļā tiks izmantota, lai formāli aprakstītu datu noliktavas shēmas izmaiņu realizācijas procesu.

Šajā nodaļā arī aprakstīti datu noliktavas shēmas metadati un atskaišu metadati, kas ir izmantoti datu noliktavas evolūcijas arhitektūrā datu noliktavas un atskaišu metadatu repozitorijos. Metadati tika būvēti atbilstoši aprakstītajam formālajam modelim. Šajā nodaļā doti arī realizācijas iespējamie modeļi metadatu glabāšanai.

Daudzversiju datu noliktavas formālais modelis un daudzversiju datu noliktavas metadatu repozitorija modelis tika aprakstīti un publicēts rakstos:

Solodovņikova D. 'Metadata to Support Data Warehouse Evolution'. *Proceedings of the 17th International Conference on Information Systems Development by Springer, Paphos, Cyprus,* 2008.

Solodovņikova D. 'The Formal Model for Multiversion Data Warehouse Evolution'. *Postconference proceedings of the 8th International Baltic Conference on Databases and Information Systems, Tallinn, Estonia, Frontiers in Artificial Intelligence and Applications by IOS Press*, 2008.

Solodovnikova D., Niedrite L. 'Evolution-Oriented User-Centric Data Warehouse'. *Proceedings of the 19th International Conference on Information Systems Development by Springer, Prague, Czech Republic, 2010 (tiks publicēts).*

7. DATU NOLIKTAVAS SHĒMAS IZMAIŅAS

7.1. Nodaļas nolūks

Šajā nodaļā tiek aprakstītas piedāvātajā pieejā atbalstītas izmaiņas, kas var notikt datu noliktavas shēmā evolūcijas rezultātā. Izmaiņu saraksts tika iegūts no literatūras avotos sastopamām izmaiņām un no izmaiņām, kas notika e-studiju datuvē (nodaļa 4.). Šīs izmaiņas var tikt ierosinātas gan ar mainīgām datu noliktavas darījumprasībām, gan ar izmaiņām datu avotos. Šādas izmaiņas ietekmē datu noliktavas shēmu, ETL procesus un atskaites, turklāt šo izmaiņu rezultātā tiek izveidota jaunā datu noliktavas shēmas versija. Tāpēc šīs izmaiņas ir jāapstrādā. Katrai izmaiņai ir jāizpilda noteikta procedūra, lai pielāgotu esošo datu noliktavu.

Katras izmaiņas apraksts sastāv no formālas procedūras, kas atbilstoši pielāgo datu noliktavas formālā modeļa instanci (kas tika aprakstīta apakšnodaļā 6.2.), un no procesa apraksta, kas ir jāveic, lai izplatītu izmaiņas piedāvātajos datu noliktavas shēmas metadatos (kas tika aprakstīti apakšnodaļā 6.3.) un fiziskajā datu noliktavas shēmā relāciju datu bāzē.

7.2. Izmaiņu vispārīgs apraksts

Izmaiņas tiek sadalītas fiziskajās, loģiskajās un semantiskajās izmaiņās pēc to darbības veida. Fiziskās izmaiņas (piemēram, jauna dimensijas atribūta pievienošana) operē ar datu bāzes objektiem un datu noliktavas fiziskā līmeņa modeļa instanci, loģiskās izmaiņas (piemēram, dimensijas piesaistīšana faktu tabulai) maina pārsvarā tikai datu noliktavas loģiskā līmeņa modeļa instanci, bet semantiskās izmaiņas (piemēram, dimensijas atribūta nozīmes maiņa) var mainīt gan datu noliktavas semantiskā līmeņa modeļa instanci, gan arī loģiskā līmeņa modeļa instanci. Loģisko izmaiņu rezultātā netiek mainīta datu noliktavas fiziskā līmeņa modeļa instance, vienīgi var parādīties jaunas atslēgas vai atslēgu kolonnas vai eksistējošas atslēgas vai atslēgu kolonnas var tikt izdzēstas. Fizisko izmaiņu rezultātā tiek pielāgota gan loģiskā, gan fiziskā līmeņa datu noliktavas modeļa instance.

Formāli katras izmaiņas rezultātā jaunā datu noliktavas shēmas versija *v'* tiek konstruēta no iepriekšējās versijas v_u. Katra izmaiņas procedūra tiek aprakstīta kā soļu secība, kas maina datu noliktavas shēmu *DWS=<F,D,FDA,V,MAP>*, datu bāzes shēmu *DBS=<T,VW,Q,CS>* un biznesa domēnu *BD=<TAX,GLOS,TGMAP,CTMAP>*. Izmaiņas procedūras rezultāts ir jaunā datu noliktavas shēma *DWS'=<F',D',FDA',V',MAP'>* loģiskajā līmenī, jaunā datu bāzes shēma *DBS'=<T',VW',Q',CS'>* fiziskajā līmenī un jaunais biznesa domēns *BD'=<TAX',GLOS',TGMAP',CTMAP'>* semantiskajā līmenī. Ja fiziskajā vai semantiskajā līmenī shēma netiek mainīta, tad tās apraksts tiek izlaists.

Ja izmaiņas rezultātā tiek izmests shēmas elements, tad iespēju veidot versijas transformāciju, kas izrēķina trūkstošu shēmas elementu no citiem shēmas versijas shēmas elementiem, nosaka datu noliktavas administrators.

Kad tiek izveidota jaunā datu noliktavas shēmas versija V_J no iepriekšējas versijas V_V, tad attiecīgi arī jāmaina metadati. Jaunās shēmas versijas izveidošanas laikā loģiskā līmeņa metadatos tabulā *ShēmasVersija* tiek veidots jauns ieraksts ar jaunās versijas metadatiem, vecajai versijai V_V tiek aizpildīts *DatumsLīdz* ar izmaiņas laiku, bet jaunajai versijai V_J – *DatumsNo*.

7.2.1. Kopējās funkcijas

Formālajā izmaiņu procedūru aprakstā tiek izmantotas sekojošas kopējās funkcijas:

Funkcija duplicate

v'=duplicate(v) ir funkcija, kas veido versiju *v'*, kas satur tos pašus elementus, kas ir iekļauti versijā *v*. Šī funkcija tiek lietota, lai izveidotu jaunu datu noliktavas shēmas versiju no iepriekšējās versijas. Tas tiek darīts sekojoši.

- Ja $v=<\tau_{from}, \tau_{to}, VT>$, tad $v'=<\tau_{now}, null, VT'>$, kur τ_{now} ir pašreizējais laiks un *VT'* ir versiju transformāciju kopa, kur katrai versijas transformācijai $vt=<el,conv,E> \in VT$ tiek uzbūvēta atbilstošā versijas transformācija $vt'=<el,conv,E> \in VT'$, tas nozīmē, ka jaunajā versijā *v'* tiek izveidotas tādas pašas versijas transformācijas, kas eksistē vecajā versijā *v*.
- Ja τ_{to} ir *null* (t.i. versija bija spēkā līdz jaunas versijas izveidi), tad versija *v* tiek atjaunināta $v=<\tau_{from}, \tau_{now}-1, VT>$ (t.i. vecajai versijai tiek ierakstīts derīguma beigu termiņš).

Funkcija vtrans

VT=vtrans(elem) ir rekursīva funkcija, kas atgriež versiju transformāciju kopu $\{vt_1,...,vt_i\}$, kas ieejā saņem elementu $elem \in ELEM$. Šīs funkcijas rezultāts ir sekojošas versiju transformācijas:

- $vt=<elem,null,null>$

vai

- $vt=<el,conv,E>$ un $elem \in E \subseteq ELEM$, kur $el \in ELEM$, *conv* ir jebkura konvertācijas funkcija. Šajā pēdējā gadījumā funkcija *vtrans(elem)* atgriež arī *vtrans(el)*.

Šī funkcija tiek izmantota, lai iegūtu visas versiju transformācijas, kurās piedalās konkrētais elements.

Funkcija primary

key=primary(cs) ir funkcija, kas atgriež kolonnu kopas *cs* primāro atslēgu. Ja $cs=<C,KEY,DOMA>$, tad $key=<'PRIMARY',KC,null> \in KEY$.

7.2.2. Lietotie apzīmējumi

Formālajā izmaiņu procedūru aprakstā tiek izmantoti sekojoši apzīmējumi:

- *new* nozīmē jaunā fiziskā, loģiskā vai semantiskā līmeņa modeļa klases instances izveide. Piemēram, c=new Kolonna nozīmē, ka tiek izveidota jaunā klases Kolonna instance.
- *Set* ir esošas fiziskā, loģiskā vai semantiskā līmeņa modeļa klases instances piešķiršana. Piemēram, Set cs=<C,KEY,DOMA> nozīmē, ka kolonnu kopai cs tiek piešķirts kortežs <C,KEY,DOMA>.
- *return* ir procedūras darbības rezultāts.

- *For each ... do* apzīmē ciklā izpildītas darbības. Piemēram, *For each* $att_j \in \{att_1,...,att_i\}$ *do* c_j*=new Kolonna* nozīmē, ka katram atribūtam kopā $\{att_1,...,att_i\}$ tiek izpildīta darbība, kas veido jaunu kolonnu.

7.3. Fiziskās izmaiņas

7.3.1. Jauna atribūta pievienošana dimensijai

Formālā definīcija

Pirmsizmaiņas stāvoklis: Dimensijai *dim=<ATT,LEV,H,AL,HL>* tiek pievienots jauns atribūts *att*. Kolonnu kopa *cs=<C,KEY,DOMA>* atbilst dimensijai *dim*. Datu noliktavas shēmas elementus apraksta biznesa domēna *BD* termini no vārdnīcas *glos*$\in$*GLOS, glos=<TERM,PTMAP,CONTMAP,SEMAP>*, kas ir saistīta ar taksonomiju *tax*$\in$*TAX* ar asociāciju.

Procedūra izmaiņām fiziskā līmenī: Jaunā kolonna *c'* tiek izveidota kolonnu kopā *cs.*

1. c'=new Kolonna;
2. Set cs=<C$\cup${c'},KEY,DOMA$\cup${<c',dom_{att}>})>, kur dom_{att} ir vērtību apgabals, kas atbilst atribūtam *att*;
3. return DBS'=<T,VW,Q,CS>.

Procedūra izmaiņām loģiskā līmenī: Jaunais atribūts *att* tiek pievienots dimensijai *dim*. Tiek uzbūvēts attēlojums ar funkciju "kopija", lai savienotu fiziskajā līmenī izveidotu kolonnu *c'* ar *att*, kas nozīmē, ka atribūts *att* iegūts pa tiešo no datu bāzes kolonnas *c'*. Tiek uzbūvētas versijas transformācijas v_u atribūtam *att*, ja to var izrēķināt no pārējiem atribūtiem.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Set dim=<ATT$\cup${att},LEV,H,AL,HL>;
3. Set v'=<τ_{now},null,VT'$\cup${<att,null,Ø>}>;
4. Ja *att* var tikt izrēķināts no pārējiem dimensijas *dim* atribūtiem $\{att_i,...,att_j\}$ ar funkciju *conv*, tad set v_u=<τ_{from},τ_{to},$VT_u\cup${<att,conv,{att_i,…,att_j}>}>;
5. return DWS'=<F,D,FDA,V$\cup${v'},MAP$\cup${<att,'kopija',{c'}>}>.

Procedūra izmaiņām semantiskā līmenī: Jaunais termins *term* ar vienu termina versiju *tv* un termina definīciju *termdef* tiek pievienots vārdnīcā *glos*. Ja biznesa domēnā *BD* neeksistē jēdziens, ko apraksta termins *term*, tad tiek arī izveidots atbilstošais jaunais jēdziens *conc* taksonomijā *tax*$\in$*TAX*. Termins *term* ir savienots ar jēdzienu *conc* ar asociāciju. Ja atbilstošais jēdziens *conc* eksistē biznesa domēna BD kādā taksonomijā *tax*, tad tas tiek savienots ar terminu *term* ar asociāciju.

1. tv=new TerminaVersija(<tvdef,{att},v'>);
2. term=new Termins(<termdef,{tv}>);
3. Set glos=<TERM$\cup${term}, PTMAP,CONTMAP,SEMAP>;

4. Ja biznesa domēnā *BD* neeksistē jēdziens, ko apraksta termins *term*, tad
   - conc=new Jēdziens;
   - tax=<CONC∪{conc}> , kur *tax* ir taksonomija, kas satur ar jēdzienu *conc* semantiski līdzīgus jēdzienus. Saistītu taksonomiju nosaka datu noliktavas administrators.
5. ctmap=new AsociācijaStarpJēdzieniemUnTerminiem(<conc,term>), kur *conc* ir jēdziens, ko apraksta termins *term* (jaunais vai esošais);
6. return BD'=<TAX,GLOS,TGMAP,CTMAP∪{ctmap}>.

Izmaiņas metadatu repozitorijā

Ja ir nepieciešams izveidot jauno atribūtu A dimensijā D, tad datubāzē tabulai D tiek pievienota jaunā kolonna A.

Metadatos fiziskajā līmenī tiek izveidots jaunais ieraksts A_F tabulā Kolonna, kas ir piesaistīts kolonnu kopai, kas atbilst tabulai D.

Metadatu loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši elementi, kas bija pēdējā versijā V_V. Tiek izveidots jaunais atribūta elements A_L, kas ir piesaistīts dimensijai D. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista kolonnu A_F fiziskā līmeņa metadatos un atribūtu A_L loģiskā līmeņa metadatos. 4. Ja atribūts A var tikt izrēķināts no pārējiem dimensijas D atribūtiem, kas eksistēja versijā V_V ar konvertācijas funkciju KF, tad tiek veidota versijas transformācija jaunajam atribūtam versijā V_V. Citādi versijas transformācija netiek veidota, kas nozīmē, ka versijā V_V atribūta A nebija.

Semantiskā līmeņa metadatos tiek izveidots jaunais termins T tabulā Termins un saistīta jaunā termina versija TV tabulā TerminaVersija. Tiek izveidots ieraksts tabulā SElementaTerminaVersija, kas savieno termina versiju TV ar atribūtu A_L loģiskā līmeņa metadatos. Ja tabulā Jēdziens eksistē atbilstošais jēdziens, ko apraksta termins T, tad tas tiek savienots ar terminu T caur tabulu JēdzienaTermins. Citādi tiek izveidots jaunais jēdziens J, kas tiek savienots ar terminu T arī tabulā JēdzienaTermins.

7.3.2. Dimensijas atribūta datu tipa maiņa

Formālā definīcija

Pirmsizmaiņas stāvoklis: Kolonnu kopas *cs=<C,KEY,DOMA>∈CS* kolonnas *c* vērtību apgabals *dom* tiek mainīts uz *dom'*. Dimensijas *d∈D* atribūts *att∈ATT* tiek iegūts no kolonnas *c* ar attēlojumu *map=<att,'kopija',{c}>*. Funkcijas $conv_{dom->dom'}$ un $conv_{dom'->dom}$ ir funkcijas, kas pārveido vērtības starp attiecīgiem vērtību apgabaliem. Atribūtu *att* apraksta biznesa domēna *BD* termins *term=<termdef,TV>*.

Procedūra izmaiņām fiziskā līmenī: Jaunā kolonna *c'* tiek izveidota kolonnu kopā *cs*.

1. c'=new Kolonna;
2. Set cs=<C∪{c'},KEY,DOMA∪{<c',dom'>})>;
3. return DBS'=<T,VW,Q,CS>.

Procedūra izmaiņām loģiskā līmenī: Jaunais atribūts *att'* tiek pievienots dimensijai *d*. Tiek uzbūvēts attēlojums, lai sasaistītu *c'* ar *att'*. Abas versijas v' un v_u tiek papildinātas ar versijas transformācijām, lai pārveidotu vērtības starp atribūta versijām.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. att'=new Atribūts;
3. Set d=<ATT∪{att'},LEV,H,AL,HL>;
4. Set v_u=<τ_{from},τ_{to},VT_u∪{<att',$conv_{dom->dom'}$,{att}>}>;
5. Set v'=<τ_{now},null,VT'∪{<att,$conv_{dom'->dom}$,{att'}>}>;
6. return DWS'=<F,D,FDA,V∪{v'},MAP∪{<att','kopija',{c'}>}>.

Procedūra izmaiņām semantiskā līmenī: Jaunā termina versija *tv'*, kas apraksta mainīto atribūtu, tiek pievienota terminam *term*.

1. tv'=new TerminaVersija(<tvdef,{att'},v'>);
2. Set term=<termdef,TV∪{tv'}>;
3. return BD'=<TAX,GLOS,TGMAP,CTMAP>.

Izmaiņas metadatu repozitorijā

Ja dimensijas D atribūtam A datu tips tiek mainīts, tad datu bāzes tabulai D tiek pievienota jauna kolonna A' ar jaunu datu tipu.

Metadatos fiziskajā līmenī tiek izveidots jaunais kolonnas elements A' kolonnu kopā, kas atbilst tabulai D.

Metadatu loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, izņemot atribūtu A. Šai jaunajai versijai V_J tiek pievienots jaunais atribūts A'. Tiek izveidots transformācijas elements ar funkciju „kopija", kas sasaista kolonnu A' fiziskajā līmenī un atribūtu A' loģiskajā līmenī, kas nozīmē, ka atribūts A' atbilst kolonnas A' datiem bez jebkādiem pārveidojumiem. Tiek veidota versijas transformācija, kas pārveido atribūtu A versijā V_V uz atribūtu A' versijā V_J ar konvertācijas funkciju, kas pārveido vērtību ar veco datu tipu par vērtību ar jauno datu tipu.

Metadatos semantiskajā līmenī atribūtu A definējošam terminam tiek izveidota jaunā termina versija, t.i. tiek izveidots jaunais ieraksts tabulā TerminaVersija, kas atbilst jaunai mainīta atribūta definīcijai. Šī jaunā termina versija tiek savienota ar atribūtu A' caur tabulu SElementaTerminaVersija.

7.3.3. Dimensijas atribūta dzēšana

Formālā definīcija

Pirmsizmaiņas stāvoklis: Atribūts *att* tiek izdzēsts no dimensijas *dim=<ATT,LEV,H,AL,HL>*. Atribūts var tikt izdzēsts tikai, ja pēdējā derīgā versijā tas nav savienots ne ar vienu līmeni. Ja savienojums ar līmeni eksistē, tad sākumā ir jāatvieno atribūts no visiem līmeņiem. Nav atļauts izdzēst atribūtu, kas atbilst kādai primārai atslēgai.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Versijas transformācija versijai *v'* var tikt uzbūvēta atribūtam *att*, ja tas var tikt izrēķināts no pārējiem šīs versijas atribūtiem. Citā gadījumā atribūts *att* un citi atkarīgi atribūti tiek izdzēsti no versijas *v'*.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Ja atribūts *att* var tikt izrēķināts no pārējiem dimensijas *dim* atribūtiem *{att_i,…,att_j}* ar funkciju *conv*, tad Set v'=<τ_{now},null,VT'∪{<att,conv,{att_i,…,att_j}>}>;
   citādi Set v'=<τ_{now},null,VT'\vtrans(att)>;
3. return DWS'=<F,D,FDA,V∪{v'},MAP>.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Ierobežojums: Atribūtu ir iespējams izdzēst tikai ja tas nav piesaistīts nevienam hierarhijas līmenim. Ja ir saistība ar līmeni, tad sākumā jāizdzēš šīs atribūts no visiem līmeņiem.

Ja ir nepieciešams izdzēst atribūtu A dimensijā D, tad fiziski datubāzē tabulā D šo atribūtu atstāj.

Metadati fiziskajā līmenī arī paliek bez izmaiņām.

Bet metadatos loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, izņemot atribūtu A. Ja atribūts A var tikt izrēķināts no pārējiem dimensijas D atribūtiem, kas eksistē versijā V_J ar konvertācijas funkciju KF, tad tiek veidota versijas transformācija izdzēstam atribūtam A versijā V_J. Citādi versijas transformācija netiek veidota, kas nozīmē, ka versijā V_J atribūta A nav.

Metadati semantiskajā līmenī paliek bez izmaiņām.

7.3.4. Dimensijas atribūta pārsaukšana

Formālā definīcija

Šai izmaiņai formālas definīcijas nav, jo formālā loģiskā un fiziskā līmeņa modeļa definīcija nesatur informāciju par datu noliktavas shēmas elementu nosaukumiem.

Izmaiņas metadatu repozitorijā

Ja dimensijas D atribūts A tika pārsaukts, tad metadatos fiziskajā līmenī tiek pārsaukta kolonna A, kas atbilst atribūtam A.

Loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, tikai atribūts A tiek izveidots ar jauno nosaukumu. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista kolonnu A fiziskajā līmenī un atribūtu A loģiskajā līmenī. Tiek veidota versijas transformācija, kas sasaista atribūtu A versijā V_V uz atribūtu A versijā V_J ar tukšu funkciju. Tas nozīmē, ka semantiski atribūti abās versijās ir identiski.

Metadati semantiskajā līmenī paliek bez izmaiņām.

7.3.5. Jaunas dimensijas izveidošana

Formālā definīcija

Pirmsizmaiņas stāvoklis: Jaunā dimensija *dim* ar atribūtiem *{att₁,...,attᵢ}* tiek pievienota. Datu noliktavas shēmas elementus apraksta biznesa domēna *BD* termini no vārdnīcas $glos \in GLOS$, $glos=<TERM,PTMAP,CONTMAP,SEMAP>$, kas ir saistīta ar taksonomiju $tax \in TAX$ ar asociāciju.

Procedūra izmaiņām fiziskā līmenī: Jaunā kolonnu kopa tiek izveidota ar tās kolonnām un to vērtību kopām.

1. cs=new KolonnuKopa;
2. For each $att_j \in \{att_1,\ldots,att_i\}$ do c_j=new Kolonna;
3. Set $cs=<\{c_1,\ldots,c_i\},Ø,\{<c_1,dom_1>,\ldots,<c_i,dom_i>\})>$, kur *dom₁,...,domᵢ* ir vērtību apgabali, kuri atbilst atribūtiem *{att₁,...,attᵢ}*;
4. return $DBS'=<T,VW,Q,CS\cup\{cs\}>$.

Procedūra izmaiņām loģiskā līmenī: Datu noliktavas shēmai tiek pievienota jauna dimensija ar atribūtiem. Tiek uzbūvēti attēlojumi, kas savieno kolonnu kopas *cs* kolonnas ar dimensijas *dim* atribūtiem.

1. $v'=duplicate(v_u)=<\tau_{now},null,VT'>$;
2. dim=new Dimensija($<\{att_1,\ldots,att_i\},Ø,Ø,Ø,Ø>$);
3. For each $att_j \in \{att_1,\ldots,att_i\}$ do map_j=new Transformācija($<att_j,$'kopija'$,\{c_j\}>$);
4. Set $v'=<\tau_{now},null,VT'\cup\{<dim,null,Ø>,<att_1,null,Ø>,...,<att_i,null,Ø>\}>$;
5. return $DWS'=<F,D,FDA,V\cup\{v'\},MAP\cup\{map_1,\ldots,map_i\}>$.

Procedūra izmaiņām semantiskā līmenī: Katram jaunas dimensijas *dim* atribūtam tiek veidots jaunais termins līdzīgā veidā kā tas tiek darīts jaunā atribūta izveidošanas izmaiņas gadījumā.

1. For each $att_j \in \{att_1,\ldots,att_i\}$ do
   - tv_j=new TerminaVersija($<tvdef_j,\{a_j\},v'>$);
   - $term_j$=new Termins($<termdefj,\{tv_j\}>$);
   - Ja biznesa domēnā *BD* neeksistē jēdziens, ko apraksta termins *termⱼ*, tad $conc_j$=new Jēdziens;
   - $ctmap_j$=new AsociācijaStarpJēdzieniemUnTerminiem($<conc_j,term_j>$), kur *concⱼ* ir jēdziens, ko apraksta termins *termⱼ* (jaunais vai esošais);
2. Set $glos=<TERM\cup\{term_1,...,term_i\}, PTMAP,CONTMAP,SEMAP>$;
3. $tax=<CONC\cup\{conc_1,...,conc_j\}>$, kur *tax* ir taksonomija, kas satur ar jēdzieniem *conc₁,...,concⱼ* semantiski līdzīgus jēdzienus;
4. return $BD'=<TAX,GLOS,TGMAP,CTMAP\cup\{ctmap_1,...,ctmap_j\}>$.

Izmaiņas metadatu repozitorijā

Ja tika izveidota jauna dimensija D, tad fiziskajā līmenī metadatos tiek izveidots jaunais ieraksts tabulās Tabula un KolonnuKopa (ar tipu 'tabula'), kas atbilst jaunai tabulai D. Tiek izveidoti kolonnu un primāras atslēgas metadati.

Loģiskajā līmenī metadatos tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V. Tiek izveidots jaunais dimensijas elements D_L ar atribūtiem, kas ir vienādi ar jaunas tabulas kolonnām. Tiek izveidotas transformācijas ar funkciju „kopija", kas sasaista tabulas D kolonnas fiziskajā līmenī un atribūtus loģiskajā līmenī. Versijas transformācija versijā V_V netiek veidota, kas nozīmē, ka versijā V_V dimensijas D nebija. Atskaites uz dimensiju D var izpildīties tikai no laika, kad tā tika izveidota.

Metadatos semantiskajā līmenī tiek izveidoti jaunie termini ar terminu versijām katram jaunās dimensijas atribūtam.

7.3.6. Dimensijas dzēšana

Formālā definīcija

Pirmsizmaiņas stāvoklis: Dimensija *dim* tiek izdzēsta. Dimensija var tikt izdzēsta tikai, ja pēdējā derīgā versijā tā nesatur nevienu līmeni vai hierarhiju un nav sasaistīta ne ar vienu faktu tabulu. Ja eksistē savienojums, tad pirms šīs izmaiņas izpildīšanas dimensijai ir jābūt atvienotai no visām faktu tabulām.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Dimensijas *dim* atribūti un citi atkarīgie shēmas elementi, kas tiek iegūti ar versijas transformācijām no dimensijas *dim* atribūtiem, tiek izdzēsti no versijas *v'*.

1. v'=duplicate(v_u);
2. v'=<τ_{now},null,VT'\(vtrans(att_1)∪...∪vtrans(att_i))>;
3. return DWS'=<F,D,FDA,V∪{v'},MAP>.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Ierobežojums: Dimensiju ir iespējams izdzēst tikai ja tā nav piesaistīta nevienai faktu tabulai. Ja ir saistība ar faktu tabulu, tad sākumā jāatvieno šī dimensija no visām faktu tabulām.

Ja ir nepieciešams izdzēst dimensiju D, tad fiziski datubāzē šo tabulu atjāj.

Metadati fiziskajā līmenī paliek bez izmaiņām.

Bet metadatos loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši objekti, kas bija pēdējā versijā V_V, izņemot dimensiju D, šīs dimensijas atribūtus, hierarhijas un līmeņus. Versijas transformācija versijā V_J netiek veidota, kas nozīmē, ka versijā V_J dimensijas D nav. Atskaites uz dimensiju D var izpildīties tikai līdz laikam, kad tā tika izdzēsta.

Metadati semantiskajā līmenī nemainās.

7.3.7. Dimensijas pārsaukšana

Formālā definīcija

Šai izmaiņai formālas definīcijas nav, jo formālā loģiskā un fiziskā līmeņa modeļa definīcija nesatur informāciju par datu noliktavas shēmas elementu nosaukumiem.

Izmaiņas metadatu repozitorijā

Ja dimensija D tika pārsaukta, tad metadatos fiziskajā līmenī tiek pārsaukta tabula D.

Loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, tikai dimensija D tiek izveidota ar jauno nosaukumu. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista dimensijas D kolonnas fiziskajā līmenī un dimensijas D atribūtiem loģiskajā līmenī. Tiek veidota versijas transformācija, kas sasaista dimensiju D versijā V_V un dimensiju D versijā V_J ar tukšu funkciju. Tas nozīmē, ka semantiski dimensijas abās versijās ir identiskas.

Metadati semantiskajā līmenī paliek bez izmaiņām.

7.3.8. Jauna mērījuma pievienošana faktu tabulai

Formālā definīcija

Pirmsizmaiņas stāvoklis: Jaunais mērījums *ms* tiek pievienots faktu tabulai *fact*=<*MS*>. Kolonnu kopa *cs*=<*C,KEY,DOMA*> atbilst faktu tabulai *fact*. Datu noliktavas shēmas elementus apraksta biznesa domēna *BD* termini no vārdnīcas $glos \in GLOS$, *glos*=<*TERM,PTMAP,CONTMAP,SEMAP*>, kas ir saistīta ar taksonomiju $tax \in TAX$ ar asociāciju.

Procedūra izmaiņām fiziskā līmenī: Jaunā kolonna *c'* tiek izveidota kolonnu kopā *cs.*

1. c'=new Kolonna;
2. Set cs=<C∪{c'},KEY,DOMA∪{<c',dom_{ms}>}>, kur dom_{ms} ir vērtību apgabals, kas atbilst mērījumam *ms*;
3. return DBS'=<T,VW,Q,CS>.

Procedūra izmaiņām loģiskā līmenī: Jaunais mērījums *ms* tiek pievienots faktu tabulai *fact*. Tiek izveidots attēlojums ar funkciju "kopija", lai savienotu fiziskajā līmenī izveidotu kolonnu *c'* ar mērījumu *ms*, kas nozīmē, ka mērījums *ms* iegūts pa tiešo no datu bāzes kolonnas *c'*. Versijas transformācija versijai v_u var tikt uzbūvēta mērījumam *ms*, ja tas var tikt izrēķināts no pārējiem mērījumiem.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Set fact=<MS∪{ms}>;
3. Set v'=<τ_{now},null,VT'∪{<ms,null,Ø>}>;
4. Ja *ms* varētu tikt izrēķināts no pārējiem faktu tabulas *fact* mērījumiem $\{ms_i,\ldots,ms_j\}$ ar funkciju *conv*, tad set v_u=<τ_{from},τ_{to},VT_u∪{<ms,conv,{ms_i,…,ms_j}>}>;
5. return DWS'=<F,D,FDA,V∪{v'},MAP∪{<ms,'kopija',{c'}>}>.

Procedūra izmaiņām semantiskā līmenī: Jaunais termins *term* ar vienu termina versiju *tv* un termina definīciju *termdef* tiek pievienots vārdnīcā *glos*. Ja biznesa domēnā *BD* neeksistē jēdziens, ko apraksta termins *term*, tad tiek arī izveidots atbilstošais jaunais jēdziens *conc* taksonomijā $tax \in TAX$. Termins *term* ir savienots ar jēdzienu *conc* ar asociāciju. Ja atbilstošais jēdziens *conc* eksistē biznesa domēna BD kādā taksonomijā *tax*, tad tas ir savienots ar terminu *term* ar asociāciju.

1. tv=new TerminaVersija(<tvdef,{ms'},v'>);
2. term=new Termins(<termdef,{tv}>);
3. Set glos=<TERM∪{term}, PTMAP,CONTMAP,SEMAP>;
4. Ja biznesa domēnā *BD* neeksistē jēdziens, ko apraksta termins *term*, tad
   - conc=new Jēdziens;
   - tax=<CONC∪{conc}> , kur *tax* ir taksonomija, kas satur ar jēdzienu *conc* semantiski līdzīgus jēdzienus.
5. ctmap=new AsociācijaStarpJēdzieniemUnTerminiem(<conc,term>), kur *conc* ir jēdziens, ko apraksta termins *term* (jaunais vai esošais);
6. return BD'=<TAX,GLOS,TGMAP,CTMAP∪{ctmap}>.

Izmaiņas metadatu repozitorijā

Ja ir nepieciešams izveidot jauno mērījumu M faktu tabulā F, tad datubāzē tabulai D tiek pievienota jaunā kolonna M_F.

Metadatos fiziskajā līmenī tiek izveidots ieraksts tabulā Kolonna, kas atbilst jaunajai kolonnai M_F kolonnu kopā, kas atbilst tabulai F.

Metadatu loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V. Tiek izveidots jaunais mērījums M_L, kas ir piesaistīts faktu tabulai F. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista kolonnu A_F fiziskajā līmenī un mērījumu M_L loģiskajā līmenī. Ja mērījums M_L var tikt izrēķināts no pārējiem faktu tabulas F mērījumiem, kas eksistēja versijā V_V ar konvertācijas funkciju KF, tad tiek veidota versijas transformācija jaunajam mērījumam versijā V_V. Citādi versijas transformācija netiek veidota, kas nozīmē, ka versijā V_V mērījuma M_L nebija.

Semantiskā līmeņa metadatos tiek izveidots jaunais termins T tabulā Termins un ar to saistīta jaunā termina versija TV tabulā TerminaVersija. Tiek izveidots ieraksts tabulā SElementaTerminaVersija, kas savieno termina versiju TV ar mērījumu M_L loģiskā līmeņa metadatos. Ja tabulā Jēdziens eksistē atbilstošais jēdziens, ko apraksta termins T, tad tas tiek savienots ar terminu T caur tabulu JēdzienaTermins. Citādi tiek izveidots jaunais jēdziens J, kas tiek savienots ar terminu T arī tabulā JēdzienaTermins.

7.3.9. Mērījuma datu tipa maiņa

Formālā definīcija

Pirmsizmaiņas stāvoklis: Kolonnu kopas $cs=<C,KEY,DOMA> \in CS$ kolonnas *c* vērtību apgabals *dom* tiek mainīts uz *dom'*. Faktu tabulas $f \in F$ mērījums $ms \in MS$ tiek iegūts no kolonnas *c* ar attēlojumu *map=<ms,'kopija',{c}>*. Funkcijas $conv_{dom->dom'}$ un $conv_{dom'->dom}$ ir funkcijas, kas pārveido vērtības starp attiecīgiem vērtību apgabaliem. Mērījumu *ms* apraksta

biznesa domēna *BD* termins *term*=<*termdef,TV*>. Šīs izmaiņas apstrāde ir ļoti līdzīga atribūta datu tipa maiņai.

Procedūra izmaiņām fiziskā līmenī: Jaunā kolonna *c'* tiek izveidota kolonnu kopā *cs*.

1. c'=new Kolonna;
2. Set cs=<C∪{c'},KEY,DOMA∪{<c',dom'>})>;
3. return DBS'=<T,VW,Q,CS>.

