LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

NEVIENDABĪGU INTEGRĒTU DATU AVOTU EVOLŪCIJAS APSTRĀDE

BAKALAURA DARBS

Autors: **Lauma Svilpe**
Studenta apliecības Nr.: ls16043
Darba vadītājs: asociētā profesore Dr. dat. Darja Solodovņikova

RĪGA 2020

ANOTĀCIJA

Neviendabīgu integrētu datu avotu evolūcija ir kļuvusi par aktuālu problēmu, jo dati mūsdienās ir ļoti daudzveidīgi un to mainība ir dinamiska. Šī iemesla dēļ nepieciešams atrast veidus, kā datus no dažādiem datu avotiem maksimāli efektīvi apkopot un uzkrāt, apstrādājot arī datu struktūrās evolūcijas rezultātā radušās izmaiņas.

Darba mērķis ir atrast risinājumu, kā apstrādāt neviendabīgu integrētu datu avotu evolūcijas rezultātā radušās izmaiņas un adaptēt tās sistēmā. Darbā veikta literatūras analīze par datu noliktavām, ETL procesiem un lielajiem datiem, kā arī izpētīta esoša datu avotu evolūcijas sistēma.

Rezultātā piedāvāts risinājums - neviendabīgu integrētu datu avotu evolūcijas apstrādes mehānisms datu bāzes struktūras un procedūru formā, kas paredzēts datu avotu izmaiņu adaptācijai esošajā sistēmā.

Atslēgvārdi: datu noliktava, lielie dati, ETL procesi, datu avotu evolūcija, metadati.

ABSTRACT

Processing of integrated heterogeneous data sources evolution

The evolution of heterogeneous integrated data sources has become a topical issue, as data today is very diverse and dynamic. For this reason, it is necessary to find ways to collect and store data from different data sources as efficiently as possible, while also processing the changes in data structures that have occurred as a result of data source evolution.

The aim of the thesis is to find a solution to process the changes caused by the evolution of integrated heterogeneous data sources and to adapt them into the system. The analysis of the literature on data warehouses, ETL processes and big data is performed in the paper, as well as the existing system of data source evolution is studied.

As a result, a solution is proposed - a mechanism for processing the evolution of integrated heterogeneous data sources in the form of a database structure and procedures for adapting changes to data sources in the existing system.

Keywords: data warehouse, big data, ETL processes, data source evolution, metadata.

SATURA RĀDĪTĀJS

APZĪMĒJUMU SARAKSTS

Apzīmējums	Skaidrojums
4GL	Saīsinājums no angļu val. – "*4th generation language*". Apzīmē jebkuru programmēšanas valodu, kura piedāvā augstāku abstrakcijas līmeni, norobežojoties no iekšējām datora aparatūras detaļām, tādā veidā padarot programmēšanas valodu universālāku un lietotājam draudzīgāku.
API	Saīsinājums no angļu val. – "*Application programming interface*". Apzīmē lietojumprocesos izmantojamu pilnu operētājsistēmas funkciju specifikāciju, kā arī šo funkciju izmantošanas procedūru aprakstu. Tīkla operētājsistēmās ar saskarni API tiek definēta standarta metode, kas nodrošina visu tīkla iespēju izmantošanu.
CSV	Saīsinājums no angļu val. – "*Comma-seperated value file*". Apzīmē teksta datni, kas sastāv no laukiem un ierakstiem, kur vērtības viena no otras atdalītas ar komatiem.
DBMS	Saīsinājums no angļu val. – "*Database management system*". Apzīmē datu bāzu pārvaldības sistēmu.
DSS	Saīsinājums no angļu val. – "*Decision support system*". Apzīmē lēmumu atbalsta sistēmu.
ETL	Saīsinājums no angļu val. – "*Extraction, Transformation, Loading*". Apzīmē datu izvilkšanu no dažādiem datu avotiem, to pārveidošanu konsekventā struktūrā un ielādēšanu kādā datu glabātuvē tālākai izmantošanai.
JSON	Saīsinājums no angļu val. – "*JavaScript Object Notation*". Apzīmē tekstuālu datu formātu, kas paredzēts strukturētu datu pārraidei.
OLAP	Saīsinājums no angļu val. – "*Online analytical processing*". Apzīmē daudzdimensionālu datu bāzes analīzes metodi, kas nodrošina specifisku datu bāzu indeksāciju, tādējādi paātrinot piekļuvi datiem gadījumos, kad jāpārskata lieli datu masīvi, kā arī ļaujot analizēt datus pēc daudziem dažādiem aspektiem.
OLTP	Saīsinājums no angļu val. – "*Online transaction processing*". Apzīmē sistēmu, kas atvieglo un pārrauga tādus transakciju orientētus lietojumus kā datu ievades un izguves transakcijas dažādās jomās, t.sk. banku darījumos, gaisa satiksmē, pasta sūtījumos, tirdzniecībā un ražošanā.

Raw	Apzīmē datu formātu, kas nonāk datu apstrādes sistēmā bez iepriekšējas to pareizības noteikšanas, sakārtošanas vai priekšapstrādes.
RDBMS	Saīsinājums no angļu val. – "*Relational database management system*". Apzīmē relāciju datu bāzu pārvaldības sistēmu.
SQL	Saīsinājums no angļu val. – "*Structured Query Language*". Apzīmē strukturētu vaicājumvalodu, ko lieto datu bāzes pārvaldības sistēmās dažāda tipa datoros. *SQL* tiek izmantota klientservera arhitektūras tīklos, lai nodrošinātu personālajiem datoriem piekļuvi kopīgi izmantojamu datu bāzu resursiem. Izmantojot šo valodu, lietotājam nav jārūpējas par to, kā fizikāli tiek īstenota piekļuve datiem un kā tiek nodrošināta piekļuve datu bāzēm, kas izvietotas gan lieldatoros, gan arī minidatoros un personālajos datoros.
XML	Saīsinājums no angļu val. – "*Extensible Markup Language*". Apzīmē paplašināmās iezīmēšanas valodu, ko lieto strukturētu datu aprakstīšanai un apmaiņai, kas ir neatkarīga no lietotās operētājsistēmas un lietojumprogrammām.

IEVADS

Ik dienu pasaulē tiek saražots milzīgs datu apjoms. Kamēr 2000.gadā šis skaitlis bija mērāms vien simtos petabaitu, 2020.gadā tiek prognozēts, ka tiks glabāti jau ap 35 zetabaitiem datu [25]. Lielā datu apjoma dēļ arvien aktuālākas kļūst datu glabāšanas, mainības un izmantošanas problēmas. Nepieciešams būvēt arvien optimālākas datu noliktavas, kurās jāpielieto efektīvi algoritmi neviendabīgu datu integrācijai un pašai datu atlasei un analīzei. Kā arī, neskatoties uz to, ka vairumā gadījumu datu ieguves avoti jau ir integrēti sistēmā, nepieciešami pēc iespējas automatizēti risinājumi, kas spēj atrisināt datu mainības un evolūcijas problēmas.

Pēdējo gadu laikā ieviesti vairāki jauni rīki, tehnoloģijas un ietvari, kas atbalsta lielo datu analītiku, piemēram, *Apache Hadoop* dalītā failu sistēma [20], *ApacheHBase* datu bāzu pārvaldības risinājums [6] vai *Hive* datu noliktavas risinājums [26]. Tomēr minētie rīki galvenokārt risina tikai tikai lielo datu apjoma pieauguma problēmu, atstājot neatrisinātas datu un to struktūras evolūcijas problēmas. Turklāt, lai apstrādātu šāda veida izmaiņas, nepieciešams liels izstrādātāja manuālā darba ieguldījums, jo esošie risinājumi neatbalsta automātisku vai daļēji automātisku datu avotu izmaiņu adaptāciju datu noliktavā [22].

Balstoties uz pastāvošajām datu avotu evolūcijas problēmām, tiek izvirzīts bakalaura darba mērķis – atrast risinājumu, kā apstrādāt neviendabīgu integrētu datu avotu evolūcijas rezultātā radušās izmaiņas. Lai sasniegtu izvirzīto mērķi, tiek uzstādīti darba uzdevumi:

- veikt literatūras analīzi par datu noliktavām, ETL procesiem un lielajiem datiem;
- izpētīt esošu datu avotu evolūcijas sistēmu;
- izstrādāt neviendabīgu integrētu datu avotu evolūcijas apstrādes mehānismu esošajai sistēmai.

Darbā pielietotās pētniecības metodes ir literatūras analīze un praktiskā uzdevuma veikšana. Literatūras analīzē izmantotie avoti ir par attiecīgajām tēmām izdotās grāmatas, ārzemju un Latvijas pētnieku izdotie zinātniskie raksti, kā arī informācija, kas pieejama plaši atzītās un uzticamās tīmekļa vietnēs.

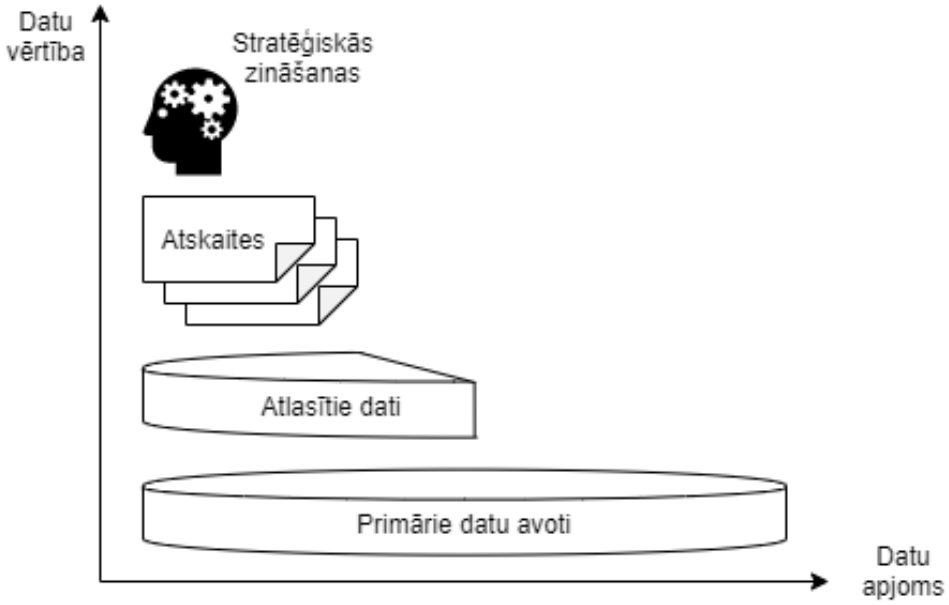
Izstrādātais darbs sastāv no 3 galvenajām nodaļām: 1.nodaļā veikta literatūras analīze par datu noliktavām, ETL procesiem un lielajiem datiem, 2.nodaļā aprakstīta esošā datu avotu evolūcijas sistēma, bet 3.nodaļā aprakstīts izstrādātais neviendabīgu integrētu datu avotu evolūcijas apstrādes mehānisms. Nodaļās aprakstītā informācija papildināta ar 22 attēliem un 10 tabulām.

1. DATU NOLIKTAVAS

Datu noliktavas jēdziens radies, attīstoties tehnoloģijām un tādā veidā pieaugot ikdienā saglabāto datu apjomam. Šīs nodaļas apakšnodaļās detalizēti aprakstīts, kā attīstījies datu noliktavas jēdziens un kāda ir tās arhitektūra ar nolūku izprast šīs struktūras nepieciešamību, aktualitāti un izmantojamību mūsdienās. Aprakstīta arī lielo datu koncepcija, kas datu noliktavu kontekstā ir nozīmīgs un saistīts jēdziens. Viens no svarīgākajiem posmiem neviendabīgu datu integrēšanai datu noliktavā ir ETL procesi, tāpēc sniegts arī šī jēdziena detalizēts skaidrojums.

1.1. Datu noliktavu attīstība

Pašos pirmsākumos datu glabāšanai tika izmantoti ļoti dārgi un ierobežotas ietilpības mehānismi – perfokartes, magnētiskās lentas, pēc tam ar pavisam jauniem papildinājumiem nāca klajā diskatmiņa. Drīz vien diskatmiņas tika papildinātas ar datu bāzu pārvaldības sistēmām jeb DBMS, kuru galvenais izmantošanas ieguvums bija ļoti ātra datu ievietošana sistēmā [9]. Līdz pat 80.gadu vidum šādās datu bāzēs tika glabāti tikai operatīvie dati – dati, kuri ir iesaistīti ikdienā veicamajos procesos. Tomēr bieži vien stratēģisku lēmumu pieņemšanai bija nepieciešama ātra pieeja glabātajiem datiem [18].



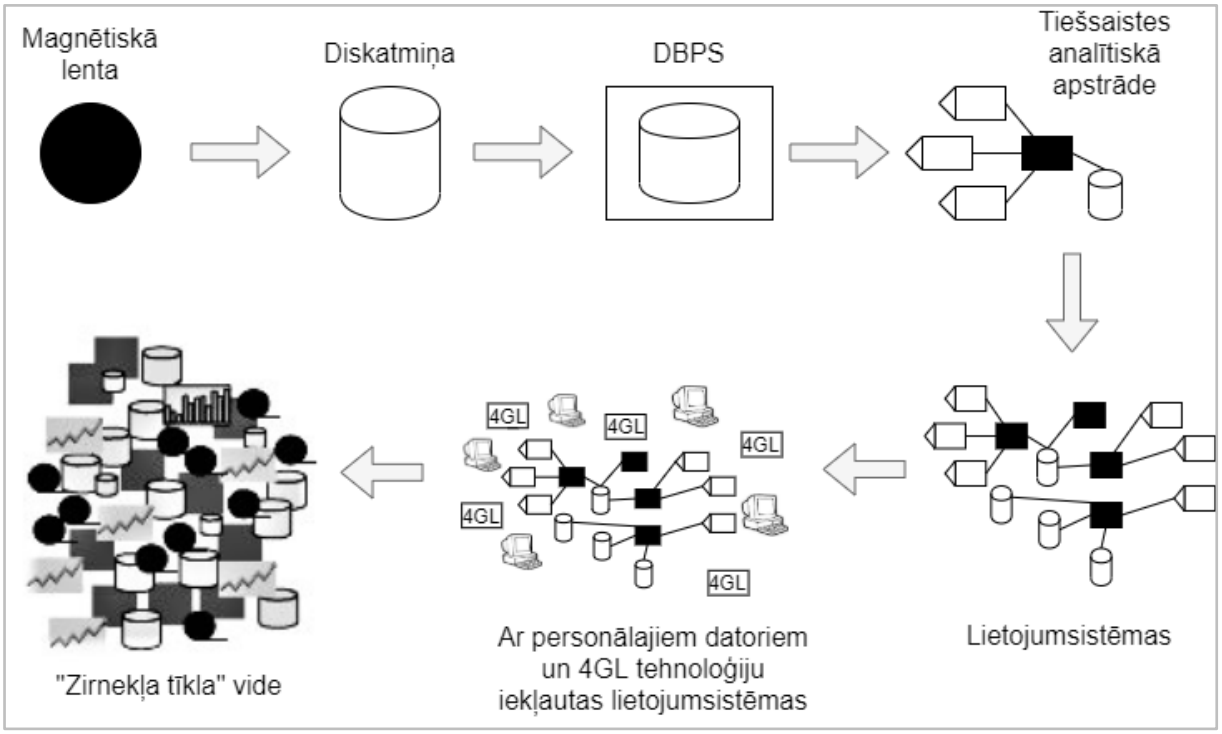
1.1. att. **Informācijas vērtības atkarība no datu daudzuma [18]**

Lai iegūtu visvērtīgāko informāciju – stratēģiskās zināšanas, nepieciešami vairāki datu stāvokļi, kur katrs no tiem satur atšķirīgu informācijas daudzumu (skat. att. 1.1.). Vislielākais

datu apjoms ir primārajiem datu avotiem, kur glabājas pilnīgi visa uzkrātā informācija. Pēc tam, izvēloties konkrētu subjektu, par kuru jāiegūst stratēģiskās zināšanas, tiek atlasīti konkrēti ar šo subjektu saistītie dati. Un tikai tad, kad izveidotas atskaites par izvēlēto subjektu, iespējams iegūt kopskatu, līdz ar to arī stratēģiskās zināšanas.