Procedūra izmaiņām loģiskā līmenī: Jaunais mērījums *ms'* tiek pievienots faktu tabulai *f*. Tiek uzbūvēts attēlojums, lai sasaistītu *c'* ar *ms'*. Abas versijas v' un v_u tiek papildinātas ar versijas transformācijām, lai pārveidotu vērtības starp mērījuma versijām.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. el'=new Mērījums;
3. Set f=<MS∪{ms'}>;
4. Set v_u=<τ_{from},τ_{to},VT_u∪{<ms',$conv_{dom->dom'}$,{ms}>}>;
5. Set v'=<τ_{now},null,VT'∪{<ms,$conv_{dom'->dom}$,{ms'}>}>;
6. return DWS'=<F,D,FDA,V∪{v'},MAP∪{<ms','kopija',{c'}>}>.

Procedūra izmaiņām semantiskā līmenī: Jaunā termina versija *tv'*, kas apraksta mainītu mērījumu, tiek pievienota terminam *term*.

1. tv'=new TerminaVersija(<tvdef,{ms'},v'>);
2. Set term=<termdef,TV∪{tv'}>;
3. return BD'=<TAX,GLOS,TGMAP,CTMAP>.

Izmaiņas metadatu repozitorijā

Ja faktu tabulas F mērījumam M datu tips tiek mainīts, tad datu bāzes tabulai F tiek pievienota jaunā kolonna M' ar jauno datu tipu.

Metadatos fiziskajā līmenī tiek izveidots jaunais kolonnas elements M' kolonnu kopā, kas atbilst tabulai F.

Metadatu loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, tikai mērījums M netiek piesaistīts, bet tiek izveidots jaunais mērījums M'. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista kolonnu M' fiziskajā līmenī un mērījumu M' loģiskajā līmenī. Tiek veidota versijas transformācija, kas pārveido mērījumu M versijā V_V uz mērījumu M' versijā V_J ar konvertācijas funkciju, kas pārveido vērtību ar veco datu tipu par vērtību ar jauno datu tipu.

Metadatos semantiskajā līmenī mērījumu M definējošam terminam tiek izveidota jaunā termina versija, t.i. tiek izveidots jaunais ieraksts tabulā TerminaVersija, kas atbilst jaunai mainītā mērījuma definīcijai. Šī jaunā termina versija tiek savienota ar mērījumu M' caur tabulu SElementaTerminaVersija.

7.3.10. Mērījuma dzēšana

Formālā definīcija

Pirmsizmaiņas stāvoklis: Mērījums *ms* tiek izdzēsts no faktu tabulas *fact*.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Versijas transformācija versijai *v'* var tikt uzkonstruēta mērījumam *ms*, ja tas var tikt izrēķināts no pārējiem mērījumiem. Citā gadījumā mērījums *ms* un pārējie atkarīgie mērījumi tiek izdzēsti no versijas *v'*.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Ja mērījums *ms* var tikt izrēķināts no pārējiem faktu tabulas *fact* mērījumiem *{ms_i,...,ms_j}* ar funkciju *conv*, tad set v'=<τ_{now},null,VT'$\cup${<ms,conv,{ms_i,…,ms_j}>}>, citādi set v'=<τ_{now},null,VT'\vtrans(ms)>.
3. return DWS'=<F,D,FDA,V$\cup${v'},MAP>.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Ja ir nepieciešams izdzēst mērījumu M faktu tabulā F, tad fiziski datubāzē tabulā F atbilstošo kolonnu atstāj.

Metadati fiziskajā līmenī arī paliek bez izmaiņām.

Bet metadatos loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, izņemot mērījumu M. Ja mērījums M var tikt izrēķināts no pārējiem faktu tabulas F mērījumiem, kas eksistē versijā V_J ar konvertācijas funkciju KF, tad tiek veidota versijas transformācija izdzēstam mērījumam M versijā V_J. Citādi versijas transformācija netiek veidota, kas nozīmē, ka versijā V_J mērījuma M nav.

Metadati semantiskajā līmenī paliek bez izmaiņām.

7.3.11. Mērījuma pārsaukšana

Formālā definīcija

Šai izmaiņai formālas definīcijas nav, jo formālā loģiskā un fiziskā līmeņa modeļa definīcija nesatur informāciju par datu noliktavas shēmas elementu nosaukumiem.

Izmaiņas metadatu repozitorijā

Ja faktu tabulas F mērījums M tika pārsaukts, tad metadatos fiziskajā līmenī tiek pārsaukta kolonna M.

Loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, tikai mērījums M tiek izveidots ar jauno nosaukumu. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista kolonnu M fiziskajā līmenī un mērījumu M loģiskajā līmenī. Loģiskajā limenī tiek veidota versijas transformācija, kas sasaista mērījumu M versijā V_V un mērījumu M versijā V_J ar tukšu funkciju. Tas nozīmē, ka semantiski mērījumi abās versijās ir identiski.

Metadati semantiskajā līmenī paliek bez izmaiņām.

7.3.12. Jaunas faktu tabulas izveidošana

Formālā definīcija

Pirmsizmaiņas stāvoklis: Jaunā faktu tabula *fact* ar mērījumiem $\{ms_1,...,ms_i\}$ tiek pievienota. Datu noliktavas shēmas elementus apraksta biznesa domēna *BD* termini no vārdnīcas $glos \in GLOS$, $glos=<TERM,PTMAP,CONTMAP,SEMAP>$, kas ir saistīta ar taksonomiju $tax \in TAX$ ar asociāciju.

Procedūra izmaiņām fiziskā līmenī: Jaunā kolonnu kopa *cs* tiek izveidota ar visām tās kolonnām un kolonnu vērtību kopām.

1. cs=new KolonnuKopa;
2. For each $ms_j \in \{ms_1,\ldots,ms_i\}$ do c_j=new Kolonna;
3. Set $cs=<\{c_1,\ldots,c_i\},Ø,\{<c_1,dom_1>,\ldots,<c_i,\ldots,dom_i>\})>$, kur $dom_1,...,dom_i$ ir vērtību kopas, kas atbilst mērījumiem $\{ms_1,...,ms_i\}$;
4. return DBS'=<T,VW,Q,CS∪{cs}>.

Procedūra izmaiņām loģiskā līmenī: Jaunā faktu tabula ar mērījumiem tiek pievienota loģiskajam modelim. Tiek izveidoti attēlojumi, lai savienotu kolonnu kopas *cs* kolonnas ar faktu tabulas *fact* mērījumiem.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. fact=new FaktuTabula(<$\{ms_1,\ldots,ms_i\}$>);
3. For each $ms_j \in \{ms_1,\ldots,ms_i\}$ do map_j=new Transformācija(<ms_j,'kopija',$\{c_j\}$>);
4. Set v'=<τ_{now},null,VT'∪{<ms_1,null,Ø>,...,<ms_i,null,Ø>}>;
5. return DWS'=<F∪fact,D,FDA,V∪{v'},MAP∪{$map_1,\ldots,map_i$}>.

Procedūra izmaiņām semantiskā līmenī: Katram jaunās faktu tabulas *fact* mērījumam tiek veidots jaunais termins līdzīgā veidā kā tas tiek darīts jauna mērījuma izveidošanas izmaiņas gadījumā.

1. For each $ms_j \in \{ms_1,\ldots,ms_i\}$ do
   - tv_j=new TerminaVersija(<$tvdef_j$,{ms_j},v'>);
   - $term_j$=new Termins(<termdefj,{tv_j}>);
   - Ja biznesa domēnā *BD* neeksistē jēdziens, ko apraksta termins $term_j$, tad $conc_j$=new Jēdziens;
   - $ctmap_j$=new AsociācijaStarpJēdzieniemUnTerminiem(<$conc_j$,$term_j$>), kur $conc_j$ ir jēdziens, ko apraksta termins $term_j$ (jaunais vai esošais);
2. Set glos=<TERM∪{$term_1,...,term_i$}, PTMAP,CONTMAP,SEMAP>;
3. tax=<CONC∪{$conc_1,...,conc_j$}> , kur *tax* ir taksonomija, kas satur ar jēdzieniem $conc_1,...,conc_j$ semantiski līdzīgus jēdzienus;
4. return BD'=<TAX,GLOS,TGMAP,CTMAP∪{$ctmap_1,...,ctmap_j$}>.

Izmaiņas metadatu repozitorijā

Ja tika izveidota jauna faktu tabula F, tad fiziskajā līmenī metadatos tiek izveidots jaunais ieraksts tabulās Tabula un KolonnuKopa (ar tipu 'tabula'), kas atbilst jaunai tabulai F. Tiek izveidoti kolonnu un primāras atslēgas metadati.

Loģiskajā līmenī metadatos tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V. Tiek izveidots jaunais faktu tabulas elements F_L ar mērījumiem, kas ir vienādi ar jaunās tabulas kolonnām. Tiek izveidoti transformācijas elementi ar funkciju „kopija", kas sasaista tabulas F kolonnas fiziskajā līmenī un mērījumus loģiskajā līmenī. Versijas transformācija netiek veidota, kas nozīmē, ka versijā V_V faktu tabulas F nebija. Atskaites uz faktu tabulu F var izpildīties tikai no laika, kad tā tika izveidota.

Metadatos semantiskajā līmenī tiek izveidoti jaunie termini ar terminu versijām katram jaunās faktu tabulas mērījumam.

7.3.13. Faktu tabulas dzēšana

Formālā definīcija

Pirmsizmaiņas stāvoklis: Faktu tabula *fact* tiek izdzēsta. Faktu tabula var tikt izdzēsta tikai, ja pēdējā derīgā versijā tā nav sasaistīta ne ar vienu dimensiju. Ja savienojums eksistē, tad sākumā faktu tabula ir jāatvieno no visām dimensijām.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Faktu tabulas *fact* mērījumi un citi atkarīgie mērījumi tiek izdzēsti no versijas *v'*.

1. $v'=\text{duplicate}(v_u)=<\tau_{now},\text{null},VT'>$;
2. Set $v'=<\tau_{now},\text{null},VT'\setminus(\text{vtrans}(ms_1)\cup...\cup\text{vtrans}(ms_i))>$;
3. return $DWS'=<F,D,FDA,V\cup\{v'\},MAP>$.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Ierobežojums: Faktu tabulu ir iespējams izdzēst tikai ja tā nav savienota ne ar vienu dimensiju. Ja faktu tabula ir savienota ar kādu dimensiju, tad sākumā jāatvieno šī faktu tabula no visām dimensijām.

Ja ir nepieciešams izdzēst faktu tabulu F, tad fiziski datubāzē šo tabulu atstāj.

Metadati fiziskajā līmenī paliek bez izmaiņām.

Bet metadatos loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, izņemot faktu tabulu F, šīs faktu tabulas mērījumus un saistības ar dimensijām. Versijas transformācija netiek veidota, kas nozīmē, ka versijā V_J faktu tabulas F nav. Atskaites uz faktu tabulu F var izpildīties tikai līdz laikam, kad tā tika izdzēsta.

Metadati semantiskajā līmenī nemainās.

7.3.14. Faktu tabulas pārsaukšana

Formālā definīcija

Šai izmaiņai formālas definīcijas nav, jo formālā loģiskā un fiziskā līmeņa modeļa definīcija nesatur informāciju par datu noliktavas shēmas elementu nosaukumiem.

Izmaiņas metadatu repozitorijā

Ja faktu tabula F tika pārsaukta, tad metadatos fiziskajā līmenī tiek pārsaukta tabula F.

Loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, tikai faktu tabula F tiek izveidota ar jauno nosaukumu. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista faktu tabulas F kolonnas fiziskajā līmenī un faktu tabulas F mērījumus loģiskajā līmenī. Tiek veidota versijas transformācija, kas sasaista faktu tabulu F versijā V_V un faktu tabulu F versijā V_J ar tukšu funkciju. Tas nozīmē, ka semantiski faktu tabulas abās versijās ir identiskas.

Metadati semantiskajā līmenī paliek bez izmaiņām.

7.4. Loģiskās izmaiņas

7.4.1. Dimensijas piesaistīšana faktu tabulai

Formālā definīcija

Pirmsizmaiņas stāvoklis: Dimensija $dim=<ATT,LEV,H,AL,HL>$ tiek savienota ar faktu tabulu $fact=<\{ms_1,...,ms_i\}>$, mērījumi $\{ms_1,...,ms_i\}$ ir attēloti kā kolonnas $\{c_1,...,c_i\}$. Kolonnu kopas cs_d un $cs_f=<C,KEY,DOMA>$ atbilst dimensijai *dim* un faktu tabulai *fact*.

Procedūra izmaiņām fiziskā līmenī: Ārējās atslēgas kolonnas tiek pievienotas kolonnu kopai cs_f un tiek savienotas ar kolonnu kopas cs_d primārās atslēgas kolonnām. Kolonnas $c'_1,...,c'_i$ tiek iekļautas kolonnu kopas primārajā atslēgā cs_f.

1. $pkey_d=primary(cs_d)=<'PRIMARY',\{kc_1,\ldots,kc_i\},null>$;
2. For each $kc_j \in \{kc_1,\ldots,kc_i\}$ do c'_j=new Kolonna;
3. fkey=new Atslēga(<'FOREIGN',$\{c'_1,\ldots,c'_i\}$,pkey>);
4. $pkey_f=primary(cs_f)=<'PRIMARY',KC,null>$;
5. Set $pkey_f=<'PRIMARY',KC\cup\{c'_1,\ldots,c'_i\},null>$;
6. Set $cs_f=<C\cup\{c'_1,\ldots,c'_i\},KEY\cup\{fkey\},DOMA\cup\{<c'_1,dom_1>,\ldots,<c'_i,\ldots,dom_i>\}>$, kur $dom_1,...,dom_i$ ir vērtību kopas, kas atbilst kolonnām $\{kc_1,...,kc_i\}$;
7. return DBS'=<T,VW,Q,CS>.

Procedūra izmaiņām loģiskā līmenī: Faktu tabulas un dimensijas asociācija tiek izveidota starp faktu tabulu *fact* un dimensiju *dim*. Tiek izveidotas versiju transformācijas faktu tabulas *fact* mērījumiem, ja tas ir iespējams.

1. $v'=duplicate(v_u)=<\tau_{now},null,VT'>$;
2. fda'=new FaktuTabulasDimensija(<dim,fact>);
3. For each $ms_j \in \{ms_1,\ldots,ms_i\}$ do ms'_j=new Mērījums;
4. Set fact=$<\{ms_1,\ldots,ms_i,ms'_1,\ldots,ms'_i\}>$;

5. For each $ms_j \in \{ms_1,\ldots,ms_i\}$ do map'$_j$=new Transformācija(<ms'$_j$,'kopija',{c_j}>);
6. Ja ir iespējams pārveidot (sadalīt) mērījumu *{ms$_1$,...,ms$_i$}* vērtības atbilstoši dimensijas *dim* ierakstiem ar funkciju *conv*, tad sekojošas versiju transformācijas tiek uzbūvētas:
   - For each $ms'_j \in \{ms'_1,\ldots,ms'_i\}$ do
     vt'$_j$=new VersijasTransformācija(<ms'$_j$,conv,{ms_j}>);
   - Set v_u=<τ_{now},null,$VT_u \cup \{vt'_1,\ldots,vt'_i\}$>;
7. Ja ir iespējams pārveidot (apkopot) mērījumu *{ms'$_1$,...,ms'$_i$}* vērtības atbilstoši dimensijas *dim* ierakstiem ar funkciju *conv'*, tad sekojošas versiju transformācijas tiek uzbūvētas:
   - For each $ms_j \in \{ms_1,\ldots,ms_i\}$ do
     vt_j=new VersijasTransformācija(<ms_j,conv,{ms'$_j$}>);
   - Set v'=<τ_{now},null,$VT' \cup \{vt_1,\ldots,vt_i\} \cup \{<fda',null,\emptyset>\}$>;

   Citādi: Set
   v'=<τ_{now},null,$VT' \setminus (vtrans(ms_1) \cup \ldots \cup vtrans(ms_i)) \cup \{<fda',null,\emptyset>\}$>;
8. return DWS'=<F,D,FDA$\cup$fda',V$\cup${v'},MAP$\cup${map'$_1$,…,map'$_i$}>.

Procedūra izmaiņām semantiskā līmenī: Ja jaunām mērījumu versijām mainījās nozīme, tad katram mērījumam tiek izveidota jaunā termina versija līdzīgi jaunā mērījuma pievienošanas izmaiņai, kas ir aprakstīta apakšnodaļā 7.3.8.

Izmaiņas metadatu repozitorijā

Lai piesaistītu dimensiju D faktu tabulai F ir nepieciešams izveidot ārējas atslēgas kolonnu (vai kolonnas) faktu tabulā F, kas tiks sasaistīta ar dimensijas primārās atslēgas kolonnu (vai kolonnām).

Fiziski datubāzē sākumā dimensijā D tiek izveidots fiktīvais ieraksts (piemēram, ar datiem 'viss kopā') ar identifikatoru I. Tiek veidota jaunā tabulas F ārējās atslēgas kolonna K (vai kolonnas, ja dimensijas primārā atslēga sastāv no vairākām kolonnām), kas tiek piesaistīta dimensijas D primārai atslēgai. Ja faktu tabulā ir ieraksti, tad tajos jaunā izveidotā kolonna jāaizpilda ar dimensijas fiktīva ieraksta identifikatoru I. Atkarībā no prasībām, jaunā kolonna K varētu tikt iekļauta faktu tabulas primārajā atslēgā.

Fiziskā līmeņa metadatos tiek izveidoti metadati par jauno kolonnu K tabulā Kolonna. Jaunā kolonna K tiek piesaistīta kolonnu kopai F, arī tiek veidoti metadati par ārējo atslēgu, kas atsaucās uz dimensijas primāro atslēgu.

Loģiskā līmeņa metadatos tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V. Tiek izveidota saite starp dimensiju D un faktu tabulu F tabulā FaktuTabulasDimensija. Lai varētu izpildīt atskaites uz divām datu noliktavas shēmas versijām, tiek izveidotas versijas transformācijas faktu tabulas F mērījumiem. Versijas transformācijas tiek veidotas divos virzienos no versijas V_V uz versiju V_J un no versijas V_J uz versiju V_V, ja tas ir iespējams. Versijas transformācijā no versijas V_V uz versiju V_J tabulā VersijasTransformācija laukā Konvertācija ieraksta funkciju, kas pārveido (sadala) mērījumu datus jaunajā versijā V_J, ja tas ir iespējams. Versijas transformācijā no versijas V_J uz versiju V_V tabulā VersijasTransformācija laukā Konvertācija ieraksta funkciju, kas pārveido (apkopo) mērījumu datus vecajā versijā V_V, ja tas ir iespējams. Ja tas nav iespējams, atskaitēs, kas

pārklāj divas versijas, dati tiek attēloti pa periodiem līdz un pēc shēmas maiņas vai saskaņā tikai ar versiju V_J vai tikai versiju V_V.

7.4.2. Dimensijas atvienošana no faktu tabulas

Formālā definīcija

Pirmsizmaiņas stāvoklis: Dimensija *dim=<ATT,LEV,H,AL,HL>* tiek atvienota no faktu tabulas *fact=<{ms₁,...,msᵢ}>*, mērījumi *{ms₁,...,msᵢ}* ir attēloti kā kolonnas *{c₁,...,cᵢ}*.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Ja tas ir iespējams, tad tiek veidotas versiju transformācijas faktu tabulas *fact* mērījumiem. Ja tas nav iespējams, tad faktu tabulas dimensijas asociācija starp faktu tabulu *fact* un dimensiju *dim* tiek izdzēsta no versijas *v'*.

1. $v'=\text{duplicate}(v_u)=<\tau_{now},null,VT'>$;
2. For each $ms_j \in \{ms_1,\ldots,ms_i\}$ do ms'_j=new Mērījums;
3. Set $fact=<\{ms_1,\ldots,ms_i,ms'_1,\ldots,ms'_i\}>$;
4. For each $ms_j \in \{ms_1,\ldots,ms_i\}$ do map'_j=new Transformācija(<ms'_j,'kopija',{c_j}>);
5. Ja ir iespējams pārveidot (sadalīt) mērījumu *{ms'₁,...,ms'ᵢ}* vērtības atbilstoši dimensijas *dim* ierakstiem ar funkciju *conv*, tad sekojošas versiju transformācijas tiek uzbūvētas:
   - For each $ms_j \in \{ms_1,\ldots,ms_i\}$ do
     vt_j=new VersijasTransformācija(<ms_j,conv,{ms'_j}>);
   - Set $v'=<\tau_{now},null,VT' \cup \{vt_1,\ldots,vt_i\}>$;

   Citādi: Set
   $v'=<\tau_{now},null,VT' \setminus vtrans(ms_1) \cup \ldots \cup vtrans(ms_i)) \setminus \{<fda',null,\emptyset>\}>$;
6. Ja ir iespējams pārveidot (apkopot) mērījumu *{ms₁,...,msᵢ}* vērtības atbilstoši dimensijas *dim* ierakstiem ar funkciju *conv'*, tad sekojošas versiju transformācijas tiek uzbūvētas:
   - For each $ms'_j \in \{ms'_1,\ldots,ms'_i\}$ do
     vt'_j= new VersijasTransformācija(<ms'_j,conv,{ms_j}>);
   - Set $v_u=<\tau_{now},null,VT_u \cup \{vt'_1,\ldots,vt'_i\}>$;
7. return $DWS'=<F,D,FDA \cup fda',V \cup \{v'\},MAP \cup \{map'_1,\ldots,map'_i\}>$.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Lai atvienotu dimensiju D no faktu tabulas F, fiziski datubāzē dimensijā D tiek izveidots fiktīvais ieraksts (piemēram, ar datiem 'viss kopā') ar identifikatoru I. Kad notiek datu ielāde, visiem jauniem faktu tabulas ierakstiem tiek piesaistīts šī fiktīvā dimensijas D ieraksta identifikators.

Fiziskā līmeņa metadatos nekas netiek mainīts.

Loģiskā līmeņa metadatos tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, izņemot asociāciju starp dimensiju D un faktu

tabulu F tabulā FaktuTabulasDimensija. Lai varētu izpildīt atskaites uz divām datu noliktavas shēmas versijām, tiek izveidotas versijas transformācijas faktu tabulas F mērījumiem. Versijas transformācijas tiek veidotas divos virzienos no versijas V_V uz versiju V_J un no versijas V_J uz versiju V_V, ja tas ir iespējams. Versijas transformācijā no versijas V_J uz versiju V_V tabulā VersijasTransformācija laukā Konvertācija ieraksta funkciju, kas pārveido (sadala) mērījumu datus vecajā versijā V_V, ja tas ir iespējams. Versijas transformācijā no versijas V_V uz versiju V_J tabulā VersijasTransformācija laukā Konvertācija ieraksta funkciju, kas pārveido (apkopo) mērījumu datus jaunajā versijā V_J, ja tas ir iespējams. Ja tas nav iespējams, atskaitēs, kas pārklāj divas versijas, dati tiek attēloti pa periodiem līdz un pēc shēmas maiņas vai saskaņā tikai ar versiju V_J vai tikai versiju V_V.

7.4.3. Jaunas dimensijas hierarhijas veidošana

Formālā definīcija

Pirmsizmaiņas stāvoklis: Jaunā hierarhija *h* tiek pievienota dimensijai *dim=<ATT,LEV,H,AL,HL>*.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Jaunā hierarhija *h* tiek pievienota dimensijai *dim* versijā *v'*.

1. $v'=\text{duplicate}(v_u)=<\tau_{now},null,VT'>$;
2. Set $dim=<ATT,LEV,H\cup h,AL,HL>$;
3. Set $v'=<\tau_{now},null,VT'\cup\{<h,null,\emptyset>\}>$;
4. return $DWS'=<F,D,FDA,V\cup\{v'\},MAP>$.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Lai izveidotu jauno dimensijas D hierarhiju H, loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V. Papildus tiek veidoti metadati par jaunu hierarhiju tabulā Hierarhija, kas tiek piesaistīta dimensijai D. Versijas transformācija versijā V_V netiek veidota, kas nozīmē, ka versijā V_V hierarhijas H nebija. Atskaites, kas izmanto jauno hierarhiju H, var izpildīties tikai no laika, kad tā tika izveidota.

7.4.4. Dimensijas hierarhijas dzēšana

Formālā definīcija

Pirmsizmaiņas stāvoklis: Hierarhija *h* tiek izdzēsta no dimensijas *dim=<ATT,LEV,H,AL,HL>*.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Hierarhija *h* tiek izdzēsta no dimensijas *dim* no versijas *v'*.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Set v'=<τ_{now},null,VT'\{<h,null,Ø>}>.
3. return DWS'=<F,D,FDA,V∪{v'},MAP>.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Lai izdzēstu dimensijas D hierarhiju H, loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, izņemot hierarhiju H, t.i. netiek veidota versijas transformācija, kas piesaista hierarhiju H dimensijai D. Tas nozīmē, ka versijā V_J hierarhijas H nav. Atskaites, kas izmanto hierarhiju H, var izpildīties tikai līdz laikam, kad tā tika izmesta.

7.4.5. Jaunais hierarhijas līmenis

Formālā definīcija

Pirmsizmaiņas stāvoklis: Jaunais līmenis *lev* tiek pievienots dimensijai *dim=<ATT,LEV,H,AL,HL>*.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Jaunais līmenis *lev* tiek pievienots dimensijai *dim*.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Set dim=<ATT,LEV∪{lev},H,AL,HL>;
3. Set v'=<τ_{now},null,VT'∪{<lev,null,Ø>}>;
4. return DWS'=<F,D,FDA,V∪{v'},MAP>.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Lai izveidotu jauno hierarhijas H līmeni L, loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V. Papildus tiek veidoti metadati, kas apraksta jauno līmeni tabulā Līmenis, kas ir piesaistīts dimensijai D, kam pieder hierarhija H. Tad šis līmenis tiek piesaistīts hierarhijai H caur tabulu HierarhijaLīmenis. Tiek ierakstīts atbilstošais indekss, t.i. līmeņa atrašanas vieta hierarhijā. Ja tabulā HierarhijaLīmenis ir ieraksts par līmeni, kas pieder hierarhijai H un kam indekss ir vienāds ar jaunā līmeņa L indeksu, tad visiem hierarhijas līmeņa ierakstiem, kam indekss ir vienāds vai lielāks par līmeņa L indeksu, tiek izveidoti jauni metadati tabulā HierarhijasLīmenis, kur indekss tiek palielināts par vienu. Visiem tabulas HierarhijasLīmenis ierakstiem, kas tika mainīti, tiek veidotas versijas transformācijas ar tukšu konvertāciju. Jaunajam līmenim L netiek veidota versijas transformācija vecajā versijā V_V, kas nozīmē, ka

versijā V_V līmeņa L nebija. Atskaites, kas izmanto jauno līmeni L, var izpildīties tikai no laika, kad tas tika izveidots.

Atribūta piesaistīšana līmenim tiek apskatīta apakšnodaļā 7.4.9. kā atsevišķa izmaiņa.

7.4.6. Līmeņa dzēšana no hierarhijas

Formālā definīcija

Pirmsizmaiņas stāvoklis: Līmenis *lev* tiek izdzēsts no dimensijas *dim* hierarhijas *h*.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Asociācija starp hierarhiju *h* un līmeni *lev* tiek izdzēsta no versijas *v'*.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Set v'=<τ_{now},null,VT'\{<<lev,h>,null,Ø>}>;
3. Ja hierarhijā *h* eksistē vēl citi līmeņi, kuru indekss ir lielāks, nekā līmeņa *lev* indekss, tad šo līmeņu indekss tiek samazināts par 1;
4. return DWS'=<F,D,FDA,V∪{v'},MAP>.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Lai izdzēstu dimensijas D līmeni L no hierarhijas H, loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, izņemot versijas transformāciju, kas piesaista versijai V_V tabulas HierarhijasLīmenis metadatus, kas atbilst hierarhijas H līmenim L. Ja šajā tabulā hierarhijai H eksistē līmeņi, kam indekss ir lielāks par līmeņa L indeksu hierarhijā H, tad visiem šiem līmeņiem tiek izveidoti jauni metadati tabulā HierarhijasLīmenis, kur indekss tiek samazināts par vienu. Versijas transformācija versijā V_J netiek veidota, kas nozīmē, ka versijā V_J hierarhijā H līmeņa L nav. Atskaites, kas izmanto šo saistību starp hierarhiju H un līmeni L, var izpildīties tikai līdz laikam, kad tā tika izmesta.

7.4.7. Līmeņa dzēšana no dimensijas

Formālā definīcija

Pirmsizmaiņas stāvoklis: Līmenis *lev* tiek izdzēsts no dimensijas *dim*. Līmenim *lev* ir jābūt izdzēstiem no visām hierarhijām, kur tas ir izmantots.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Līmenis *lev* tiek izdzēsts no versijas *v'*.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Set v'=<τ_{now},null,VT'\{<lev,null,Ø>}>;
3. return DWS'=<F,D,FDA,V∪{v'},MAP>.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Lai izdzēstu dimensijas D līmeni L, loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, izņemot līmeni L. Arī jaunajā versijā V_J netiek veidotas šī līmeņa saistības ar visām hierarhijām tabulā HierarhijasLīmenis. Katrai saistītai hierarhijai tiek atrasti visi līmeņi, kam indekss ir lielāks par līmeņa L indeksu katrā hierarhijā. Visiem šiem līmeņiem indekss tiek samazināts par vienu. Versijas transformācija versijā V_J netiek veidota, kas nozīmē, ka versijā V_J līmeņa L nav. Atskaites, kas izmanto līmeni L, var izpildīties tikai līdz laikam, kad tas tika izmests.

7.4.8. Līmeņa pievienošana hierarhijai

Formālā definīcija

Pirmsizmaiņas stāvoklis: Eksistējošais līmenis *lev* tiek savienots ar dimensijas *dim=<ATT,LEV,H,AL,HL>* hierarhiju *h*.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Jaunā līmeņa un hierarhijas asociācija starp līmeni *lev* un hierarhiju *h* tiek pievienota dimensijai *dim*.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Set dim=<ATT,LEV,H,AL,HL∪{<lev,h>}>;
3. Set v'=<τ_{now},null,VT'∪{<<lev,h>,null,Ø>}>;
4. Ja hierarhijā *h* ir vēl citi līmeņi, kam indekss ir vienāds vai lielāks par līmeņa *lev* indeksu, tad šo līmeņu indekss tiek palielināts par 1.
5. return DWS'=<F,D,FDA,V∪{v'},MAP>.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Lai pievienotu eksistējošo dimensijas D līmeni L hierarhijai H, loģiskajā līmenī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V. Papildus līmenis L tiek piesaistīts hierarhijai H caur tabulu HierarhijasLīmenis, t.i. tiek veidoti metadati šajā tabulā, kas tiek piesaistīti versijai V_J ar versijas transformāciju. Tabulā HierarhijasLīmenis tiek ierakstīts atbilstošais indekss, t.i. līmeņa atrašanas vieta hierarhijā. Ja tabulā HierarhijasLīmenis ir metadati par līmeni, kas pieder hierarhijai H un kam indekss ir vienāds ar jaunā līmeņa L indeksu, tad visiem hierarhijas līmeņa ierakstiem, kam indekss ir vienāds vai lielāks par līmeņa L indeksu, indekss tiek palielināts par vienu. Visiem tabulas HierarhijasLīmenis ierakstiem, kas tika mainīti, tiek veidotas versijas transformācijas ar tukšu konvertāciju. Jaunajam ierakstam, kas atbilst līmenim L, versijas transformācija netiek veidota, kas nozīmē, ka versijā V_V līmenis L nebija piesaistīts hierarhijai H. Atskaites, kas izmanto hierarhijas H līmeni L, var izpildīties tikai no laika, kad tas tika izveidots.

7.4.9. Atribūta piesaistīšana līmenim

Formālā definīcija

Pirmsizmaiņas stāvoklis: Eksistējošais atribūts *att* tiek savienots ar dimensijas *dim=<ATT,LEV,H,AL,HL>* līmeni *lev*.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Jaunā atribūta un līmeņa asociācija starp atribūtu *att* un līmeni *lev* tiek pievienota dimensijai *dim*.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Set dim=<ATT,LEV,H,AL∪{<att,lev>},HL>;
3. Set v'=<τ_{now},null,VT'∪{<<att,lev>,null,Ø>}>;
4. return DWS'=<F,D,FDA,V∪{v'},MAP>.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Lai pievienotu dimensijas D atribūtu A līmenim L, loģiskajā modelī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V. Papildus tabulā LīmeņaAtribūts tiek veidots ieraksts, kas sasaista atribūtu A ar līmeni L. Šai jaunajai atribūta un līmeņa saistībai netiek veidota versijas transformācija versijā V_V, kas nozīmē, ka versijā V_V atribūts A nebija piesaistīts līmenim L. Atskaites, kas izmanto saistību starp atribūtu A un līmeni L, var izpildīties tikai no laika, kad A tika piesaistīts L.

7.4.10. Atribūta atvienošana no līmeņa

Formālā definīcija

Atribūts *att* tiek atvienots no līmeņa *lev*.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Atribūta un līmeņa asociācija starp atribūtu *att* un līmeni *lev* tiek izdzēsta no versijas *v'*.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Set v'=<τ_{now},null,VT'\{<<att,lev>,null,Ø>}>;
3. return DWS'=<F,D,FDA,V∪{v'},MAP>.

Procedūra izmaiņām semantiskā līmenī: Bez izmaiņām.

Izmaiņas metadatu repozitorijā

Lai atvienotu dimensijas D atribūtu A no līmeņa L, loģiskajā modelī tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, izņemot tabulas LīmeņaAtribūts metadatus, kas sasaista atribūtu A ar līmeni L. Versijas

transformācija versijā V_J netiek veidota, kas nozīmē, ka versijā V_J līmenis L nav saistīts ar atribūtu A. Atskaites, kas izmanto šo saistību starp līmeni L un atribūtu A, var izpildīties tikai līdz laikam, kad tā tika izmesta.

7.5. Semantiskās izmaiņas

7.5.1. Atribūta nozīmes maiņa

Formālā definīcija

Pirmsizmaiņas stāvoklis: Dimensijas *dim* atribūtam *att* tiek mainīta nozīme no *tvdef* uz *tvdef'*. Atribūtu apraksta termina *term*=<*termdef,TV*> versija *tv*. Atribūtam atbilst kolonna *c*.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Jaunajā versijā *v'* tiek izveidots jaunais atribūts *att'*. Atribūts *att* tiek izmests no shēmas versijas, bet visās šī atribūta saistībās ar dimensijas līmeņiem tiek ierakstīts jaunais atribūts *att'*.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Set dim=<ATT∪{att'},LEV,H,AL,HL>;
5. For each al∈AL & al=<att,lev> (kur *lev* ir jebkurš līmenis) do
   Set dim=<ATT,LEV,H,AL∪{<att',lev>}\{al},HL>;
4. Set dim=<ATT\{att},LEV,H,AL,HL>;
5. Set v'=<τ_{now},null,VT'\vtrans(att)∪{<att',null,Ø>}>;
6. return DWS'=<F,D,FDA,V∪{v'},MAP∪{<att','kopija',{c}>}>.

Procedūra izmaiņām semantiskā līmenī: Jaunā termina *term* versija *tv* tiek izveidota un piesaistīta atribūtam *att'*.

1. tv=new TerminaVersija(<tvdef',{att'},v'>);
2. Set term=<termdef,TV∪{tv}>;
3. return BD'=<TAX,GLOS,TGMAP,CTMAP>.

Izmaiņas metadatu repozitorijā

Lai mainītu atribūtam nozīmi, loģiska līmeņa metadatos tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, izņemot atribūtu A, kam mainījās nozīme. Tabulā Atribūts tiek izveidoti metadati par jauno atribūtu A', kas atbilst jaunai atribūta A nozīmei. Šis atribūts A' tiek piesaistīts jaunajai versijai V_J caur tabulu VersijasTransformācija. Jaunajā versijā V_J atribūtam A' tiek arī izveidoti metadati par šī atribūta saistību ar tiem pašiem hierarhiju līmeņiem tabulā LīmeņaAtribūts, ar kuriem tika sasaistīts atribūts A.

Semantiskā līmeņa metadatos tiek izveidota jauna termina versija tabulā TerminaVersija, kas satur jauno atribūta nozīmi, atbilst terminam T, kas apraksta atribūtu A. Šī jaunā termina versija tiek piesaistīta shēmas versijai V_J un atribūtam A' caur tabulu SElementaTerminaVersija.

7.5.2. Mērījuma nozīmes maiņa

Formālā definīcija

Pirmsizmaiņas stāvoklis: Faktu tabulas *fact* mērījumam *ms* tiek mainīta nozīme no *tvdef* uz *tvdef'*. Mērījumu apraksta termina *term=<termdef,TV>* versija *tv*. Mērījumam atbilst kolonna *c*.

Procedūra izmaiņām fiziskā līmenī: Bez izmaiņām.

Procedūra izmaiņām loģiskā līmenī: Jaunajā versijā *v'* tiek izveidots jaunais mērījums *ms'*. Mērījums *ms* tiek izmests no shēmas versijas *v'*.

1. v'=duplicate(v_u)=<τ_{now},null,VT'>;
2. Set fact=<MS∪{ms'}\{ms}>;
3. Set v'=<τ_{now},null,VT'\vtrans(ms)∪{<ms',null,Ø>}>;
4. return DWS'=<F,D,FDA,V∪{v'},MAP∪{<ms','kopija',{c}>}>.

Procedūra izmaiņām semantiskā līmenī: Jaunā termina *term* versija *tv* tiek izveidota un piesaistīta mērījumam ms*'*.

1. tv=new TerminaVersija(<tvdef',{ms'},v'>);
2. Set term=<termdef,TV∪{tv}>;
3. return BD'=<TAX,GLOS,TGMAP,CTMAP>.

Izmaiņas metadatu repozitorijā

Lai mainītu mērījumam nozīmi, loģiskā līmeņa metadatos tiek izveidota versija V_J, kurai tiek piesaistīti visi tie paši shēmas elementi, kas bija pēdējā versijā V_V, izņemot mērījumu M, kam mainījās nozīme. Tabulā Mērījums tiek izveidoti metadati par jauno mērījumu M', kas atbilst jaunai mērījuma M nozīmei. Šis mērījums M' tiek piesaistīts jaunajai versijai V_J caur tabulu VersijasTransformācija.

Semantiskā līmeņa metadatos tiek izveidota jauna termina versija tabulā TerminaVersija, kas satur jauno mērījuma nozīmi, atbilst terminam T, kas apraksta mērījumu M. Šī jaunā termina versija tiek piesaistīta shēmas versijai V_J un mērījumam A' caur tabulu SElementaTerminaVersija.

7.6. Nodaļas secinājumi

Šajā nodaļā tika specificētas izmaiņu procedūras, kas ir jāizpilda, lai izplatītu datu noliktavas shēmā notikušās izmaiņas formālajā modelī un tam atbilstošajos metadatos. Katrai izmaiņai tiek dots vispārējs izmaiņas apraksts un procedūra, kas ir jāveic, lai izplatītu izmaiņas daudzversiju datu noliktavas modelī. Tiek aprakstīti nosacījumi, kam jāizpildās, lai varētu izpildīt konkrēto izmaiņu. Kur tas ir iespējams, tiek aprakstītas versiju transformācijas, kas tiek izveidotas, lai padarītu iespējamu datu transformāciju starp vairākām versijām.

Šajā nodaļā aprakstītās izmaiņas tiek atbalstītas izstrādātajos metadatos. Šo izmaiņu saraksts tika iegūts, balstoties uz izmaiņām, kas notika e-studiju datuves projektā, kas tika

aprakstīts nodaļā 4., un kas tika minētas zinātniskajos rakstos par datu noliktavas evolūciju [HMV99], [BSH99], [Bla00], [KPR04], [WB05], [GLR+04], [GLR+06] u.c.

Izmaiņu aprakstā tika apskatīts gadījums, kad katra datu noliktavas dimensija un faktu tabula atbilst tieši vienai fiziskai tabulai relāciju datu bāzē un katrs faktu tabulas mērījums un dimensijas atribūts atbilst tieši vienai tabulas kolonnai. Šis gadījums tika realizēts ar funkciju 'kopija', kas ir izmantota mērījumu un atribūtu attēlojumos. Fiziskā un loģiskā līmeņa metadatos ir iespējams aprakstīt arī gadījumu, kad atribūti un mērījumi tiek izrēķināti no vairāku tabulas kolonnu kombinācijas ar noteiktu funkciju. Šīs funkcijas var būt ļoti dažādas un nav iespējams vispārīgi nodefinēt visu piedāvātajā modelī atbalstītu izmaiņu apstrādes procesu jebkurai patvaļīgai funkcijai. Ja mērījums vai atribūts tiek iegūts no tabulu kolonnu kombinācijas ar funkciju, tad izmaiņas apstrādes procedura ir jāveido atšķirīga konkrētai situācijai.

Izmaiņu apstrādes procedūru apraksti ir doti ar mērķi apskatīt tipiskas izmaiņas datu noliktavas shēmā un pārbaudīt izstrādāto metadatu pilnību, lai noteiktu, vai tiek nodrošināta visu iespējamo izmaiņu rezultātu aprakstīšana metadatos un vai ir iespējams izveidot versijas mainītiem shēmas elementiem metadatos. Tas ir nepieciešams, lai promocijas darba ietvaros izstrādātais atskaišu definēšanas un izpildīšanas rīks varētu korekti interpretēt notikušās izmaiņas. Izmaiņu apstrādes procedūras tika pārbaudītas visām atbalstītajām izmaiņām ar testa piemēriem, kas ir aprakstīti nodaļā 10.

Šajā nodaļā aprakstītās izmaiņu apstrādes procedūras nav realizētas automātiskajā vai automatizētajā rīkā. Lai pārbaudītu atskaišu izpildīšanu izstrādātajā atskaišu rīkā, tika apstrādātas fiziskās un loģiskās izmaiņas, bet šo izmaiņu apstrādes procedūru soļi tika izpildīti manuāli.

Daudzversiju datu noliktavas izmaiņu procedūras tika aprakstītas un publicētas rakstos:

Solodovņikova D. 'The Formal Model for Multiversion Data Warehouse Evolution'. *Postconference proceedings of the 8th International Baltic Conference on Databases and Information Systems, Tallinn, Estonia, Frontiers in Artificial Intelligence and Applications by IOS Press*, 2008.

Solodovņikova D. 'Metadata to Support Data Warehouse Evolution'. *Proceedings of the 17th International Conference on Information Systems Development, Paphos, Cyprus,* 2008.

8. VAICĀJUMU ĢENERĒŠANA DATU NOLIKTAVAI UN ATSKAIŠU IZMANTOŠANA

8.1. Nodaļas nolūks

Šajā nodaļā ir piedāvāta pieeja SQL vaicājumu būvēšanai, izmantojot iepriekš aprakstītos datu noliktavas shēmas un atskaišu metadatus, kā arī izveidotas atskaites lietošanas un datu analīzes iespējas. Šajā nodaļā aprakstītais vaicājumu būvēšanas algoritms tika realizēts promocijas darba ietvaros izstrādātajā atskaišu attēlošanas rīkā. Kad atskaite tiek izpildīta, vaicājumi tiek ģenerēti uz vienu vai vairākām datu noliktavas shēmas versijām saskaņā ar piedāvāto pieeju, kas ir aprakstīta šajā nodaļā.

8.2. Vaicājumu ģenerēšanas process

Lietotājs var *definēt* atskaiti ar atskaites attēlošanas rīku, izvēloties terminus vai terminu versijas no semantiskā līmeņa metadatiem. Atskaites definēšanas laikā atskaišu attēlošanas rīks ģenerē automātiski atskaites metadatus, atbilstoši lietotāja izveidotai atskaitei. Atskaites metadati glabā atskaites *definīciju*.

Kad lietotājs *izpilda* atskaiti ar atskaišu attēlošanas rīku, tiek prasīts ievadīt parametru vērtības, ja parametri iepriekš tika nodefinēti atskaitei atskaišu metadatos, kad atskaite tika definēta. Tālāk vaicājuma būvēšanas algoritms, balstoties uz datu noliktavas shēmas un atskaišu metadatiem un ievadītajām parametru vērtībām, ģenerē SQL vaicājumu un izpilda to.

Ģenerēto SQL teikumu struktūra ir redzama attēlā 30. Vaicājuma struktūrā ir izmantotas SQL standarta konstrukcijas.

```
SELECT      {DISTINCT} <izvēlēto elementu saraksts>
FROM        <kolonnu kopu saraksts>
{WHERE      (<savienojumu saraksts>)}
{AND        (<nosacījumu saraksts>)}
{GROUP BY   <grupēšanas elementu saraksts>}
{HAVING     <grupēšanas nosacījumu saraksts>}
{ORDER BY   <kārtošanas elementu saraksts>}
```

Att. 30. Ģenerēta SQL vaicājuma struktūra

Izvēlēto elementu saraksta elementi var būt dimensijas atribūti, faktu tanulas mērījumi, agregēti mērījumi, funkcijas, kuru argumenti ir dimensijas atribūti, vai funkcijas, kuru argumenti ir dimensijas atribūti un mērījumi. Kolonnu kopu saraksts ir atskaitē izmantoto tabulu, skatu vai apakšvaicājumu saraksts. Nosacījumu saraksts ir formula no nosacījumiem, kas ir saistīti ar AND un OR loģiskiem operatoriem. Grupēšanas elementu saraksts ir formula no nosacījumiem uz agregātfunkcijām, kas ir saistīti ar AND un OR loģiskiem operatoriem. Kārtošanas elementu saraksts sastāv no atskaitei izvēlētiem elementiem, kas tiek izmantoti, lai kārtotu atskaites datus. Izvēlēto elementu, kolonnu kopu, savienojumu, nosacījumu, grupēšanas elementu, grupēšanas nosacījumu un kārtošanas elementu sarakstu izveides apraksts tiek dots apakšnodaļās 8.2.1.-8.2.6.

Ja tas ir iespējams, SQL teikumi tiek sadalīti vairākos neatkarīgos teikumos, kas tiek izpildīti atsevišķi. SQL teikumu sadalīšana ir iespējama, ja elementi tiek atlasīti no dažādām tabulām, kas nav savā starpā savienotas un neeksistē nosacījumi, kas sasaista šīs tabulas.

SQL vaicājumu ģenerēšana notiek ar šajā nodaļā aprakstīto algoritmu vairākos soļos:

- izvēlēto elementu analīze un izmantoto tabulu noteikšana,
- savienojumu analīze,
- nosacījumu kopas ģenerēšana,
- grupēšana un nosacījumu būvēšana agregātfunkcijām,
- vaicājumu vienkāršošana un optimizācija,
- lietotāja tiesību ierobežojumu pievienošana,
- versiju analīze.

8.2.1. Izvēlēto elementu analīze un izmantoto kolonnu kopu noteikšana

Šajā solī tiek analizēti izvēlētie elementi, ko lietotājs atlasīja atskaitei to definējot.

1. Analizējot atskaišu metadatos izvēlētos elementus tabulā *Izvēlēts*, tiek iegūtas kalkulācijas, kas aprēķina konkrēto izvēlēto elementu. Katra kalkulācija satur kalkulācijas daļas, kas tiek izmantotas aprēķināšanas formulā AF, kas ir definēta atskaišu metadatu tabulā *Kalkulācija*. Tiek noskaidrots aprēķināšanas formulas veids:
   A. Neagregātfunkcija no vienas vai vairākām kolonnām – $F(i_1,i_2,...,i_n)$, kur $i_1...i_n$ ir indeksi kalkulācijas daļām, kas ir kolonnas, un F ir kaut kāda neagregācijas funkcija, kas arī varētu būt vienkārši viena kolonna;
   B. Agregātfunkcija no vienas vai vairākām kolonnām – $AGR(F(i_1,i_2,...,i_n))$, kur AGR ir kāda no agregātfunkcijām (SUM, COUNT, AVG, MIN, MAX), $i_1...i_n$ ir indeksi kalkulācijas daļām, kas ir kolonnas, un F ir kaut kāda neagregācijas funkcija, kas arī varētu būt vienkārši viena kolonna;
   C. Vairāku agregātfunkciju apvienojums ar neagregātfunkcijām – $G(AGR_1(F_1(i_1,i_2,...,i_n)),...,AGR_j(Fp(i_{n+1},i_{n+2},...,i_{n+m})),F_{p+1}(i_{n+m+1},i_{n+m+2},...,i_{n+m+k}))$, kur $AGR_1...AGR_j$ ir agregātfunkcijas, $i_1...i_n...i_{n+m}...i_{n+m+k}$ ir indeksi kalkulācijas daļām, kas ir kolonnas, $F_1...F_{p+1}$ un G ir neagregātfunkcijas.
2. Tālāk aprēķināšanas formulas, kas ir vairāku agregātfunkciju apvienojums ar neagregātfunkcijām, tiek pārveidotas postfiksa pierakstā. Tiek apstrādāts postfiksa pieraksts. Tiek apstrādāts katrs aprēķināšanas formulas arguments, sākot no postfiksa pieraksta beigām. Tiek atrastas formulas daļas, kas nesatur agregātfunkcijas un kas tiek izmantotas vaicājuma GROUP BY daļā.
3. Tālāk tiek apstrādāta katra kalkulācijas daļa. Atkarībā no kalkulācijas daļas tipa (kas ir iegūstams no kolonnas *Tips* tabulā *KalkulācijasDaļa*), katra kalkulācijas daļa tiek apstrādāta rekursīvā veidā sekojoši:
   3.1. Ja kalkulācijas daļa ir kolonna, tad
      3.1.1. Tiek pārbaudīts, vai šī kolonna eksistē datu bāzē, vai lietotājam ir tiesības uz šo kolonnu.

3.1.2. Ja tiesības ir lasīt informāciju no kolonnas, tad aprēķināšanas formulā AF apstrādātās kalkulācijas daļas indeksa vietā tiek ievietots kolonnas nosaukums ar kolonnu kopas pseidonīmu. Kolonnu kopai tiek ģenerēts pseidonīms, savienojot burtu T ar kolonnu kopas identifikatoru KOLONNU_KOPA_ID no fiziskā līmeņa metadatu tabulas KolonnuKopa.

3.1.3. Kolonna un tai atbilstošā kolonnu kopa tiek pievienotas izmantoto kolonnu sarakstam. Izmantoto kolonnu kopu sarakstā tiek pievienots pseidonīms tabulai, skatam vai vaicājumam, kas satur šo kolonnu.

3.2. Ja kalkulācijas daļa ir parametrs, tad aprēķināšanas formulā AF1 apstrādātās kalkulācijas daļas indeksa vietā tiek ievietota lietotāja ievadītā parametra vērtība.

4. Šajā solī arī tiek sastādīts atsevišķs saraksts ar A veida aprēķināšanas funkcijām un C veida aprēķināšanas funkciju neagregācijas daļām $F_{p+1}(i_{n+m+1},i_{n+m+2},...,i_{n+m+k})$. Šis saraksts tiks izmantots, lai būvētu grupēšanas elementu sarakstu algoritma solī, kas ir aprakstīts apakšnodaļā 8.2.4.

Izvēlēto elementu analīzes un izmantoto kolonnu kopu noteikšanas soļa rezultātā tiek iegūts saraksts ar izvēlētiem atskaites elementiem, kas sastāv no kolonnu un parametru nosaukumiem, kā arī no formulām, kas no kolonnām un parametriem aprēķina atskaites elementus. Šis saraksts tiek izmantots SQL vaicājuma SELECT sadaļā. Šajā sarakstā tiek iekļauti tikai tie elementi, kas nav paslēpti. Paslēptiem izvēlētiem elementiem atskaišu metadatu tabulā *Izvēlēts* ir pazīme kolonnā *Paslēpts*. Tas nozīmē, ka šādi paslēpti izvēlētie elementi ir izmantoti vai nu nosacījumos vai citu izvēlētu elementu aprēķināšanai.

Katra izvēlētā elementa SQL pieraksts tiek piesaistīts elementa identifikatoram no tabulas *Izvēlēts* no atskaišu metadatiem. Tas tiek izmantots tālākajos algoritma soļos.

Papildus šajā algoritma solī tiek iegūts saraksts ar tabulu, skatu un apakšvaicājumu nosaukumiem, kas tiks izmantots SQL vaicājuma FROM daļā. Šajā sarakstā viena kolonnu kopa var parādīties vairākas reizes ar dažādiem pseidonīmiem, ja atskaitē tiek izmantotas fiziskā līmeņa metadatu kolonnas no vairākām kolonnu kopām, kas definē vienu un to pašu tabulu, skatu vai apakšvaicājumu.

Šajā algoritma solī tiek izveidots arī kārtošanas elementu saraksts, kurā tiek iekļauti izvēlētie elementi tādā secība, kas atbilst katra izvēlētā elementa kārtošanas indeksam, kas tiek glabāts kolonnā *Kārtošanas indekss* atskaišu metadatu tabulā *Izvēlēts*.

Vēl atsevišķi tiek izveidots saraksts ar izmantotām kolonnām, kas tālāk tiek lietots, lai pievienotu vaicājumam lietotāja tiesību ierobežojumus algoritma solī, kas ir aprakstīts apakšnodaļā 8.2.6.

8.2.2. Savienojumu ģenerēšana

Nākošajā solī tiek analizēti iespējamie savienojumi starp tabulām, skatiem vai apakšvaicājumiem, kas tika iekļauti izmantoto kolonnu kopu sarakstā, kas tika iegūts iepriekšējā algoritma solī.

Savienojumi starp kolonnu kopām tiek glabāti tabulā Savienojums atskaišu metadatos. Šī tabula tiek aizpildīta, definējot atskaiti, automātiski no atslēgu informācijas no fiziskā līmeņa metadatiem. Arī lietotājs, kas veido atskaiti, var savienot kolonnu kopas manuāli, ja

tās nav savienotas fiziskajā līmenī, un var atvienot kolonnu kopas, t.i. neizmantot datu bāzē eksistējošo atslēgu. Šāda veida savienojumi arī ir saglabāti tabulā Savienojums.

1. Katram savienojumam tiek ģenerēts SQL pieraksts formā $t_1.k_1=t_2.k_2<(+)>$, kur t_1 un t_2 ir savienojamu kolonnu kopu nosaukumi vai pseidonīmi, k_1 un k_2 ir, respektīvi, kolonnu kopas t_1 un t_2 kolonnas. Ja savienojums ir ārējs, tad papildus tiek pielikts *(+)* simbols. (+) simbols nav SQL standarta apzīmējums ārējiem savienojumiem. Tas tika izmantots, lai atvieglotu ārējo savienojumu būvēšanu. Lai izveidotu ārējus savienojumus atbilstoši SQL standartam, ir savādāk jāģenerē izmantoto kolonnu kopu saraksts, sastādot to no kolonnu kopām, atdalītām ar atslēgas vārdiem t_1 LEFT | RIGHT OUTER JOIN t_2, un pievienojot izvēlēto kolonnu sarakstam arī savienošanas nosacījumus formā ON $t_1.k_1=t_2.k_2$.
2. Tālāk savienojumi tiek apvienoti ar AND loģiskiem operatoriem un tiek iegūta savienojumu formula.

Šī soļa rezultātā tiek iegūta savienojumu formula SQL pierakstā, kas tālāk tiek izmantota SQL vaicājuma WHERE sadaļā.

8.2.3. Nosacījumu kopas ģenerēšana

Šajā solī tiek analizēti nosacījumi tabulās *Nosacījums* un *NosacījumuKopa* atskaišu metadatos, ko ir definējis lietotājs. Tiek ģenerēts nosacījumu saraksts SQL pierakstā.

1. Vispirms tiek apstrādātas nosacījumu formulas, kas tiek glabātas kolonnā *Formula* tabulā *NosacījumuKopa* atskaišu metadatos. Šīs formulas tiek sadalītas divās grupās: nosacījumu formulas, kas satur nosacījumus ar agregātfunkcijām, un pārējās formulas. Šajā solī tālāk tiek apstrādātas tikai nosacījumu formulas, kas nesatur agregātfunkcijas. Pārējie nosacījumi tiek apstrādāti nākošajā algoritma solī, kas aprakstīts apakšnodaļā 8.2.4. Katrai formulai tiek iegūti nosacījumi no tabulas *Nosacījums* atskaišu metadatos.
2. Katram nosacījumam tiek ģenerēts SQL pieraksts E_1 *operators* E_2, kur E_1 un E_2 ir operandi, kuru tipi ir glabāti tabulā Nosacījums kolonnās *Objekta1Tips* un *Objekta2Tips* atskaišu metadatos, un *operators* ir nosacījuma operators, kas tiek iegūts no tabulas *Nosacījums* kolonnas *Operators.* Nosacījumi tiek ģenerēti ar sekojošu algoritmu.
   2.1. Ja nosacījuma operators ir *'IS NULL', 'IS NOT NULL', 'EXISTS'* vai *'NOT EXISTS'*, tad otrais izvēlētais elements E_2 netiek izmantots un nosacījuma pieraksts ir E_1 *IS NULL,* vai E_1 *IS NOT NULL,* vai *EXISTS* E_1*,* vai *NOT EXISTS* E_1.
   2.2. Pārējiem operatoriem tiek izmantots pieraksts E_1 *operators* E_2.
3. Tiek analizēti nosacījumā izmantotie operandi E_1 un E_2.
   3.1. Ja operands ir izvēlēts atskaites elements, tad tā SQL pierakstu var iegūt no apstrādātā izvēlētā elementa saraksta, kas tika iegūts 1. algoritma solī, kas tika aprakstīts apakšnodaļā 8.2.1.
   3.2. Ja operands ir parametrs, tad nosacījuma pierakstā tiek ievietota lietotāja ievadītā parametra vērtība.
   3.3. Ja operands ir konstante, tad tā vērtība tiek iegūta no tabulas *Nosacījums* kolonnas *Objekta1Vērtība* vai *Objekta2Vērtība.* Šī vērtība ir izmantota nosacījuma pierakstā.

3.4. Ja operands ir apakšvaicājums, tas nozīmē, ka tas jau ir SQL pierakstā un tiek izmantots bez izmaiņām. Apakšvaicājumam tiek pārbaudīts, vai lietotājam ir lasīšanas tiesības uz visām kolonnām un kolonnu kopām, kas ir izmantotas apakšvaicājumā.

4. Tālāk nosacījumi tiek apvienoti nosacījumu kopā, kas tālāk tiek izmantota SQL vaicājuma WHERE daļā. Tas tiek darīts sekojoši. Atskaišu metadatos nosacījumu formulās nosacījumu vietā ir glabāti nosacījumu indeksi. Lai iegūtu nosacījumu formulas SQL pierakstu, šie indeksi tiek nomainīti uz nosacījumu SQL pierakstiem.
5. Lai tālāk algoritma solī, kas ir aprakstīts apakšnodaļā 8.2.7., iegūtu datu noliktavas shēmas versijas, kas bija spēkā laika periodā, kurā izpilda atskaiti, tiek sastādīts saraksts ar nosacījumiem, kas satur laika dimensijas ierobežojumus.

Šī soļa rezultātā tiek iegūta nosacījumu kopa, kas ir apvienota nosacījumu formulā SQL pierakstā un tiek izmantojama SQL vaicājuma WHERE daļā.

8.2.4. Grupēšana un nosacījumu būvēšana agregātfunkcijām

Šajā solī atkal tiek analizēti izvēlētie elementi un tiek atdalīti elementi, kas veido grupēšanu, kā arī tiek analizēti nosacījumi, kas satur agregātfunkcijas.

1. Sākumā tiek apstrādātas A veida aprēķināšanas formulas un C veida aprēķināšanas formulu neagregācijas daļas, kas tika iegūtas izvēlēto elementu analīzes un izmantoto kolonnu kopu noteikšanas solī, kas tika aprakstīts apakšnodaļā 8.2.1. A veida aprēķināšanas formulas varētu tikt izmantotas kā grupēšanas elementi bez izmaiņām. No C veida aprēķināšanas formulām tiek izvēlētas neagregācijas daļas $F_{p+1}(i_{n+m+1},i_{n+m+2},...,i_{n+m+k})$, kas tiek izmantotas kā grupēšanas elementi SQL vaicājuma GROUP BY sadaļā.
2. Tālāk tiek apstrādāti nosacījumi ar agregātfunkcijām, kas tika atdalīti nosacījumu kopas ģenerēšanas solī. Katram nosacījumam tiek ģenerēts SQL pieraksts līdzīgi kā nosacījumu kopas ģenerēšanas solī, kas tika aprakstīts apakšnodaļā 8.2.3. Tālāk nosacījumi tiek apvienoti nosacījumu kopā sekojošā veidā. Nosacījumu formula tiek glabāta kolonnā *Formula* tabulā *NosacījumuKopa* atskaišu metadatos, kur nosacījumu vietā tiek glabāti nosacījumu indeksi. Lai iegūtu nosacījumu formulas SQL pierakstu, šie indeksi tiek nomainīti uz nosacījumu SQL pierakstiem. Iegūtā nosacījumu formula tiek izmantota vaicājuma HAVING sadaļā.

Šī soļa rezultātā tiek iegūts saraksts ar grupēšanas elementiem, kas tiek izmantots SQL vaicājuma GROUP BY daļā, un grupēšanas nosacījumu formula, kas tiek izmantota SQL vaicājuma HAVING daļā.

8.2.5. Lietotāja tiesību ierobežojumu pievienošana

Šajā solī SQL vaicājums tiek papildināts ar ierobežojumiem, kas kontrolē lietotāja tiesības uz izmantotajām kolonnām un kolonnu kopām.

Tiek apstrādāts vaicājumā izmantoto kolonnu saraksts, kas tika iegūts izvēlēto elementu analīzes un izmantoto kolonnu kopu noteikšanas solī. Katrai kolonnai tiek meklēti

tiesību ierobežojumi konkrētam lietotājam tabulā *Tiesības* fiziskajā metadatu līmenī. Ja lietotājam ir datu ierobežojums uz šo kolonnu, tad tas ir ierakstīts fiziskā līmeņa metadatu tabulā *Tiesības* kolonnā *Nosacījums*. Šis nosacījums ir izteiksme ar vienas kolonnu kopas kolonnām, kur kolonnu nosaukumi ir formā *T||setid.colname*, kur *setid* ir kolonnu kopas identifikators un *colname* ir kolonnas nosaukums. Šis nosacījums tiek pievienots SQL vaicājuma WHERE daļai.

8.2.6. Vaicājumu vienkāršošana un optimizācija

Šajā solī notiek iegūtā SQL vaicājuma vienkāršošana, lai paātrinātu vaicājuma izpildi, kā arī vaicājuma papildināšana ar konstrukcijām, kas iegūst papildus datus, lai varētu ātri veikt OLAP operācijas ar atskaites datiem.

1. Izvēlētie elementi un grupēšanas elementi, kas atkārtojas, tiek atfiltrēti. Tiek analizēti nosacījumi un savienojumi, kur arī tiek izmesti nosacījumi, kas atkārtojas, ja tie nemaina vaicājuma rezultātu.
2. Datu noliktavas atskaitēm jāatbalsta mijiedarbība ar lietotāju, t.i. OLAP operācijas: agregācija (rollup), detalizācija (drill-down), sagriešana (slicing). Veicot šādu operāciju, lietotājs vēlētos iegūt datus analīzei pēc iespējas ātrāk. Tādēļ vēl ir iespējams izmantot ROLLUP un GROUPING SETS SQL paplašinājumus [GCB+97], lai iegūtu rezultātus, kas tiks izmantoti, ja lietotājs veiks rollup, drill-down un slicing operācijas.
    2.1. Paplašinājums ROLLUP grupē izvēlētos ierakstus, balstoties uz pirmo *n, n-1,...,0* grupēšanas izteiksmju vērtībām un atgriež agregācijas vērtības katrai grupai.
        2.1.1. Tas tiek izmantots SQL vaicājuma GROUP BY sadaļā, ja lietotājs ir izvēlējies mērījumus ar agregātfunkcijām un vienas vai vairāku dimensiju atribūtus, kas piedalās kādā hierarhijā loģiskajā metadatu līmenī. Saistība ar loģisko metadatu līmeni notiek caur tabulu KalkulācijasDaļa. Tad katrai hierarhijai, kurā piedalās katrs izvēlētais atribūts, tiek konstruēts teikums $ROLLUP(A_1,A_2,...,A_n)$, kur $A_1,...,A_n$ ir atribūti, kas veido noteiktas hierarhijas līmeņus ar pieaugušo detalizācijas pakāpi.
        2.1.2. Vairāki ROLLUP teikumi tiek apvienoti vaicājuma GROUP BY sadaļā kopā ar pārējiem izvēlētiem grupēšanas elementiem, kas nav dimensiju atribūti vai nepiedalās nevienā hierarhijā: *GROUP BY* $ROLLUP(A_1,A_2,...,A_n)$, $ROLLUP(B_1,B_2,...,B_k)$,..., C_1, C_2,...C_m, kur $A_1,...,A_n$ un $B_1,...,B_k$ ir atribūti, kas veido noteiktas hierarhijas līmeņus, un $C_1,...,C_m$ ir pārējie atribūti vai izrēķinātas vērtības.
        2.1.3. Tad vaicājuma SELECT daļā tiek pievienoti arī pārējo katras hierarhijas līmeņu atribūti, uz kuriem lietotājam ir tiesības. Šāda vaicājuma rezultātā tiek iegūtas mērījumu agregācijas vērtības katram dimensijas hierarhijas līmenim, un ja lietotājs veic agregācijas (rollup) vai detalizācijas (drill-down) operācijas, šo operāciju rezultāts var tikt iegūts bez savienošanās ar datu bāzi.
    2.2. Paplašinājums GROUPING SETS ļauj specificēt vairākas grupēšanas kopas.
        2.2.1. Tas tiek izmantots SQL vaicājuma GROUP BY sadaļā, ja lietotājs ir izvēlējis vienu vai vairākus elementus, kas tiek attēloti atskaitē kā lappušu

objekti (page items). Lappušu objektos parādās visas konkrēta elementa vērtības un vēl vērtība „Viss kopā". Skatoties atskaiti, lietotājs var izvēlēties katra lappušu objekta vienu vērtību vai vērtību „Viss kopā".

2.2.2. Lai iegūtu datus katrai iespējamai šo vērtību kombinācijai, tiek būvēta grupu kopa, kur katra grupa atbilst kādai lappušu objektu kombinācijai, t.i. ja atskaitē ir *n* lappušu objekti, tad tiek veidotas 2^n grupas.

2.2.3. Papildus katrā grupā tiek salikti visi pārējie izvēlētie grupēšanas elementi, kas netiek attēloti atskaitē kā lappušu objekti.

2.2.4. Šāda vaicājuma rezultātā tiek iegūtas mērījumu agregācijas vērtības katrai lappušu objektu vērtību kombinācijai, un ja lietotājs izvēlas kāda lappušu objekta citu vērtību, atskaites datus var iegūt bez savienošanās ar datu bāzi.

2.3. Ja vaicājumā tika pievienoti paplašinājumi ROLLUP vai GROUPING SETS, tad vaicājuma SELECT daļa arī tiek papildināta ar DISTINCT operatoru.

8.2.7. Versiju analīze

Šajā solī tiek analizēti atskaitei izvēlētie elementi no atskaišu metadatu tabulas *Izvēlēts,* un katram elementam tiek iegūts saraksts ar datu noliktavas versijām, kad šis elements bija spēkā. Tas tiek darīts, lai paziņotu lietotājam par gadījumu, kad atskaite tiek izpildīta uz vairākām shēmas versijām.