Tā kā operatīvo datu apjoms jau tobrīd pieauga eksponenciāli, datori tika atzīti par vienīgo iespējamo rīku, ar ko šādā daudzpakāpju veidā apstrādāt un analizēt informāciju. Tā rezultātā tika ieviests jēdziens "lēmumu atbalsta sistēmas" (angļu val. - "*Decision support systems"* [1] jeb DSS). Par DSS tiek saukts paplašināmu un interaktīvu metožu un rīku kopums, kas paredzēts vadības stratēģisko lēmumu pieņemšanas atbalstam, iekļaujot tajā datu apstrādi un analīzi [18].

Izmantojot DSS rīkus, arvien biežāk analītiskā apstrāde tika veikta tiešsaistē. Attīstoties tehnoloģijām, tika veidotas arvien lielākas lietojumsistēmas. Lai apmierinātu lietotāju vajadzību pēc iespējami ātras un ērtas piekļuves datiem, radās divas tehnoloģijas – personālie datori un 4GL. Personālie datori ļāva jebkuram lietotājam individuāli apstrādāt datus, taču 4GL tehnoloģijas ideja bija padarīt programmatūras izstrādi tik vienkāršu, lai to varētu darīt jebkurš [9].

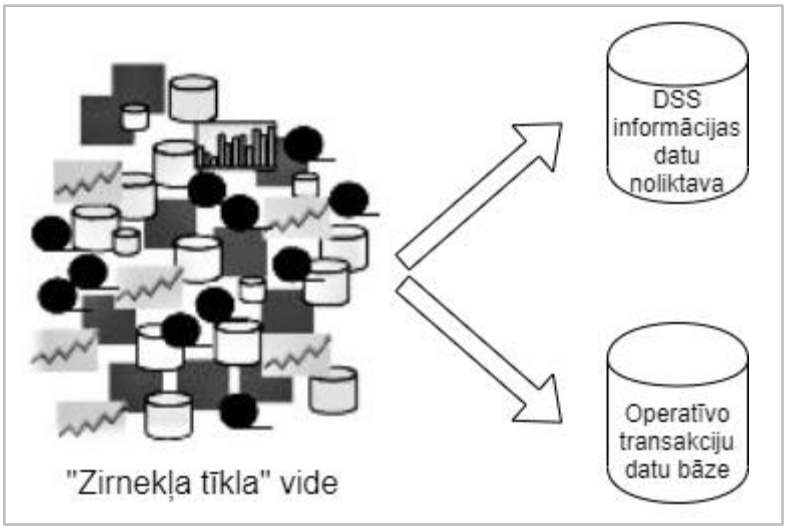


1.2. att. **"Zirnekļa tīkla" vides attīstība [9]**

Iepriekšminētās attīstības rezultātā radās nekārtība, ko mēdz saukt par "zirnekļa tīkla" vidi (skat. att. 1.2. – adaptēts un latviskots no [9]). Šāds "zirnekļa tīkls" vairākās koorporatīvās vidēs mēdza izaugt neiedomājami sarežģīts, kā rezultātā ar laiku šādu vidi vairs nebija iespējams ne uzturēt, ne pilnveidot. Tai tika iedots nosaukums "dabiski attīstīta

arhitektūra", jo bija maz iespēju to ietekmēt. Jo lielāka kļuva šī arhitektūra, jo vairāk izaicinājumu tā radīja:

- datu ticamība – dati tika analizēti ar dažādiem algoritmiem, nepastāvēja datu ieguves līmeņi un katrs ar datiem rīkojās atšķirīgi un izvēlējās dažādus datu avotus, tāpēc rezultāti atskaitēs vairāku analītiķu starpā kļuva pārāk atšķirīgi, līdz ar to – neuzticami;
- produktivitāte – lai analizētu datus no dažādiem datu avotiem, nepieciešams izstrādāt vairākas individuāli pielāgotas programmas, kas savāc datus no katra avota, līdz ar to analītiķim nepieciešams iedziļināties dažādās failu struktūrās, kas ir laikietilpīgi un neproduktīvi;
- nespēja pārveidot datus par informāciju – dati no dažādiem datu avotiem ir par dažādiem laika posmiem un ar atšķirīgām konceptuālajām nozīmēm, ko DSS analītiķim ir gandrīz neiespējami apstrādāt ar izstrādāto rīku palīdzību [8].



1.3. att. **Pāreja no "Zirnekļa tīkla" vides uz datu noliktavas vidi [9]**

Drīz vien pēc DSS jēdziena ieviešanas klajā nāca arī datu noliktavas, kuras tiek uzskatītas par visplašāk izmantoto DSS veidu [18]. Datu noliktavas būtiski izmainīja IT speciālistu domāšanu – līdz šim pastāvēja uzskats, ka datu bāze ir paradzēta, lai glabātu jebkādam nolūkam paredzētus datus. Tomēr līdz ar datu noliktavas jēdziena parādīšanos kļuva acīmredzams, ka ir nepieciešamas dažādu veidu datu bāzes [9], tādēļ "Zirnekļa tīkla" vide tika sadalīta divās atsevišķās daļās, lai glabātu datus pēc to nozīmes datu analītikas kontekstā (skat. att. 1.3. – adaptēts un latviskots no [9]).

Datu noliktava ir datu bāze ar unikālu struktūru, kas ļauj relatīvi ātri un efektīvi apstrādāt lielus datu apjomus [15]. Taču par datu noliktavas formālā jēdziena ieviesēju tiek uzskatīts amerikāņu datorzinātnieks Bils Inmons, kurš to definējis sekojoši: "Datu noliktava ir

uz subjektiem orientēta, integrēta, laika atkarīga un nemainīga datu glabāšanas struktūra, kas paredzēta vadības lēmumu pieņemšanas atbalstam." [16]. Definīcijas pamatā minētas četras galvenās datu noliktavas īpašības:

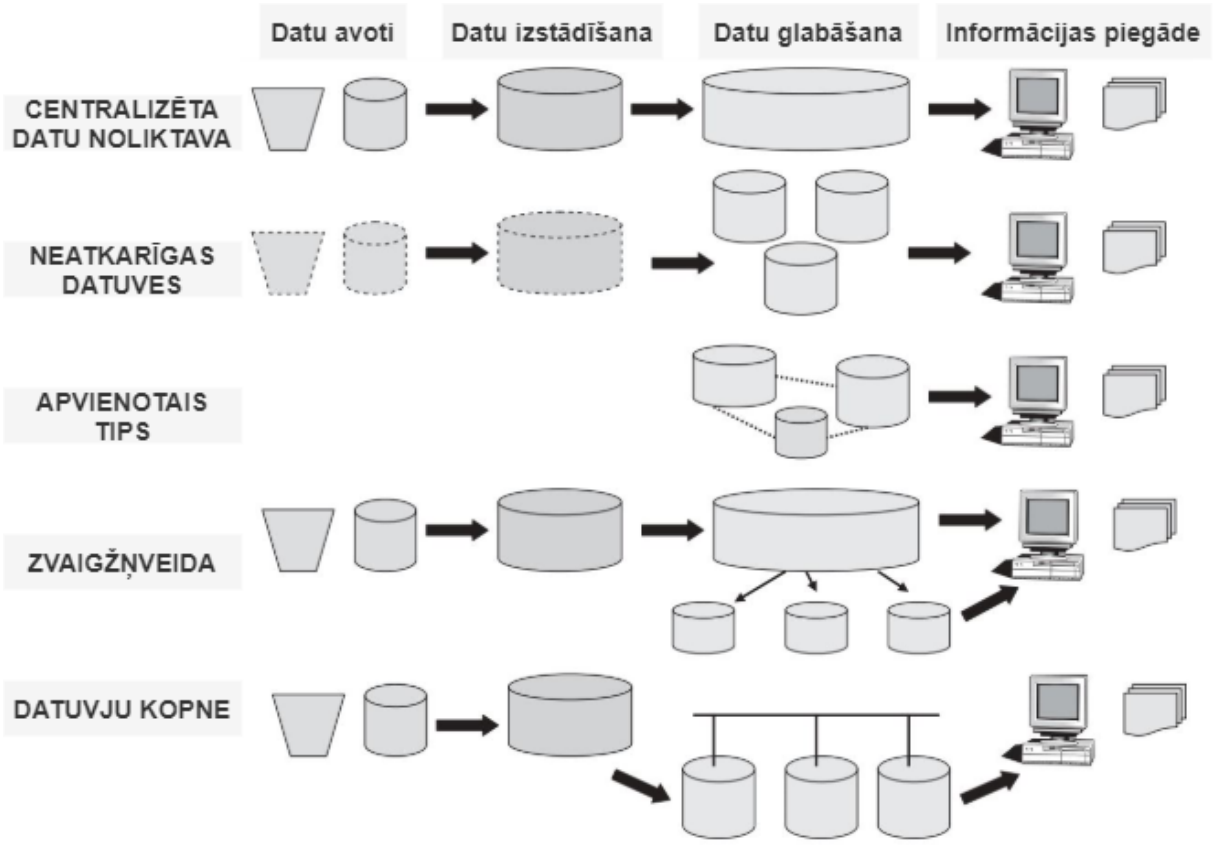
- uz subjektiem orientēta – izveidotajā datu noliktavā tiek glabāti dati par kādu konkrētu subjektu;
- integrēta – datu noliktavas dati tiek iegūti no dažādiem avotiem, līdz ar to tie tiek pārveidoti tā, lai struktūra būtu konsekventa;
- laika atkarīga – lai pieņemtu stratēģiskus biznesa lēmumus, datiem jāatspoguļo laikā notikušās izmaiņas, līdz ar to datu noliktava glabā ne tikai esošo situāciju, bet visus ar subjektu saistītos datus, atšķirībā no OLTP sistēmām, kur vēsturiskie dati tiek pārvietoti uz arhīviem;
- nemainīga – dati, kas ienākuši noliktavā, netiek mainīti, lai nodrošinātu izmaiņu analīzi [14].

Datu noliktava pamatā tiek projektēta datu atlases un analīzes vaicājumu efektīvai izpildei, mazāk pievēršot uzmanību transakciju apstrādei. Tā tiek realizēta kā relāciju datu bāze, kas papildināta ar ETL risinājumiem, OLAP funkcionalitāti, kā arī dažādiem klienta pusē izmantojamiem rīkiem, kas palīdz apstrādāt un nogādāt datus gala lietotājam saprotamā formā [3]. Par datu noliktavas pamatvienību var uzskatīt datuvi. Tā ir primārajā datu noliktavā glabāto datu apakškopa vai apkopojums. Datuve ietver informācijas vienību, kas attiecas uz konkrētu uzņēmējdarbības jomu, departamentu vai lietotāju grupu [18].

1.2. Datu noliktavas arhitektūra

Datu noliktavas arhitektūrai nepieciešamas sekojošas īpašības:

- atdalāmība – analītiskajām un transakcionālajām darbībām jābūt pēc iespējas nošķirtām;
- mērogojamība – tā kā datu apjoms, kas jāpārvalda un jāapstrādā, kā arī lietotāju skaita prasības pakāpeniski palielinās, arhitektūrai jābūt viegli paplašināmai;
- paplašināmība – arhitektūrai jāspēj uzņemt jaunas lietojumsistēmas un tehnoloģijas tā, lai nebūtu jāpārprojektē visa sistēma;
- piekļuves drošības uzraudzība – tā kā datu noliktavā glabājas uzņēmuma stratēģiskie dati, šis ir ļoti svarīgs faktors;
- administrējamība – datu noliktavas pārvaldība jāprojektē tā, lai tā būtu pēc iespējas vienkāršāka [11].



1.4. att. **Datu noliktavas arhitektūru tipi [16]**

Datu noliktavām gadu gaitā attīstoties, tiek izdalīti vairāki datu noliktavu arhitektūru tipi (skat. att. 1.4.). Tie ir atšķirīgi gan pēc tā, kā dati tiek glabāti, gan pēc relācijām starp datu noliktavu un datuvēm. Sīkāk datu noliktavas arhitektūru tipi aprakstīti nākamajās nodaļās.

1.2.1. Centralizēta datu noliktava

Centralizētas datu noliktavas arhitektūras tips ņem vērā individuālās uzņēmuma līmeņa informācijas glabāšanas un lietošanas veidu prasības. Tiek izveidota vispārējā infrastruktūra, kur normalizēti dati zemākajā detalizācijas pakāpē tiek glabāti trešajā normālformā, taču dažkārt tiek iekļauti arī apkopotie dati. Vaicājumi un lietojumsistēmas normalizētajiem datiem piekļūst no centrālās datu noliktavas. Šajā arhitektūrā nepastāv atsevišķas datuves [16].

1.2.2. Neatkarīgas datuves

Datu noliktavas tips, kur visa pamatā ir neatkarīgas datuves, raksturīgs uzņēmumiem, kur dažādas organizatoriskās vienības izstrādā savas datuves, balstoties uz specifiskiem mērķiem. Lai arī katra datuve paredzēta konkrētai organizatoriskajai vienībai, šīs atsevišķās datuves nesniedz vienu vienīgu patieso datu attēlojuma versiju. Datuves savā starpā ir neatkarīgas, kā rezultātā datu definīcijas un standarti var būt nekonsekventi. Šādas atšķirības var kavēt datu analīzi starp datuvēm [16]. Piemēram, ja sistēmā ir divas neatkarīgas datuves, viena klientiem, bet otra – pārdošanas datiem, klientu un pārdošanas datus kopā analizēt ir sarežģīti, neskatoties uz to, ka būtībā tie ir savā starpā cieši saistīti subjekti. Tomēr bieži vien šāda pieeja tiek aizstāta ar citām arhitektūrām, kas ir ērtākas datu integrācijai un atskaišu ģenerēšanai [18].

1.2.3. Apvienotais datu noliktavas tips

Dažkārt uzņēmumiem, pirms tie sāk datus glabāt datu noliktavā, ir jau pastāvošas lēmumu pieņemšanas struktūras dažādu sistēmu, jau iegūtu datu kopu vai primitīvu datuvju formā. Šādā gadījumā nav saprātīgi esošos datus dzēst, tāpēc tiek izmantots apvienotais arhitektūras tips, kurā datus iespējams apvienot gan fiziski, gan loģiski, izmantojot dažādus atslēgu laukus, metadatus, vaicājumus un citus mehānismus. Šim arhitektūras tipam nav vienas vispārējas datu noliktavas [16].