Versiju izvēle

1. Sākumā tiek analizēti dimensiju atribūti un mērījumi, kas ir piesaistīti kalkulācijas daļām. Katram atribūtam un mērījumam tiek meklētas visas versijas, kad tas bija spēkā, caur tabulu *VersijasTransformācija* loģiskā līmeņa metadatos. Tālāk no visu versiju saraksta tiek atfiltrētas tās versijas, kas nebija spēkā laika periodā, uz kuru izpilda atskaiti, ja tas ir nodefinēts nosacījumos uz laika dimensiju, kas bija apkopoti atsevišķā sarakstā nosacījumu kopas ģenerēšanas solī.
2. Ja pēc atfiltrēšanas paliek tikai viena versija, tad vaicājuma rezultāti tiek attēloti atskaitē visi kopā. Ja paliek vairākas versijas, kas bija spēkā atskaites laika periodā, tad lietotājam tiek piedāvāts izvēlēties vienu no datu attēlošanas veidiem, kas ir iespējami konkrētai atskaitei. Atkarībā no dažādu elementu esamības shēmas versijās ir iespējami trīs atskaites datu attēlošanas veidi:
   A. Attēlot atskaites datus saskaņā ar vienu konkrēto shēmas versiju;
   B. Attēlot elementus no vairākām shēmas versijām vienā atskaitē;
   C. Attēlot datus vairākās atskaitēs atsevišķi katrai versijai.

   Var izvēlēties attēlot datus saskaņā ar vienu versiju, ja šajā versijā eksistē visi mērījumi un atribūti, kas ir izmantoti atskaitē, un visās pārējās versijās, kuru spēkā būšanas periodi pārklājas ar atskaites laika dimensijas ierobežojumu, vai nu eksistē atskaitei izvēlētie shēmas elementi, vai tos var iegūt no pārējiem šīs versijas shēmas elementiem ar versijas transformācijām.

   Ja nav iespējams attēlot datus saskaņā ar vienu konkrēto versiju tādēļ, ka neeksistē kāda versija, kas satur visus atskaitē izmantotos shēmas elementus, tad lietotājam var piedāvāt attēlot elementus no vairākām versijām vienā atskaitē, kad katrs shēmas elements vai nu

eksistē katrā versijā vai to var dabūt ar versijas transformācijām no pārējiem shēmas elementiem, t.i. nav situācijas, kad elements vispār nekādā veidā nav iegūstams kādā brīdī atskaites laika periodā.

3. Lai noskaidrotu, vai ir iespējams attēlot atskaiti saskaņā ar kādu versiju vai apvienot elementus no vairākām versijām, tiek būvēta versiju un elementu attiecību tabula, kur tabulas kolonnas ir versijas un rindas ir elementi, bet tabulas šūnās tiek saglabāti veidi, kā var iegūt elementu versijā.
    3.1. Ja elements eksistē versijā (tiek iegūts ar versijas transformāciju bez jebkādas konvertācijas), tad attiecīgajā tabulas šūnā tiek ierakstīts 1.
    3.2. Ja elements neeksistē versijā, bet iegūstams ar versijas transformāciju ar konvertācijas funkciju no pārējiem elementiem, tad attiecīgajā tabulas šūnā tiek ierakstīts 2.
    3.3. Ja elements neeksistē versijā un nav iegūstams ar kādu konvertācijas funkciju (nav versijas transformācijas), tad attiecīgajā tabulas šūnā tiek ierakstīta 0.
4. Versiju un elementu attiecību tabulas šūnu vērtības tiek analizētas un tiek iegūti iespējamie atskaites attēlošanas veidi.
    4.1. Atskaiti ir iespējams attēlot saskaņā ar kādu versiju, ja visās tabulas versijai atbilstošās kolonnas šūnās ir ierakstīti 1 un pārējās tabulas šūnās ir vai nu 1 vai 2.
    4.2. Ja neviena versija nesatur visus elementus, tad elementus no vairākām versijām ir iespējams attēlot vienā atskaitē, ja visās tabulas šūnās ir ierakstīti 1 vai 2.
    4.3. Ja kādā tabulas šūnā ir ierakstīta 0, tad atskaiti var attēlot tikai atsevišķi katrai versijai.

Piemēram, tabulā 8. ir attēlots versiju un elementu attiecību tabulas piemērs, kur ir redzams, ka atskaiti ir iespējams attēlot saskaņā ar versiju 1. vai versiju 2, jo šīm versijām atbilstošajās kolonnās ir ierakstīti 1, t.i. visi atskaitē izmantotie elementi eksistē šajās divās versijās, un pārējās kolonnās ir ierakstīti 1 vai 2, t.i. atskaitē izmantotie elementi eksistē pārējās versijās vai tos ir iespējams iegūt no pārējiem versijās eksistējošiem elementiem ar konvertācijas funkcijām.

Tabula 8. Versiju un elementu attiecību tabulas piemērs

		Versija ID				
		1	***2***	***3***	***4***	***5***
Elements ID	***1***	1	1	1	1	2
	2	1	1	1	2	2
	3	1	1	1	1	1
	4	1	1	2	2	2
	5	1	1	1	1	1
	6	1	1	2	2	2

Dažādo vaicājumu ģenerēšana atsevišķajām versijām

Ja lietotājs ir izvēlējies attēlot datus atsevišķi katrai versijai, tad iepriekšējos soļos iegūtais SQL vaicājums tiek izmantots katrai versijai ar sekojošām izmaiņām:

1. tiek pievienots nosacījums, kas ierobežo laika dimensiju ar periodu, kad konkrēta versija bija spēkā;
2. no vaicājuma tiek izmesti izvēlētie elementi, kas izmanto kalkulācijas daļas, kas ir vienādas ar atribūtiem vai mērījumiem, kas neeksistē konkrētā versijā.

Katra SQL vaicājuma rezultāti tiek attēloti lietotājam atsevišķā tabulā kopā ar versijas metadatiem (versijas nosaukumu un derīguma periodu).

Viena vaicājuma ģenerēšana vairākām versijām

Ja lietotājs ir izvēlējies attēlot elementus no vairākām versijām $V_1 \ldots V_n$ vienā atskaitē, tad katrai no versijām V_i, kas bija spēkā atskaites laika periodā, tiek būvēts vaicājums Q_i saskaņā ar algoritmu, kura soļi ir aprakstīti apakšnodaļās 8.2.1.-8.2.6.

1. Katram vaicājumam Q_i tiek pievienots laika dimensijas ierobežojums, kas atbilst atbilstošās versijas V_i spēkā būšanas periodam.
2. Katram elementam (mērījumam vai atribūtam), kas ir izmantots atskaitē, tiek pārbaudīts, vai tas ir iekļauts kārtējā shēmas versijā V_i.
    2.1. Ja elements eksistē versijā (ir piesaistīts shēmas versijai V_i caur versijas transformāciju bez konvertācijas funkcijas), tas tiek iekļauts vaicājumā Q_i bez pārveidošanas.
    2.2. Ja elements nav iekļauts versijā, tad tas ir iegūstams ar versijas transformāciju no citiem elementiem, kas eksistē versijā V_i. Šajā gadījumā mainītai dimensijai vai faktu tabulai tiek būvēts apakšvaicājums, kas adaptē mainītu dimensiju vai faktu tabulu tā, lai tā saturētu elementu. Apakšvaicājumā mainītie elementi tiek pierakstīti kā versijas transformācijas funkcijas no citiem shēmas elementiem, kas eksistē kārtējā versijā V_i. Ja mainās kāda mērījuma detalizācija, tad apakšvaicājumam tiek pielietota grupēšana un nosacījumu būvēšana agregātfunkcijām, ņemot vērā pieļaujamās agregācijas, kas ir definēts tabulā *PieļaujamaAgregācija* loģiskā līmeņa metadatos. Ja elementa izrēķināšanai ir nepieciešama papildus kolonnu kopa, kas nebija sākotnēji izmantota, tad tā tiek ielikta FROM sadaļā apakšvaicājumā un WHERE apakšvaicājuma sadaļa tiek papildināta ar atbilstošiem savienojuma nosacījumiem. Iegūtais SQL apakšvaicājums tiek iekļauts vaicājumā Q_i FROM daļā atbilstošās tabulas nosaukuma vietā, t.i. tiek izmantots kā apakšvaicājums, kas definē tabulu, no kuras tiek atlasīts konkrēts shēmas elements.
3. Beigās visi iegūtie vaicājumi $Q_1 \ldots Q_n$ tiek apvienoti ar UNION operatoru. Visu iegūto vaicājumu apvienojuma rezultāts tiek attēlots lietotājam.

Viena vaicājuma ģenerēšana vienai versijai

Ja lietotājs ir izvēlējies attēlot atskaiti saskaņā ar vienu no versijām, piemēram, V_i, tad sākumā SQL vaicājums Q_i tiek sastādīts balstoties uz versiju V_i, kas satur visus versijas elementus.

1. Vaicājumam Q_i tiek pievienots laika dimensijas ierobežojums, kas atbilst versijas V_i spēkā būšanas periodam.

2. Tālāk tiek uzkonstruēti atsevišķi vaicājumi katrai no versijām, kas bija spēkā atskaites laika periodā. Pieņemsim, ka vaicājumi $Q_1...Q_i...Q_n$ atbilst versijām $V_1...V_i...V_n$. Tie tiek uzkonstruēti sekojošā veidā:
    2.1. Vaicājums Q_{i-1} ($2 \le i \le n+1$) ir uzkonstruēts no vaicājuma Q_i, kam tiek pievienots laika dimensijas ierobežojums ar versijas V_{i-1} spēkā būšanas periodu.
    2.2. Tiek atrasti elementi, kas mainījās no versijas V_{i-1} līdz versijai V_i un kas tiek izmantoti atskaitē. Katram šim elementiem tiek meklēta versijas transformācija, kas ļauj iegūt elementu no pārējiem versijas elementiem. Ja elementam eksistē versijas transformācija no versijas V_{i-1} elementiem, tad tiek būvēts SQL vaicājums *SQ*, kas pārveido dimensiju vai faktu tabulu, kas satur elementu, no versijas V_{i-1} uz V_i. Šajā vaicājumā *SQ* mainītie elementi tiek pierakstīti kā versijas transformācijas funkcijas no citiem elementiem, kas eksistē versijā V_{i-1}. Ja mainās kāda mērījuma detalizācija, tad vaicājumam *SQ* tiek pielietota grupēšana un nosacījumu būvēšana agregātfunkcijām, ņemot vērā pieļaujamās agregācijas, kas ir definētas tabulā *PieļaujamaAgregācija* loģiskā līmeņa metadatos. Ja elementa izrēķināšanai ir nepieciešama papildus kolonnu kopa, kas nebija sākotnēji izmantota, tad tā tiek ielikta FROM sadaļā vaicājumā *SQ* un WHERE sadaļa tiek papildināta ar savienojuma nosacījumiem. Iegūtais SQL vaicājums *SQ* tiek iekļauts vaicājumā Q_{i-1} FROM daļā atbilstošas tabulas nosaukuma vietā, t.i. tiek izmantots kā apakšvaicājums, kas definē tabulu, no kuras tiek atlasīts konkrēts elements.
3. Pārveidošanas rezultātā iegūtais SQL vaicājums Q_{i-1} atgriež datus saskaņā ar versiju V_i. Šis vaicājums tiek apvienots ar vaicājumu Q_i ar UNION. Līdzīgi no vaicājuma Q_i tiek konstruēti arī pārējie vaicājumi un apvienoti ar UNION ar Q_i. Tā turpinot procesu tiek apvienoti visi vaicājumi $Q_1...Q_i...Q_n$. Visu iegūto vaicājumu apvienojuma rezultāts tiek attēlots lietotājam.

8.3. Atskaites datu analīze

8.2.8. Hierarhiju versiju izmantošana

Datu noliktavas atskaitēm ir jābūt interaktīvām, t.i. ir jāatbalsta dažādas OLAP operācijas, piemēram, agregācija (rollup), detalizācija (drill-down), u.c. Tādēļ evolūcijas arhitektūras atskaišu rīkam jāvar attēlot datus atskaitē dažādos veidos saskaņā ar lietotāju vajadzībām. Viens no veidiem kā varētu analizēt datus, ir izmantot hierarhijas, kas ir definētas loģiskajos metadatos. Saskaņā ar piedāvāto metamodeli, var tikt izveidotas vairākas versijas hierarhijām, līmeņiem un asociācijām starp atribūtiem un hierarhiju līmeņiem. Kad lietotājs izpilda atskaiti, vaicājuma būvēšanas algoritms noskaidro visas hierarhijas un to struktūru, kas pastāv konkrētajā datu noliktavas versijā, ko ir izvēlējies lietotājs, lai attēlotu atskaites datus. Ja lietotājs ir izvēlējies apvienot elementus no dažādām versijām vienā atskaitē, tad ir pieejamas tikai hierarhijas, kas pastāv visās versijās, kā arī hierarhijas, kas sastāv no atribūtiem, kas vai nu eksistē versijās vai var būt iegūti ar versijas transformācijām no eksistējošiem atribūtiem.

8.2.9. Terminu versiju izmantošana

Datu noliktavas evolūcijas arhitektūras atskaitēs terminu versijas tiek izmantotas, lai atšķirtu viena un tā paša shēmas elementa dažādas nozīmes. Ja kādam shēmas elementam ir vairākas terminu versijas, kad lietotājs izpilda atskaiti, kas satur šo shēmas elementu, tad viņam tiek paziņots, ka tam pašam shēmas elementam eksistē divas terminu versijas. Lietotājam ir jāizvēlas vēlamā termina versija un tad atbilstoša shēmas elementa versija tiek iekļauta atskaitē.

8.4. Nodaļas secinājumi

Šajā nodaļā tika izklāstīts vaicājumu ģenerēšanas algoritms, kas būvē SQL vaicājumus uz vienu vai vairākām datu noliktavas shēmas versijām. Piedāvātais algoritms ir viens no šī darba galvenajiem jauninājumiem. Algoritms sastāv no 7 soļiem un balstās uz atskaišu definīcijām metadatos un uz datu noliktavas shēmas metadatiem. Šis algoritms ļauj lietotājiem izvēlēties dažādus datu attēlošanas veidus, ja atskaite tiek izpildīta uz vairākām datu noliktavas shēmas versijām. Šie datu attēlošanas veidi ir: attēlot atskaites datus saskaņā ar vienu konkrēto shēmas versiju, attēlot elementus no vairākām shēmas versijām vienā atskaitē un attēlot datus vairākās atskaitēs atsevišķi katrai versijai.

Lai nodrošinātu atskaites datu analīzi, izmantojot OLAP operācijas (agregāciju, detalizāciju un sagriešanu), algoritms papildina ģenerēto vaicājumu ar SQL paplašinājumiem, kas izrēķina dažādu iespējamo OLAP operāciju rezultātu datus, baltoties uz datu noliktavas shēmas metadatos definētajām hierarhijām un atskaites elementu izvietojumu.

Piedāvātais algoritms konstruē vaicājumu atbilstoši SQL standartam, kas nozīmē, ka algoritmu var realizēt jebkurā relāciju datu bāzē, kas atbalsta SQL standartu. Konkrētajā algoritma realizācijā istrādātajā atskaišu attēlošanas rīkā izmanto (+) simbolu ārējo savienojumu konstruēšanai, lai vienkāršotu algoritma darbību, bet algoritma apraksts satur arī darbības, kas jāveic, lai uzkonstruētu ārējus savienojumus atbilstoši SQL standartam.

Aprakstītais algoritms tika realizēts vaicājumu definēšanas un attēlošanas rīkā, kas atbalsta daudzversiju datu noliktavas. Algoritma darbības piemērs tiek apskatīts nodaļā 10.

Atskaišu definēšanas un vaicājuma konstruēšanas process uz vairākām datu noliktavas shēmas versijām, izmantojot datu noliktavas metadatus, tika aprakstīts un publicēts rakstā:

Solodovņikova D. 'Building Queries on Multiple Versions of Data Warehouse'. *Proceedings of the 8th International Baltic Conference on Databases and Information Systems, Tallinn, Estonia*, 2008.

9. ATSKAIŠU RĪKA TEHNISKĀ REALIZĀCIJA

9.1. Nodaļas nolūks

Šīs nodaļas nolūks ir aprakstīt iepriekšējās nodaļās izklāstīto teorētisko ideju aprobāciju, pielietojot tās atskaišu definēšanas un attēlošanas rīka realizācijā. Šajā nodaļā ir aprakstītas dažādo ar atskaišu definēšanas un attēlošanas rīka tehnisko realizāciju saistīto problēmu risinājumi. Tiek dotas detalizētas prasības atskaišu definēšanas un attēlošanas rīkam, kas tika izstrādāts darba ietvaros. Tiek apskatīta rīka realizācijai izmantotā arhitektūra un tehnoloģijas. Tiek aplūkoti arī atskaites noformējuma iespējas un realizācija.

9.2. Prasības atskaišu rīkam

Atskaišu rīks sastāv no *atskaišu attēlošanas rīka*, ar ko lietotāji veido un izpilda atskaites uz vienas vai vairākām datu noliktavas shēmas versijām, un *atskaišu definēšanas rīka*, ar ko izstrādātājs pārvalda datu noliktavas un atskaišu metadatus un definē lietotāju tiesības uz datiem atskaitēs un uz atskaitēm.

9.2.1. Atskaišu attēlošanas rīka prasības

Atskaišu attēlošanas rīkā jābūt iespējām:

- definēt datus, kas tiek attēloti atskaitēs,
- definēt atskaites datu izvietojumu (kolonnas, rindas, lappušu objekti (page items),
- definēt nosacījumus datiem,
- parametrizēt atskaites,
- veidot aprēķinātas vērtības (kalkulācijas) no izvēlētiem datiem,
- veidot kopsavilkumus kolonnām, rindām,
- nodrošināt datu attēlošanu grafikā,
- nodrošināt datu analīzi, izmantojot hierarhijas,
- nodrošināt datu eksportu uz MS Excel,
- vākt statistiku par rīka lietošanu,
- nodrošināt lietotāju pieejas tiesības atskaitēm un datiem,
- ģenerēt SQL vaicājumus vienai vai vairākām datu noliktavas shēmas versijām atbilstoši lietotāju ievadītajām paramteru vērtībām, atskaišu metadatiem un datu noliktavas shēmas metadatiem, ko sagatavo atskaišu definēšanas rīks, kas savukārt baltīts uz promocijas darbā piedāvāto metadatu modeli,
- nodrošināt dažādus atskaites datu attēlošanas veidus, atkarībā no shēmas versiju laika ierobežojumiem un shēmas elementu esamības datu noliktavas shēmas versijās,
- attēlot SQL vaicājumu tekstus, kas tiek izpildīti, lai iegūtu datus atskaitei,
- definēt tiesības lietotājiem uz atskaitēm.

Atskaišu attēlošanas rīkam jābūt platformneatkarīgam, t.i. lietotājiem jāvar izmantot atskaišu rīku ar dažādām operētājsistēmām un pārlūkprogrammām.

9.2.2. Atskaišu definēšanas rīka prasības

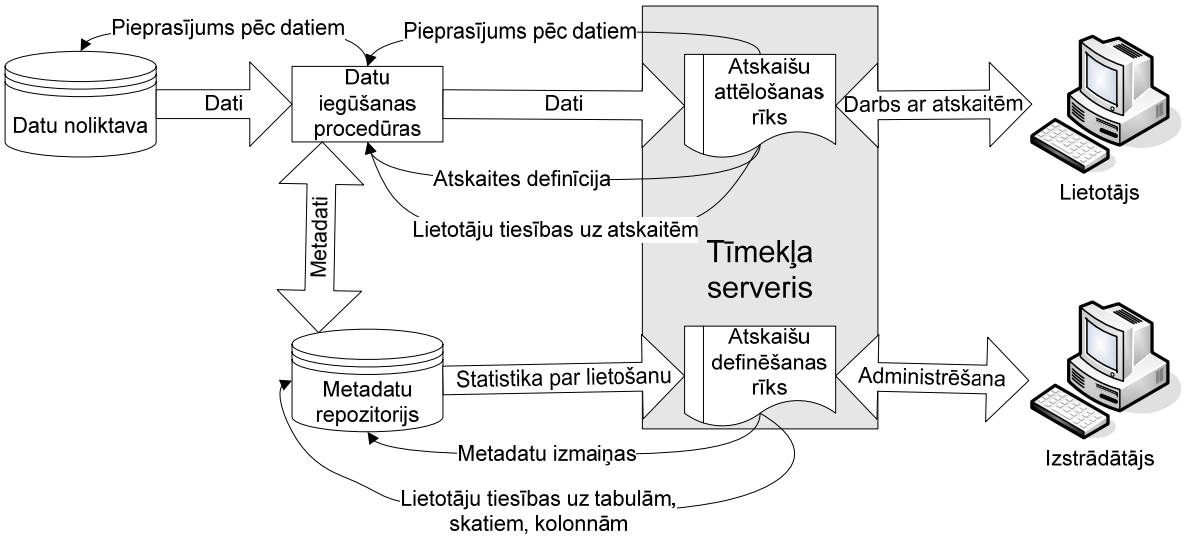
Atskaišu definēšanas rīkam jābūt iespējām:

- veidot, mainīt un dzēst metadatus, kas apraksta datu noliktavas shēmas,
- definēt tiesības lietotājiem uz noteiktām tabulām, kolonnām, datiem,
- aplūkot statistiku par atskaišu rīka lietošanu.

Rīkam jābūt metadatu repozitorijam saskaņā ar darba ietvaros izstrādāto datu noliktavas metadatu modeli, kur tiek glabāti datu noliktavas shēmu apraksti, atskaišu definīcijas u.c.

9.3. Atskaišu rīka arhitektūra

Atskaišu rīkam tiek izmantota trīs slāņu arhitektūra, kas ir redzama attēlā 31. Rīka arhitektūra sastāv no servera ar relāciju datu bāzi datu noliktavas datu un metadatu glabāšanai, datu iegūšanas procedūrām, kuras pārvalda atskaišu un datu noliktavas shēmas metadatus, atskaišu attēlošanas un definēšanas rīkiem, kas atrodas uz tīmekļa servera. Rīka realizācijā aprobācijas projekta ietvaros tika izmantota relāciju datu bāzes pārvaldības sistēma Oracle. Datu iegūšanas procedūras tika realizētas ar PL/SQL procedūrām. Atskaišu rīku izvietošanai tika izmantots tīmekļa serveris Tomcat. Atskaites definēšanas un attēlošanas rīki tika veidoti kā Java serversīklietotnes, kuras ģenerē HTML kodu, ko ir iespējams lietot pārklūkprogrammās bez papildus programmatūras instalācijas. Atskaišu grafiskai attēlošanai tika izmantots atvērtā koda atskaišu dzinējs JasperReports.



Att. 31. Atskaišu rīka arhitektūra

9.3.1. Metadatu pārvaldība

Izstrādātajā atskaišu rīkā, atbilstoši apakšnodaļā 9.2.2. aprakstītajām prasībām, datu noliktavas shēmas metadati tiek pārvaldīti ar atskaišu definēšanas rīku, kas ir atsevišķa lietojumprogramma. Atskaišu definēšanas rīku izmanto datu noliktavas izstrādātāji, kas definē

jaunu datu noliktavas shēmu, jaunu shēmas versiju vai veic izmaiņas datu noliktavas shēmas metadatos. Visi jauni datu noliktavas shēmas metadati tiek definēti un izmaiņas esošajos metadatos tiek veiktas atskaišu definēšanas rīkā un saglabātas metadatu repozitorijā.

Ar atskaišu definēšanas rīku varētu apskatīt arī statistiku par atskaišu veidošanu un lietošanu.

Tika izstrādāts atskaišu definēšanas rīka prototips, kas ļauj pārvaldīt datu noliktavas shēmas metadatus, bet nerāda statistiku par atskaišu lietošanu, kaut arī šī statistika tiek automātiski vākta un varētu būt apskatāma pēc atskaišu definēšanas rīka pilnas funkcionalitātes realizācijas. Atskaišu definēšanas rīka ievadformas piemērs ir redzams attēlā 20. Šī rīka funkcionalitāti realizēja pēc promocijas darba autores projektējuma saskaņā ar prasībām, kas tika aprakstītas apakšnodaļā 9.2.2.

9.3.2. Atskaites definēšana

Izstrādātajā atskaišu attēlošanas rīkā atskaites definē izstrādātāji un lietotāji, kam ir tiesības veidot vai rediģēt atskaites. Lietotāju tiesības uz atskaitēm tiek definētas atskaišu metadatos (skatīt apakšnodaļu 6.3.4.). Kad atskaite tiek definēta vai modificēta, tad atskaišu attēlošanas rīks ģenerē atskaites jauno aprakstu XML formātā un sūta to datu iegūšanas procedūrai, kas saglabā jauno atskaites definīciju atskaišu metadatos.

Atskaitei arī tiek definētas lietotāju tiesības atbilstoši atskaišu metadatiem. Lietotāju tiesības definē atskaišu rīka administrators vai lietotāji, kam ir tiesības mainīt citu lietotāju tiesības.

9.3.3. Atskaites izpildīšana

Kad lietotājs izpilda atskaiti ar atskaišu attēlošanas rīku, atskaišu attēlošanas rīks noformē pieprasījumu pēc datiem XML formātā un aizsūta to datu iegūšanas procedūrām. Pieprasījums pēc datiem satur informāciju par to, kāda atskaite ir jāizpilda un kādas parametru vērtības tika ievadītas, ja atskaite ir parametrizēta.

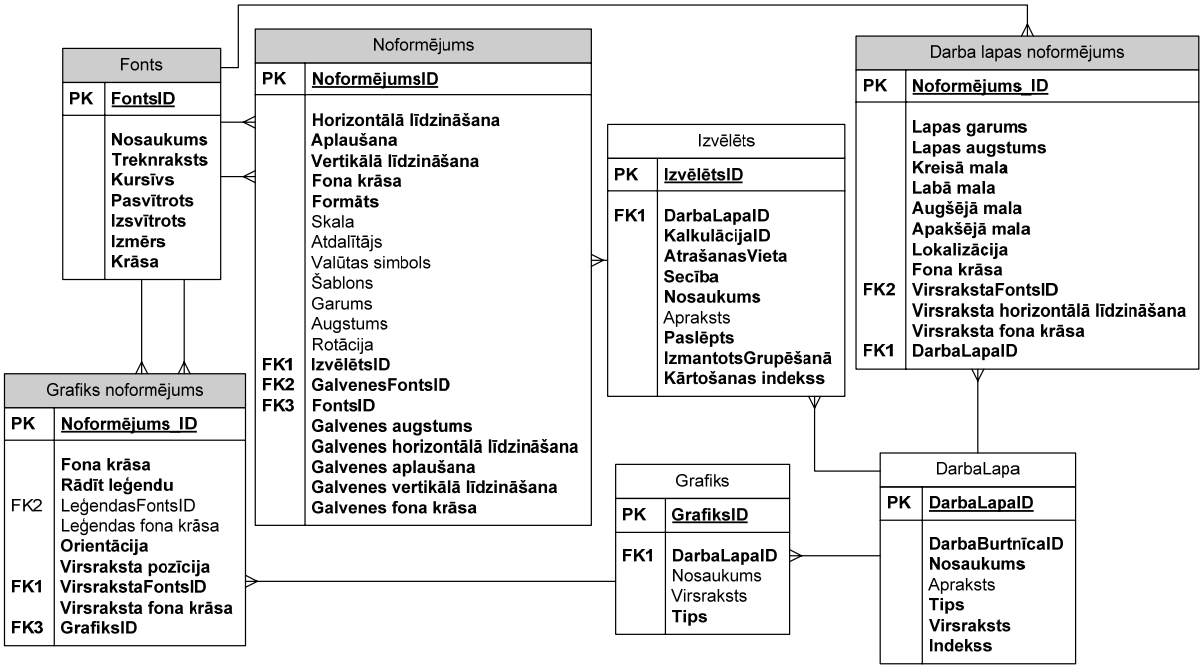
Datu iegūšanas procedūras atrod atskaites definīciju atskaišu metadatos. Izmantojot atskaišu metadatus un saistības starp fizisko un loģisko datu noliktavas shēmas metadatu līmeņiem, datu iegūšanas procedūras pārformulē pieprasījumu pēc datiem SQL vaicājumā atbilstoši vaicājuma konstruēšanas algoritmam, kas tika aprakstīts nodaļā 8.

Ģenerētais SQL vaicājums tiek izpildīts datubāzē un tiek atgriezti dati atskaišu attēlošanas rīkam, kas apstrādā datus un sagatavo atskaiti.

Lietotājiem ir iespēja analizēt datus sagatavotajās atskaitēs, izmantojot OLAP operāciju - sagriešanu (slicing), ja atskaitei tika nodefinēti izvēlētie elementi, kas tiek attēloti kā lappušu objekti (page items). Hierarhiju grafiskā attēlošana pagaidām nav realizēta, tāpēc pagaidām atskaišu attēlošanas rīka funkcionalitātē nav nodrošinātas OLAP operācijas agregācija (roll-up) un detalizācija (drill-down).

9.4. Atskaites noformējuma metadati

Ja lietotājs vēlas mainīt noklusēto atskaites vai atskaites elementa noformējumu, tad metadati par to tiek reģistrēti sekojošajā struktūrā (attēls 32.). Ja tiek izmantots noklusētais noformējums, tad metadatos nav atbilstoša noformējuma ieraksta.



Att. 32. Atskaites noformējuma metadati

Izvēlēta elementa noformējums

Tabulā *Noformējums* tiek veidots ieraksts katram izvēlētam atskaites elementam. Tiek saglabāti metadati par izvēlētā elementa izskatu, piemēram, līdzināšana, aplaušana, fona krāsa, formāts (iespējamās vērtības: skaitlis, valūta, datums, laiks, procenti, teksts, lietotāja definēts formāts), skala, atdalītājs, valūtas simbols, šablons (gatavs formāta šablons, tiek izmantots formātiem: datums, laiks, lietotāja definēts formāts), kolonnas garums, rindas augstums, rotācija. Ja izvēlētam elementam ir galvene, tad tās noformējums arī tiek glabāts tabulā *Noformējums*, ir iespējams noformēt galvenes augstumu, līdzināšanu, aplaušanu un fona krāsu.

Fonta iestatījumi

Izvēlētiem atskaites elementiem un šo elementu galvenēm ir iespējams nodefinēt arī konkrētu fontu. Fonta iestatījumi tiek glabāti tabulā *Fonts*, un piesaistīti tabulai *Noformējums* caur kolonnām FontsID un GalvenesFontsID, kā arī citām noformējuma metadatu tabulām. Fonta iestatījumi iekļauj fonta nosaukumu, fonta formātu (vai fonts ir treknrakstā, kursīvā, pasvītrots, izsvītrots), izmēru un krāsu.

Grafika noformējums

Grafika noformējumam ir paredzēta tabula *Grafika noformējums*. Grafikam var uzstādīt fona krāsu, orientāciju, virsraksta pozīciju, fonu un fontu, var arī noformēt leģendu.

Darba lapas noformējums

Visas darba lapas noformējuma datu glabāšanai tiek izmantota tabula *Darba lapas noformējums*. Darba lapai var norādīt lapas garumu un augstumu, malu platumu, lokalizāciju, fona krāsu, atskaites virsraksta fontu, fonu un līdzināšanu.

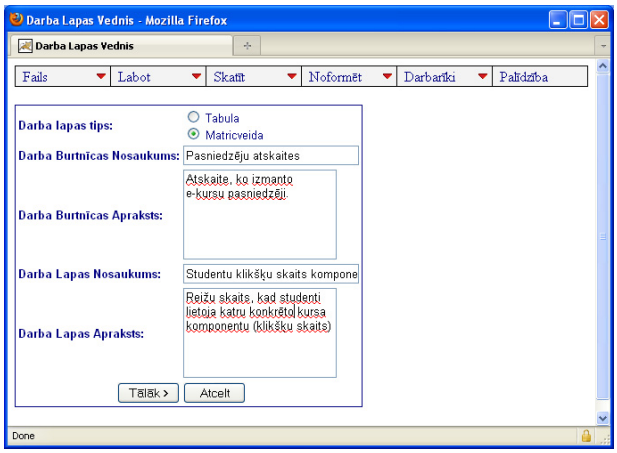
9.5. Saskarne

Šajā apakšnodaļā tiek parādīta izstrādāta atskaišu attēlošanas rīka saskarne, lai ilustrētu realizētās prasības atskaišu rīkam aprobācijas projektā.

Lai lietotāji varētu definēt atskaites, datu noliktavas izstrādātājam ir jānodefinē datu noliktavas shēmas metadati, termini un terminu versijas, kas shēmas elementus apraksta. Vajadzīgie metadati tiek definēti ar atskaišu definēšanas rīku. Metadatu izveidošanas apraksts tika dots apakšnodaļā 9.3.1.

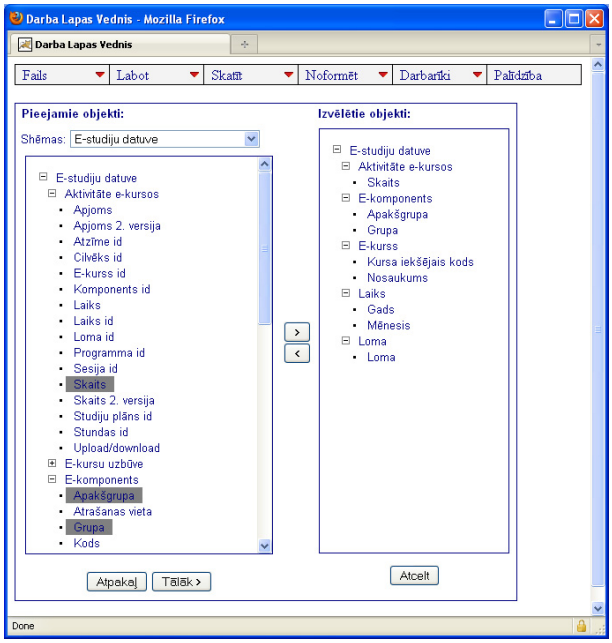
9.5.1. Atskaites definēšana

Atskaišu rīks tiek palaists no tīmekļa pārlūkprogrammas un ir pieejams caur internetu. Veidojot jaunu atskaiti, pirmajā solī lietotājs izvēlas atskaites veidu (tabulas vai matricveida atskaite) un ievada darba burtnīcas un darba lapas nosaukumu un aprakstu (attēls 33.). Ja atskaite tiek izveidota eksistējošā burtnīcā, tad darba burtnīcas nosaukums un apraksts nav jāievada.



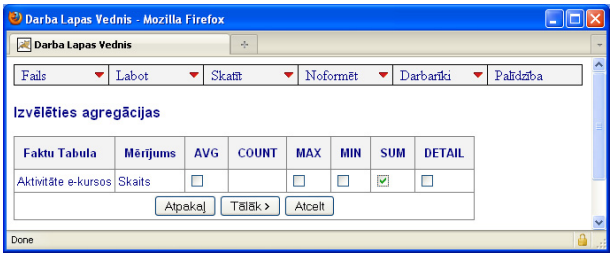
Att. 33. Jaunas atskaites definēšana – nosaukumi un apraksti

Tālāk lietotājam jāizvēlas datu noliktavas shēma un jāizvēlas šīs shēmas elementi, ko lietotājs grib redzēt atskaitē (attēls 34.). Shēmas elementi tiek attēloti kā terminu versijas, kas šos elementus apraksta. Terminu versijas var būt aprakstītas vienam un tam pašam terminam, piemēram, faktu tabulas *Aktivitāte* ir divas viena termina versijas: skaits un skaits 2. versija.



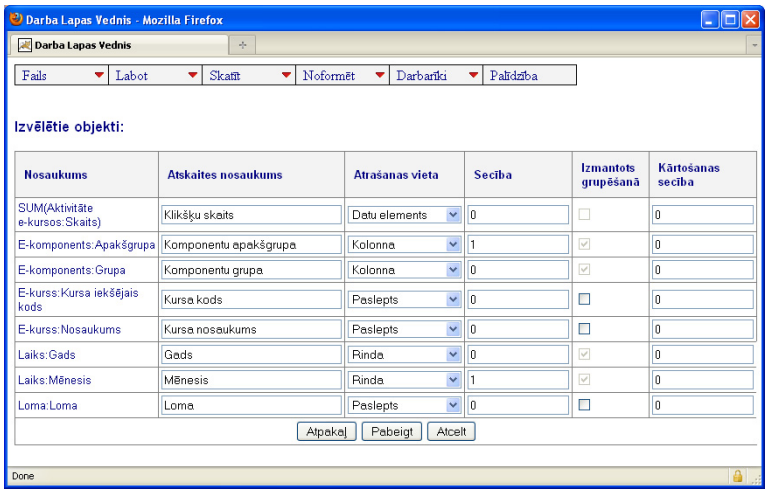
Att. 34. Jaunas atskaites definēšana – shēmas elementi

Nākošajā solī lietotājs izvēlas agregācijas funkcijas, ko pielietot mērījumiem (attēls 35.). Tiek attēlotas tikai tādas agregācijas funkcijas, kas ir pieļaujamas izvēlētiem mērījumiem attiecībā pret izvēlētām dimensijām. Šī informācija par pieļaujamām agregācijām tiek iegūta no loģiskā līmeņa metadatiem.



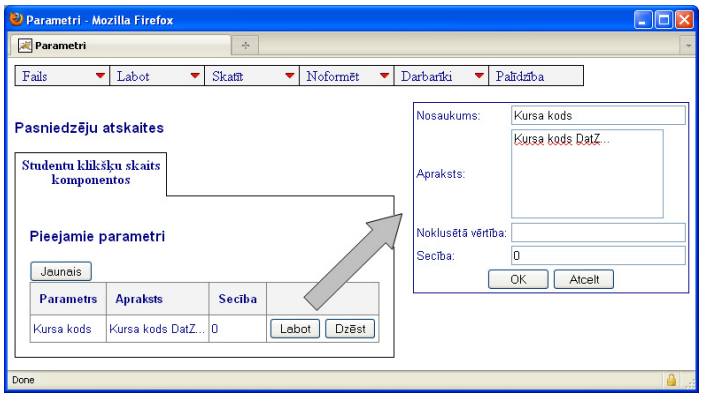
Att. 35. Jaunas atskaites definēšana – mērījumu agregācijas funkcijas

Pēdējā solī lietotājs izvēlas nosaukumus atskaites elementiem, to atrašanās vietas atskaitē, secību, definē, vai elements ir izmantots grupēšanai (ja elements ir paslēpts) un kā dati tiek sakārtoti (attēls 36.). Pēc šī soļa izpildīšanas atskaite ir jāsaglabā un to var jau izpildīt.



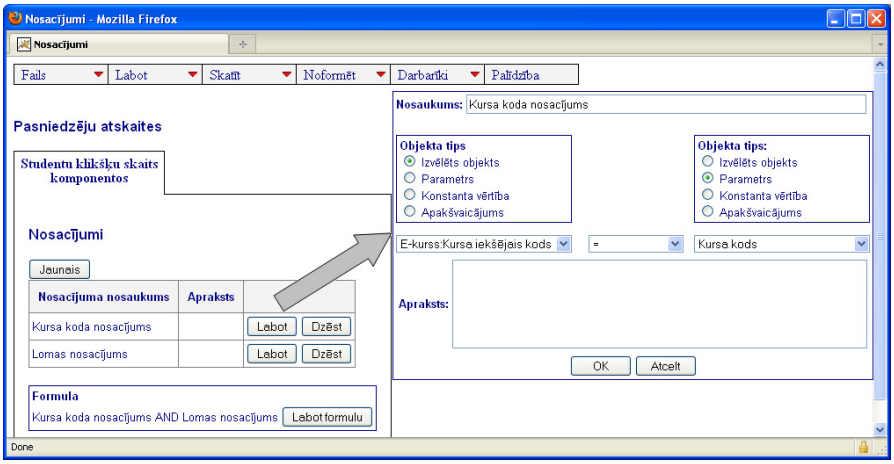
Att. 36. Jaunas atskaites definēšana – atskaites elementu iestatījumi

Atskaišu rīkā ir iespējams arī nodefinēt parametrus, kam vērtības piešķir lietotājs, kad izpilda atskaiti. Parametri tiek definēti visai darba burtnīcai, kas satur vairākas darba lappuses ar atskaitēm. Attēlā 37. ir redzams parametru saraksta un jaunā parametra veidošanas piemērs. Parametram definē nosaukumu, aprakstu, noklusēto vērtību un secību, kādā parametrs tiek parādīts lietotājam visu parametru sarakstā, kad lietotājam būs jāievada parametru vērtības atskaites izpildes laikā.



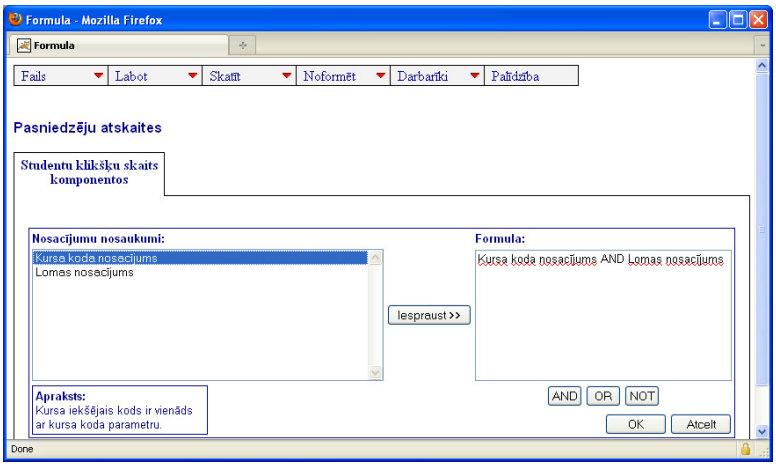
Att. 37. Jaunas atskaites definēšana – parametri

Atskaitei arī var definēt dažādus nosacījumus (attēls 38.). Nosacījumiem tiek definēts nosaukums, salīdzināmo elementu tipi (izvēlēts atskaites elements, parametrs, konstanta vērtība vai apakšvaicājums), salīdzināšanas operators un apraksts.



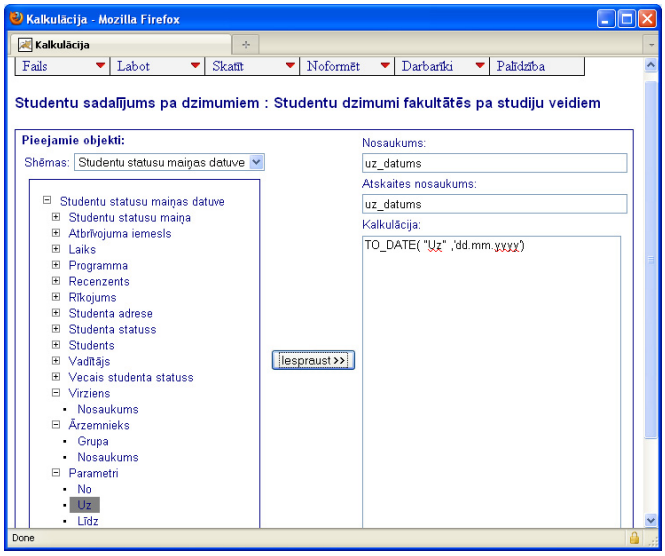
Att. 38. Jaunas atskaites definēšana – nosacījumi

Nosacījumi tiek apvienoti nosacījumu formulā, kas sastāv no nosacījumiem un loģiskiem operatoriem, kas definē, kādai nosacījumu kombinācijai jāizpildās (attēls 39.). Nosacījumu formulas ekrānā lietotājs izvēlas izveidotus nosacījumus no saraksta, aplūko katra nosacījuma aprakstu un pievieno nosacījumu formulas logā.



Att. 39. Jaunas atskaites definēšana – nosacījumu formula

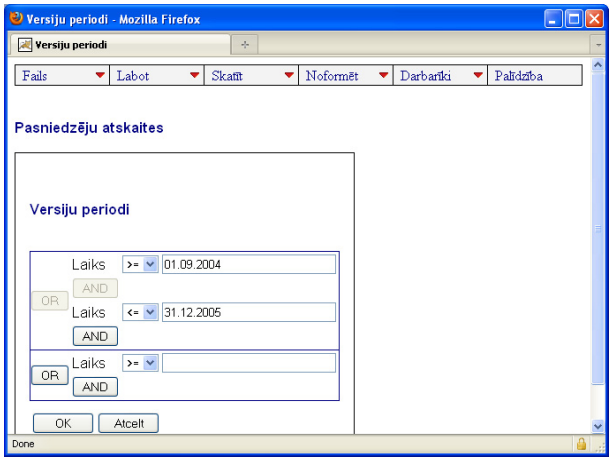
Atskaišu rīkā var nodefinēt arī papildus atskaites elementus, kas tiek izrēķināti no izvēlētiem elementiem un parametriem ar kādu formulu (attēls 40.). Atskaitēm var nodefinēt arī papildus savienojumus starp tabulām, kopsummas, kas apkopo kādas kolonnas vai rindas datus, grafikus, kas attēlo atskaites datus grafiski (ir atbalstīti 16 veidu grafiki), un noformējumus dažādiem izvēlētiem elementiem, atskaites virsrakstam, rindu un kolonnu galvenēm un visai atskaitei kopā.



Att. 40. Jaunas atskaites definēšana – izrēķinātie elementi

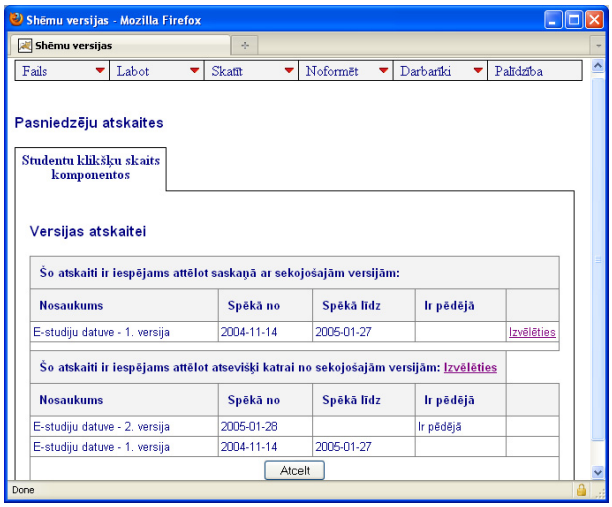
9.5.2. Atskaites izpildīšana

Pirms izpildīt atskaiti uz vairākām datu noliktavas shēmas versijām, ir jādefinē laika periodi, kuros spēkā esošas versijas tiks izmantotas atskaitē (attēls 41.). Versiju laika periodus veido lietotāji vai izstrādātāji, kas definē atskaiti vai kad izpilda to. Šie laika periodi tiek definēti kā nosacījumi ar laiku, t.i. tiek definēts, ka laiks ir mazāks vai lielāks par konkrētu datumu.



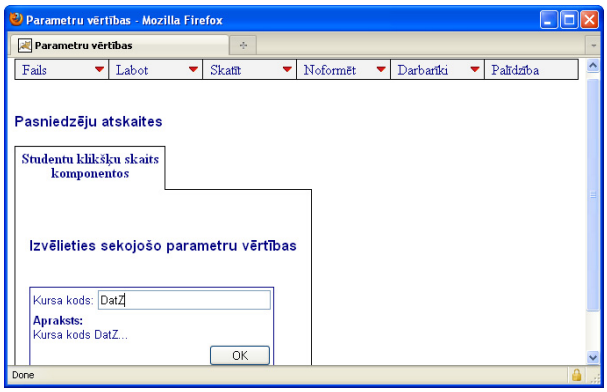
Att. 41. Atskaites izpildīšana – versiju periodi

Pēc versiju periodu definēšanas tiek izpildīta atskaite, vaicājuma konstruēšanas algoritms (kas tika aprakstīts nodaļā 8.) atrod versijas, kas bija spēkā definētajā versiju laika periodā un iegūst iespējamos atskaites datu attēlošanas veidus. Šie datu attēlošanas veidi tiek parādīti lietotājam (attēls 42.), kurš izvēlas interesējošo datu attēlošanas veidu.



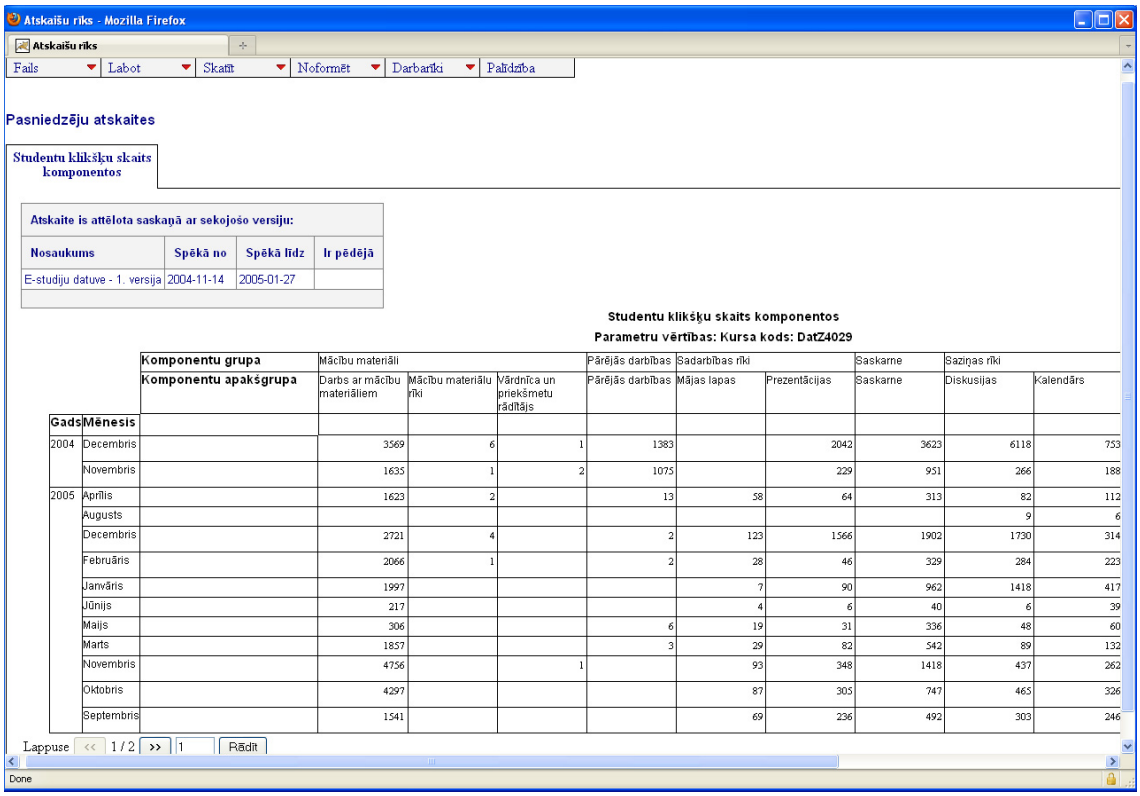
Att. 42. Atskaites izpildīšana – versiju izvēle

Ja atskaitē ir definēti parametri, tad lietotājam tiek piedāvāts ievadīt šo parametru vērtības (attēls 43.). Katram parametram arī tiek attēlots apraksts, kas parasti satur komentāru, kādi dati ir jāievada konkrētajā parametrā. Ja parametram ir nodefinēta noklusēta vērtība, tad tā tiek automātiski ierakstīta parametra vērtības laukā un lietotājs var šo vērtību pamainīt vai atstāt noklusēto.



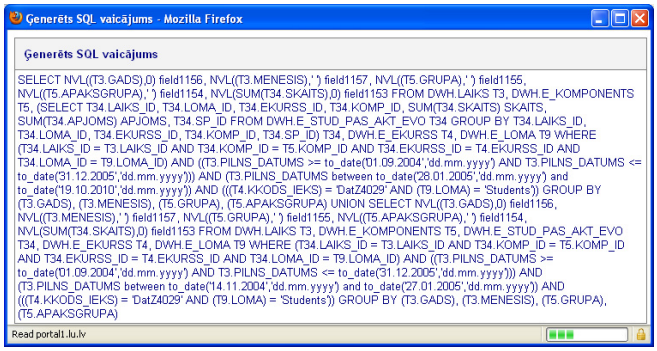
Att. 43. Atskaites izpildīšana – parametru ievads

Pēc parametru vērtību ievada vaicājuma konstruēšanas algoritms ģenerē SQL vaicājumu vienai vai vairākām shēmas versijām atbilstoši atskaišu metadatiem un lietotāja versiju izvēlei. Rezultātā iegūtā atskaite tiek attēlota tabulas vai matricas veidā un kopā ar atskaites datiem satur arī izvēlēto datu noliktavas shēmas versiju metadatus (attēls 44.).



Att. 44. Atskaites izpildīšana – rezultāta atskaite

Atskaišu rīkā ir arī iespējas gatavo atskaiti eksportēt MS Excel vai PDF formātā. Atskaitei var apskatīt arī ģenerēto SQL vaicājumu (attēls 45.).

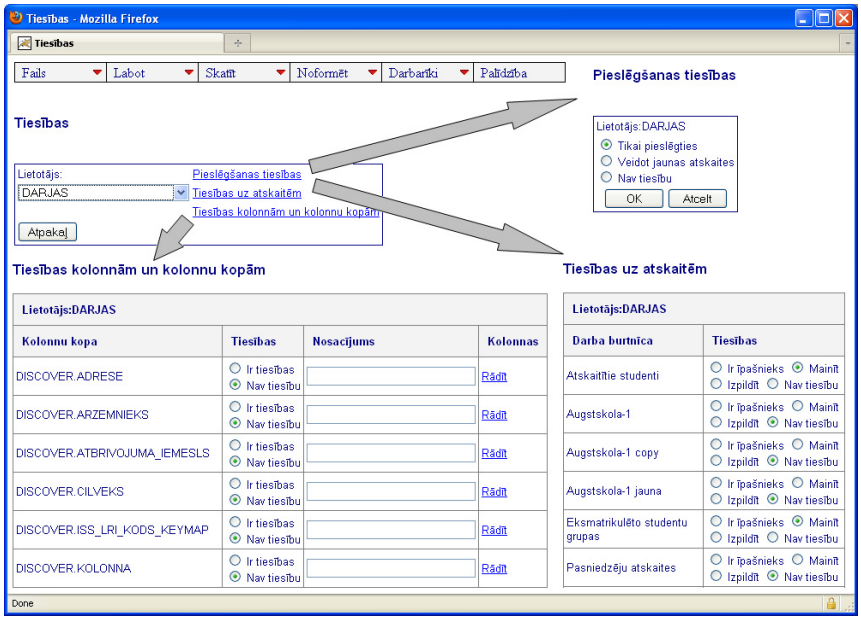


Att. 45. Atskaites izpildīšana – ģenerētais SQL vaicājums

9.5.3. Lietotāju tiesības

Atskaišu rīkā datu noliktavas administrators var definēt lietotāju pieslēgšanās tiesības, tiesības uz atskaitēm, uz kolonnām un kolonnu kopām (attēls 46.). Pieslēgšanas tiesības definē, vai lietotājs drīkst pieslēgties atskaišu rīkam un vai viņš vai viņa var veidot jaunas atskaites. Tiesības uz atskaitēm definē katrai atskaitei, vai lietotājs ir atskaites īpašnieks, vai drīkst mainīt atskaites definīciju, vai drīkst atskaiti tikai izpildīt. Tiesības kolonnām un kolonnu kopām definē katrai kolonnu kopai no fiziskā līmeņa metadatiem, vai lietotājam ir tiesības apskatīt informāciju no šīs kolonnu kopas. Ja lietotājam ir tiesības uz kādu kolonnu kopu, tad var norādīt arī nosacījumu, kas definē konkrētu datu apakškopu, ko lietotājs var

redzēt konkrētajā kolonnu kopā. Var atsevišķi iedot tiesības uz kolonnu kopas konkrētajām kolonnām.



Att. 46. Lietotāju tiesības

Atskaišu attēlošanas un definēšanas rīkos no promocijas darba rezultātiem ir realizēti:

- Datu noliktavas shēmas versiju atbalsts metadatos;
- Shēmas versiju izvēle atskaišu attēlošanas rīka saskarnē;
- Algoritms SQL vaicājumu konstruēšanai uz vienu vairākām datu noliktavas shēmas versijām.

Atskaišu attēlošanas rīks sekmīgi pielietots aprobācijas projektā.

10. PIEDĀVĀTĀS PIEEJAS IZVĒRTĒJUMS

10.1. Nodaļas nolūks

Šīs nodaļas mērķis ir novērtēt promocijas darbā piedāvātās pieejas priekšrocības un ierobežojumus, salīdzināt to ar tradicionālu datu noliktavas risinājumu. Ilustrācijai tiek izmantota aprobācijas projekta ietvaros izstrādātā datuve un tās evolūcijas problēmas. Otrs paņēmiens, kas pielietots izvērtēšanā, ir piedāvātās pieejas testēšana ar piemēriem no citu autoru zinātniskiem rakstiem par datu noliktavas evolūciju.

10.2. Piedāvātās pieejas salīdzinājums ar risinājumu bez shēmas versijām

10.3.1. Problēmas apraksts

Lai izvērtētu piedāvāto pieeju, tika izmantota e-studiju datuve. Izmaiņas, kas ir notikušas ar šo datuvi, kā arī datuves shēmas versijas, tika aprakstītas nodaļā 4. Aprobācijas projekta ietvaros tika izmantotas divas konkrētas aktivitātes zvaigznes shēmas versijas, kas ir redzamas attēlā 10. un attēlā 14.

Saistībā ar aktivitātes zvaigznes shēmas evolūciju tiek risināti sekojošie jautājumi:

- notikušo shēmas izmaiņu apstrāde;
- divu shēmas versiju glabāšana relāciju datu bāzē;
- zvaigznes shēmas aprakstīšana metadatos;
- zvaigznes shēmas datu attēlošana atskaitēs.

Atskaites piemērs

Aprobācijas projekta ietvaros atskaišu vaicājumu konstruēšanai tika izmantotas atskaites, ko izpilda e-studiju datuvē. Lai ilustrētu promocijas darbā piedāvāto pieeju, tālāk promocijas darbā tiek izmantota atskaite, kas attēlo reižu skaitu, kad studenti lietoja konkrēta e-kursa komponentus (klikšķu skaits). Šī atskaite tika izstrādāta abās aktivitātes zvaigznes shēmas versijās, kuru shēmas redzamas attēlā 10. un attēlā 14. Šo atskaiti izmanto akadēmiskā departamenta darbinieki e-studiju procesa analīzei, kā arī šī atskaite varētu interesēt konkrēto kursu pasniedzējus, kas ir ieinteresēti sīkākā viņu docēto kursu analīzē.

10.2.2. Risinājums bez shēmas versijām

Šajā apakšnodaļā tiek apskatīts risinājums datu noliktavas evolūcijas problēmām, kas tika lietots reālā datu noliktavas projektā pirms piedāvātās pieejas izstrādes. Sākotnējā datuves aktivitātes zvaigznes shēmas versija tika izveidota atbilstoši projektējumam un fiziski datu bāzē tā sastāvēja no tabulām, kas ir redzamas attēlā 14. Kad mainījās datuves darījumprasības, un datuves aktivitātes zvaigznes shēma tika pārveidota, tad fiziski datu bāzē tika izveidotas jaunas tabulas: *Aktivitāte-jauna, Cilvēks, Sesija, Atzīme, Studiju programma, Laiks-jauna*, kuru kolonnas atbilst jaunajai zvaigznes shēmas versijai, kas ir redzama attēlā 10. Datu iegūšanas, pārveidošanas un ielādes (ETL) procesi tika pārveidoti un kopš otrās shēmas

versijas parādīšanas (01.02.2005.) ETL procesi tika izpildīti otrajai shēmas versijai, t.i. pirmā zvaigznes shēmas versija vairs netika atjaunināta.

Tādēļ, ka iepriekš projektā izmantotais atskaišu rīks Oracle Discoverer neatļauj vienlaicīgu darbu ar vairākām shēmas versijām, šajā rīkā atskaites tika izveidotas atsevišķi pirmajai un otrajai zvaigznes shēmas versijai. Datiem par periodu no 01.09.2004. līdz 31.01.2005. lietotājiem bija jāizmanto atskaites, kas balstījās uz pirmo zvaigznes shēmas versiju, bet datiem par periodu no 01.02.2005. lietotāji izmantoja atskaites uz otro zvaigznes shēmas versiju. Ja bija nepieciešams iegūt datus par periodu, kas pārklāj abu shēmas versiju derīguma termiņu, tad lietotājiem bija jāizpilda divas atskaites un dati jāsummē manuāli.

```
SELECT KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
FROM E_KURSS, KOMPONENTS, LAIKS, AKTIVITATE, LOMA
WHERE ((LOMA.ID=AKTIVITATE.LOMA_ID) AND
(E_KURSS.ID=AKTIVITATE.E_KURSS_ID) AND
(KOMPONENTS.ID=AKTIVITATE.KOMPONENTS_ID) AND
(LAIKS.ID=AKTIVITATE.LAIKS_ID) AND
(LAIKS.DATUMS BETWEEN TO_DATE(:"no",'dd.mm.yyyy') AND
TO_DATE(:"līdz",'dd.mm.yyyy'))) AND
(E_KURSS.NOSAUKUMS=:"kurss") AND
LOMA.LOMA='Students'
GROUP BY KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS;
```

Att. 47. Vaicājums sākotnējai aktivitātes zvaigznes shēmas versijai

Vaicājumi, kurus ģenerēja Oracle Discoverer aprobācijas atskaitei, ir redzami attēlā 47. un attēlā 48. Lai uzlabotu vaicājumu lasāmību, vaicājumi tika minimāli pārveidoti (tika mainīti tabulu aizstājvārdi ar tabulu nosaukumiem). Atskaitei tika izmantoti parametri „no" un "līdz", kas definē periodu, par kuru lietotājs vēlas attēlot datus atskaitē, un parametrs „kurss" definē kursa nosaukumu, par kuru interesē informācija. Šie divi vaicājumi atšķīras ar to, ka dati tajos ir atlasīti no divām dažādām faktu tabulām *Aktivitāte* un *Aktivitāte-jauna*.

```
SELECT KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE_JAUNA.HITU_SKAITS)
FROM E_KURSS, KOMPONENTS, LAIKS, AKTIVITATE_JAUNA, LOMA
WHERE ((LOMA.ID=AKTIVITATE_JAUNA.LOMA_ID) AND
(E_KURSS.ID=AKTIVITATE_JAUNA.E_KURSS_ID) AND
(KOMPONENTS.ID=AKTIVITATE_JAUNA.KOMPONENTS_ID) AND
(LAIKS.ID=AKTIVITATE_JAUNA.LAIKS_ID) AND
(LAIKS.DATUMS BETWEEN TO_DATE(:"no",'dd.mm.yyyy') AND
TO_DATE(:"līdz",'dd.mm.yyyy'))) AND
(E_KURSS.NOSAUKUMS=:"kurss") AND
LOMA.LOMA='Students'
GROUP BY KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS;
```

Att. 48. Vaicājums otrajai aktivitātes zvaigznes shēmas versijai

10.2.3. Piedāvātās pieejas pielietojums aprobācijas projektā

Šajā apakšnodaļā, lai ilustrētu piedāvāto pieeju, tiek risinātas aktivitātes zvaigznes shēmas evolūcijas problēmas saskaņā ar piedāvāto pieeju. Tiek aprakstīts, kā aprobācijas

projekta ietvaros, tika būvēta daudzversiju datu noliktava un ar promocijas darba ietvaros piedāvāto rīku ģenerēti vaicājumi, kas veido atskaitei nepieciešamos datus.

Lai izveidotu jaunu zvaigznes shēmas versiju, tika izpildītas sekojošas fiziskās un loģiskās izmaiņas:

- Fiziskās izmaiņas:
  - Jaunu dimensiju *Cilvēks*, *Studiju programma, Atzīme* un *Sesija* pievienošana;
  - Jaunu dimensijas atribūtu pievienošana: *Laiks* un *Stunda* dimensijai *Laiks*, *Apakšgrupa, Grupa, Obligāts* un *Tips* dimensijai *Komponents*;
  - Jauna mērījuma *Laiks* pievienošana faktu tabulai *Aktivitāte*.
- Loģiskās izmaiņas:
  - Dimensiju *Cilvēks, Studiju programma, Atzīme* un *Sesija* piesaistīšana faktu tabulai *Aktivitāte*;
  - Jauno attiecīgo dimensiju hierarhiju pievienošana dimensijām *Cilvēks, Studiju programma, Atzīme* un *Sesija*;
  - Jauno dimensiju hierarhiju (*Grupa, Tips, Obligātums*) pievienošana eksistējošai *Komponentu* dimensijai;
  - Jauno hierarhiju līmeņu pievienošana: *Obligāts* un *Nosaukums* komponentu *Obligātumu* hierarhijai; *Tips* un *Nosaukums* komponentu *Tipu* hierarhijai; *Grupa, Apakšgrupa, Nosaukums* komponentu *Grupu* hierarhijai; *Laiks* un *Stunda* laika *Kalendārai* un *Mācību gadu* hierarhijām.
  - *Komponentu* dimensijas atribūtu *Nosaukums, Obligāts, Tips, Apakšgrupa* un *Grupa* un *Laika* dimensijas atribūtu *Datums* un *Stunda* piesaistīšana attiecīgajiem hierarhiju līmeņiem.

Fiziskais līmenis

Saskaņā ar piedāvāto pieeju fiziski relāciju datu bāzē visas shēmas versijas tiek glabātas vienā fiziskajā struktūrā. Sākotnējā apskatītās aktivitātes zvaigznes shēmas versija tika izveidota atbilstoši shēmai, kas ir redzama attēlā 14. Saskaņā ar izmaiņām datuves darījumprasībās, kas izraisīja aktivitātes zvaigznes shēmas evolūciju, datu bāzē ir jāizveido jaunā zvaigznes shēmas versija.

Lai to izdarītu, datu bāzē tika izveidotas jaunas tabulas: *Cilvēks, Sesija, Atzīme* un *Studiju programma*, kuru kolonnas atbilst jaunajai zvaigznes shēmas versijai, kas ir redzama attēlā 10. Katrā šajā tabulā tiek izveidots fiktīvs ieraksts ar datiem „Viss kopā" ar primārās atslēgas identifikatoriem, respektīvi, C, S, A un SP.

Tabulai *Aktivitāte* tiek pievienotas papildus kolonnas: Cilveks_ID, Sesija_ID, Atzime_ID un Studiju_programma_ID, kas ir aizpildītas ar identifikatoriem C, S, A un SP visiem tabulā esošiem datiem.

Tabula *Laiks* ir papildināta ar kolonnām *Laiks* un *Stunda*, lai glabātu laiku ar precizitāti līdz sekundei. Esošiem datiem dimensijā *Laiks* jaunās kolonnas tiek aizpildītas ar noklusētiem datiem, piemēram, '00:00:00'.

Tabulai *Komponents* tika pieliktas jaunas kolonnas *Obligāts*, *Tips*, *Apakšgrupa* un *Grupa*, lai tālāk izveidotu atbilstošas hierarhijas. Šajā gadījumā ir iespējams aizpildīt jaunās

kolonnas manuāli, jo informācija par komponentu klasifikāciju ir pieejama visiem komponentiem un ierakstu skaits tabulā ir salīdzinoši mazs.

Loģiskais līmenis

Loģiskā līmeņa metadatos tiek veidotas divas shēmas versijas atbilstoši shēmām, kas ir redzamas attēlā 10. un attēlā 14. Elementi, kas paliek nemainīgi abās versijās tiek piesaistīti abām versijām ar versiju transformācijām bez konvertācijas funkcijām.

Jauni elementi, kas ir piesaistīti ar versiju transformācijām bez konvertācijas tikai otrajai versijai ir:

- mērījums *Laiks* faktu tabulā *Aktivitāte;*
- dimensijas *Cilvēks, Sesija, Atzīme* un *Studiju programma* ar atbilstošajām hierarhijām;
- faktu-tabulu-dimensiju asociācijas starp dimensijām *Cilvēks, Sesija, Atzīme, Studiju programma* un faktu tabulu *Aktivitāte*;
- atribūti *Laiks* un *Stunda* dimensijā *Laiks*;
- atribūti *Obligāts, Tips, Apakšgrupa* un *Grupa* dimensijā *Komponents*;
- *Grupu, Tipu, Obligātumu* hierarhijas *Komponentu* dimensijā;
- līmeņi *Laiks* un *Stunda Laika* dimensijas hierarhijās;
- jauni mērījumi, *Hitu_skaits_2_versija*, *Datu_apjoms_2_versija.*

Jauni mērījumi tiek izveidoti tādēļ, ka semantiski tagad tajos glabājas citi dati, nekā pirmās versijas mērījumos, tie ir ar citu detalizācijas pakāpi.