1.2.4. Zvaigžņveida datu noliktava

Līdzīgi kā centralizētās datu noliktavas arhitektūrā, arī šeit pastāv uzņēmuma mēroga kopīga datu noliktava. Zvaigžņveida (angļu val. - "*hub-spoke*" [1]) datu noliktavas arhitektūrai raksturīgs tas, ka dati trešajā normālformā tiek glabāti centralizētā datu noliktavā, taču galvenais ieguvums šādai arhitektūrai ir tāds, ka tiek ieviestas datuves, kuras tiek aizpildītas no datu noliktavas. Datuves var tikt veidotas visdažādākajām vajadzībām: analītikai, vaicājumiem, datu ieguvei u.c. Visbiežāk vaicājumu izpildei tiek izmantotas tieši datuves nevis pati datu noliktava [16].

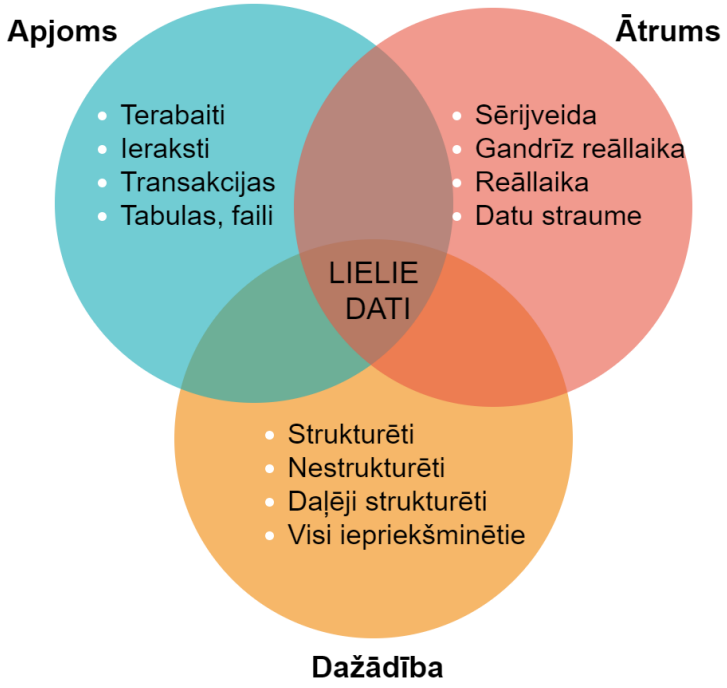
1.2.5. Datuvju kopne

Šīs arhitektūras būvēšanas sākumā nepieciešams izanalizēt prasības konkrētajam biznesa subjektam un pēc tam, balstoties uz iegūtajām uzņēmējdarbības dimensijām, tiek būvēta pirmā datuve. Pēc tam iegūtās dimensijas tiek izmantotas arī būvējot pārējās datuves [16]. Šāda pieeja nodrošina loģisku datuvju integrāciju, kas rezultējas ar iespēju iegūt plašu skatījumu uz analizēto informāciju [18].

1.3. Lielie dati

Astoņdesmito gadu beigās, kad pasaulē tika ieviesta datu noliktavas koncepcija, radās iespēja veikt plašu spektru ar datu manipulācijām - veikt vēsturisko datu analīzi, paredzēt turpmāko datu virzību, novērot dažādas tendences ilgākos laika posmos. Deviņdesmito gadu vidū, popularizējoties jēdzienam "internets", ikdienā iegūto un glabāto datu apjoms eksponenciāli pieauga, kas šī gadsimta sākumā iezīmēja pavisam jaunas tendences. Pasaulē ienāca meklētājprogrammas, dažādi mobilie risinājumi, sociālie tīkli un citas vēl nebijušas tehnoloģijas, kas būtiski palielināja glabāto datu apjomu.

Jauni biznesa modeļi un iespējas, kas radās līdz ar datu apjoma palielināšanos, izraisīja nepieciešamību pēc jaudīgiem risinājumiem, kas, analizējot iegūtos datus, spētu piedāvāt personalizētus risinājumus tā brīža vajadzību apmierināšanai. Tādas kompānijas kā *Google, Yahoo, Facebook* u.c. jau kopš pašiem pirmsākumiem investējušas tehnoloģisko risinājumu attīstīšanā, sniedzot iespēju ikdienā strādāt ar milzīgu daudzumu dažādu formātu datiem, lai izveidotu jaudīgu stratēģisku lēmumu pieņemšanas atbalsta platformu [12]. Līdz ar šīm pārmaiņām tika ieviests lielo datu (angļu val. - "*big data*" [1]) jēdziens.



1.5. att. **Lielo datu raksturlielumi**

Lielos datus var definēt kā liela apjoma datu kopumus dažādās sarežģītības pakāpēs, kas ģenerēti ar dažādu ātrumu un neskaidrības pakāpi. Šādus datus nevar apstrādāt, izmantojot tradicionālās metodes, tehnoloģijas un algoritmus [12]. Lai raksturotu lielos datus, bieži vien tiek izmantots "trīs V" princips: apjoms (angļu val. - "*volume*" [1]), ātrums (angļu val. - "*velocity*" [1]) un dažādība (angļu val. - "*variety*" [1]) (skat. att. 1.5.) [4]. Katra no pazīmēm sīkāk aprakstīta nodaļās 1.3.1., 1.3.2., 1.3.3.

1.3.1. Lielo datu apjoms

Datu apjoms tiek noteikts pēc tā, cik daudz datu nepārtraukti tiek ģenerēts. Liela apjoma datus sastāda dažādu iekārtu radītie (gan ikdienā lietojamo, gan industriālo), lietojumsistēmu žurnalēšanas, klikšķu secības, globālās pozicionēšanas sistēmas (GPS) un vēl dažādu citu veidu dati [12]. Ikdienā saglabāto datu apjoms laika gaitā ir ievērojami palielinājies. 2000.gadā visā pasaulē saglabāti dati, kas aizņem ap 800 000 petabaitu, taču 2020.gadā plānots, ka šis skaitlis palielināsies līdz pat 35 zetabaitiem. Piemēram, *Facebook* dienā saražo ap 10 terabaitiem datu, *Twitter* – 7 terabaitus, kā arī eksistē uzņēmumi, kas vairākus terabaitus datu saražo ik stundu.

1.3.2. Lielo datu ieplūšanas ātrums

Lielo datu ātruma raksturlielumu var definēt kā lietojumsistēmas spēju nepārtraukti apstrādāt datu plūsmu konkrētā ātrumā. Šis ātrums kļūst par kritisku raksturlielumu, ja datus nepieciešams analizēt gandrīz reāllaikā [25]. Dati var ieplūst dažādos ātrumos:

- sērijveidā – dati tiek saņemti pa partijām, tādā veidā starp partiju ienākšanas reizēm sniedzot sistēmai laiku tos apstrādāt;
- gandrīz reāllaikā – dati tiek saņemti ar nelielu aizkavi;
- reāllaikā – dati nonāk sistēmā uzreiz, kolīdz tie tiek savākti;
- straumē – dati sistēmā ienāk nepārtraukti (piemēram, sensoru rādījumi).

1.3.3. Lielo datu dažādība

Ļoti reti dati tiek iegūti sakārtoti un gatavi tūlītējai apstrādei. Pārsvarā tie ir ļoti dažādi un tos sākotnēji nav iespējams pielāgot nekādai konkrētai struktūrai. Piemēram, teksti sociālajos tīklos, attēli, neapstrādāti sensoru dati nebūs iekļaujami vienā un tajā pašā struktūrā [4]. Lai lielo datu sistēma spētu apstrādāt jaunus datu formātus, tai jāpiemīt sekojošām īpašībām:

- mērogojamība – jaunu datu formātu pievienošanai nav nepieciešama sistēmas pārprojektēšana;
- dalītā datu apstrāde - apstrādes, uzglabāšanas un vadības funkcijas, kā arī ievadizvades funkcijas ir sadalītas starp vairākām attālām datu apstrādes sistēmām [1].
- attēlu apstrādes iespējas – attēlu apstrāde būtiski atšķiras no dažādu citu datu avotu apstrādes, jo tie ir pilnīgi nestrukturēti dati [12].

Papildus iepriekšminētajām pamatīpašībām (apjoms, ātrums un dažādība) mūsdienās izdalītas arī tādas papildus īpašības kā ticamība, nepastāvība, kvalitāte, neskaidrība, izturība un izplatība [12, 25].

Apstrādājot datus, jārēķinās ar visām iepriekšminētajām lielo datu īpašībām. Lai dažādas struktūras, apjoma un kvalitātes datus novietotu datu noliktavā tālākai analīzei, tiek veidoti ETL (angļu val. - "*extraction, transformation, loading*" [1]) procesi, kas aprakstīti nodaļā 1.4.

1.3.4. Lielie dati datu noliktavu kontekstā

Pastāv vairāki aspekti, kas lielo datu un datu noliktavas jēdzienus padara ļoti viegli saistāmus. Piemēram, vairāki lielo datu avoti jau iekļauj savu kvalitatīvi izstrādātu metadatu struktūru, kā arī e-komercijas vietnēs parasti ir labi nodefinēti saites elementi. Šo iemeslu dēļ lielos datus ir samērā vienkārši iekļaut datu noliktavā tālākās analīzes veikšanai [7].

Mūsdienās klasisko datu noliktavu rīkiem un tehnoloģijām kļūst arvien grūtāk apstrādāt visu analītisko procesu slodzi, tāpēc, lai paplašinātu esošos datu noliktavu risinājumus, tiek ieviestas lielo datu tehnoloģijas [19]. Arvien biežāk tiek izmantota *Apache Hadoop* tehnoloģija, kas par zemām izmaksām spēj apstrādāt liela apjoma un dažādu struktūru datus [5]. *Apache Hadoo*p ir uz programmēšanas valodas *Java* bāzēts ietvars, kas atbalsta uzticamu, mērogojamu un nodalītu lielu datu kopu apstrādi uz ļoti plaša spektra programmatūras [27]. Tā kā *Apache Hadoop* nav pārāk ērta gala lietotājam, jo tajā nepieciešams izstrādāt *MapReduce* programmas, kā arī tai nepiemīt *SQL* līdzīga un dabiski saprotama saskarne, tika izstrādāta *Apache Hive* platforma [24]. *Hive* tiek uzskatīta par galveno *Hadoop* sistēmas sastāvdaļu, kas sniedz iespēju ērti veidot dažādus datu kopsavilkumus [27].

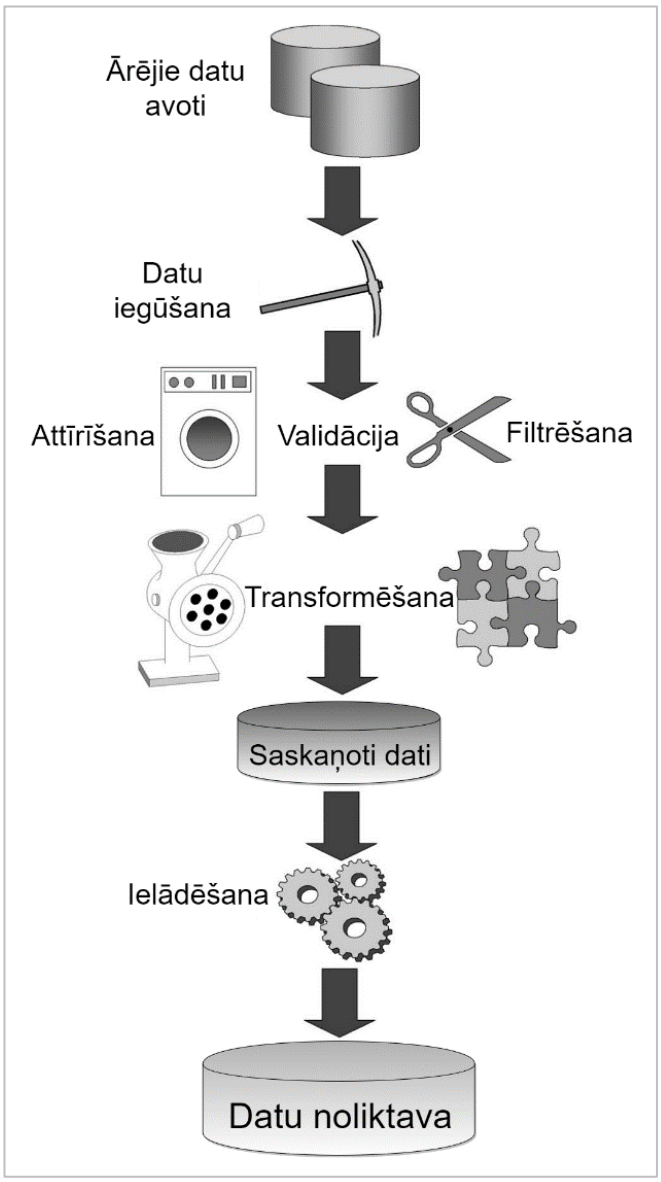
Tomēr realitātē tehnoloģijām nepieciešami vairāki gadi vai pat dekādes, lai tās nobriestu un spētu atbalstīt visdažādākās uzņēmumu izvirzītās prasības. Lai arī *Apache Hadoop* tehnoloģija ir ideāli piemērota lielo datu analīzei, tomēr tā nav pilnībā aizstājusi tradicionālās datu noliktavas – organizācijas, kuras vēlas modernizēt savu datu vidi izstrādā arhitektūras, kurās vienlīdz darbojas gan *Apache Hadoop*, gan tradicionālās datu noliktavas [5].

1.4. ETL procesi

ETL procesi nodrošina datu ievākšanu no dažādiem datu avotiem, transformēšanu vienotā struktūrā un ielādēšanu datu noliktavā tālākai izmantošanai. Sistēmas ETL procesiem ir vairākas funkcijas:

- datu loģiskā pārveidošana;
- datu avota domēna pārbaude;
- konvertēšana no vienas DBMS uz citu;
- noklusējuma vērtību izveidošana;
- datu apkopošana;
- laika vērtību pievienošana;
- datu pārstrukturēšana;

- ierakstu apvienošana;
- nevajadzīgu vai lieku datu dzēšana [9].



1.6. att. **ETL procesi datu pārveidošanā ievietošanai datu noliktavā [9]**

ETL procesi ir kā pāreja starp datiem no dažādiem ārējiem avotiem uz datu noliktavu (skat. att. 1.6.). Kad dati tiek iegūti, tos nepieciešams attīrīt no nevajadzīgiem un liekiem ierakstiem, filtrēt, kāda informācija tieši ir nepieciešama, kā arī validēt datus, ja tas nepieciešams un iespējams. Pēc tam seko datu transformēšana, kur ar noteiktām formulām un atsevišķiem procesiem neviendabīgi dati tiek pārveidoti par konsekventas struktūras saskaņotiem datiem. Un tikai tad, kad dati ir sastrukturizēti, tos iespējams ievietot datu noliktavā. Sīkāk par katru no ETL procesu soļiem skatīt nodaļās 1.4.1., 1.4.2. un 1.4.3.

1.4.1. Ievākšana

Datu ievākšanas fāze galvenokārt attiecas uz dažādu tehnisko jautājumu risināšanu, kas saistīti ar neviendabīgiem datiem, kā arī datu importēšanu sistēmā tālākai to apstrādei [13]. Pie šīs fāzes pieder arī datu tīrīšana (angļu val. - "*data cleaning*" [1]), kuras laikā tiek atklātas un novērstas kļūdas un nepilnības datos, lai uzlabotu to kvalitāti. Datu noliktavās nepārtraukti tiek ielādēts milzīgs datu daudzums no dažādiem avotiem, tāpēc ir liela varbūtība, ka daļa datu būs neatbilstoši vai nederīgi. Tā kā datu noliktavas tiek izmantotas dažādu stratēģisku lēmumu pieņemšanai, dublēta vai trūkstoša informācija var radīt nepareizu vai maldinošu priekšstatu par esošo situāciju. Lai arī datu attīrīšana datu noliktavu kontekstā tiek izmantota plaši, daudzo iespējamu datu neatbilstību veidu un milzīgā datu apjoma dēļ tā tiek uzskatīta par vienu no lielākajām ar datu noliktavām saistītajām problēmām [17].