Mainītajiem mērījumiem tiek ģenerētas versiju transformācijas, kas ir attēlotas tabulā 9. Versiju transformācijas mērījumiem *Hitu skaits* un *Datu apjoms* tiek ģenerētas tikai vienā virzienā no otrās versijas uz pirmo versiju, jo nav iespējams sadalīt apkopotos mērījumu datus, lai atbilstu otrās versijas shēmai. Tiek izveidotas arī versijas transformācijas *Komponentu* dimensijas jaunajiem atribūtiem, jo to vērtības eksistējošiem datiem tika aizpildītas fiziskajā līmenī. Versijas transformācijas jauniem *Laika* dimensijas atribūtiem netiek veidotas, jo šie atribūti tika aizpildīti ar fiktīvām vērtībām esošiem datiem.

Tabula 9. Versiju transformācijas

NoVersija	**UzVersija**	**UzObjekts**	**Konvertācija**
V_2	V_1	Hitu_skaits	SUM(Aktivitāte.Hitu_skaits_2_versija)
V_2	V_1	Datu_apjoms	SUM(Aktivitāte.Datu_apjoms_2_versija)
V_1	V_2	Apakšgrupa	Apakšgrupa
V_1	V_2	Grupa	Grupa
V_1	V_2	Tips	Tips
V_1	V_2	Obligāts	Obligāts

Semantiskais līmenis

Semantiskajā līmenī tika izveidotas jaunas terminu versijas faktu tabulas *Aktivitāte* mērījumiem *Hitu skaits* un *Datu apjoms*, jo mainījās mērījumu detalizācijas pakāpe, kas nozīmē arī to, ka semantiski mērījumi attēlo dažādu informāciju divās shēmas versijās. Terminu versijas, kas eksistēja mainītajiem mērījumiem, un kas tika izveidotas jaunajiem mērījumiem, ir dotas tabulā 10.

Tabula 10. Terminu versijas

Mērījums	Termins	Termina versija	Shēmas versija
Hitu_skaits	Klikšķu skaits	Lietotāju summārais klikšķu skaits pa lomām	V_1
Hitu_skaits_2_versija	Klikšķu skaits	Katra lietotāja klikšķu skaits	V_2
Datu_apjoms	Pārsūtīto datu apjoms	Lietotāju summārais pārsūtīto datu apjoms pa lomām	V_1
Datu_apjoms_2_versija	Pārsūtīto datu apjoms	Katra lietotāja pārsūtīto datu apjoms	V_2

Atskaites definēšana

Kad tiek izveidotas visas nepieciešamas datu struktūras datu bāzē un shēmas metadati, atskaišu metadatos tiek definēta aprobācijas ilustrācijas atskaite. Šai atskaitei tiek izveidoti sekojošie atskaišu metadati:

- Tabulā *Izvēlēts* aprakstīti izvēlētie elementi: komponenta nosaukums, datums un klikšķu skaits, kā arī paslēpti izvēlētie elementi: kursa nosaukums un loma;
- Tabulā *Savienojums* tiek izveidoti savienojumi starp faktu tabulu *Aktivitāte* un dimensiju tabulām *Loma*, *E-kurss, Komponents* un *Laiks*;
- Tabulā *Parametrs* tiek aprakstīti parametri: konkrētā kursa nosaukums, datums no un datums līdz, kas definē atskaites datu laika ierobežojumu;
- Tabulā *Nosacījums* tiek izveidoti lietotāja definēti nosacījumi: *Loma='Students'*, *Kursa nosaukums=kursa nosaukums*, *Datums>=datums no, Datums<=datums līdz* (kursa nosaukums, datums no un datums līdz ir parametri), kā arī tiek izveidoti versiju periodu nosacījumi: *Laiks>='01.01.2004'* un *Laiks<='01.09.2005'*.

Atskaites izpildīšana

Kad lietotājs izpilda atskaiti, viņam ir jāievada parametru vērtības, kas aprobācijas ilustrācijas atskaitei ir kursa nosaukuma parametrs „kurss", datumi, kas raksturo atskaites laika periodu: „datums no" un „datums līdz". No ievadītajām versijas laika ierobežojuma vērtībām tiek iegūts laika periods, kurā ir spēkā atskaitē izmantotās shēmas versijas. Lietotājs ir ievadījis laika ierobežojuma parametru vērtības'01.01.2004' un '01.09.2005', kas pārklāj abas divas aktivitātes zvaigznes shēmas versijas.

Tabula 11. Versiju un elementu attiecību tabula 1. gadījuma atskaitei

		Shēmas versija	
		V_1	**V_2**
Shēmas elements	**Kurss.Nosaukums**	1	1
	Aktivitāte.Hitu_skaits	1	2
	Laiks.Datums	1	1
	Loma.Loma	1	1
	Komponents.Nosaukums	1	1

Gadījums 1. Šajā gadījumā saskaņā ar algoritma soli, kas ir aprakstīts apakšnodaļā 8.2.7., tiek būvēta versiju un elementu attiecību tabula, kas ir attēlota tabulā 11. Atskaitē

izmantotie shēmas elementi tiek iegūti no izvēlētajiem elementiem atskaišu metadatu tabulā *Izvēlēts*. Aprobācijas ilustrācijas atskaiti ir iespējams attēlot saskaņā ar pirmo shēmas versiju, tādēļ lietotājam tiek piedāvāti divi atskaites datu attēlošanas veidi: attēlot atskaites datus saskaņā ar pirmo shēmas versiju un attēlot datus vairākās atskaitēs atsevišķi katrai versijai.

Gadījums 2. Ja lietotājs modificētu atskaiti un izvēlētos citu detalizācijas pakāpi, piemēram, attēlot mērījuma datus pēc komponenta apakšgrupas (nevis nosaukuma), tad versiju un elementu attiecību tabula izskatītos savādāk, kā tas ir redzams tabulā 12. Šajā gadījumā atskaites datus nav iespējams attēlot saskaņā ar vienu konkrēto shēmas versiju, bet tomēr ir iespējams attēlot elementus no vairākām shēmas versijām vienā atskaitē, jo ir versijas transformācija komponenta apakšgrupai. Šādā gadījumā lietotājam piedāvātu sekojošus datu attēlošanas veidus: attēlot elementus no vairākām shēmas versijām vienā atskaitē un attēlot datus vairākās atskaitēs atsevišķi katrai versijai.

Tabula 12. Versiju un elementu attiecību tabula 2. gadījuma atskaitei

		Shēmas versija	
		V_1	**V_2**
Shēmas elements	**Kurss.Nosaukums**	1	1
	Aktivitāte.Hitu_skaits	1	2
	Laiks.Datums	1	1
	Loma.Loma	1	1
	Komponents.Apakšgrupa	2	1

Pieņemsim, ka aprobācijas ilustrācijas atskaitei (gadījums 1.) lietotājs ir izvēlējies attēlot atskaites datus saskaņā ar pirmo shēmas versiju, lai apskatītu piemēru, kad dati tiek pārveidoti ar versiju transformācijām. Tad tālākajos soļos tiek ģenerēts SQL vaicājums saskaņā ar piedāvāto algoritmu, kas ir aprakstīts apakšnodaļās 8.2.1.-8.2.7.

Ģenerētā vaicājuma struktūra ir redzama attēlā 30. Sākotnējais vaicājums tiek konstruēts saskaņā ar pirmās zvaigznes shēmas versijas struktūru. Vaicājums tiek konstruēts dažos soļos. Tālāk tiek apskatīts vaicājuma konstruēšanas algoritms piemēra atskaitei.

Pirmais algoritma solis ir atskaitē izvēlēto elementu analīze un izmantoto kolonnu kopu noteikšana (apakšnodaļa 8.2.1.). Aprobācijas ilustrācijas atskaitei tiek iegūti 3 redzami izvēlētie elementi: komponenta nosaukums, datums un summārais klikšķu skaits, kā arī 2 paslēptie izvēlētie elementi: kursa nosaukums un loma, kas netiek iekļauti vaicājuma SELECT daļā. Vaicājumā iekļautie pirmie divi elementi ir kolonnu nosaukumi, bet pēdējais elements ir iegūts ar agregācijas funkciju no kolonnas. Vienlaicīgi šajā solī tiek iegūts arī saraksts ar izmantotajām tabulām: e-kurss, komponents, laiks, aktivitāte, loma, un saraksts ar kārtošanas elementiem. Pirmajā solī iegūtais rezultāts ir redzams attēlā 49.

```
SELECT KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
FROM E_KURSS, KOMPONENTS, LAIKS, AKTIVITATE, LOMA
ORDER BY KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
```

Att. 49. Izvēlētie elementi un kolonnu kopu saraksts

Atbilstoši apakšnodaļā 8.2.2. dotajam aprakstam otrajā algoritma solī tiek ģenerēti izmantoto kolonnu kopu savienojumi. Tiek analizētas izmantotās tabulas, kas ir iegūtas iepriekšējā solī un tiek ģenerēti savienojumi, kas atbilst ārējām atslēgām aktivitātes faktu tabulai ar dimensijām, jo nekādi papildus savienojumi atskaitei netika definēti. Tiek apstrādāti savienojumu metadati no atskaišu metadatu tabulas *Savienojums*, un tiek iegūti savienojumu predikāti atbilstoši vaicājuma konstruēšanas algoritmam. Predikāti tiek apvienoti ar AND operatoru un tiek iekļauti vaicājuma WHERE daļā. Iegūtais rezultāts ir dots attēlā 50.

```
SELECT KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
FROM E_KURSS, KOMPONENTS, LAIKS, AKTIVITATE, LOMA
WHERE (LOMA.ID=AKTIVITATE.LOMA_ID AND
E_KURSS.ID=AKTIVITATE.E_KURSS_ID AND
KOMPONENTS.ID=AKTIVITATE.KOMPONENTS_ID AND
LAIKS.ID=AKTIVITATE.LAIKS_ID)
ORDER BY KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU SKAITS)
```

Att. 50. Savienojumi starp izmantotajām tabulām

Atbilstoši apakšnodaļā 8.2.3. dotajam aprakstam algoritma trešajā solī tiek būvēti lietotāja definētie nosacījumi, kas nesatur agregācijas funkcijas. Piemēra atskaitei tika definēti 4 nosacījumi, kas ierobežo laika periodu, kursu un lietotāja lomu. Nosacījumos tiek izmantoti parametri, kuru vērtības jau ir zināmas uz atskaites izpildīšanas brīdi un vaicājuma konstruēšanas brīdi, jo lietotājs parametru vērtības jau ir ievadījis. Tādēļ parametru nosaukumu vietā tiek uzreiz iekļautas atbilstošo parametru vērtības. Šajā solī iegūtais rezultāts ir attēlots attēlā 51.

```
SELECT KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
FROM E_KURSS, KOMPONENTS, LAIKS, AKTIVITATE, LOMA
WHERE (LOMA.ID=AKTIVITATE.LOMA_ID AND
E_KURSS.ID=AKTIVITATE.E_KURSS_ID AND
KOMPONENTS.ID=AKTIVITATE.KOMPONENTS_ID AND
LAIKS.ID=AKTIVITATE.LAIKS_ID) AND
(LAIKS.DATUMS >= TO_DATE('01.09.2004','dd.mm.yyyy') AND
LAIKS.DATUMS <= TO_DATE('01.09.2005','dd.mm.yyyy') AND
E_KURSS.NOSAUKUMS='Datu noliktavas' AND
LOMA.LOMA='Students')
ORDER BY KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
```

Att. 51. Lietotāja definēti nosacījumi

Atbilstoši apakšnodaļā 8.2.4. dotajam aprakstam algoritma ceturtajā solī atkal tiek analizēti izvēlētie elementi un tiek iegūti elementi, kas ir jāizmanto grupēšanai, un nosacījumi, kas satur agregācijas funkcijas. Piemēra atskaitei ir divi izvēlētie elementi (komponenta nosaukums un datums), kas ir jāizmanto grupēšanai. Pārējie izvēlētie elementi vai nu ir paslēpti un tādēļ nav izmantoti grupēšanai, vai tiek aprēķināti ar agregācijas funkcijām, tādēļ tie nav jāiekļauj vaicājuma GROUP BY daļā. Iegūtais vaicājums redzams attēlā 52.

Aprobācijas ilustrācijas atskaitei nav nosacījumu, kas satur agregācijas funkcijas, tādēļ būvētais SQL vaicājums netiek papildināts ar HAVING sadaļu un nosacījumiem ar agregācijas funkcijām.

```
SELECT KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
FROM E_KURSS, KOMPONENTS, LAIKS, AKTIVITATE, LOMA
WHERE (LOMA.ID=AKTIVITATE.LOMA_ID AND
E_KURSS.ID=AKTIVITATE.E_KURSS_ID AND
KOMPONENTS.ID=AKTIVITATE.KOMPONENTS_ID AND
LAIKS.ID=AKTIVITATE.LAIKS_ID) AND
(LAIKS.DATUMS >= TO_DATE('01.09.2004','dd.mm.yyyy') AND
LAIKS.DATUMS <= TO_DATE('01.09.2005','dd.mm.yyyy') AND
E_KURSS.NOSAUKUMS='Datu noliktavas' AND
LOMA.LOMA='Students')
GROUP BY KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS
ORDER BY KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
```

Att. 52. Grupēšanas elementi

Nākošajā solī, atbilstoši darba apakšnodaļā 8.2.5. dotajam aprakstam, vaicājums tiek papildināts ar lietotāja tiesību ierobežojumiem, ja tādi eksistē. Izmantotajām tabulām un kolonnām ierobežojumi tiek iegūti kā nosacījumi, kas tiek pielikti vaicājuma WHERE daļai. Pieņem, ka lietotājam eksistē tiesības uz visām atskaitē izmantotajām kolonnu kopām. Aprobācijas ilustrācijas atskaitei nav nekādu tiesību ierobežojumu, tādēļ šajā solī vaicājums paliek bez izmaiņām.

Sestajā algoritma solī, atbilstoši darba apakšnodaļā 8.2.6. dotajam aprakstam, vaicājums tiek vienkāršots un papildināts ar papildus izteiksmēm. Pirmkārt, ja ir atkārtojošas konstrukcijas (izvēlētie elementi vai nosacījumi), kas nemaina vaicājuma rezultātu, tie tiek izņemti. Aprobācijas vaicājumā nav konstrukciju, kas atkārtojas. Tālāk vaicājums tiek papildināts ar ROLLUP un GROUPING SETS SQL papildinājumiem saskaņā ar piedāvāto algoritmu. Piemēra atskaitei nav izvēlēto elementu, kas ir attēloti atskaitē kā lappušu objekti (page items), tādēļ paplašinājums GROUPING SETS netiek pievienots.

Tālāk sestā algoritma solis tiek apskatīts piemēra atskaitei gadījumam 1., kad atskaites dati ir attēloti katram komponenta nosaukumam, un gadījumam 2., kad atskaites dati ir attēloti katrai komponentu apakšgrupai.

Gadījums 1. Aprobācijas ilustrācijas atskaite izmanto atribūtus no dažām hierarhijām, piemēram, datumu hierarhija (*Datums, Mēnesis, Gads*) un semestru hierarhija (*Datums, Semestris, Studiju gads*). ROLLUP papildinājums tiek konstruēts katrai hierarhijai un tiek iekļauts vaicājuma GROUP BY daļā. Arī vaicājuma SELECT daļa tiek papildināta ar pārējiem katras hierarhijas atribūtiem un DISTINCT operatoru. Rezultātā tiek iegūti summēti mērījumi katram hierarhijas līmenim. Ja lietotājs veic roll-up vai drill-down operācijas, nepieciešamie dati tiek attēloti atskaitē uzreiz, neizpildot papildus vaicājumus. Šajā solī iegūtais rezultāts ir redzams attēlā 53.

```
SELECT DISTINCT KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
LAIKS.MENESIS, LAIKS.GADS, LAIKS.SEMESTRIS,
LAIKS.STUDIJU_GADS, KOMPONENTS.APAKSGRUPA, KOMPONENTS.GRUPA,
SUM(AKTIVITATE.HITU_SKAITS)
FROM E_KURSS, KOMPONENTS, LAIKS, AKTIVITATE, LOMA
WHERE (LOMA.ID=AKTIVITATE.LOMA_ID AND
E_KURSS.ID=AKTIVITATE.E_KURSS_ID AND
KOMPONENTS.ID=AKTIVITATE.KOMPONENTS_ID AND
LAIKS.ID=AKTIVITATE.LAIKS_ID) AND
(LAIKS.DATUMS >= TO_DATE('01.09.2004','dd.mm.yyyy') AND
LAIKS.DATUMS <= TO_DATE('01.09.2005','dd.mm.yyyy') AND
E_KURSS.NOSAUKUMS='Datu noliktavas' AND
LOMA.LOMA='Students')
GROUP BY ROLLUP(LAIKS.GADS, LAIKS.MENESIS, LAIKS.DATUMS),
ROLLUP(LAIKS.STUDIJU_GADS, LAIKS.SEMESTRIS, LAIKS.DATUMS
ORDER BY KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
```

Att. 53. Optimizēts vaicājums 1. gadījuma atskaitei

Gadījums 2. Ja lietotājs izvēlētos attēlot atskaitē datus pa komponentu apakšgrupām, kas ir iekļautas tikai otras versijas shēmā, bet varētu tikt iegūti ar versijas transformācijām arī pirmās versijas shēmā, tad, apvienojot elementus no divām versijām, tiktu iegūta papildus komponentu grupu hierarhija (Nosaukums, Apakšgrupa, Grupa) un ģenerētais vaicājums šajā solī izskatītos savādāk, kā tas ir redzams attēlā 54.

.

```
SELECT DISTINCT KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
LAIKS.MENESIS, LAIKS.GADS, LAIKS.SEMESTRIS,
LAIKS.STUDIJU_GADS, KOMPONENTS.APAKSGRUPA, KOMPONENTS.GRUPA,
SUM(AKTIVITATE.HITU_SKAITS)
FROM E_KURSS, KOMPONENTS, LAIKS, AKTIVITATE, LOMA
WHERE (LOMA.ID=AKTIVITATE.LOMA_ID AND
E_KURSS.ID=AKTIVITATE.E_KURSS_ID AND
KOMPONENTS.ID=AKTIVITATE.KOMPONENTS_ID AND
LAIKS.ID=AKTIVITATE.LAIKS_ID) AND
(LAIKS.DATUMS >= TO_DATE('01.09.2004','dd.mm.yyyy') AND
LAIKS.DATUMS <= TO_DATE('01.09.2005','dd.mm.yyyy') AND
E_KURSS.NOSAUKUMS='Datu noliktavas' AND
LOMA.LOMA='Students')
GROUP BY ROLLUP(LAIKS.GADS, LAIKS.MENESIS, LAIKS.DATUMS),
ROLLUP(LAIKS.STUDIJU_GADS, LAIKS.SEMESTRIS, LAIKS.DATUMS),
ROLLUP(KOMPONENTS.GRUPA, KOMPONENTS.APAKSGRUPA,
KOMPONENTS.NOSAUKUMS)
ORDER BY KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
```

Att. 54. Optimizēts vaicājums 2. gadījuma atskaitei

Pēdējā solī uzkonstruētais vaicājums 1. gadījuma atskaitei tiek pārveidots, lai iegūtu datus atskaitē no divām versijām saskaņā ar pirmo versiju. Pirmkārt, iegūtais vaicājums pirmajai versijai tiek papildināts ar nosacījumu, kas uzstāda pirmās versijas derīguma periodu (skatīt attēlu 55.).

```
SELECT DISTINCT KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
LAIKS.MENESIS, LAIKS.GADS, LAIKS.SEMESTRIS,
LAIKS.STUDIJU_GADS, KOMPONENTS.APAKSGRUPA, KOMPONENTS.GRUPA,
SUM(AKTIVITATE.HITU_SKAITS)
FROM E_KURSS, KOMPONENTS, LAIKS, AKTIVITATE, LOMA
WHERE (LOMA.ID=AKTIVITATE.LOMA_ID AND
E_KURSS.ID=AKTIVITATE.E_KURSS_ID AND
KOMPONENTS.ID=AKTIVITATE.KOMPONENTS_ID AND
LAIKS.ID=AKTIVITATE.LAIKS_ID) AND
(LAIKS.DATUMS >= TO_DATE('01.09.2004','dd.mm.yyyy') AND
LAIKS.DATUMS <= TO_DATE('01.09.2005','dd.mm.yyyy') AND
E_KURSS.NOSAUKUMS='Datu noliktavas' AND
LOMA.LOMA='Students') AND
LAIKS.DATUMS BETWEEN TO_DATE('01.09.2004','dd.mm.yyyy') AND
TO_DATE('31.01.2005','dd.mm.yyyy')
GROUP BY ROLLUP(LAIKS.GADS, LAIKS.MENESIS, LAIKS.DATUMS),
ROLLUP(LAIKS.STUDIJU_GADS, LAIKS.SEMESTRIS, LAIKS.DATUMS
ORDER BY KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
```

Att. 55. Vaicājums pirmajai aktivitātes zvaigznes shēmas versijai

Tālāk iepriekšējos soļos uzbūvētais vaicājums, kas nesatur pirmās versijas laika ierobežojumus, kas ir redzams attēlā 54., tiek izmantots, lai uzkonstruētu vaicājumu atsevišķi tikai otrajai mainītas zvaigznes shēmas versijai. Pirmkārt, šis vaicājums tiek papildināts ar otrās versijas derīguma perioda nosacījumu (skatīt attēlu 56.).

```
SELECT DISTINCT KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS, LAIKS.MENESIS,
LAIKS.GADS, LAIKS.SEMESTRIS, LAIKS.STUDIJU_GADS,
KOMPONENTS.APAKSGRUPA, KOMPONENTS.GRUPA,
SUM(AKTIVITATE.HITU_SKAITS)
FROM E_KURSS, KOMPONENTS, LAIKS, LOMA,
(SELECT LAIKS_ID, LOMA_ID, E_KURSS_ID, KOMPONENTS_ID,
SUM(HITU_SKAITS) HITU_SKAITS, SUM(DATU_APJOMS) DATU_APJOMS,
STUDIJU_PLANS_ID
FROM AKTIVITATE
GROUP BY LAIKS_ID, LOMA_ID, E_KURSS_ID, KOMPONENTS_ID,
STUDIJU_PLANS_ID) AKTIVITATE
WHERE (LOMA.ID=AKTIVITATE.LOMA_ID AND
E_KURSS.ID=AKTIVITATE.E_KURSS_ID AND
KOMPONENTS.ID=AKTIVITATE.KOMPONENTS_ID AND
LAIKS.ID=AKTIVITATE.LAIKS_ID) AND
(LAIKS.DATUMS >= TO_DATE('01.09.2004','dd.mm.yyyy') AND
LAIKS.DATUMS <= TO_DATE('01.09.2005','dd.mm.yyyy') AND
E_KURSS.NOSAUKUMS='Datu noliktavas' AND
LOMA.LOMA='Students') AND
LAIKS.DATUMS BETWEEN TO_DATE('01.02.2005','dd.mm.yyyy') AND
SYSDATE
GROUP BY ROLLUP(LAIKS.GADS, LAIKS.MENESIS, LAIKS.DATUMS),
ROLLUP(LAIKS.STUDIJU_GADS, LAIKS.SEMESTRIS, LAIKS.DATUMS)
ORDER BY KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
```

Att. 56. Vaicājums otrajai aktivitātes zvaigznes shēmas versijai

Tādēļ, ka otrā versija ir joprojām aktuāla, tad tai derīguma perioda beigu vietā ir izmantota funkcija SYSDATE, kas atgriež pašreizējo laiku. Tālāk tiek analizēti izmantotie shēmas elementi, kas atšķīrās abās versijās. Tas ir mērījums *Hitu skaits*. Algoritms atrod versiju transformāciju šim elementam no otrās versijas uz pirmo un aizvieto FROM vaicājuma daļā faktu tabulu *Aktivitāte*, kas satur mainīto mērījumu, ar apakšvaicājumu, kas pārveido faktu tabulu *Aktivitāte* uz pirmo versiju (skatīt attēlu 56.). Šajā apakšvaicājumā mainītais mērījums tiek aizvietots ar versijas transformāciju, kas summē mērījuma datus, un apakšvaicājums tiek papildināts ar grupēšanas elementiem.

```
SELECT DISTINCT KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS, LAIKS.MENESIS,
LAIKS.GADS, LAIKS.SEMESTRIS, LAIKS.STUDIJU_GADS,
KOMPONENTS.APAKSGRUPA, KOMPONENTS.GRUPA,
SUM(AKTIVITATE.HITU_SKAITS)
FROM E_KURSS, KOMPONENTS, LAIKS, AKTIVITATE, LOMA
WHERE (LOMA.ID=AKTIVITATE.LOMA_ID AND
E_KURSS.ID=AKTIVITATE.E_KURSS_ID AND
KOMPONENTS.ID=AKTIVITATE.KOMPONENTS_ID AND
LAIKS.ID=AKTIVITATE.LAIKS_ID) AND
(LAIKS.DATUMS >= TO_DATE('01.09.2004','dd.mm.yyyy') AND
LAIKS.DATUMS <= TO_DATE('01.09.2005','dd.mm.yyyy') AND
E_KURSS.NOSAUKUMS='Datu noliktavas' AND
LOMA.LOMA='Students') AND
LAIKS.DATUMS BETWEEN TO_DATE('01.09.2004','dd.mm.yyyy') AND
TO_DATE('31.01.2005','dd.mm.yyyy')
GROUP BY ROLLUP(LAIKS.GADS, LAIKS.MENESIS, LAIKS.DATUMS),
ROLLUP(LAIKS.STUDIJU_GADS, LAIKS.SEMESTRIS, LAIKS.DATUMS
UNION
SELECT DISTINCT KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS, LAIKS.MENESIS,
LAIKS.GADS, LAIKS.SEMESTRIS, LAIKS.STUDIJU_GADS,
KOMPONENTS.APAKSGRUPA, KOMPONENTS.GRUPA,
SUM(AKTIVITATE.HITU_SKAITS)
FROM E_KURSS, KOMPONENTS, LAIKS, LOMA,
(SELECT LAIKS_ID, LOMA_ID, E_KURSS_ID, KOMPONENTS_ID,
SUM(HITU_SKAITS) HITU_SKAITS, SUM(DATU_APJOMS) DATU_APJOMS,
STUDIJU_PLANS_ID
FROM AKTIVITATE
GROUP BY LAIKS_ID, LOMA_ID, E_KURSS_ID, KOMPONENTS_ID,
STUDIJU_PLANS_ID) AKTIVITATE
WHERE (LOMA.ID=AKTIVITATE.LOMA_ID AND
E_KURSS.ID=AKTIVITATE.E_KURSS_ID AND
KOMPONENTS.ID=AKTIVITATE.KOMPONENTS_ID AND
LAIKS.ID=AKTIVITATE.LAIKS_ID) AND
(LAIKS.DATUMS >= TO_DATE('01.09.2004','dd.mm.yyyy') AND
LAIKS.DATUMS <= TO_DATE('01.09.2005','dd.mm.yyyy') AND
E_KURSS.NOSAUKUMS='Datu noliktavas' AND
LOMA.LOMA='Students') AND
LAIKS.DATUMS BETWEEN TO_DATE('01.02.2005','dd.mm.yyyy') AND
SYSDATE
GROUP BY ROLLUP(LAIKS.GADS, LAIKS.MENESIS, LAIKS.DATUMS),
ROLLUP(LAIKS.STUDIJU_GADS, LAIKS.SEMESTRIS, LAIKS.DATUMS)
ORDER BY KOMPONENTS.NOSAUKUMS, LAIKS.DATUMS,
SUM(AKTIVITATE.HITU_SKAITS)
```

Att. 57. Vaicājums uz abām shēmas versijām

Algoritma darbības beigās abi vaicājumi uz abām shēmas versijām tiek apvienoti ar UNION operatoru un tiek iegūts gala vaicājums, kas ir redzams attēlā 57. Šīs vaicājums tiek izpildīts un tā rezultāts tiek piegādāts lietotājam.

10.2.4. Pieeju salīdzinājums

Šajā nodaļā tiek salīdzināta piedāvātā pieeja ar komerciālo atskaišu rīku, kas neatbalsta datu noliktavas shēmas versijas. Sākumā pieejas tiek skatītas no lietotāju skata punkta. Šajā kontekstā piedāvātās pieejas galvenā priekšrocība ir jaunas iespējas, kas nebija līdz šim realizētas komerciālajā rīkā. Promocijas darba ietvaros izstrādātajā rīkā lietotājam ir iespēja analizēt informāciju vairākās shēmas versijās gan atsevišķās atskaitēs uz katru versiju vai visu kopā, kas ir ērtāk, bet iepriekš lietotajā atskaišu rīkā lietotājam bija nepieciešams manuāli apvienot informāciju no vairākām atskaitēm.

Atskaišu definēšanas process ir līdzīgs abās pieejās un ir balstīts uz metadatiem, kas apraksta datu noliktavas shēmu. Abās pieejās metadatus, kas apraksta datu noliktavas shēmu, iepriekš definē datu noliktavas administrators. Vienīgi piedāvātajos metadatos datu noliktavas shēma tiek skaidri aprakstīta kā daudzdimensiju modelis saskaņā ar CWM standartu, bet iepriekš lietotajā rīkā shēma ir aprakstīta kā mapes ar elementiem, kas neņem vērā vairākas daudzdimensiju modeļa īpašības. Abiem salīdzinātiem rīkiem ir interneta saskarne, kas neprasa papildus instalācijas.

Diska vieta, kas ir nepieciešama vairāku datu noliktavas shēmas versiju glabāšanai, ir mazāka piedāvātajā pieejā tāpēc, ka saskaņā ar to fiziski relāciju datu bāzē glabājas tikai viens mainītas dimensijas vai faktu tabulas eksemplārs, piemēram, faktu tabula *Aktivitāte* un dimensija *Laiks* apskatītajā piemērā, bet versijas tiek realizētas tikai loģiskā līmeņa metadatos. Bet pieejā, kas neatbalsta datu noliktavas shēmas versijas, bija jāizveido jaunas tabulas *Aktivitāte-jaunā* un *Laiks-jaunā*, kas palielināja datu dublēšanos.

Viena mainītas tabulas eksemplāra glabāšana piedāvātajā pieejā arī atvieglo referenciālas integritātes nodrošināšanu, jo dati netiek dublēti vairākās tabulās.

Piedāvātajā pieejā ir nepieciešami metadati, kas arī aizņem diska vietu, glabājot shēmas metadatus un versijas transformācijas vairākām versijām. Lai samazinātu repozitorija diska vietu, ir iespējams arhivēt vai dzēst vecākās datu noliktavas shēmas versijas saskaņā ar administratora lēmumu, ja šīs versijas vairāk netiek izmantotas.

Atskaišu izpildīšanas laiks bija līdzīgs abās pieejās, jo ģenerētie vaicājumi pēc uzbūves ir līdzīgi. Galvenā atšķirība ir tā, ka piedāvātajā pieejā ir izmantota viena mainīta tabula un analizējamie mērījumi tiek pārveidoti automātiski.

Piedāvātajā rīkā ir iespējams definēt gan lietotāju tiesības uz atskaitēm, gan uz konkrētiem datiem, bet salīdzinātais iepriekš lietotais rīks ļauj definēt tiesības tikai uz atskaitēm. Oracle Discoverer rīkā lietotājam jābūt pieejai visām atskaitē izmantotajām tabulām datu bāzē, bet izstrādātais rīks konstruē vaicājumu datu bāzes pusē un attēlo lietotājam tikai vaicājuma rezultātu, t.i. lietotājam nav tiešas pieejas pie datu bāzes tabulām, kas paaugstina sistēmas drošības līmeni.

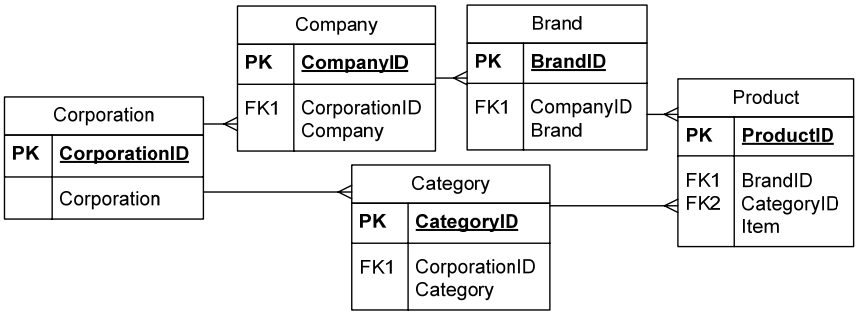
10.3. Piedāvātās pieejas testa piemēri

Šajā apakšnodaļā tiek apskatīti četri testa piemēri piedāvātajai pieejai no zinātniskajām publikācijām par datu noliktavas evolūciju, kā arī divi speciāli izveidoti testa piemēri. Šie piemēri tiek izmantoti, lai pārbaudītu piedāvātas pieejas izmantošanu datu noliktavas evolūcijas problēmu risinājumiem. Šie testa piemēru rezultāti apliecina, ka promocijas darbā piedāvātajā pieejā ir atbalstītas datu noliktavas evolūcijas tipiskas situācijas, piedāvātie datu noliktavas shēmas metadati pilnīgi apraksta datu noliktavas shēmas versijas, un darbā aprakstītās izmaiņu procedūras spēj apstrādāt dažāda veida izmaiņas datu noliktavas shēmā. Piemēri tika izvēlēti tā, lai pārklātu pēc iespējas lielāku promocijas darbā piedāvātajā pieejā atbalstīto izmaiņu kopu. Netika apskatīti tādi piemēri no literatūras, kas satur vienīgi izmaiņas, kas jau tika izpildītas apakšnodaļā 10.2. aprakstītajā piedāvātas pieejas izvērtējumā, balstoties uz e-studiju datuvi.

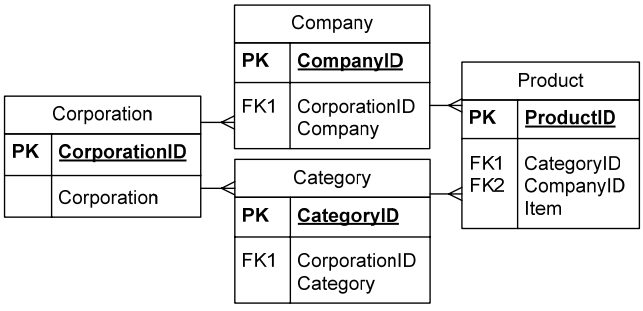
10.3.1. Testa piemērs 1.

Situācijas apraksts

Rakstā [HMV99] viens no aprakstītajiem piemēriem ir dimensijas *Product* evolūcija, kas ir redzama attēlos 58. un 59. Šīs evolūcijas rezultātā dimensijā *Product* tika izdzēsts dimensijas līmenis *Brand*. Lai pārklātu pēc iespējas vairāk piedāvātajā pieejā atbalstīto izmaiņu, tiek pieņemts, ka pēc dimensijas līmeņa dzēšanas tiek izdzēsts arī atribūts *Brand*. Lai pielietotu testa piemēram 1. promocijas darbā piedāvāto pieeju, tiek apskatīts gadījums, kad dimensijai *Product* tiek izveidotas divas shēmas versijas.



Att. 58. Dimensijas *Product* shēma pirms izmaiņām



Att. 59. Dimensijas *Product* shēma pēc izmaiņām

Lai izveidotu jaunu dimensijas *Product* shēmas versiju, tiek izpildītas sekojošas fiziskas un loģiskas izmaiņas:

- Fiziskā izmaiņa:
  - Atribūta *Brand* dzēšana no dimensijas *Product*.
- Loģiskā izmaiņa:
  - Līmeņa *Brand* dzēšana no dimensijas *Product*.

Fiziskais līmenis

Saskaņā ar piedāvāto pieeju fiziski relāciju datu bāzē visas shēmas versijas tiek glabātas vienā fiziskajā struktūrā. Sākotnējā dimensijas versija tiek izveidota atbilstoši shēmai, kas ir redzama attēlā 58. Kad notiek izmaiņas dimensijā, tad fiziski datubāzē tabulā *Product* kolonna *Brand* tiek atstāta, bet vairs netiek aizpildīta ar ETL procesiem. Metadati fiziskajā līmenī paliek bez izmaiņām.

Loģiskais līmenis

Metadatos loģiskajā līmenī tiek izveidotas divas dimensijas versijas atbilstoši shēmām, kas ir redzamas attēlā 58. un attēlā 59. Dimensijas atribūti, hierarhijas un līmeņi, kas paliek nemainīgi abās versijās tiek piesaistīti abām versijām ar versiju transformācijām bez konvertācijas funkcijām.

Atribūts *Brand* tiek pievienots tikai pirmajai dimensijas versijai. Ja šis atribūts var tikt izrēķināts no pārējiem dimensijas *Product* atribūtiem ar konvertācijas funkciju F, tad tiek veidota versijas transformācija izdzēstam atribūtam *Brand* jaunākajā shēmas versijā, kas ir redzama attēlā 59. Ja atribūtu *Brand* izrēķināt nevar, tad versijas transformācija šim atribūtam jaunajā versijā netiek veidota.

Līmenis *Brand* arī tiek pievienots tikai pirmajai dimensijas versijai ar versijas transformāciju bez konvertācijas funkcijas. Arī jaunajā shēmas versijā netiek veidotas šī līmeņa saistības ar produktu kompāniju hierarhiju (*Corporation->Company->Product*). Šai hierarhijai līmeņiem *Company* un *Corporation* līmeņa indekss tiek samazināts par vienu. Versijas transformācija jaunajā shēmas versijā izdzēstajam līmenim netiek veidota.

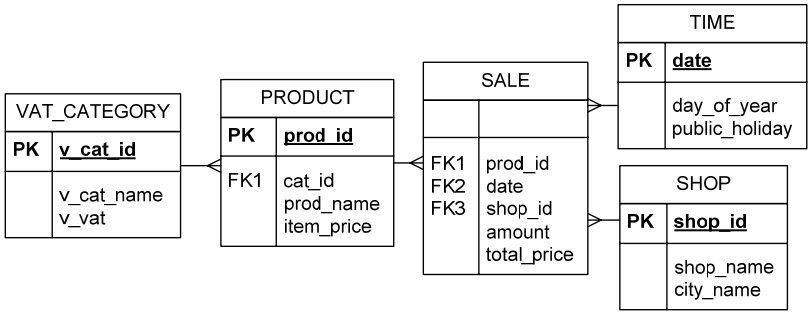
Semantiskais līmenis

Metadati semantiskajā līmenī paliek bez izmaiņām.

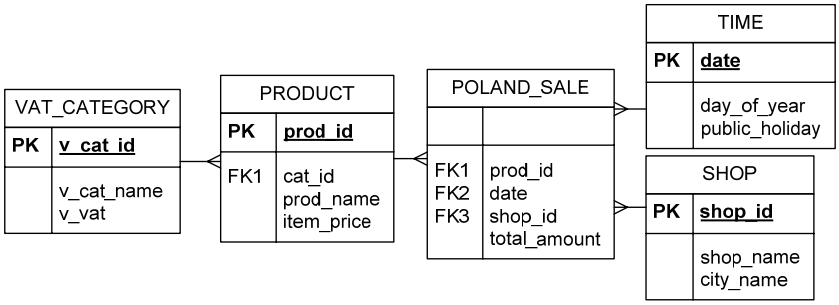
10.3.2. Testa piemērs 2.

Situācijas apraksts

Rakstā [MW04] tiek aprakstītas dažādas izmaiņas, kas notiek produktu pārdošanas zvaigznes shēmā. Lai ilustrētu promocijas darbā piedāvāto pieeju, testa piemērā 2. tiek izmantotas izmaiņas, kas netika ilustrētas iepriekšējā testa piemērā 1. un e-studiju datuves evolūcijas gadījumā. Attēlos 60. un 61. tiek dotas produktu pārdošanas zvaigznes shēmas versijas. Pirmajā produktu pārdošanas zvaigznes shēmas evolūcijas rezultātā tika izdzēsts mērījuma *total_price* no faktu tabulas *SALE*, kas savukārt tika pārsaukta par *POLAND_SALE*, kā arī tika mainīts mērījuma *amount* nosaukums uz jaunu nosaukumu *total_amount*.



Att. 60. Produktu pārdošanas zvaigznes shēmas pirmā versija



Att. 61. Produktu pārdošanas zvaigznes shēmas otrā versija

Lai izveidotu otru produktu pārdošanas zvaigznes shēmas versiju, tiek izpildītas sekojošās fiziskas un semantiskas izmaiņas:

- Fiziskās izmaiņas:
  - Faktu tabulas *SALE* pārsaukšana par *POLAND_SALE*;
  - Mērījuma *total_price* dzēšana;
  - Mērījuma *amount* pārsaukšana par *total_amount*,
- Semantiskā izmaiņa:
  - Dimensijas *PRODUCT* atribūta *item_price* nozīmes maiņa no 'Cena Polijas zlotos' uz 'Cena eiro'.

Fiziskais līmenis

Saskaņā ar piedāvāto pieeju fiziski relāciju datu bāzē visas shēmas versijas tiek glabātas vienā fiziskajā struktūrā. Sākotnējā zvaigznes shēmas versija tiek izveidota atbilstoši shēmai, kas ir redzama attēlā 60. Kad notiek shēmas izmaiņas, tad fiziski datubāzē un fiziskā līmeņa metadatos faktu tabula *SALE* tiek pārsaukta par *POLAND_SA*LE un tabulas *SALE* kolonna *amount* tiek pārsaukta par *total_amount*. Faktu tabulas kolonna *total_price* datubāzē un fiziskā līmeņa metadatos tiek atstāta.

Loģiskais līmenis

Loģiskā līmeņa metadatos tiek izveidotas divas zvaigznes shēmas versijas atbilstoši shēmām, kas ir redzamas attēlā 60. un attēlā 61. Shēmas elementi, kas paliek nemainīgi abās versijās tiek piesaistīti abām versijām ar versiju transformācijām bez konvertācijas funkcijām.

Mērījums *total_price* tiek pievienots tikai pirmajai shēmas versijai. Ja šis mērījums var tikt izrēķināts no pārējiem faktu tabulas *SALE* mērījumiem ar konvertācijas funkciju F, tad tiek veidota versijas transformācija izdzēstajam mērījumam *total_price* jaunākajā shēmas

versijā, kas ir redzama attēlā 61. Ja mērījumu *total_price* izrēķināt nevar, tad versijas transformācija šim mērījumam jaunajā versijā netiek veidota.

Loģiskā līmeņa metadatos otrajā shēmas versijā mērījums *amount* tiek izveidots ar jauno nosaukumu *total_amount*. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista kolonnu *total_amount* fiziskajā līmenī un mērījumu *total_amount* loģiskajā līmenī. Tiek veidota versijas transformācija, kas sasaista mērījumu *amount* pirmajā versijā un mērījumu *total_amount* otrajā versijā ar tukšu funkciju.

Loģiskajā līmenī otrajā shēmas versijā faktu tabula *SALE* tiek izveidota ar jauno nosaukumu *POLAND_SALE*. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista faktu tabulas *POLAND_SALE* kolonnas fiziskajā līmenī un faktu tabulas *POLAND_SALE* mērījumus loģiskajā līmenī. Tiek veidota versijas transformācija, kas sasaista faktu tabulu *SALE* pirmajā versijā un faktu tabulu *POLAND_SALE* otrajā versijā bez konvertācijas funkcijas.

Visbeidzot loģiskā līmeņa metadatos otrajai shēmas versijai netiek piesaistīts atribūts *item_price*, kam mainījās nozīme, bet tiek piesaistīts atribūts *item_price_2._versija* caur tabulu VersijasTransformācija. Atribūts *item_price* nepiedalās nevienā hierarhijā, tāpēc arī atribūtam *item_price_2._versija* netiek izveidoti metadati par šī atribūta saistību ar kādiem hierarhiju līmeņiem.

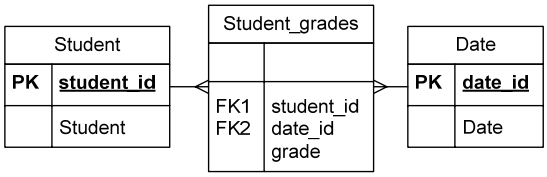
Semantiskais līmenis

Semantiskā līmeņa metadatos tiek izveidota jauna termina versija 'Cena eiro', kas atbilst terminam 'Cena', kas apraksta atribūtu *item_price*. Šī jaunā termina versija tiek piesaistīta jaunajai shēmas versijai un atribūtam *item_price_2._versija*.

10.3.3. Testa piemērs 3.

Situācijas apraksts

Rakstā [MW04] arī tiek apskatīts piemērs ar augstskolu informācijas sistēmu, kur faktu tabulā *Student_grades* glabājas studentu atzīmes (attēls 62.). Vienā šīs shēmas versijā atzīmes tiek glabātas ASV standartā kā vērtības A, B, C, un D, un otrajā shēmas versijā atzīmes tiek glabātas Polijas standartā kā vērtības 5, 4, 3, un 2.



Att. 62. Studentu atzīmju zvaigznes shēma

Lai izveidotu otru studentu atzīmju zvaigznes shēmas versiju, jāizpilda sekojošā fiziskā izmaiņa:

- Fiziskā izmaiņa:
  - Mērījuma *grade* datu tipa maiņa no simbolu virknes, piemēram, *varchar*, uz skaitli, piemēram, *integer*.

Fiziskais līmenis

Saskaņā ar piedāvāto pieeju fiziski relāciju datu bāzē visas shēmas versijas tiek glabātas vienā fiziskajā struktūrā. Sākotnēja zvaigznes shēmas versija tiek izveidota atbilstoši shēmai, kas ir redzama attēlā 62. Kad notiek shēmas izmaiņas, tad fiziski datubāzē un fiziskā līmeņa metadatos tabulai *Student_grades* tiek pievienota jauna kolonna *grade_2_versija* ar jauno datu tipu *integer*.

Loģiskais līmenis

Loģiskā līmeņa metadatos tiek izveidotas divas zvaigznes shēmas versijas. Pirmā zvaigznes shēmas versija atbilst zvaigznes shēmai, kas ir redzama attēlā 62. Otrā zvaigznes shēmas versija tiek izveidota ar visiem tiem pašiem shēmas elementiem, kas bija pirmajā versijā, tikai mērījums *grade* netiek izveidots. Otrajā shēmas versijā tiek izveidots jaunais mērījums *grade_2_versija*. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista kolonnu *grade_2_versija* fiziskajā līmenī un mērījumu *grade_2_versija* loģiskajā līmenī. Tiek veidota versijas transformācija, kas pārveido mērījumu *grade* pirmajā versijā uz mērījumu *grade_2_versija* otrajā versijā ar konvertācijas funkciju: *CASE WHEN grade_2_versija=5 THEN grade='A' WHEN grade_2_versija=4 THEN grade='B' WHEN grade_2_versija=3 THEN grade='C' WHEN grade_2_versija=2 THEN grade='D' END.* Ir iespējams izveidot arī versijas transformāciju mainītajam mērījumam no otrās versijas uz pirmo ar līdzīgu konvertācijas funkciju.

Semantiskais līmenis

Metadatos semantiskajā līmenī mērījumu *grade* definējošam terminam tiek izveidota jauna termina versija 'Atzīmes Polijas standartā'. Šī jaunā termina versija tiek savienota ar mērījumu *grade_2_versija*.

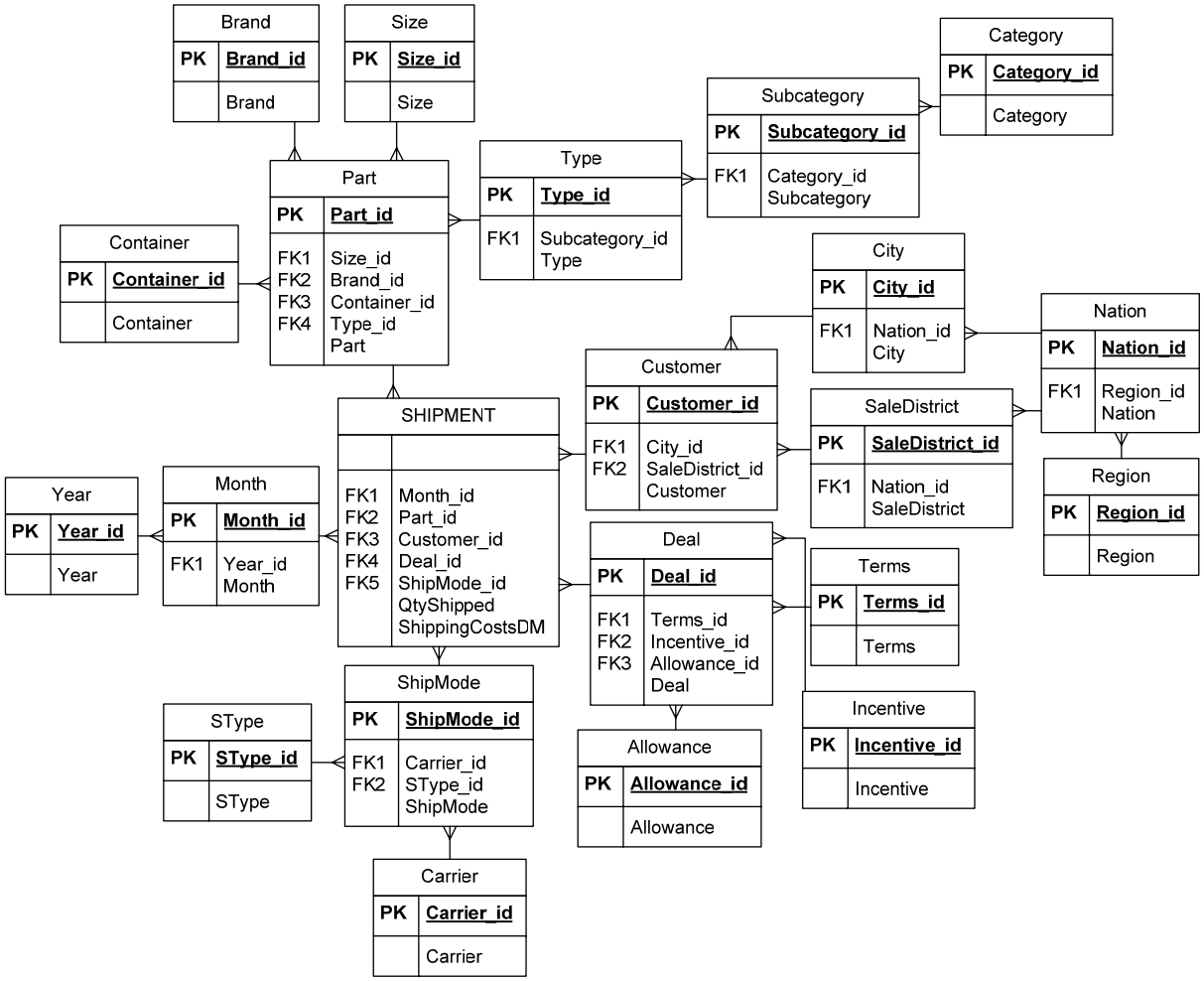
10.3.4. Testa piemērs 4.

Situācijas apraksts

Rakstā [GLR+06] tiek apskatīts piemērs ar detaļu sūtījumu zvaigznes shēmas versijām. Lai analizētu iepriekšējos piemēros neapskatītās izmaiņas datu noliktavas shēmā, šajā piemērā ir atstātas tikai divas izmaiņas no rakstā minētajām, kuru rezultātā tiek izveidotas divas zvaigznes shēmas versijas, kas ir redzamas attēlos 63. un 64. Šīs zvaigznes shēmas evolūcijas rezultātā tika izdzēsta pirmajā shēmas versijā eksistējošā dimensija *ShipMode*.

Lai izveidotu otru sūtījumu zvaigznes shēmas versiju, tiek izpildītas sekojošas fiziskas un loģiskas izmaiņas:

- Fiziskā izmaiņa:
  - Dimensijas *ShipMode* dzēšana.
- Loģiskā izmaiņa:
  - Dimensijas *ShipMode* atvienošana no faktu tabulas *SHIPMENT*.



Att. 63. Sūtījumu zvaigznes shēmas pirmā versija

Fiziskais līmenis

Lai varētu izpildīt fizisko izmaiņu (dimensijas dzēšana), sākumā ir jāizpilda loģiskā izmaiņa (dimensijas atvienošana no faktu tabulas). Saskaņā ar piedāvāto pieeju fiziski relāciju datu bāzē visas shēmas versijas tiek glabātas vienā fiziskajā struktūrā. Sākotnējā zvaigznes shēmas versija tiek izveidota atbilstoši shēmai, kas ir redzama attēlā 63. Kad notiek shēmas izmaiņas, tad fiziski datubāzē dimensijas tabulā *ShipMode* tiek izveidots fiktīvs ieraksts (piemēram, ar datiem 'viss kopā') ar identifikatoru *FiktīvsID*. Datubāzē un arī fiziskā līmeņa metadatos dimensija netiek izdzēsta.

Loģiskais līmenis

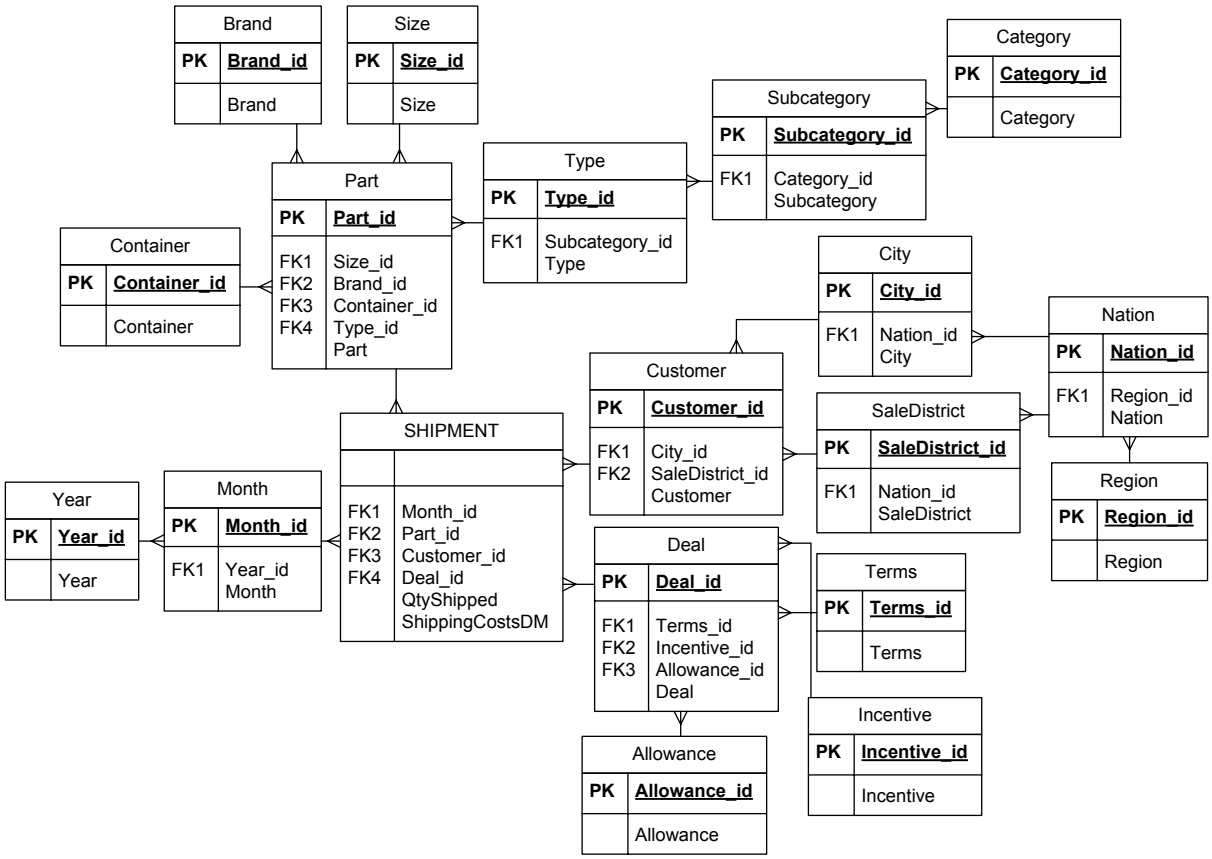
Loģiskā līmeņa metadatos tiek izveidotas divas zvaigznes shēmas versijas. Pirmā zvaigznes shēmas versija atbilst zvaigznes shēmai, kas ir redzama attēlā 63. Otrā zvaigznes shēmas versija tiek izveidota ar visiem tiem pašiem shēmas elementiem, kas bija pirmajā versijā, izņemot dimensiju *ShipMode*, šīs dimensijas atribūtus, hierarhijas un līmeņus, kā arī savienojumu starp dimensiju *ShipMode* un faktu tabulu *SHIPMENT*.

Lai varētu izpildīt atskaites uz divām datu noliktavas shēmas versijām, tiek izveidotas versijas transformācijas faktu tabulas *SHIPMENT* mērījumiem, ja tās ir iespējams izveidot. Versijas transformācijā no pirmās versijas uz otro versiju apkopo mērījumu datus otrajā

versijā. Versijas transformācijā no otrās versijas uz pirmo versiju sadala mērījumu datus otrajā versijā, ja tas ir iespējams.

Semantiskais līmenis

Metadati semantiskajā līmenī paliek bez izmaiņām.

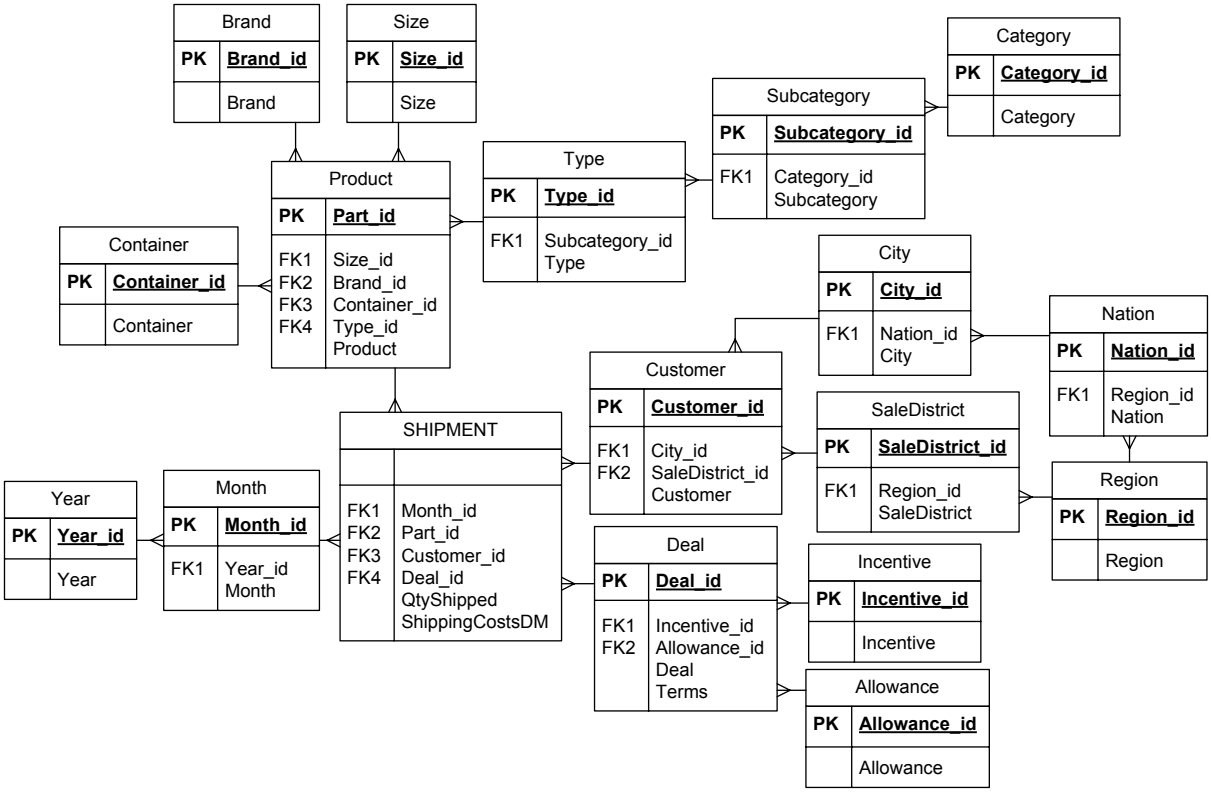


Att. 64. Sūtījumu zvaigznes shēmas otrā versija

10.3.5. Testa piemērs 5.

Situācijas apraksts

Šis testa piemērs tika izveidots, lai analizētu izmaiņas datu noliktavas shēmā, kas netika aprakstītas iepriekšējos testa piemēros 1.-4, kas ir sastopami zinātniskajos rakstos par datu noliktavas evolūciju. Par pamatu testa piemēram 5. tiek izmantota sūtījumu zvaigznes shēma no raksta [GLR+06]. Lai atspoguļotu vairākas izmaiņas, tiek izveidota sūtījumu zvaigznes shēmas trešā versija (attēls 65.), kas tiek iegūta no otrās shēmas versijas (attēls 64.). Šajā trešajā zvaigznes shēmas versijā tiek pārsaukta dimensija *Part* par *Product* un atribūts *Part* tiek pārsaukts par *Product*, atribūtam *Incentive* tiek mainīts datu tips no skaitļa uz simbolu virkni, tiek mainīta rajonu hierarhijas struktūra, jo no tās tiek izmests līmenis *Nation*, kā arī tiek mainīta noteikumu hierarhijas struktūra, jo no tās tiek izmests līmenis *Terms*, un visbeidzot tiek mainīta nozīme mērījumam *QtyShipped*.



Att. 65. Sūtījumu zvaigznes shēmas trešā versija

Lai izveidotu trešo sūtījumu zvaigznes shēmas versiju, jāizpilda sekojošas fiziskas, loģiskas un semantiskas izmaiņas:

- Fiziskās izmaiņas:
  - Dimensijas *Part* pārsaukšana par *Product*;
  - Dimensijas atribūta *Part* pārsaukšana par *Product*;
  - Dimensijas atribūta *Incentive* datu tipa maiņa no *integer* uz *varchar*.
- Loģiskās izmaiņas:
  - Līmeņa *Nation* dzēšana no pārdošanas rajonu hierarhijas (hierarhija pirms izmaiņas: *Region->Nation->SaleDistrict->Customer*, hierarhija pēc izmaiņas: *Region-> SaleDistrict->Customer*),
  - Atribūta *Terms* atvienošana no līmeņa *Terms*;
  - Dimensijas *Deal* noteikumu hierarhijas (*Terms->Deal*) dzēšana.
- Semantiskā izmaiņa:
  - Faktu tabulas *SHIPMENT* mērījuma *QtyShipped* nozīmes maiņa no 'Nosūtīts preču skaits' uz 'Nosūtīts konteineru skaits'.

Fiziskais līmenis

Saskaņā ar piedāvāto pieeju fiziski relāciju datu bāzē visas shēmas versijas tiek glabātas vienā fiziskajā struktūrā. Sākotnējā zvaigznes shēmas versija tiek izveidota atbilstoši shēmai, kas ir redzama attēlā 64. Kad notiek shēmas izmaiņas, tad fiziski datubāzē un fiziskā līmeņa metadatos dimensijas tabula *Part* tiek pārsaukta par *Product* un tabulas *Product* kolonna *Part* tiek pārsaukta par *Product*, kā arī tabulai *Incentive* tiek pievienota jaunā kolonna *Incentive_3_versija* ar jauno datu tipu *varchar*.

Loģiskais līmenis

Loģiskā līmeņa metadatos tiek izveidotas divas zvaigznes shēmas versijas atbilstoši shēmām, kas ir redzamas attēlā 64. un attēlā 65. Shēmas elementi, kas paliek nemainīgi abās versijās tiek piesaistīti abām versijām ar versiju transformācijām bez konvertācijas funkcijām.

Loģiskā līmeņa metadatos trešajā shēmas versijā (attēls 66.) atribūts *Part* tiek izveidots ar jauno nosaukumu *Product*. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista kolonnu *Product* fiziskajā līmenī un atribūtu *Product* loģiskajā līmenī. Tiek veidota versijas transformācija, kas sasaista atribūtu *Part* otrajā versijā un atribūtu *Product* trešajā versijā bez konvertācijas funkcijas.

Arī loģiskajā līmenī trešajā shēmas versijā dimensija *Part* tiek izveidota ar jauno nosaukumu *Product*. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista dimensijas tabulas *Product* kolonnas fiziskajā līmenī un dimensijas *Product* atribūtus loģiskajā līmenī. Tiek veidota versijas transformācija, kas sasaista dimensiju *Part* otrajā versijā un dimensiju *Product* trešajā versijā bez konvertācijas funkcijas.

Loģiskajā līmenī trešā shēmas versija arī nesatur versijas transformāciju, kas piesaista versijai metadatus par atribūta *Terms* savienojumu ar līmeni *Terms*, versijas transformāciju, kas piesaista versijai līmeni *Terms*, un versijas transformāciju, kas piesaista versijai dimensijas *Deal* noteikumu hierarhiju.

Arī loģiskā līmeņa metadatos trešajai zvaigznes shēmas versijai netiek piesaistīts atribūts *Incentive*. Trešajā shēmas versijā tiek izveidots jaunais atribūts *Incentive_3_versija*. Tiek izveidota transformācija ar funkciju „kopija", kas sasaista kolonnu *Incentive_3_versija* fiziskajā līmenī un atribūtu *Incentive_3_versija* loģiskajā līmenī. Tiek veidota versijas transformācija, kas pārveido atribūtu *Incentive* otrajā zvaigznes shēmas versijā uz atribūtu *Incentive_3_versija* trešajā versijā ar konvertācijas funkciju: *TO_CHAR(Incentive)*.

Visbeidzot loģiskā līmeņa metadatos trešajai shēmas versijai netiek piesaistīts mērījums *QtyShipped*, kam mainījās nozīme, bet tiek piesaistīts mērījums *QtyShipped_3_versija* caur tabulu VersijasTransformācija.

Semantiskais līmenis

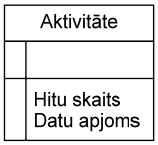
Semantiskā līmeņa metadatos tiek izveidota jauna termina versija 'Nosūtīts konteineru skaits', kas atbilst terminam 'Nosūtīts skaits', kas apraksta mērījumu *QtyShipped*. Šī jaunā termina versija tiek piesaistīta jaunajai shēmas versijai un mērījumam *QtyShipped_3_versija*.

Arī metadatos semantiskajā līmenī atribūtu *Incentive* definējošam terminam tiek izveidota jaunā termina versija 'Incentive conditions'. Šī jaunā termina versija tiek savienota ar atribūtu *Incentive_3_versija*.

10.3.6. Testa piemērs 6.

Situācijas apraksts

Šis testa piemērs tika izveidots, lai atspoguļotu divas datu noliktavas shēmas izmaiņas: faktu tabulas izveidošanu un dzēšanu. Šo izmaiņu atspoguļošanai tiek izmantota aktivitātes faktu tabula no e-studiju datuves piemēra ar diviem mērījumiem *Hitu skaits* un *Datu apjoms*.



Att. 66. Aktivitātes faktu tabula

Lai izveidotu jaunu faktu tabulu un pēc tam izdzēstu to, lai atspoguļotu abas iepriekš neapskatītās izmaiņas ar faktu tabulu (dzēšanu un izveidošanu), ir jāizpilda fiziskās izmaiņas:

- Fiziskās izmaiņas:
  - Jauna faktu tabula *Aktivitāte*;
  - Faktu tabulas *Aktivitāte* dzēšana.

Fiziskais līmenis

Kad tiek izveidota jauna faktu tabula *Aktivitāte*, tad datubāzē tiek izveidota jauna tabula, atbilstoši shēmai attēlā 66., fiziskajā līmenī metadatos tiek izveidoti jaunas tabulas metadati tabulās Tabula un KolonnuKopa. Tiek izveidoti tabulas *Aktivitāte* kolonnu metadati.