1.4.2. Transformācija

Datu transformācijas fāzē tie tiek pārveidoti, piemērojot nepieciešamos uzņēmējdarbības virzienā noteiktos noteikumus un apstrādājot datu saturu, lai iegūtu viendabīgu datu struktūru. Šajā fāzē tiek izpildītas vairākas darbības, līdz ar to tā ātri var kļūt grūti pārvaldāma. Pēc katras no izpildītajām datu transformācijas darbībām tiek iegūts starprezultāts (tipiski tie ir kādi jauni metadati vai transformētas datu kopas) [12].

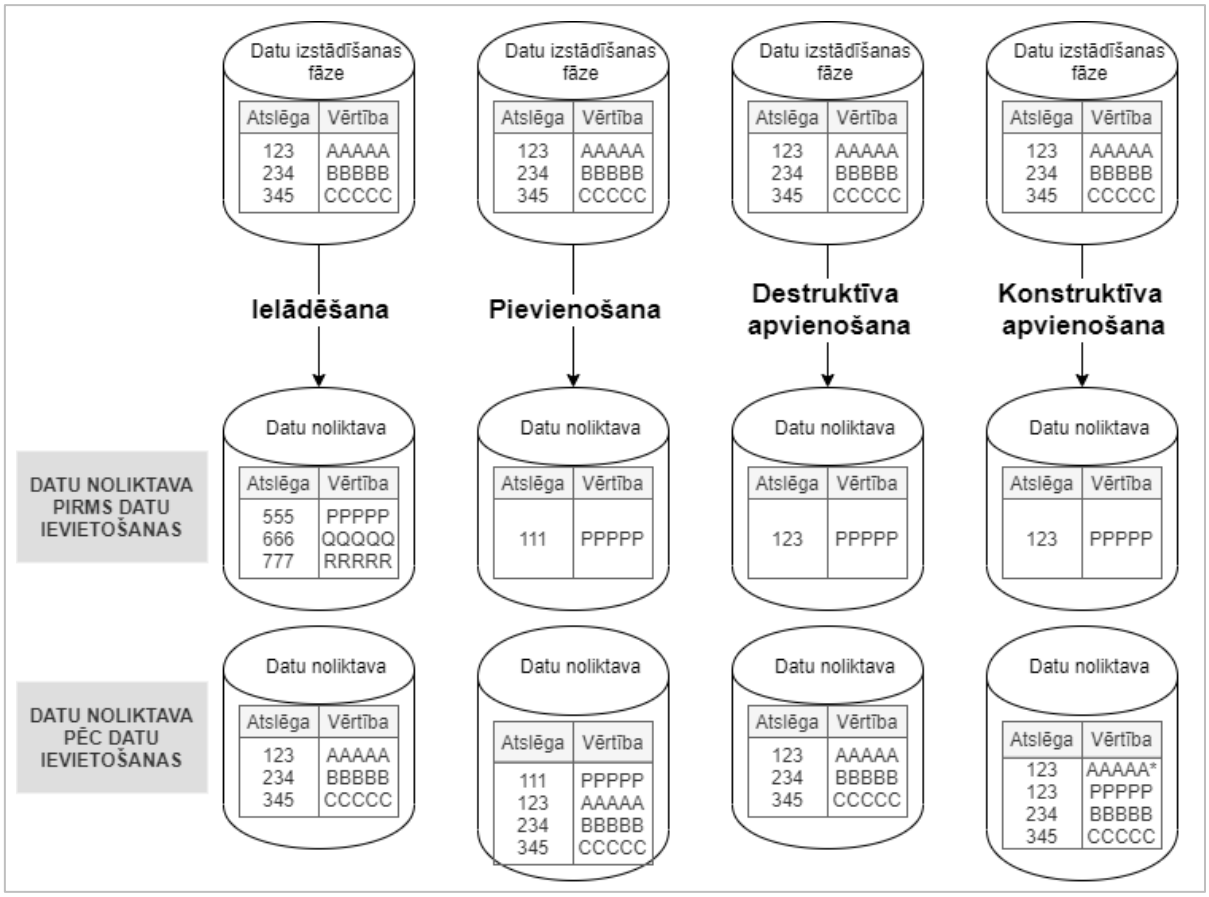
Būtībā datu transformācijas fāzē tiek atrisināta datu neviendabīguma problēma. Par neviendabīgiem sauc jebkādus datus, kuriem ir dažādi datu tipi un formāti [28], bet datu noliktavu kontekstā tie ir dati, kuri ir iegūti no atšķirīgiem avotiem, taču tiek uzkrāti un attēloti vienotā stilā [2]. Pamatuzdevumi datu transformēšanai ir sekojoši:

- atlase – tiek izvēlēti pilni ieraksti vai daļa no tiem, kurus nepieciešams transformēt;
- sadalīšana un apvienošana – izvēlēto ierakstu dati tiek sadalīti sīkāk vai apvienoti kopā (datu noliktavu kontekstā biežāk izmantota tieši apvienošana);
- konvertācija – tiek veiktas plašas datu lauku izmaiņas, lai standartizētu datu ieguvi no atšķirīgiem avotiem, kā arī lai datu lauku vērtības būtu saprotamas gala lietotājiem;
- apkopošana – daļā gadījumu datu noliktavā nav nepieciešams glabāt datus zemākajā detalizācijas pakāpē, tāpēc vairāki lauki tiek apkopoti, par pamatu ņemot kādu konkrētu uzņēmējdarbības subjektu;

- bagātināšana – nepieciešamības gadījumā dati tiek dublēti, lai datu noliktavas kontekstā iegūtu daudzveidīgāku skatījumu uz izvēlēto uzņēmējdarbības subjektu [16].

1.4.3. Ievietošana datu noliktavā

Datu ielādēšanas posmā tie tiek ievietoti datu noliktavas tabulās. Datu noliktavās tiek izmantoti dažādi indeksēšanas mehānismi un partīciju dalīšanas metodes, tādēļ šī fāze var aizņemt daudz laika [10].



1.7.att. **Datu ievietošanas datu noliktavā veidi [16]**

Pirms nonākšanas datu noliktavā dati atrodas izstādīšanas fāzē (angļu val. - "*staging phase*" [1]), kur tie tiek transformēti. Pēc tam datu noliktavā tie var tikt ievietoti četros pamatveidos (skat. att. 1.7.):

- ielādēšana;
- pievienošana;
- destruktīva apvienošana;
- konstruktīva apvienošana.

Ielādēšana

Ja gan mērķa tabula, gan dati tabulā jau eksistē, ielādēšanas process nodzēš eksistējošos datus un aizvieto tos ar ienākošajiem. Ja tabulas pirms datu ielādēšanas ir tukšas, ieraksti vienkārši tiek pievienoti tabulai [16].

Pievienošana

Ja tiek veikta datu pievienošana, tabulā eksistējošie dati tiek saglabāti un ienākošie vienkārši pievienoti tabulai. Iespējams integrēt apstrādes ierakstiem, kas dublējas (dublikātu saglabāšana vai dzēšana) [16].

Destruktīvā apvienošana

Šis datu ievietošanas veids pārbauda ievietojamo un eksistējošo datu atslēgas. Ja pastāv ieraksts ar konkrēto atslēgu, tas tiek atjaunināts. Pretējā gadījumā tabulai tiek pievienots jauns ieraksts [16].

Konstruktīvā apvienošana

Konstruktīvās apvienošanas datu ievietošanas veids līdzinās destruktīvajai apvienošanai. Atšķirība ir tāda, ka šeit, konstatējot meklētās atslēgas eksistenci, esošais ieraksts tiek atstāts tabulā, bet jaunais ieraksts pievienots un atzīmēts kā veco ierakstu aizstājošs [16].

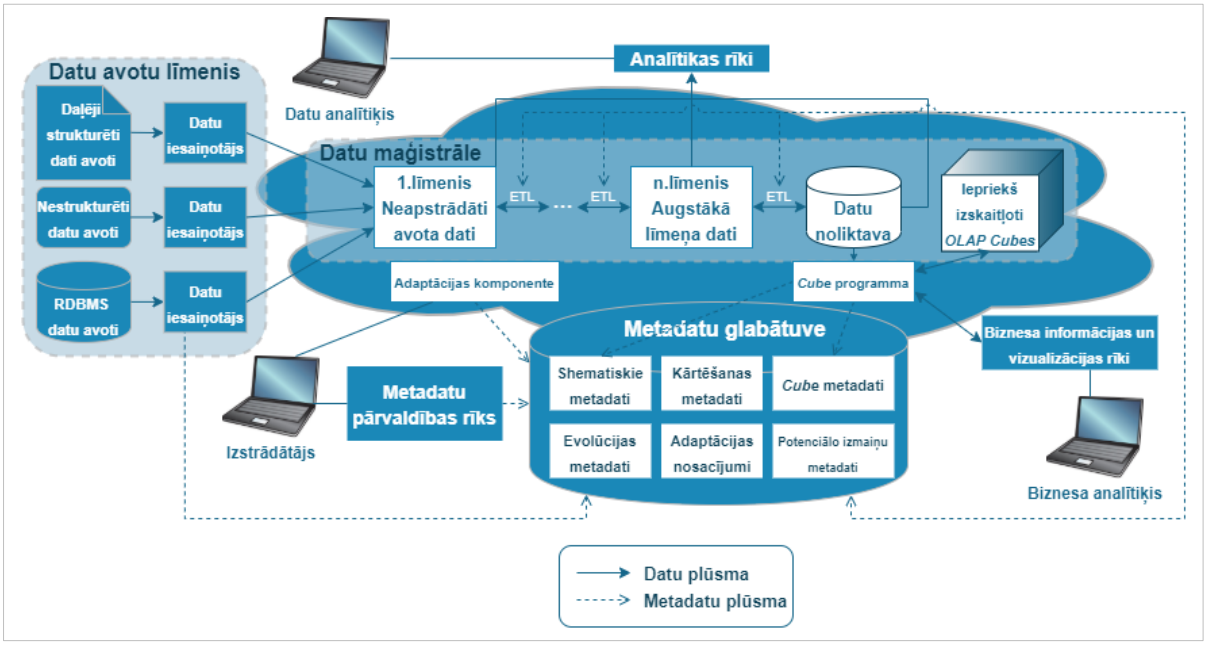
2. DATU AVOTU EVOLŪCIJAS SISTĒMA

Latvijas Universitātes Datorikas fakultātē piedāvāts datu noliktavas risinājums lielo datu analīzei, kurā iekļauti algoritmi dažādu evolūcijas rezultātā radušos izmaiņu atklāšanai. Risinājumā iekļauta lielo datu analīzei paredzēta arhitektūra, kas ļauj izpildīt OLAP operācijas un cita veida analīzi datu noliktavā integrētiem datu avotiem, kā arī spēj atklāt dažādas izmaiņas strukturētos, daļēji strukturētos un nestrukturētos datos, balstoties uz tos aprakstošajiem metadatiem. Piedāvātās sistēmas arhitektūra un izmaiņu noteikšanas un apstrādes mehānismi aprakstīti nodaļās 2.1. un 2.2.

Esošā datu avotu evolūcijas sistēmas arhitektūra bakalaura darba ietvaros papildināta ar datu avotu evolūcijas rezultātā radušos izmaiņu adaptācijas mehānismu, kas sīkāk aprakstīts 3.nodaļā.

2.1. Arhitektūra

Datu avotu evolūcijas sistēma sastāv no vairākām komponentēm, kas nodrošina datu plūsmu un apstrādi no avotu līmeņa līdz pat datu noliktavā ievietotiem datiem.



2.1. att. **Datu avotu evolūcijas sistēmas arhitektūra [23]**

Datu avotu evolūcijas sistēmas arhitektūras (skat. att. 2.1. – adaptēts un latviskots no [23]) pamatkomponentes ir datu avoti, datu maģistrāle, metadatu glabātuve un adaptācijas komponente. Biznesa analītiķi var piekļūt jau apstrādātiem datiem no iepriekš izskaitļotiem

OLAP kubiem (angļu val. – "*OLAP cubes*" [1]), taču datu analītiķi iesaistās arī pašā datu izgūšanas un konvertācijas procesā. Izstrādātāji, izmantojot metadatu pārvaldības rīku, piekļūst metadatiem. Sīkāk katra no minētajām datu avotu evolūcijas sistēmas arhitektūras komponentēm aprakstīta nodaļās 2.1.1., 2.1.2., 2.1.3 un 2.1.4.

2.1.1. Datu avotu līmenis

Datu avotu līmenī neviendabīgi dati tiek iegūti no dažādiem avotiem un ielādēti sistēmā tālākai to apstrādei. Tā kā lielie dati ietver dažādu veidu datus, sistēma atbalsta strukturētus avotus (datu bāzu tabulas), daļēji strukturētus datus (piemēram, XML vai JSON formātā), kā arī nestrukturētus datus (žurnālfaili, fotogrāfijas, video ieraksti).

Datu avotu ielādēšanu veic datu iesaiņotāji (angļu val. – "*wrappers*" [1]), kuri satur konkrētajam datu avotam atbilstošo iegūšanas saskarni (piemēram, tīmekļa pakalpojumi, API u.c.). Piemēram, dati no RDBMS sistēmā tiek ielādēti sagrupēti pa partijām, taču sensoru mērījumi varētu tikt ielādēti kā datu straume [22]. Visi dati to sākotnējā formātā tiek ielādēti datu maģistrāles 1.līmenī, kas sīkāk aprakstīts nodaļā 2.1.2.