Kad faktu tabula *Aktivitāte* tiek izdzēsta, tad fiziski datubāzē atbilstošo tabulu atstāj. Arī metadati fiziskajā līmenī paliek bez izmaiņām.

Loģiskais līmenis

Kad jaunā faktu tabula *Aktivitāte* tiek izveidota, loģiskajā līmenī metadatos tiek izveidota jaunā datuves shēmas versija, tiek izveidots jaunais faktu tabulas elements *Aktivitāte* ar mērījumiem *Hitu skaits* un *Datu apjoms*. Tiek izveidoti transformācijas elementi ar funkciju „kopija", kas sasaista tabulas *Aktivitāte* kolonnas fiziskajā līmenī un mērījumus loģiskajā līmenī.

Kad faktu tabula *Aktivitāte* tiek izdzēsta, tad metadatos loģiskajā līmenī tiek izveidota jaunā datu noliktavas shēmas versija V_2, kurai netiek piesaistīta faktu tabula *Aktivitāte* un šīs faktu tabulas mērījumi, ja shēmā vēl eksistē citi shēmas elementi. Ja shēmā vairs nav citu elementu, izņemot faktu tabulu *Aktivitāte*, tad jaunā shēmas versija netiek izveidota, bet esošai shēmas versijai tiek aizpildīts derīguma beigu datums ar izmaiņas izpildīšanas datumu.

Semantiskais līmenis

Kad jaunā faktu tabula tiek izveidota, metadatos semantiskajā līmenī tiek izveidoti jaunie termini (piemēram, 'Klikšķu skaits' un 'Pārsūtīto datu apjoms') ar terminu versijām (piemēram, 'Kopējais klikšķu skaits' un 'Kopējais pārsūtīto datu apjoms'), lai aprakstītu jaunās faktu tabulas mērījumus *Hitu skaits* un *Datu apjoms* un lai izmantotu mērījumus atskaišu definēšanā.

Kad faktu tabula *Aktivitāte* tiek izdzēsta, tad metadati semantiskajā līmenī nemainās.

10.4. Nodaļas secinājumi

Šajā nodaļā aprakstīti testa piemēri no zinātniskiem rakstiem par datu noliktavas evolūciju un pielietojuma piemērs, kas ir balstīts uz e-studiju datuves evolūciju, kas tika aprakstīts apakšnodaļā 10.2., lai pārbaudītu visas promocijas darbā piedāvātajā pieejā atbalstītās datu noliktavas shēmas izmaiņas. Tabulā 13. tiek apkopotas izpildītās fiziskās, loģiskās un semantiskās izmaiņas datu noliktavas shēmā, kas analizētas testa piemēros un piemērā no aprobācijas projekta. Katrai izmaiņai tabulā ir atzīmēti testa piemēri, kuros šī izmaiņa tika realizēta, ar simbolu ✓.

Apskatītie testa piemēri pārbauda piedāvātās pieejas metadatu pilnīgumu un spēju aprakstīt atbalstītās izmaiņas datu noliktavas shēmā. Testa piemēru analīzes rezultātā tika konstatēts, ka promocijas darbā piedāvātie metadati spējīgi korekti aprakstīt visus nepieciešamos datu noliktavas shēmas elementus, kā arī vairākas shēmas versijas, un darbā piedāvātās izmaiņu apstrādes procedūras korekti pielāgo esošās datu noliktavas shēmas realizāciju datu bāzē, fiziskā, loģiskā un semantiskā līmeņa metadatus atbalstītajām izmaiņām. Ja datu noliktavā notiek citas promocijas darbā piedāvātajā pieejā neatbalstītas izmaiņas, tad šīs izmaiņas jāizpēta, jāizveido izmaiņu apstrādes procedūras, kas pielāgo esošo datu noliktavu un metadatus, kā arī nepieciešamības gadījumā jāveic piedāvāto metadatu papildināšana.

Piedāvātās pieejas daudzversiju datu noliktavas atbalstam un atskaišu uz vairākām versijām ģenerēšanai salīdzinājuma rezultāti ar tradicionālo pieeju tika publicēti rakstā:

Solodovnikova D. 'An Approach to Supporting Data Warehouse Schema Versions: Overview and Case Study'. *Proceedings of the 13th International Conference on Advances in Databases and Information Systems, Riga, Latvia*, 2009.

Tabula 13. Datu noliktavas shēmas izmaiņu piemēru apkopojums

		E-studiju datuves piemērs	Testa piemērs 1.	Testa piemērs 2.	Testa piemērs 3.	Testa piemērs 4.	Testa piemērs 5.	Testa piemērs 6.
Fiziskās izmaiņas	Jauna atribūta pievienošana dimensijai	✓						
	Dimensijas atribūta datu tipa maiņa						✓	
	Dimensijas atribūta dzēšana		✓					
	Dimensijas atribūta pārsaukšana						✓	
	Jaunas dimensijas izveidošana	✓						
	Dimensijas dzēšana					✓		
	Dimensijas pārsaukšana						✓	
	Jauna mērījuma pievienošana faktu tabulai	✓						
	Mērījuma datu tipa maiņa				✓			
	Mērījuma dzēšana			✓				
	Mērījuma pārsaukšana			✓				
	Jaunas faktu tabulas izveidošana							✓
	Faktu tabulas dzēšana							✓
	Faktu tabulas pārsaukšana			✓				
Loģiskās izmaiņas	Dimensijas piesaistīšana faktu tabulai	✓						
	Dimensijas atvienošana no faktu tabulas					✓		
	Jaunā dimensijas hierarhija	✓						
	Dimensijas hierarhijas dzēšana						✓	
	Jaunais hierarhijas līmenis	✓						
	Līmeņa dzēšana no hierarhijas						✓	
	Līmeņa dzēšana no dimensijas		✓					
	Līmeņa pievienošana hierarhijai	✓						
	Atribūta piesaistīšana līmenim	✓						
	Atribūta atvienošana no līmeņa						✓	
Semantiskās izmaiņas	Atribūta nozīmes maiņa			✓				
	Mērījuma nozīmes maiņa						✓	

NOBEIGUMS

Promocijas darbā tika izpētītas problēmas, kas parādās datu noliktavas dzīves cikla laikā. Šīs problēmas ir datu noliktavas uzturēšana saskaņā ar izmaiņām datu avotu datos, pielāgošana, kad mainās datu noliktavas shēma saskaņā ar jaunām darījumprasībām vai citu iemeslu dēļ, adaptācija, kad mainās datu avotu shēma, un uzturēšana pēc adaptācijas. Katrai datu noliktavas evolūcijas problēmai tika izpētīti iespējamie risinājumi literatūrā. Šie risinājumi ir atkarīgi no datu noliktavas jēdziena interpretācijas. Tika izdalītas divas datu noliktavas interpretācijas, kas ir materializēto skatu kopa uz datu avotu datiem un daudzdimensiju datu bāze, kur ETL procesi veic datu iegūšanu no datu avotiem, pārveidošanu un ielādi datu noliktavā.

Tika konstatēts, ka literatūrā minētie un darbā analizētie risinājumi var tikt piemēroti tikai viena veida evolūcijas problēmām un nevar tikt kombinēti, bet autore uzskata, ka datu noliktavas attīstības laikā ir jāspēj atrisināt pēc iespējas vairāk problēmu. Vairāki no eksistējošiem risinājumi nesaglabā datu noliktavas attīstības vēsturi. Lai korekti tiktu apstrādātas visas datu noliktavas evolūcijas situācijas un netiktu pazaudētas datu vēsturiskas izmaiņas, ir vēlams izmantot datu noliktavas shēmu versiju pieeju, kas arī ir jaunākais ar datu noliktavas evolūciju saistītais pētījumu virziens. Šī virziena būtība ir tā, ka tā paredz vienlaicīgu vairāku shēmu versiju esamību. Shēmas versija ir shēma, kas atspoguļo darījumprasības noteiktā laika periodā, ko sauc par versijas derīguma periodu, kas sākas pēc shēmas izveides un ilgst līdz nākamās versijas izveidošanai. Šajā jomā arī pastāv vairākas neatrisinātas problēmas, piemēram, daudzversiju datu noliktavas shēmu glabāšana relāciju datu bāzē, vairāku versiju atspoguļošana metadatos, atskaišu un vaicājumu konstruēšana un eksistējošo atskaišu adaptēšana, lai atspoguļotu vairāku versiju esamību.

Šī darba galvenais ieguldījums ir pieeja daudzversiju datu noliktavas atbalstam, kas iekļauj gan fizisko shēmas versiju reprezentāciju datu bāzē, gan iespēju modelēt datu noliktavas loģisko shēmu vairākām versijām, kā arī algoritms atskaišu konstruēšanai un izpildīšanai uz vairākām datu noliktavas shēmas versijām.

Viens no galvenajiem promocijas darba rezultātiem ir piedāvātais datu noliktavas metamodelis, kas veidots balstoties uz CWM metadatu standartu, atbilstošās pakotnes paplašinot ar shēmas versiju aprakstīšanai vajadzīgiem metadatiem. Metadati sastāv no fiziskā, loģiskā un semantiskā līmeņa un atskaišu specifikācijas. Fiziskajā līmenī tiek aprakstīta datu noliktavas shēmas struktūra relāciju datu bāzē, loģiskajā līmenī tiek atspoguļoti daudzdimensiju datu noliktavas elementi un shēmas versijas, semantiskajā līmenī tiek aprakstīti datu noliktavas shēmas elementi lietotājiem saprotamā biznesa valodā, bet atskaišu metadatos tiek glabāta lietotāju definēto atskaišu uz datu noliktavas shēmām specifikācija. Metadati balstās uz Common Warehouse Metamodel metadatu standartu, kas atļauj veikt metadatu integrāciju un apmaiņu ar citām datu noliktavu lietojumprogrammām, kas arī atbalsta šo standartu. Standarta metadati tika papildināti ar metadatiem, kas apraksta datu noliktavas shēmas versijas un terminu versijas. Galvenā informācija par elementu saistību dažādās versijās tiek aprakstīta ar versiju transformācijām, kas definē, kādā veidā mainītais vai izdzēstais datu noliktavas shēmas elements varētu tikt iegūts no pārējiem shēmas

elementiem. Šīs transformācijas definē datu noliktavas administrators, kad tiek veidota jaunā datu noliktavas shēmas versija.

Promocijas darbā piedāvātais metadatu modelis tika veidots atbilstoši formālajam modelim daudzversiju datu noliktavai, kas apraksta datu noliktavu loģiskajā, fiziskajā un semantiskajā līmenī, kā arī modelis apraksta atskaites uz datu noliktavas shēmas versijām.

Promocijas darbā tika izpētītas arī izmaiņas, kas notiek datu noliktavas evolūcijas rezultātā. Tika iegūts izmaiņu saraksts no literatūras avotos sastopamām izmaiņām un no izmaiņām, kas notika promocijas darba autores praksē, izstrādājot un uzturot datu noliktavas. Tika aprakstītas piedāvātajā pieejā atbalstītās izmaiņas, kas var tikt ierosinātas gan ar mainīgām datu noliktavas darījumprasībām, gan ar izmaiņām datu avotos. Katrai izmaiņai tika aprakstīta formālā procedūra, kas atbilstoši pielāgo datu noliktavas formālā modeļa instanci, un procesa apraksts, kas ir jāveic, lai izplatītu izmaiņas piedāvātajos datu noliktavas shēmas metadatos un fiziskajā datu noliktavas shēmā relāciju datu bāzē.

Viens no darba jauninājumiem ir algoritms SQL vaicājumu konstruēšanai uz vienu vai vairākām datu noliktavas shēmas versijām. Šis algoritms būvē SQL vaicājumu saskaņā ar atskaites definīciju atskaišu metadatos un automātiski pārveido vaicājumu, kas ir konstruēts uz vienu no shēmas versijām, citā vaicājumā, kas ir saskaņots ar citu shēmas versiju, izmantojot versiju transformācijas, kas ir definētas loģiskā līmeņa metadatos. Ja nav atbilstošo versiju transformāciju, tad atskaites rezultāti tiek attēloti atsevišķi dažādām shēmas versijām.

Lai aprobētu promocijas darbā piedāvāto algoritmu SQL vaicājumu būvēšanai un metadatus, tika izstrādāts atskaišu definēšanas un attēlošanas rīks, kas atbalsta daudzversiju datu noliktavas. Galvenais datu noliktavas un atskaišu rīka nolūks ir nodrošināt ātru un ērtu veidu informācijas analīzei, t.i. datu noliktavas atskaitēm jāatbalsta mijiedarbība ar lietotāju, tai skaitā OLAP operācijas rollup, drill-down, slicing. Šim nolūkam algoritms papildina ģenerēto vaicājumu ar SQL paplašinājumiem, lai jau iepriekš aprēķinātu iespējamu OLAP operāciju rezultāta datus.

Promocijas darbā piedāvātās pieejas priekšrocības un ierobežojumi tika novērtēti, baltoties uz aprobācijas projekta ietvaros izstrādātās datuves piemēra, kas tika salīdzināts ar tradicionālu datu noliktavas risinājumu. Tika izveidoti metadati atbilstoši divām aprobācijas projekta ietvaros izstrādātās datuves shēmas versijām un versiju transformācijas. Tika uzbūvētas un izpildītas atskaites promocijas darba ietvaros izstrādātajā atskaišu attēlošanas rīkā. Tika konstatētas vairākas priekšrocības piedāvātai pieejai salīdzinājumā ar tradicionālu datu noliktavas risinājumu, kur pēc autores viedokļa galvenā priekšrocība ir iespēja lietotājam vienkārši, ātri un automātiski attēlot datus no vairākām datu noliktavas shēmas versijām vienā atskaitē.

Promocijas darbā piedāvātā pieeja vēl tika izvērtēta, baltoties uz testa piemēriem no citu autoru zinātniskiem rakstiem par datu noliktavas evolūciju. Ar testa piemēriem tika pārbaudītas visas piedāvātajā pieejā atbalstītās izmaiņas datu noliktavas shēmā un tika konstatēts, ka promocijas darbā piedāvātie metadati spējīgi korekti aprakstīt visus nepieciešamos datu noliktavas shēmas elementus, kā arī vairākas shēmas versijas. Tika secināts, ka darbā piedāvātās izmaiņu apstrādes procedūras korekti pielāgo esošās datu noliktavas shēmas realizāciju datu bāzē, kā arī pielāgo atbalstītajām izmaiņām fiziskā, loģiskā un semantiskā līmeņa metadatus.

Darbā arī ir piedāvāta datu noliktavas evolūcijas arhitektūra, kas atbalsta dažāda veida evolūcijas problēmas, tai skaitā datu noliktavas adaptāciju un pielāgošanu. Arhitektūras daļēja funkcionalitāte tika realizēta prototipā, kurā tika izmantots komerciāls datu noliktavas izstrādes rīks Oracle Warehouse Builder. Arhitektūra sastāv no vairākiem komponentiem, kurus ir iespējams aizvietot ar cita veida komponentiem ar atbilstošu funkcionalitāti, ja ir tāda vajadzība. Mainot vienu arhitektūras komponentu uz citu ir nepieciešamas minimālas vai nekādas izmaiņas citos komponentos, lai arhitektūra strādātu saskaņā ar konkrētas situācijas prasībām.

Evolūcijas arhitektūra tapa papildinot datu noliktavas adaptācijas arhitektūru, kas tika izstrādāta un izmantota datu noliktavas adaptācijai pēc izmaiņām datu avotu shēmās. Izstrādātie metadati un atskaišu rīks ir šīs arhitektūras sastāvdaļas. Izstrādātais uz datu noliktavas shēmas evolūciju orientētais atskaišu rīks tiek lietots Latvijas Universitātē atskaišu definēšanai.

Ir plānots papildināt piedāvāto pieeju dažos virzienos, tai skaitā sarežģītāku versijas transformāciju definēšana, ģenerēto vaicājumu tālāka optimizācija, atskaišu daudzveidības paplašināšana, atskaišu personalizācija.

Pētījumi datu noliktavu jomā ir turpināmi. Viens no tālāka darba iespējamiem virzieniem varētu būt politikas, saskaņā ar kurām varētu automātiski vai automatizēti veidot versijas transformācijas. Saistībā ar semantiskajām izmaiņām ir arī iespējami pētījumu: kā noteikt šāda veida izmaiņas datu noliktavā, vai kā varētu veidot datu versijas.

LITERATŪRA

[AAS+97] Agrawal D., Abbadi A.El., Singh A., Yurek T. „Efficient view maintenance at data warehouses", Proceedings of the 1997 ACM SIGMOD International Conference on Management of data', Tucson, Arizona, USA, 1997, pp. 417-427.

[AMM98] Arocena G., Mendelzon A., Mihaila G.A. „Query Languages for the Web", W3C Workshop: QL, Boston, Massachussets, USA, 1998, [tiešsaiste] – [atsauce 19.09.2009.]. Pieejams: http://www.w3.org/TandS/QL/QL98/pp/wql.html

[Ban07] Banerjee S. „Data Warehouse Schema Evolution with Extended Hierarchy Semantics", PhD thesis, University of Cincinatti, USA, 2007, p. 259.

[BCV99] Bergamaschi S., Castano S., Vincini M. „Semantic Integration of Semistructured and Structured Data Sources", SIGMOD Record 28(1), 1999, pp. 54-59.

[BD09] Banerjee S., Davis, K.C. „Modeling Data Warehouse Schema Evolution over Extended Hierarchy Semantics". Journal on Data Semantics XIII, LNCS, vol. 5530, Springer, Heidelberg, 2009, pp. 72-96.

[Bel02] Bellahsene Z. "Schema Evolution in Data Warehouses", Knowledge and Information Systems, Volume 4(3), Springer, 2002, pp. 283-304.

[BFB08] Bentayeb F., Favre C., Boussaid O. "A User-driven Data Warehouse Evolution Approach for Concurrent Personalized Analysis Needs", Integrated Computer-Aided Engineering, IOS Press, Volume 15, Number 1, 2008, pp. 21-36

[BFM99] Bouzeghoub M., Fabret F., Matulovic-Broque M. „Modeling the Data Warehouse Refreshment Process as a Workflow Application", Proceedings of the Intl. Workshop on Design and Management of Data Warehouses, Heidelberg, Germany, 1999, pp. 6.1–6.12.

[Bla00] Blaschka M. „FIESTA: A Framework for Schema Evolution in Multidimensional Databases", PhD thesis, Technische Universität München, Germany, 2000, p. 214.

[BMB+03] Body M., Miquel M., Bedard Y., Tchounikine A. „Handling Evolutions in Multidimensional Structures", Proceedings of the 19th International Conference on Data Engineering, Bangalore, India, 2003.

[BSH99] Blaschka M., Sapia C., Höfling G. „On Schema Evolution in Multidimensional Databases", Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery, London, UK, 1999, pp. 153–164.

[CAM01] Cobena G., Abiteboul S., Marian A. „Detecting Changes in XML Documents", Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, 2001, pp. 41–52.

[CAW98] Chawathe S.S., Abiteboul S., Widom J. „Representing and Querying Changes in Semistructured Data", Proceedings of the Fourteenth International Conference on Data Engineering, Washington, DC, USA, 1998, pp. 4–13.

[CD97] Chaudhuri S., Dayal U. „An Overview of Data Warehousing and OLAP Technology", SIGMOD Record, Volume 26, Number 1, 1997, pp. 65–74.

[CMM01] Crescenzi V., Mecca G., Merialdo P. „RoadRunner: Towards Automatic Data Extraction from Large Web Sites", Proceedings of 27th International Conference on Very Large Data Bases, Roma, Italy, 2001, pp. 109-118.

[CMM+03] Czejdo B., Messa K., Morzy T., Morzy M., Czejdo J. „Data Warehouses with Dynamically Changing Schemas and Data Sources", Proceedings of the 3rd International Economic Congress, Opportunieties of Change, Sopot, Poland, 2003, p. 10.

[CZR06] Chen S., Zhang X., Rundensteiner E.A. „A Compensation-Based Approach for View Maintenance in Distributed Environments", IEEE Transactions on Knowledge and Data Engineering, Volume 18, Number 8, 2006, pp. 1068-1081.

[EKM01] Eder J., Koncilia C., Morzy T. „A Model for a Temporal Data Warehouse", Proceedings of OES-SEO 2001 Workshop, Rome, Italy, 2001, pp. 48-57.

[EKM02] Eder J., Koncilia C., Morzy T. „The COMET Metamodel for Temporal Data Warehouses", Lecture Notes In Computer Science, Vol. 2348, 2002, pp. 83-99.

[GCB+97] Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M. „Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals", Data Min. Knowl. Discov. 1(1), 1997, pp. 29-53.

[GLR+04] Golfarelli M., Lechtenbörger J., Rizzi S., Vossen G. „Schema Versioning in Data Warehouses", Lecture Notes In Computer Science, Vol. 3289, 2004, pp. 415-428.

[GLR+06] Golfarelli, M., Lechtenbörger, J., Rizzi, S., Vossen, G. „Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation", Data Knowl. Eng. 59(2), 2006, pp. 435-459.

[GM95] Gupta A., Mumick I. S. „Maintenance of Materialized Views: Problems, Techniques, and Applications", IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Warehousing, Volume 18, Nr. 2, 1995, pp. 19-28.

[GMR+01] Gupta A., Mumich I.S., Rao J., Ross K.A. „Adapting Materialized Views after Redefinitions: Techniques and a Performance Study", Information Systems, Volume 26, 2001, pp. 323-362.

[GMR95] Gupta A., Mumich I.S., Ross K.A. „Adapting Materialized Views after Redefinitions", Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, 1995, pp. 211-222.

[GRV+98] Gruser J.-R., Raschid L., Vidal M.-L., Bright L. „Wrapper Generation for Web Accessible Data Sources", Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems, New York City, New York, USA, 1998, pp. 14-23.

[HMV99] Hurtado C.A., Mendelzon A.O., Vaisman A.A. „Maintaining Data Cubes under Dimension Updates", Proceedings of the 15th International Conference on Data Engineering, Washington, DC, USA, 1999, pp. 346-357.

[HRU96] Harinarayan V, Rajaraman A., Ullman J.D „Implementing Data Cubes Efficiently', Proceedings of the 1996 ACM SIGMOD International Conferenceon Management of Data, Montreal, Quebec, Canada, 1996, pp. 205-216.

[Huy97] Huyn N. "Multiple View Self-Maintenance in Data Warehousing Environments", Proceedings of the 23rd International Conference on Very Large Data Bases, San Francisco, California, USA, 1997, pp. 26-35.

[JLV+03] Jarke M., Lenzerini M., Vassiliou Y., Vassiliadis P., Fundamentals of Data Warehouses, Springer, 2003, p. 219.

[KM00] Kimball R., Merz R., The Data Warehouse Toolkit: Building the Web-Enabled Data Warehouse, John Wiley, 2000, p. 464.

[KPR04] C.E. Kaas, T.B. Pedersen, B.D. Rasmussen. „Schema Evolution for Stars and Snowflakes", Proceedings of the 6th Intl. Conference on Enterprise Information Systems, Porto, Portugal, 2004, pp. 425-433.

[KR02] Kimball R., Ross M., The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, John Wiley, 2002, p. 421.

[KS95] Konopnicki D., Shmueli O. „W3QS: A Query System for the World-Wide Web", Proceedings of 21th International Conference on Very Large Data Bases, Zurich, Switzerland, 1995, pp. 64-65.

[LB03] Lourenco A., Belo O. „Enhancing Web Analysis Through Data Warehousing Enforcement", 1º Simpósio Doutoral do Departamento de Informática, Braga, Outubro, 2003.

[LCR02] Lui B., Chen S., Rundensteiner E.A. „Batch Data Warehouse Maintenance in Dynamic Environments", Proceedings of the Eleventh International Conference on Information and Knowledge Management, McLean, Virginia, USA, 2002, pp. 68-75.

[LNR97] Lee A.J., Nica A., Rundensteiner E.A. „The EVE Framework: View Synchronization In Evolving Environments", Computer Science Technical Report, Worcester Polytechnic Institute, Worcester, MA, USA, 1997, p. 38.

[LSS96] Lakshmanan L.V.S., Sadri F., Subramanian I.N. „A Declarative Language for Querying and Restructuring the Web", Proceedings of Sixth International Workshop on Research Issues in Data Engineering - Interoperability of Nontraditional Database Systems, New Orleans, Louisiana, USA, 1996, pp. 12-21.

[Mar00] Marotta A. „Data Warehouse Design and Maintenance through Schema Transformations", Master thesis, Instituto de Computación – Facultad de Ingenería, Universidad de la República Uruguay, 2000, p. 141.

[MD96] Mohania M., Dong G. „Algorithms for Adapting Materialised Views in Data Warehouses", Proceedings of the International Symposium on Cooperative

Database Systems for Advanced Applications, Kyoto, Japan, 1996, pp. 309-316.

[ML01] Mengchi L., Ling T.W. „A Conceptual Model and Rule-based Query Language for HTML", World Wide Web, Volume 4, 2001, pp. 49-77.

[Moh97] Mohania M. „Avoiding Re-computation: View Adaptation in Data Warehouses", Proceedings of the 8th International Database Workshop, Hong Kong, 1997, pp. 151-165.

[MP03] McBrien P., Poulovassilis A. „Data Integration by Bi-Directional Schema Transformation Rules'. Proceedings of the 19th Intl. Conference on Data Engineering, Bangalore, India, 2003, pp. 227-238

[MV00] Mendelzon, A.O., Vaisman, A.A.: „Temporal Queries in OLAP", Proceedings of the 26th International Conference on Very Large Databases, Cairo, 2000, pp. 242-253

[MW04] Morzy T., Wrembel R. „On Querying Versions of Multiversion Data Warehouse", Proceedings of the 7th ACM International Workshop on Data Warehousing and OLAP, Washington, DC, USA, 2004, pp. 92-101.

[NR98] Nica A., Rundensteiner E.A. „View Maintenance after View Synchronization, Computer Science Technical Report", Worcester Polytechnic Institute, Worcester, MA, USA, 1998, p. 44.

[OMG02] Object Management Group (OMG), Common Warehouse Metamodel (CWM) Specification, Version 1.1, Volume 1, 2002, p. 576

[Ora03] Oracle Corporation, Oracle® Database Data Warehousing Guide 10g Release 1 (10.1), 2003, p. 806

[Ora03a] Oracle Corporation, Oracle® Warehouse Builder User's Guide 10g Release 1 (10.1), 2003, p. 760

[PCT+03] Poodle J., Chang D., Tolbert D., Mellor D., Common Warehouse Metamodel Developer's Guide, Wiley Publishing, 2003, p. 716

[QGM+96] Quass D., Gupta A., Mumich I. S., Windom J. „Making Views Self-Maintainable for Data Warehousing", Proceedings of the Forth International Conference on Parallel and Distributed Information Systems, Miami Beach, Florida, USA, 1996, pp. 158-169.

[Qui99] Quix C. „Repository Support for Data Warehouse Evolution", Proceedings of the Intl. Workshop on Design and Management of Data Warehouses, Heidelberg, Germany, 1999, pp. 4.1-4.9.

[RAL+06] Rizzi S., Abelló A., Lechtenbörger J., Trujillo J., "Research in data warehouse modeling and design: dead or alive?", Proceedings of the 9th ACM International Workshop on Data Warehousing and OLAP (DOLAP '06), Arlington, Virginia, USA, 2006, pp. 3-10.

[RKZ00] Rundensteiner E.A., Koeller A., Zhang X. „Maintaining Data Warehouses over Changing Information Sources", Communications of the ACM, Volume 43, Issue 6, New York, NY, USA, 2000, pp. 57-62.

[RLN97] Rundensteiner E.A., Lee A.J., Nica A. „On Preserving Views In Evolving Environments", Proceedings of the 4th KRDB Workshop, Athens, Greece, 1997, pp. 13.1-13.11.

[Rod95] Roddick J.F. „A Survey of Schema Versioning Issues for Database Systems", Information and Software Technology, Vol.37, No.7, 1995, pp. 383-393.

[SMR+98] Samtani S., Mohania M., Kumar V., Kambayashi Y. „Recent Advances and Research Problems in Data Warehousing", Proceedings of the Workshops on Data Warehousing and Data Mining: Advances in Database Technologies, London, UK, 1998, pp. 81-92.

[SN05] Solodovņikova D., Niedrīte L. „Using Data Warehouse Resources for Assessment of E-Learning Influence on University Processes", Proceedings of the 9th East-European Conference on Advances in Databases and Information Systems, Tallinn, Estonia, 2005, pp. 233–248.

[SN06] Solodovņikova D., Niedrīte L. 'Data Warehouse Adaptation after Changes in Source Schemata'. Proceedings of the 7th International Baltic Conference on Databases and Information Systems, Vilnius, Lithuania, 2006, pp. 52-63.

[SN10] Solodovnikova D., Niedrite L. 'Evolution-Oriented User-Centric Data Warehouse'. Proceedings of the 19th International Conference on Information Systems Development by Springer, Prague, Czech Republic, 2010 (tiks publicēts).

[SNP05] Shahzad, M.K., Nasir, J.A., Pasha, M.A. „CEV-DW: Creation and Evolution of Versions in Data Warehouse", Asian Journal of Information Technology, 4(10), 2005, pp. 910-917

[Sol07] Solodovņikova D. 'Data Warehouse Evolution Framework'. Proceedings of the Fourth Spring Young Researchers Colloquium on Databases and Information Systems, SYRCoDIS'2007, Moscow, Russia, 2007.

[Sol08a] Solodovņikova D. 'The Formal Model for Multiversion Data Warehouse Evolution'. Postconference proceedings of the 8th International Baltic Conference on Databases and Information Systems, Tallinn, Estonia, Frontiers in Artificial Intelligence and Applications by IOS Press, 2008, pp. 91-102.

[Sol08b] Solodovņikova D. 'Metadata to Support Data Warehouse Evolution'. Proceedings of the 17th International Conference on Information Systems Development, Paphos, Cyprus, 2008, pp. 627-635.

[Sol08c] Solodovņikova D. 'Building Queries on Multiple Versions of Data Warehouse'. Proceedings of the 8th International Baltic Conference on Databases and Information Systems, Tallinn, Estonia, 2008, pp. 75-86.

[Sol09] Solodovnikova D. 'An Approach to Supporting Data Warehouse Schema Versions: Overview and Case Study'. Proceedings of the 13th International Conference on Advances in Databases and Information Systems, Riga, Latvia, 2009, pp. 258-265.

[VGD99] Vavouras A., Gatziu S., Dittrich K.R. „Modeling and Executing the Data Warehouse Refreshment Process", Proceedings of International Symposium on

Database Applications in Non-Traditional Environments, Kyoto, Japan, 1999, pp. 66-73.

[VMP03] Velegrakis Y., Miller R.J., Popa L. „Mapping Adaptation under Evolving Schemas", Proceedings of 29th International Conference on Very Large Data Bases, Berlin, Germany, 2003, pp. 584-595.

[VMR+02] Vaisman A.A., Mendelzon A.O., Ruaro W., Cymerman S.G. „Supporting Dimension Updates in an OLAP Server", Proceedings of Advanced Information Systems Engineering: 14th International Conference, Toronto, Canada, 2002, pp. 67-82.

[W3Ca] World Wide Web Consortium (W3C), Logging Control In W3C httpd, The Common Logfile Format, [tiešsaiste] – [atsauce 19.09.2009.]. Pieejams: http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format

[W3Cb] World Wide Web Consortium (W3C), Extensible Markup Language (XML), [tiešsaiste] – [atsauce 19.09.2009.]. Pieejams: http://www.w3.org/XML/

[W3Cc] World Wide Web Consortium (W3C), XQuery 1.0: An XML Query Language, [tiešsaiste] – [atsauce 19.09.2009.]. Pieejams: http://www.w3.org/TR/xquery/

[WB05] Wrembel R., Bebel B. „Metadata Management in a Multiversion Data Warehouse". Proceedings of International Conference on Ontologies, DataBases, and Applications of Semantics, Agia Napa, Cyprus, 2005, pp. 1347-1364

[Web05] WebCT, Inc. Technical Communications Department, WebCT Designer and Instructor Reference WebCT Campus Edition 6.0, 2005, p. 822

[Wid95] Widom J. „Research Problems in Data Warehousing", Proceedings of the Fourth International Conference on Information and Knowledge Management, Baltimore, Maryland, USA, 1995, pp. 25-30.

[Wie98] Wiederhold, G. „Value-added middleware: Mediators" (Prepublication draft), 1998, [tiešsaiste] – [atsauce 19.09.2009.] Pieejams: http://www-db.stanford.edu/pub/gio/1998/dbpd.html

[Yan01] Yang J. „Temporal Data Warehousing", PhDThesis, Stanford University, USA, 2001, p. 222.

[ZB01] Zhu Y., Bornhövd C. „Data Transformation for Warehousing Web Data", Proceedings of the Third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, Washington, DC, USA, 2001, pp. 74-85.

[ZGH+95] Zhuge Y., Garcia-Molina H., Hammer J., Widom J. „View Maintenance in a Warehousing Environment", Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, 1995, pp. 316-327.

[ZGW96] Zhuge Y., Garcia-Molina H., Wiener J.L. „The Strobe Algorithms for Multi-Source Warehouse Consistency", Proceedings of the International Conference on Parallel and Distributed Information Sources, Miami Beach, Florida, USA, 1996, pp. 146-157.

[ZGW98] Zhuge Y., Garcia-Molina H., Wiener J.L. „Consistency Algorithms for Multi-Source Warehouse View Maintenance", Distributed and Parallel Databases, Volume 6, Issue 1, 1998, pp. 7-40.

[Zhu99] Zhu Y. „A Framework for Warehousing the Web Contents", Proceedings of the Third Int'l Workshop Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'01), Hong Kong, 1999, p. 10

[ZR00] Zhang X., Rundensteiner E.A. „DyDa: Data Warehouse Maintenance in a Fully Concurrent Environment", Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery, London, UK, 2000, pp. 94-103.

[ZR99] Zhang X., Rundensteiner E.A. „The SDCC Framework For Integrating Existing Algorithms for Diverse Data Warehouse Maintenance Tasks", Proceedings of the 1999 International Symposium on Database Engineering & Applications, Montreal, Canada, 1999, pp. 206-214.