2.1.2. Datu maģistrāle

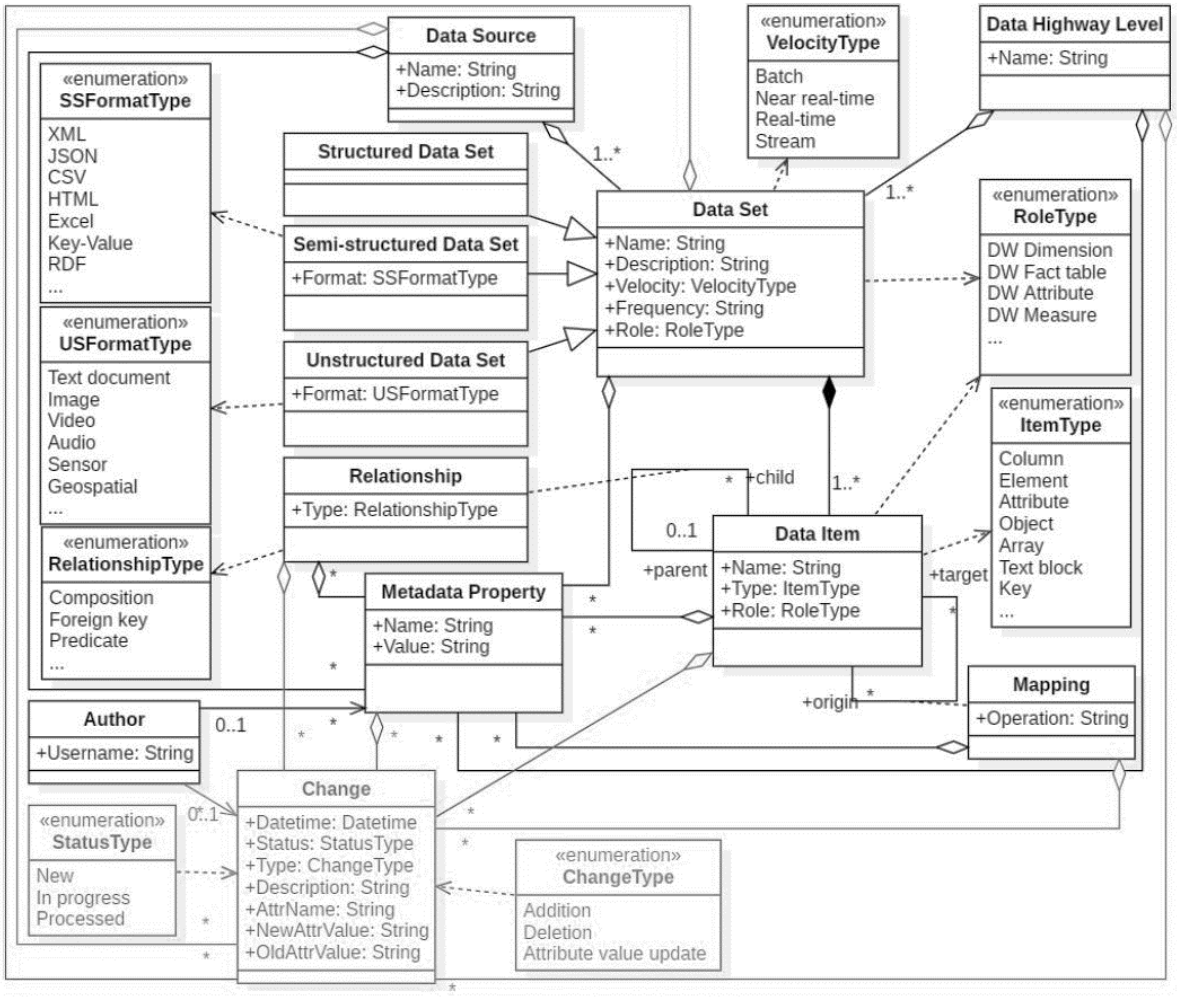
Datu maģistrāle sastāv no vairākiem līmeņiem. Pirmajā līmenī tiek glabāti neapstrādāti dati *Raw* formātā, kas iegūti pa tiešo no datu avotiem. Katra nākamā maģistrāles līmeņa dati tiek iegūti no iepriekšējā līmeņa un tiek atjaunināti arvien retāk. Līmeņu skaitu un to atjaunināšanas biežumu nosaka konkrētās sistēmas prasības. Tā kā gan pirmajā datu maģistrāles līmenī, gan pārējos starpposmos dati tiek glabāti to sākotnējā formātā un ir tikai daļēji integrēti, to glabāšanai tiek izmantota *Data Lake* struktūra [22]. Datu maģistrāles līmeņi tiek iegūti izmantojot ETL procesus, kas veic datu pārveidi, apkopošanu un saistīto datu vienību integrēšanu [21].

Tā kā ETL procesi papildina zemāka līmeņa datus ar informāciju, kas iegūta, veicot informācijas apstrādi un transformāciju augstākos līmeņos, iespējams savienot dažādu maģistrāles līmeņu datus, lai iegūtu vēl vērtīgākus datu analīzes rezultātus [22].

2.1.3. Metadatu glabātuve

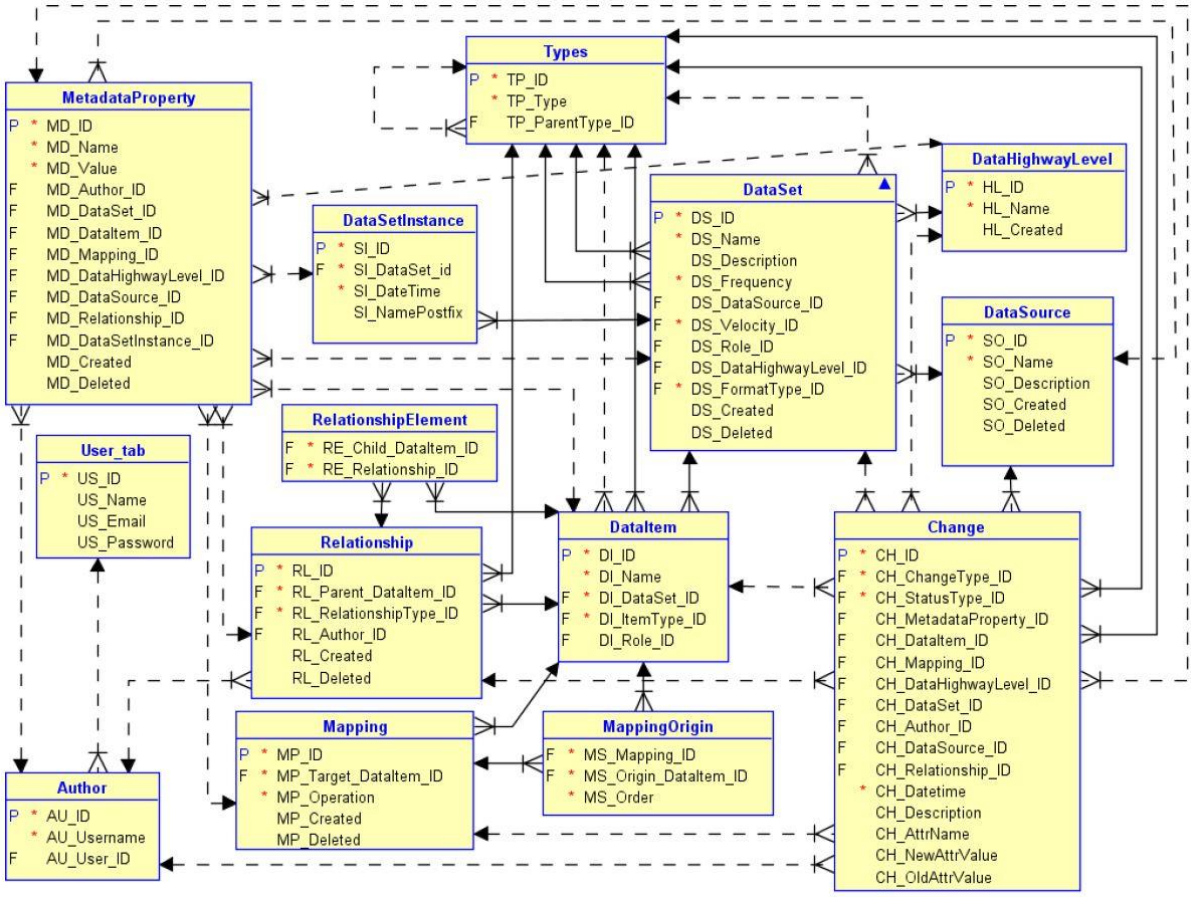
Sistēmas arhitektūras darbība pamatā balstīta uz metadatu glabātuvē esošajiem datiem. Izmantojot metadatu pārvaldības rīku un definējot dažādus metadatus, izstrādātājs nosaka, kā darbosies sistēma. Metadatu glabātuvē glabājas sešu veidu metadati:

- *OLAP* kubu (angļu val. – "*OLAP cubes*" [1]) metadati – tiek izmantoti kubu dzinējos (angļu val. – "*cube engine*" [1]) un vaicājumu izpildē;
- kartēšanas metadati – tiek izmantoti, lai definētu ETL procesus (saites starp datu avotiem un datu vienībām dažādos datu maģistrāles līmeņos, kā arī datu ielādēšanai nepieciešamās transformācijas);
- evolūcijas un potenciālo izmaiņu metadati, adaptācijas nosacījumi – tiek glabāta informācija par iespējamajām izmaiņām datu avotos un iespējām, kā šīs izmaiņas adaptēt sistēmā;
- shematiskie metadati – tiek izmantoti sistēmas datu struktūru glabāšanai. [22]



2.2. att. **Konceptuālais metadatu glabātuves modelis [23]**

Konceptuālo metadatu modeli skatīt attēlā 2.2. (adaptēts no [23]). Datu kopa (klase *Data Set*) tiek apzīmēta kā datu vienību (*Data Item*) kopums un ir sadalīta trīs apakšklasēs – strukturēta datu kopa (*Structured Data Set*), daļēji strukturēta datu kopa (*Semi-structured Data Set*) un nestrukturēta datu kopa (*Unstructured Data Set*). Datu kopa var tikt iegūta no datu avota (*Data Source*) vai tā var būt kāda datu maģistrāles līmeņa (*Data Highway Level*) sastāvdaļa. Tiek glabāta informācija arī par datu kopu informācijas iegūšanas ātrumu un biežumu (*VelocityType*). Ja pastāv saite starp datu vienībām vienā un tajā pašā vai dažādās datu kopās, tiek izmantota klase *Relationship*, kas savieno attiecīgos bērnu un vecāku objektus. Kartēšanas metadati (klase *Mapping*) norāda, kādas transformācijas pielietojamas, lai no zemāka datu maģistrāles līmeņa datu vienība tiktu pārvietota uz augstāku maģistrāles līmeni. Klase *Change* paredzēta evolūcijas rezultātā notikušo izmaiņu reģistrēšanai un tālākai izmaiņu apstrādei.



2.3. att. **Fiziskais metadatu glabātuves datu bāzes modelis [21]**

Aprakstītais konceptuālais modelis realizēts relāciju datu bāzē (skat. att. 2.3. – adaptēts no [21]). Tipu (klases *SSFormatType, USFormatType, RelationshipType, StatusType, ChnageType, VelocityType, RoleType* un *ItemType*) glabāšanai izmantota tabula *Types*. Lai nodrošinātu kartēšanu, izmantotas tabulas *Mapping* un *MappingOrigin* – tā kā tabulā

Mapping norādītā formula, pēc kuras iegūta datu vienības vērtība, var tikt izmantota vairākām datu vienībām, tabula *MappingOrigin* izmantota kā starptabula N:N saites nodrošināšanai. Tas pats princips izmantots arī tabulās *Relationship* un *RelationshipElement,* kā arī *DataSet* un *DataSetInstance* tabulās.

Tabula *MetadataProperty* izmantota jebkādu aprakstošo Atslēga:Vērtība pāru glabāšanai, piemēram, failu nosaukumi, izmēri, rakstzīmju kopas, ierobežojumu pārbaudes datu kopām, tipi u.c. [21].

2.1.4. Adaptācijas komponente

Lielo datu noliktavas pamatelements, kas atbild par datu avotu un informācijas prasību izmaiņu apstrādi, ir adaptācijas komponente. Tās galvenais uzdevums ir atpazīt notikušās vai ģenerēt potenciālās izmaiņas un pēc tam ļaut izstrādātājam izvēlēties vispiemērotākās izmaiņas, kas jāievieš. Lai nodrošinātu vēlamo funkcionalitāti, adaptācijas komponente izmanto metadatu glabātuvē esošos datus, kā arī var tikt izmantoti papildus dati, kurus nevar automātiski identificēt – šādā gadījumā izstrādātājs šos datus piegādā manuāli, izmantojot adaptācijas komponenti [22]. Datu izmaiņu noteikšana un apstrāde detalizēti aprakstīta nodaļās 2.2. un 2.3.

Bakalaura darba ietvaros adaptācijas komponente papildināta ar mehānismu, kas datu avotu evolūcijas rezultātā radušās izmaiņas adaptē sistēmas metadatos.

2.2. Izmaiņu identificēšana

Lai identificētu metadatos notikušās izmaiņas, tiek izmantots algoritms, kas vispirms apstrādā datu avotus un datu maģistrāles līmeņus, tad datu kopas un datu vienības, bet pēc tam kartēšanas metadatus un attiecības. Identificētās izmaiņas tiek saglabātas tabulā *Change*.

Apstrādājot datu avotus, tiek izmantota procedūra, kas atklāj jaunus vai nepieejamos datu avotus, kā arī identificē izmaiņas avotu datu kopās. Kā ieejas parametri tiek saņemts datu avots, kā arī to aprakstošie metadati – saņemtajam avotam atbilstošie un metadatu glabātuvē esošie. Šie ieejas dati savā starpā tiek salīdzināti, kā rezultātā iespējams identificēt datu avota pievienošanu vai dzēšanu. Apstrādājot katru avotam piederošo datu kopu, iespējams identificēt metadatu īpašību izmaiņas – īpašības pievienošanu, dzēšanu vai vērtības atjaunināšanu.

Lai identificētu izmaiņas datu maģistrāles līmeņos, datu kopās vai datu vienībās, algoritms sakrīt ar datu avotu izmaiņu noteikšanai izmantoto, tikai šajā gadījumā tiek apstrādāts attiecīgi katrs datu maģistrāles līmenis, datu kopa vai datu vienība.

Kartēšanas metadatu izmaiņu identificēšanai svarīgi ir tas, ka pirms tās tiek apstrādātas datu vienību izmaiņas, jo šajā algoritmā tiek pārbaudīts, vai neviena no kartēšanas metadatos izmantotajām datu vienībām nav atzīmēta kā dzēsta kādā no algoritma iepriekšējiem soļiem.

Lai noteiktu metadatu attiecību izmaiņas, pirmkārt, tiek reģistrētas no jauna pievienotās attiecības. Pēc tam tiek apstrādāti gadījumi, kad kāda iemesla dēļ attiecība ir dzēsta vai vairs nav izmantojama. Tie var būt gadījumi, kad kāda no attiecībās izmantotajām datu vienībām iepriekšējos algoritma soļos ir dzēsta vai izstrādātājs manuāli rediģējis attiecības [21].

2.3. Izmaiņu apstrāde

Par reģistrēto izmaiņu apstrādi tiek uzskatīta tās adaptācija sistēmas metadatos. Izmaiņas adaptācijas soļi var būt gan automātiski, gan manuāli veicami, kā arī izstrādātājs var pieņemt lēmumus par izmaiņas adaptācijas gaitu.

2.1. tabula

Evolūcijas rezultātā radušos izmaiņu apstrāde

Izmaiņa [23]	Klase	Izmaiņas apstrāde
Datu avota pievienošana	*Data Source*	Jaunais datu avots tiek pievienots pirmajam datu maģistrāles līmenim.
Datu avota dzēšana	*Data Source*	Datus no avota aizstāj ar datiem no citiem avotiem vai pavisam dzēš avotu.
Datu maģistrāles līmeņa pievienošana	*Data Highway Level*	Definē atbilstošos ETL procesus un datu avotus, ja tas nepieciešams.
Datu maģistrāles līmeņa dzēšana	*Data Highway Level*	Pārdefinē ETL procesus, lai aizvietotu trūkstošo datu maģistrāles līmeni, vai dzēš no tā atkarīgos datu avotus.
Datu kopas pievienošana	*Data Set*	Definē visas ar jauno datu kopu saistītās vienības (datu maģistrāles līmenis, ETL procesi, metadatu īpašības, datu avots).
Datu kopas dzēšana	*Data Set*	Datus no datu kopas aizstāj ar datiem no citu avotu kopām vai pavisam dzēš datu kopu.
Datu kopas formāta maiņa	*Data Set* *Data Item* *Relationship*	Pārdefinē ETL procesus vai ievērojamu izmaiņu gadījumā datu kopa tiek dzēsta.

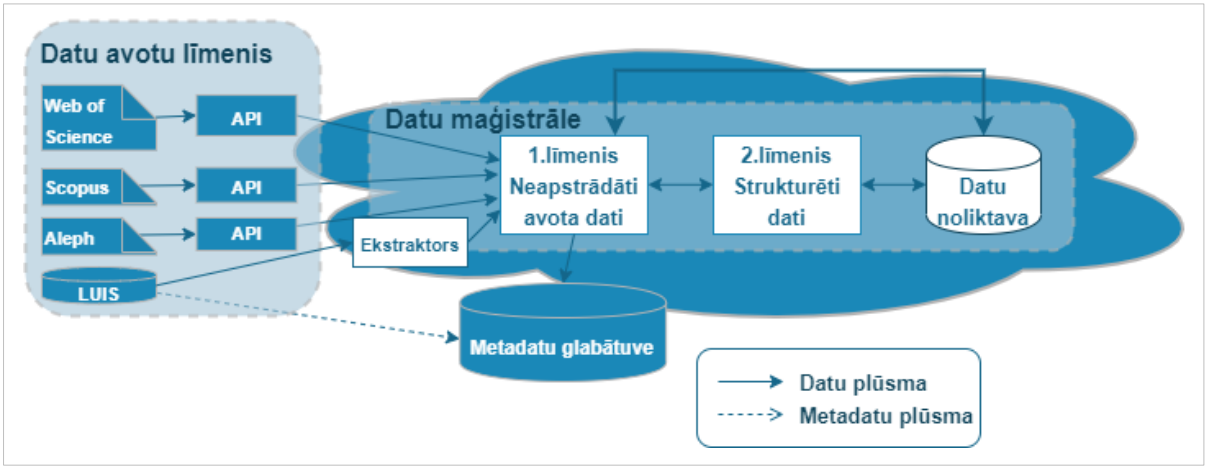
Izmaiņa [23]	Klase	Izmaiņas apstrāde
Datu kopas nosaukuma maiņa	*Data Set*	Pārdefinē ETL procesus.
Datu vienības pievienošana	*Data Item*	Definē visas ar jauno datu kopu saistītās vienības (datu kopa, ETL procesi, metadatu īpašības, datu avots).
Datu vienības nosaukuma maiņa	*Data Item*	Pārdefinē ETL procesus.
Datu vienības veida maiņa	*Data Item*	Pārdefinē ETL procesus.
Datu vienības dzēšana	*Data Item*	Datu vienību aizstāj ar datiem no citiem avotiem vai datu kopām, vai pavisam dzēš datu vienību.
Attiecības pievienošana	*Relationship*	Pārdefinē ETL procesus.
Attiecības dzēšana	*Relationship*	Pārdefinē ETL procesus.
Kartēšanas pievienošana	*Mapping*	Definē transformācijas funkciju.
Kartēšanas dzēšana	*Mapping*	Pārdefinē transformācijas funkciju vai dzēš kartēšanas datus.
Metadatu īpašības pievienošana	*Metadata Property*	Izstrādātājs pieņem lēmumu par tālāko īpašības izmantošanu.
Metadatu īpašības dzēšana	*Metadata Property*	Pārdefinē ETL procesus.
Atribūta vērtības maiņa	Atribūtu saturošā klase	Pārdefinē ETL procesus.

Iespējamos izmaiņu veidus un vispārīgu to adaptācijas aprakstu skat. tabulā 2.1. Izmaiņās iesaistītās klases kontekstā skat. att. 2.2.

Katrs no izmaiņu adaptācijas scenārijiem ir vairāku izpildāmo soļu un pārbaudāmo nosacījumu kopums, kas saistīts ar dažādu struktūru pārveidi un pārbaudi. Izmaiņu adaptācijas scenāriju glabāšanas un izpildes tehnisko realizāciju skatīt 3.nodaļā.

2.4. Datu avotu evolūcijas sistēmas pielietojums

Lai praktiski pierādītu izstrādātās lielo datu analīzes sistēmas arhitektūras metadatu modeļa darbību, tā izmantota publikāciju lielo datu sistēmā. Publikāciju sistēmas mērķis ir no vairākiem neviendabīgiem datu avotiem integrēt datus par Latvijas Universitātes darbinieku un studentu publikācijām un nodrošināt šo datu analīzi datu noliktavā [23].



2.4.att. **Publikāciju lielo datu sistēmas arhitektūra [23]**

Publikāciju lielo datu sistēmas arhitektūras shēmu skatīt attēlā 2.4. (adaptēts un latviskots no [23]). Strukturēti dati tiek iegūti no Latvijas Universitātes informācijas sistēmas *LUIS*, taču daļēji strukturēti dati - no bibliotēkas datu pārvaldības sistēmas *Aleph*, kā arī citātu indeksācijas sistēmām *Scopus* un *Web of Science*. Sistēmas arhitektūra sastāv no trīs datu maģistrāles līmeņiem, kur pēdējā no tiem dati tiek glabāti datu noliktavā, kas realizēta, balstoties uz *Apache Hive* tehnoloģiju. Pārejas starp datu maģistrāles līmeņiem nodrošina kartēšanas metadatos aprakstītie ETL procesi [23].

2.2. tabula

Publikāciju lielo datu sistēmas kartēšanas metadatu piemērs [23]

Mērķa datu vienība	Oriģinālā datu vienība	Kartēšanas operācija	Apraksts
Level3. Faculty. **Department**	Level1. Luis_person. **Affiliation**	?	Autora fakultāte tiek noteikta pēc tabulas *LUIS_person* kolonnas *Affiliation.*
Level3. Publications. **Number of Scopus citations**	Level2. Scopus_publ. **citedby_count**	SUM(?)	Scopus citātu skaits tiek izrēķināts kā individuālo publikāciju skaita summas, kas iegūta no *Scopus* datu avota.
Level3. Publications. **Number of publications**	Level1. LUIS_publ. **PUBL_ID**	COUNT(?) + COUNT(?) + COUNT(?) + COUNT(?)	Publikāciju skaits ir izrēķināts kā visu datu avotu identifikatoru summa. Šajā gadījumā tiek definētas attiecības, lai izvairītos no vienas un tās pašas publikācijas pieskaitīšanas vairākkārt, ja tā pieejama vairākos datu avotos.
	Level2. Aleph. **doc_number**		
	Level2.Scopus_publ. **identifier**		
	Publ_IDLevel2. WOS. **uid**		

Publikāciju lielo datu sistēmas kartēšanas metadatu piemērus skatīt tabulā 2.2. (adaptēts un latviskots no [23]). Datu vienības apzīmētas ar diviem prefiksiem. Pirmais apzīmē datu maģistrāles līmeni, bet otrais – datu kopas nosaukumu. Kartēšanas operācijās ar jautājuma zīmi (“?”) tiek apzīmēta datu vienība, bet agregāciju operāciju funkcionalitāte sakrīt ar *SQL* izmantoto.

2.3. tabula

Publikāciju lielo datu sistēmas izmaiņu piemēri [23]

Izmaiņa	Izmaiņas apstrāde
Datu kopai *Scopus_metrics* pievienota datu vienība *citeScoreYearInfoList.*	Jaunā datu vienība (XML elements) sastāv no vairākiem apakšelementiem, kuri neeksistēja arī iepriekšējās datu kopās, tomēr metadatos tiek izveidota tikai viena izmaiņa, kas attiecināta tikai uz augšējo apakšelementu.
No datu kopas *Scopus_metrics* izdzēsta datu vienība *IPP*.	Šī izmaiņa ietekmē datu ielādēšanas procesu. Tā kā izdzēsto datu vienību nav iespējams aizstāt ar nevienu no vienībām, kas atrodas citos datu avotos, metadatos tiek reģistrēta kartēšanas dzēšana.
Pievienots jauns datu avots *DSpace*.	Saskaņā ar jaunajām analīzes prasībām, sistēma ir jāpapildina ar datiem, kas satur pirmsdrukas vai publicētus darbus pilnos tekstus. Jaunais datu avots satur nestrukturētus datus (pilna teksta failus) un ar tiem saistītos metadatus kā birkas (angļu val. - “*tags*” [1]).
Atjaunināta datu kopas *Scopus_metrics* metadatu īpašības *API request* vērtība.	Šī izmaiņa tiek atklāta datu iegūšanas laikā no *API*. Šādu izmaiņu nepieciešams apstrādāt manuāli, jo nav iespējams automātiski izmainīt *API* pieprasījumu, balstoties uz jauno metadatu īpašību.

Datu avotu evolūcijas rezultātā iespējamas izmaiņas datu avotos. Lai būtu iespējams turpināt datu ielasīšanu datu noliktavā no evolucionējušiem avotiem, nepieciešams veikt izmaiņu adaptāciju. Bakalaura darba ietvaros izstrādātais esošajā sistēmā iekļaujamais izmaiņu adaptācijas mehānisms aprakstīts 3.nodaļā. Daži izstrādātajā mehānismā iekļautie un 3.nodaļā aprakstītie izmaiņu veidi kā konkrēti publikāciju lielo datu sistēmas piemēri aprakstīti tabulā 2.3.

3. EVOLŪCIJAS APSTRĀDES MEHĀNISMS

Lai apstrādātu datu avotu evolūcijas rezultātā radušās izmaiņas un veiksmīgi adaptētu tās sistēmas metadatos, bakalaura darba ietvaros izstrādāts mehānisms, kurš darbojas, balstoties uz dažādiem metadatiem:

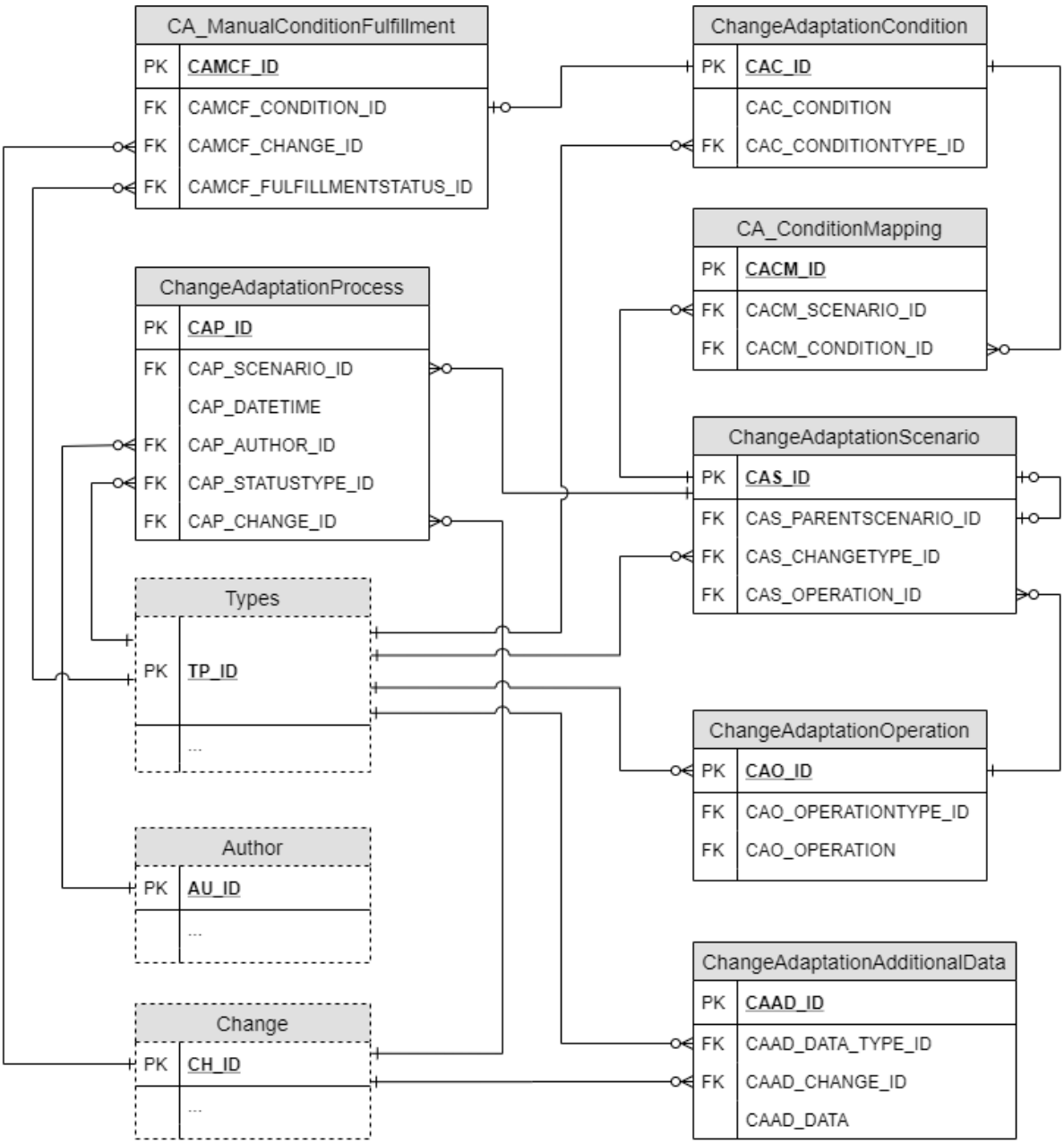
- dati par radušos izmaiņu;
- dati par manuāli veicamajām darbībām;
- dati par automātiski izpildāmām darbībām;
- kopējie izmaiņu adaptācijas scenāriji;
- izstrādātāja pievienotie papildus dati;
- scenāriju zarošanās nosacījumi.

Izstrādātais mehānisms nodrošina dažādu metadatu glabātuves daļu aizpildīšanu: shematiskie metadati, adaptācijas nosacījumi, evolūcijas un potenciālo izmaiņu metadati, kā arī kārtēšanas metadati (metadatu glabātuves aprakstu skatīt nodaļā 2.1.3.).

Tāpat kā jau esošās sistēmas metadatu glabāšanas mehānisms, arī evolūcijas apstrādes mehānisms izstrādāts, izmantojot *Oracle SQL* relāciju datu bāzi. Darba gaitā izstrādāti adaptācijas scenāriji tikai reāli publikāciju lielo datu sistēmā notikušajiem izmaiņu veidiem. Izstrādātā adaptācijas funkcionalitāte pārbaudīta uz publikāciju lielo datu sistēmas metadatiem.

3.1. Evolūcijas apstrādes metadati un to glabāšana

Evolūcijas metadatu glabāšanai datu bāze papildināta ar vairākām tabulām, kuras ar esošajām sistēmas tabulām saistītas ar ārējām atslēgām. Lai realizētu evolūcijas metadatu glabāšanu, nepieciešamas trīs esošās sistēmas tabulas. Tabulā *Change* glabājas informācija par identificētajām izmaiņām, kā arī, apstrādājot šīs tabulas ierakstus, iespējams noteikt, kāda tipa izmaiņa ir radusies. Uz šīs tabulas ierakstu balstās viss kopējais izmaiņas adaptācijas scenārijs un scenārija izpildes procesa apraksts. Tabulā *Author* glabājas informācija par sistēmas lietotājiem. Šī tabula izmantota, lai identificētu, kurš lietotājs darbojies ar konkrētas izmaiņas adaptāciju sistēmā. Savukārt tabula *Type* sistēmā izmantota kā klasifikators, kurš nosaka dažādu sistēmā izmantoto vienumu tipus. Evolūcijas mehānisma realizācijai izveidoti vairāki jauni tipi un apakštipi, kas paredzēti dažādu statusu un datu veidu glabāšanai jaunizveidotajās tabulās.



3.1. att. **Evolūcijas apstrādes metadatu glabāšanai paredzētās datu bāzes shēma**

Attēlā 3.1. redzama nepieciešamā datu bāzes shēma, lai glabātu datu avotu evolūcijas apstrādes metadatus. Ar raustīto līniju apzīmētas esošās sistēmas tabulas.

Izmaiņu adaptācijas scenāriju un veicamo darbību glabāšanai paredzētas tabulas *ChangeAdaptionOperation*, *ChangeAdaptionScenario*, kurās jau pirms izmaiņas rašanās glabājas katram izmaiņu veidam atbilstošie adaptācijas scenāriji. Šo tabulu aizpildīšana ir manuāls darbs. Arī tabulā *ChangeAdaptationProcess* glabājas izmaiņas adaptācijas scenāriji, taču šī tabula tiek aizpildīta automātiski adaptācijas procesa laikā – ieraksti tiek ievietoti tikai tad, kad ir notikusi reāla izmaiņa un tiek glabāta informācija par katras atsevišķas izmaiņas adaptācijas procesu.

Lai glabātu scenāriju zarošanās nosacījumus un pārvaldītu nosacījumu izpildi, paredzētas tabulas *ChangeAdaptationCondition, CA_ConditionMapping*, kur glabājas katram izmaiņu veidam jau iepriekš aprakstīti darbību nosacījumi. Šo tabulu aizpildīšana ir manuāls darbs. Savukārt tabula *CA_ManualConditionFulfillment* tiek aizpildīta automātiski konkrētas izmaiņas adaptācijas procesa laikā. Ja izmaiņu adaptācijas laikā nepieciešama kāda papildus informācija no izstrādātāja, tā tiek saglabāta tabulā *ChangeAdaptationAdditionalData.* Sīkāk tabulu struktūra aprakstīta nodaļās 3.1.1., 3.1.2. un 3.1.3.

3.1.1. Izmaiņu adaptācijas scenāriji un operācijas

Izmaiņu adaptācijas operācijas ir darbības jeb soļi, kas jāveic, lai adaptētu izmaiņas sistēmā. Tās ir iespējami īsas un universālas.

3.1. tabula
Datu bāzes tabula *ChangeAdaptationOperation*

Kolonnas nosaukums	Datu tips	Obligāts	Ierobežojums
CAO_ID	number(10,0)	Jā	Primārā atslēga
CAO_OPERATIONTYPE_ID	varchar2(10 byte)	Jā	Ārējā atslēga
CAO_OPERATION	varchar2(4000 byte)	Jā	Nav

Katra no darbībām tiek glabāta kā ieraksts tabulā *ChangeAdaptationOperation* (tabulas aprakstu skat. tabulā 3.1.). Darbībai ir tips, kas norāda uz to, vai tā ir manuāli veicama (šādā gadījumā tabulā glabājas tekstuāls apraksts ar to, kas izstrādātājam jāveic, lai paveiktu darbību) vai automātiski izpildāma (šajā gadījumā tabulā glabājas izpildāmās procedūras nosaukums). Vairākas dažādu tipu izmaiņu adaptāciju veikšanai izmantotās darbības pārklājas, tāpēc tās tiek glabātas atsevišķā tabulā, lai novērstu informācijas dublēšanos datu bāzē.

Izmaiņu adaptācijas scenārijs ir secīgu darbību virkne, kas tiek veikta, lai veiksmīgi adaptētu izmaiņu sistēmā. Katram no izmaiņu veidiem ir atšķirīgs izmaiņu adaptācijas scenārijs.

3.2. tabula
Datu bāzes tabula *ChangeAdaptationScenario*

Kolonnas nosaukums	Datu tips	Obligāts	Ierobežojums
CAS_ID	number(10,0)	Jā	Primārā atslēga
CAS_PARENTSCENARIO_ID	number(10,0)	Jā	Ārējā atslēga
CAS_CHANGETYPE_ID	number(10,0)	Jā	Ārējā atslēga
CAS_OPERATION_ID	varchar2(10 byte)	Jā	Ārējā atslēga

Katra scenārija soļi glabājas tabulā *ChangeAdaptationScenario* (tabulas aprakstu skat. tabulā 3.2.). Katrs no scenārija soļiem satur ārējo atslēgu uz adaptācijas darbību (tabulu *ChangeAdaptationOperation*), izmaiņas tipu, kā arī vecākieraksta identifikatoru – ārējo atslēgu uz šo pašu tabulu. Vecākieraksta identifikatora glabāšana nodrošina darbību secības uzturēšanu, kā arī atvieglo korekciju veikšanu (piemēram, pievienojot scenārija vidū kādu darbību, jārediģē tikai divi tabulas ieraksti, taču, ja secība tiktu glabāta kā kārtas skaitļi, jārediģē būtu gandrīz viss scenārijs).

Lai sekotu līdzi izmaiņu adaptācijas procesam un glabātu informāciju par katras darbības izpildi, nepieciešama tabula, kurā glabājas katrai reāli notikušai izmaiņai atbilstošais adaptācijas scenārijs.

3.3. tabula
Datu bāzes tabula *ChangeAdaptationProcess*

Kolonnas nosaukums	Datu tips	Obligāts	Ierobežojums
CAP_ID	number(10,0)	Jā	Primārā atslēga
CAP_SCENARIO_ID	number(10,0)	Jā	Ārējā atslēga
CAP_DATETIME	timestamp(6)	Jā	Nav
CAP_AUTHOR_ID	number(10,0)	Jā	Ārējā atslēga
CAP_STATUSTYPE_ID	varchar2(10 byte)	Jā	Ārējā atslēga
CAP_CHANGE_ID	number(10,0)	Jā	Ārējā atslēga

Tabula *ChangeAdaptationProcess* (tabulas aprakstu skat. tabulā 3.3.) glabā arējo atslēgu uz tabulu *ChangeAdaptationScenario*, kas nodrošina katras operācijas atšķiramību, kā arī ārējo atslēgu uz tabulu *Change,* lai identificētu, kura no izmaiņām tiek adaptēta. Lai sekotu līdzi izmaiņu adaptācijas procesam, tiek glabāts arī izpildāmās darbības statuss (darbība ir vai nav izpildīta), kā arī datums, laiks un lietotājs, kurš piedalījies šīs darbības izpildē.

3.1.2. Izmaiņu adaptācijas scenāriju zarošanās nosacījumi

Neskatoties uz to, ka izmaiņas tips ir nosakāms jau pie tās rašanās, tas negarantē viennozīmīga izmaiņas adaptācijas scenārija esamību. Ir dažādi nosacījumi, kuru izpildes rezultātā scenārijs var zaroties. Tie var būt gan automātiski izpildāmi (šajā gadījumā sistēma pati nosaka, pa kuru no scenārija ceļiem nepieciešams virzīties), gan manuāli apstrādājami (šajā gadījumā izstrādātājam nepieciešams izvērtēt konkrēto situāciju un pieņemt lēmumu).

3.4. tabula
Datu bāzes tabula *ChangeAdaptationCondition*

Kolonnas nosaukums	Datu tips	Obligāts	Ierobežojums
CAC_ID	number(10,0)	Jā	Primārā atslēga
CAC_CONDITON	varchar2(4000 byte)	Jā	Nav
CAC_CONDITIONTYPE_ID	number(10,0)	Jā	Ārējā atslēga

Tāpat kā darbības, arī nosacījumi dažādu tipu izmaiņu adaptāciju veikšanā pārklājas, līdz ar to tie tiek glabāti atsevišķā tabulā *ChangeAdaptationCondition* (tabulas aprakstu skat. tabulā 3.4.). Katram ierakstam tiek glabāts nosacījuma tips (nosacījums ir manuāli vai automātiski izpildāms), kā arī pats nosacījums – ja tas ir manuāli izpildāms, tiek glabāts tekstuāls apraksts, kas jāizvērtē pašam izstrādātājam. Ja nosacījums ir automātiski izpildāms, tiek glabāts izpildāmās funkcijas nosaukums.

3.5. tabula
Datu bāzes tabula *CA_ConditionMapping*

Kolonnas nosaukums	Datu tips	Obligāts	Ierobežojums
CACM_ID	number(10,0)	Jā	Primārā atslēga
CACM_SCENARIO_ID	number(10,0)	Jā	Ārējā atslēga
CACM_CONDITION_ID	number(10,0)	Jā	Ārējā atslēga

Tā kā katrai no scenārija ietvaros izpildāmajām darbībām var būt vairāki nosacījumi, tabula *CA_ConditionMapping* (tabulas aprakstu skat. tabulā 3.5.) tiek izmantota, lai nodrošinātu N:N saiti starp tabulām *ChangeAdaptationCondition* un *ChangeAdaptationScenario*.

Automātiski izpildāmos nosacījumus iespējams pārbaudīt pirms katras darbības veikšanas un tam nav nepieciešama izstrādātāja iejaukšanās, līdz ar to iespējams automātiski virzīties pa kādu no scenārijā paredzētajiem ceļiem. Tomēr, veicot manuālu nosacījuma apstrādi (izstrādātājs pieņem lēmumu par turpmāko scenārija gaitu), nepieciešams glabāt

informāciju par to, kādu lēmumu ir pieņēmis izstrādātājs, lai šis lēmums tiktu ņemts vērā visā izmaiņas adaptācijas procesā.

3.6. tabula
Datu bāzes tabula *CA_ManualConditionFulfillment*

Kolonnas nosaukums	Datu tips	Obligāts	Ierobežojums
CAMCF_ID	number(10,0)	Jā	Primārā atslēga
CAMCF_CONDITION_ID	number(10,0)	Jā	Ārējā atslēga
CAMCF_CHANGE_ID	number(10,0)	Jā	Ārējā atslēga
CAMCF_FULFILLMENTSTATUS_ID	varchar2(10 byte)	Jā	Ārējā atslēga

Lēmumu glabāšanai izmantota tabula *CA_ManualConditionFulfillment* (tabulas aprakstu skat. tabulā 3.6.), kurā tiek glabāta ārējā atslēga uz tabulu *Change*, kā arī uz nosacījumu un nosacījuma izpildes statusu (nosacījums izpildīts vai nav izpildīts).

3.1.3. Izmaiņu adaptācijas procesa papildus informācija

Izmaiņu adaptācijas procesa laikā no izstrādātāja var tikt prasīti dažādi papildus dati, kas būs nepieciešami tālāko scenāriju darbību izpildei un nosacījumu pārbaudei.

3.7. tabula
Datu bāzes tabula *ChangeAdaptationAdditionalData*

Kolonnas nosaukums	Datu tips	Obligāts	Ierobežojums
CAAD_ID	number(10,0)	Jā	Primārā atslēga
CAAD_DATA_TYPE_ID	varchar2(10 byte)	Jā	Ārējā atslēga
CAAD_CHANGE_ID	number(10,0)	Jā	Ārējā atslēga
CAAD_DATA	clob	Jā	Nav

Šiem papildus datiem tiek izmantota tabula *ChangeAdaptationAdditionalData* (tabulas aprakstu skat. tabulā 3.7.), kas glabā ārējo atslēgu uz tabulu *Change*, datu tipu (informācija par to, kādam nolūkam izmantojami papildus dati), kā arī paši glabājamie dati. Glabājamo datu formāts atkarīgs no datu veida. Piemēram, glabājot tikai identifikatoru, tas būs vienkāršs skaitlis, taču iespējamas situācijas, kad dati būs kādā citā formātā (JSON, XML, CSV u.c.), tāpēc izmantots universāls datu tips CLOB, kam ir liela ietilpība.

3.2. Evolūcijas apstrādes funkcionalitāte

Kad izmaiņu apstrādes scenāriji saglabāti tabulās *ChangeAdaptationOperation*, *ChangeAdaptationScenario*, *ChangeAdaptationCondition* un *CA_ConditionMapping*, iespējams sākt izmaiņu apstrādi. Lai izmaiņu veiksmīgi adaptētu sistēmas metadatos, izstrādāta funkcionalitāte, kas, savācot tabulā *Change* saglabātos izmaiņas metadatus un izmantojot noteiktos apstrādes scenārijus, aizpilda pārējās evolūcijas apstrādei paredzētās tabulas un pēc iespējas mēģina pielietot automātiskos nosacījumus un darbības.

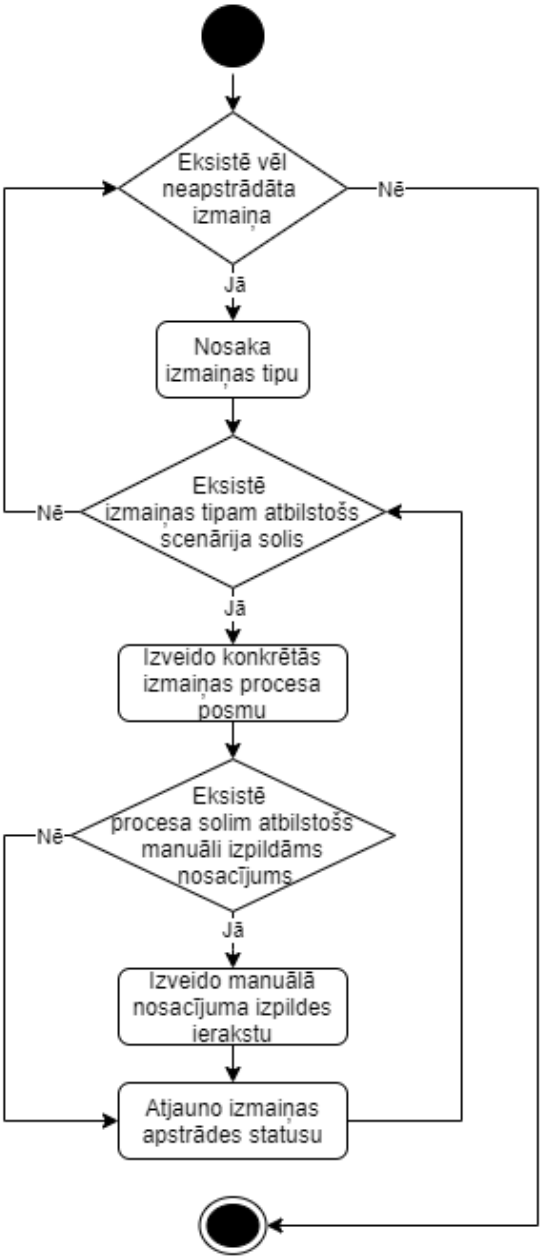
Tiek izdalīti divi evolūcijas apstrādes posmi:

- pimreizējā izmaiņas apstrāde – nosaka izmaiņas adaptācijas scenāriju un izveido sākotnējos ierakstus tabulās *ChangeAdaptationProcess* un *CA_ManualConditionFulfillment*;
- scenārija izpilde – pēc iespējas tiek izpildīti automātiskie nosacījumi un operācijas, kā arī tabulā *ManualConditionFulfillment* saglabātas izstrādātāja manuālās izvēles un tabulā *ChangeAdaptationAdditionalData* saglabāti papildus dati, ja tādi nepieciešami.

Katrs no minētajiem evolūcijas apstrādes posmiem detalizēti aprakstīts nodaļās 3.2.1., 3.2.2. un 3.2.3.

3.2.1. Pirmreizējā izmaiņas apstrāde

Pirmreizējā izmaiņas apstrādē tiek noteikts izmaiņas tips un atrasts izmaiņas tipam atbilstošais adaptācijas scenārijs. Katrai no scenārijā izpildāmajām darbībām tiek izveidots ieraksts tabulā *ChangeAdaptationProcess* ar statusu “Nav adaptēts”. Tiek saglabāts arī laiks, kad procesa solis izveidots, kā arī procesa soļa autors (šajā gadījumā tabulā *Author* jābūt definētam lietotājam “Sistēma”).



3.2. att. **Pirmreizējās izmaiņas apstrādes aktivitāšu diagramma**

Veicot pirmreizējo izmaiņas apstrādi (skat. att. 3.2.), tiek apstrādātas tikai izmaiņas ar statusu "Jauns". Nosakot izmaiņas tipu (izmaiņu tipus skatīt nodaļā 3.3.), pareizajā secībā tiek atlasīti visi izmaiņas tipam atbilstošie scenārija soļi, pēc kā izveidoti konkrētajai izmaiņai atbilstošie procesa soļi. Pēc tam tiek atlasīti konkrētajam procesa solim atbilstošie scenārija soļu manuālie nosacījumi un saglabāti ar statusu "Nav izpildīts". Līdz ar konkrētās izmaiņas procesu izveidošanu, izmaiņas statuss tiek nomainīts uz "Procesā", līdz ar to šīs izmaiņas adaptāciju var uzskatīt par sākušos.

```
PROCEDURE Create_change_adaptation_processes
IS
  v_change_type     types.tp_id%TYPE;
  v_process_created BOOLEAN;
  v_process_id      changeadaptationprocess.cap_id%TYPE;
BEGIN
    -- loops through all new changes
    FOR new_change IN (SELECT *
                       FROM   CHANGE c
                       WHERE  c.ch_statustype_id = const_new_change
                       ORDER  BY c.ch_id) LOOP
        v_process_created := FALSE;

        v_change_type := Define_change_type(new_change);

        IF v_change_type IS NOT NULL THEN
          --  for each new change finds all possible integration scenario steps
          FOR scenario_step IN (SELECT cas.cas_id
                                FROM   changeadaptationscenario cas
                                WHERE  cas.cas_changetype_id = v_change_type
                                START WITH cas.cas_parentscenario_id IS NULL
                                CONNECT BY PRIOR
                                cas.cas_id = cas_parentscenario_id
                                ORDER  BY cas.cas_operation_id) LOOP
              v_process_id := Insert_change_adaptation_process( scenario_step.cas_id,
                                                                new_change.ch_id);

              Insert_manual_condition_fulfillment(v_process_id,
                                                  scenario_step.cas_id,
                                                  new_change.ch_id);

              v_process_created := TRUE;
          END LOOP;

          IF v_process_created THEN
            Update_change_in_progress(new_change.ch_id);
          END IF;
        END IF;
    END LOOP;
END create_change_adaptation_processes;
```

3.3. att. **Pirmreizējās izmaiņas apstrādes programmkoda fragments**

Attēlā 3.3. redzams programmkoda fragments algoritmam, kas veic pirmreizējo izmaiņu apstrādi. Funkcija *Define_change_type*, balstoties uz ierakstu tabulā *Change,* nosaka izmaiņas tipu (to iespējams noteikt, pārbaudot, kuri no tabulas *Change* laukiem ir aizpildīti). Katram no procesa soļiem tiek izsaukta funkcija *Insert_change_adaptation_process,* kura ievieto atbilstošo ierakstu tabulā *ChangeAdaptationProcess*. Funkcijā *Insert_manual_condition_fulfillment* tiek atlasīti scenārija solim atbilstošie manuālās izpildes nosacījumi un saglabāti tabulā *CA_ManualConditionFulfillment*. Funkcija *Update_change_in_progress* uzstāda izmaiņas statusu "Procesā".

3.2.2. Scenārija iegūšana

Lai iegūtu konkrētajai izmaiņai atbilstošu scenāriju, jāņem vērā gan izpildāmās darbības, gan nosacījumi, kas attiecas uz katru no tām.

```
SELECT op_t.tp_type             operation_type,
       cao.cao_operationtype_id operation_type_id,
       cao.cao_operation        operation_instruction,
       pr_st.tp_type            operation_status,
       proc.cap_statustype_id   operation_status_id,
       proc.manual_condition    manual_condition,
       proc.automatic_condition automatic_condition,
       proc.cap_id              process_id
FROM   (SELECT cap.cap_statustype_id,
               cap.cap_id,
               cas.*,
               (SELECT Listagg(cac.cac_condition, '; ')
                         within GROUP (ORDER BY cac.cac_id DESC)
                FROM   ca_conditionmapping cm
                       inner join changeadaptationcondition cac
                               ON cac.cac_id = cm.cacm_condition_id
                WHERE  cm.cacm_scenario_id = cas.cas_id
                       AND cac.cac_conditiontype_id = const_manual_condition)
              manual_condition,
               (SELECT Listagg(cac.cac_condition, '; ')
                         within GROUP (ORDER BY cac.cac_id DESC)
                FROM   ca_conditionmapping cm
                       inner join changeadaptationcondition cac
                               ON cac.cac_id = cm.cacm_condition_id
                WHERE  cm.cacm_scenario_id = cas.cas_id
                       AND cac.cac_conditiontype_id = const_automatic_condition)
              automatic_condition
        FROM   changeadaptationprocess cap
               inner join changeadaptationscenario cas
                       ON cas.cas_id = cap.cap_scenario_id
        WHERE  cap.cap_change_id = in_change_id) proc
       inner join changeadaptationoperation cao
               ON cao.cao_id = proc.cas_operation_id
       inner join types op_t
               ON op_t.tp_id = cao.cao_operationtype_id
       inner join types pr_st
               ON pr_st.tp_id = proc.cap_statustype_id
START WITH proc.cas_parentscenario_id IS NULL
CONNECT BY PRIOR proc.cas_id = proc.cas_parentscenario_id;
```

3.4. att. **Izmaiņai atbilstoša adaptācijas scenārija datu atlases vaicājums**

Attēlā 3.4. redzams SQL datu atlases vaicājums, kas paredzēts secīga konkrētās izmaiņas scenārija atlasei. Ieejas parametrs ir izmaiņas identifikators (attēlā att. Izmaiņai atbilstoša adaptācijas scenārija datu atlases vaicājums – *in_change_id*). Vaicājumā izmantota *Oracle* iebūvētā funkcija *Listagg*, kas ļauj vairāku atsevišķu ierakstu datus sarindot vienā kolonnā, atdalot tos ar izvēlēto atdalītāju. Šādā veidā tiek nodrošināta vieglāk lasāma informācija par katrai operācijai atbilstošajiem nosacījumiem.

3.2.3. Scenārija izpilde

Scenārija izpilde ir balstīta uz nosacījumu pārbaudēm un uz darbību izpildes. Ja scenārija izpildei nepieciešama izstrādātāja iejaukšanās, algoritms tiek pārtraukts un turpināts tikai tad, kad izstrādātājs ir izdarījis savu izvēli par manuālajiem nosacījumiem vai izpildījis noteikto darbību.

```
---- Tries to execute adaptation scenario for specific change
---- (only consecutive, not already adapted and automatic
----  change scenario operations can be executed)
PROCEDURE Run_change_adaptation_scenario(in_change_id IN CHANGE.ch_id%TYPE)
IS
  v_change_scenario SYS_REFCURSOR;
  v_scenario_step   T_SCENARIO_STEP;
BEGIN
    v_change_scenario := Get_change_adaptation_scenario(in_change_id);
    LOOP
        FETCH v_change_scenario INTO v_scenario_step;
        exit WHEN v_change_scenario%NOTFOUND;

        -- processes only not adapted automatic changes
        IF v_scenario_step.operation_status_id = const_not_adapted THEN
          --check conditions
          IF v_scenario_step.operation_type_id = const_automatic
             AND Conditions_fulfilled(in_change_id,
                                      v_scenario_step.automatic_condition,
                                      v_scenario_step.process_id) THEN

            Execute_adaptation_procedure(in_change_id,
                                         v_scenario_step.operation_instruction);

            Set_process_adapted(v_scenario_step.process_id);
          END IF;
          exit;
        END IF;
    END LOOP;
END run_change_adaptation_scenario;
```

3.5. att. **Scenārija izpildes programmkoda fragments**

Attēlā 3.5. redzama datu bāzes procedūra *Run_change_adaptation_scenario*, kas, ieejas parametrā saņemot izmaiņas identifikatoru, apstrādā izmaiņu. Vispirms ar funkciju *Get_change_adaptation_scenario* tiek iegūts kursors ar pašu scenāriju. Šajā funkcijā tiek izmantots nodaļā 3.2.2. aprakstītais datu atlases vaicājums. Tālāk adaptācijas process tiek turpināts tikai tad, ja nepieciešams izpildīt automātisko darbību, kā arī atbilstošajiem nosacījumiem jābūt izpildītiem. Manuālie nosacījumi tiek pārbaudīti, izmantojot tabulu *CA_ManualConditionFulfillment*. Automātisko nosacījumu pārbaudei tiek izmantots dinamiskais *SQL* izsaukums, kas ļauj no tabulas iegūto procedūras nosaukumu izpildīt kā procedūru, iekļaujot to "*execute immediate"* blokā. Pēc šī paša principa tiek izsauktas arī procedūras izmaiņu adaptācijas procesa soļu izpildei.

3.3. Izmaiņu adaptācijas scenāriji

Katras izmaiņas adaptācijai sistēmā atbilst scenārijs jeb veicamo darbību kopums, kas nepieciešams izmaiņas veiksmīgai apstrādei. Darba ietvaros netiek apskatīti visu iespējamo izmaiņu apstrādes scenāriji – aprakstīti reāli notikušo izmaiņu adaptācijas scenāriji. Katra izstrādātā scenārija mērķis ir panākt, lai tas būtu pēc iespējas automātiski izpildāms (lai būtu nepieciešama pēc iespējas mazāka izstrādātāja iejaukšanās izmaiņu adaptācijas procesā), tomēr konceptuālu lēmumu pieņemšana par datu izmantošanu sistēmā nav iespējama bez cilvēka iejaukšanās.

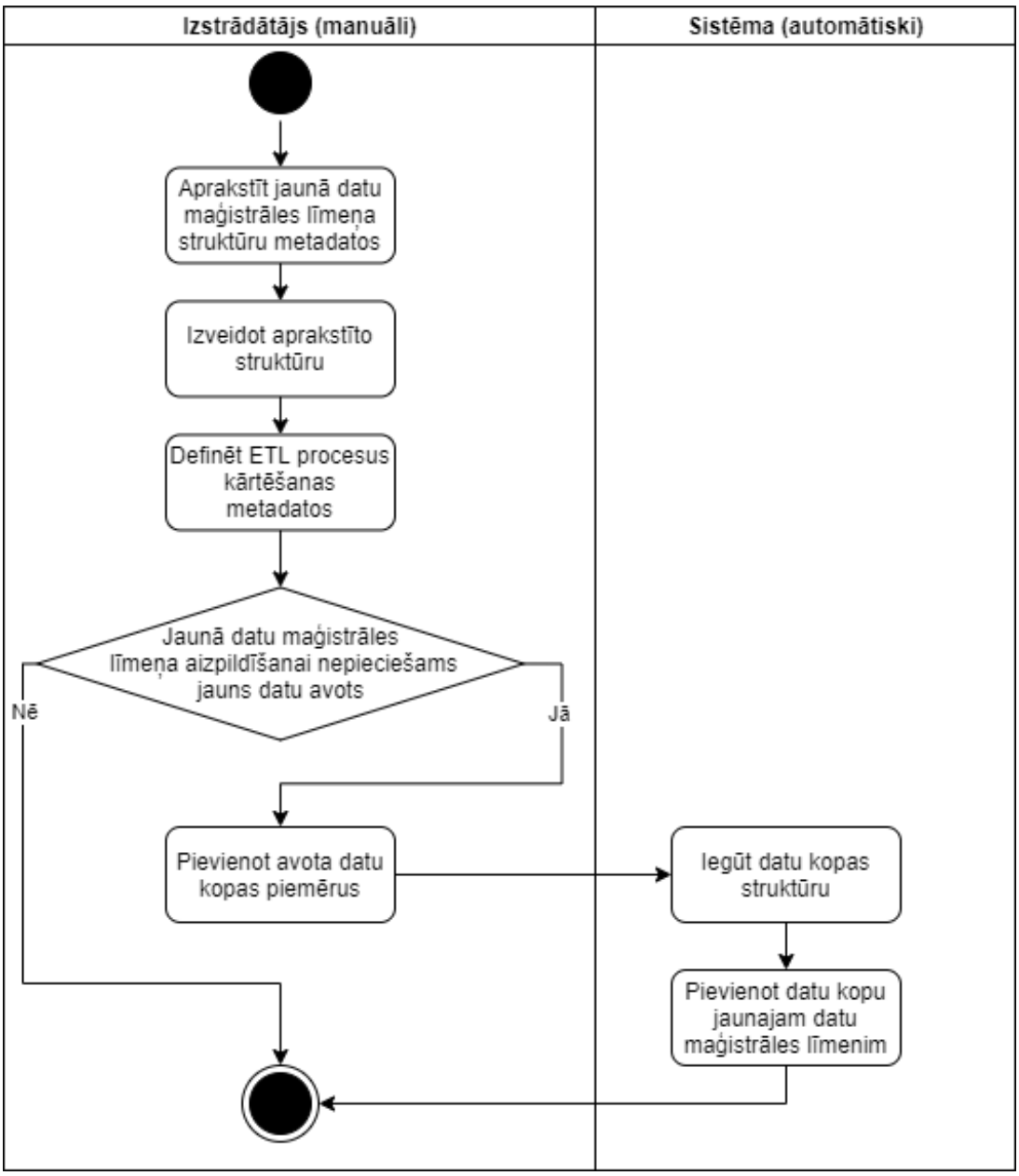
Izmaiņu adaptācijas scenāriju izpildes laikā dati tiek saglabāti gan evolūcijas apstrādes mehānisma ietvaros izstrādātajās datu bāzes tabulās, gan sistēmas esošajās tabulās, kas aprakstītas nodaļā 2.1.3.

Ja izmaiņu adaptācijas scenārijā ietilpst kāda jauna vienuma (datu maģistrāles līmeņa, datu avota, datu kopas, datu vienības) pievienošana, metadati par to tiek aprakstīti tabulā *MetadataProperty*. Pamatdati par katru jaunu vienumu tiek glabāti tiem atbilstošajās tabulās – datu vienības tabulā *DataItem*, datu kopas tabulā *DataSet*, datu avoti tabulā *DataSource,* bet datu maģistrāles līmeņi tabulā *DataHighwayLevel*. Pievienojot jaunas datu vienības, nepieciešams noteikt to savstarpējās attiecības – tās tiek aprakstītas tabulās *Relationship* un *RelationshipElement*. Lielākajā daļā scenāriju pēc nepieciešamības iekļauta jaunu datu avotu pievienošana, kuras gaitā nepieciešams definēt datu maģistrāles līmeņus. Šādā gadījumā, lai noteiktu datu atbilstības starp dažādiem datu maģistrāles līmeņiem un aprakstītu funkcijas, ar kuru palīdzību tiek izrēķinātas datu vienību vērtības, tiek izmantotas tabulas *Mapping* un *MappingOrigin*.

Tuprmākajās apakšnodaļās detalizēti aprakstīti 7 izmaiņu adaptāciju scenāriji.

3.3.1. Datu maģistrāles līmeņa pievienošana

Lai pievienotu jaunu datu maģistrāles līmeni, izstrādātājam nepieciešams sagatavot gan jaunā datu maģistrāles līmeņa struktūras aprakstu, gan identificēt nepieciešamos ETL procesus, kā arī nepieciešamības gadījumā sagatavot datu kopu piemērus.

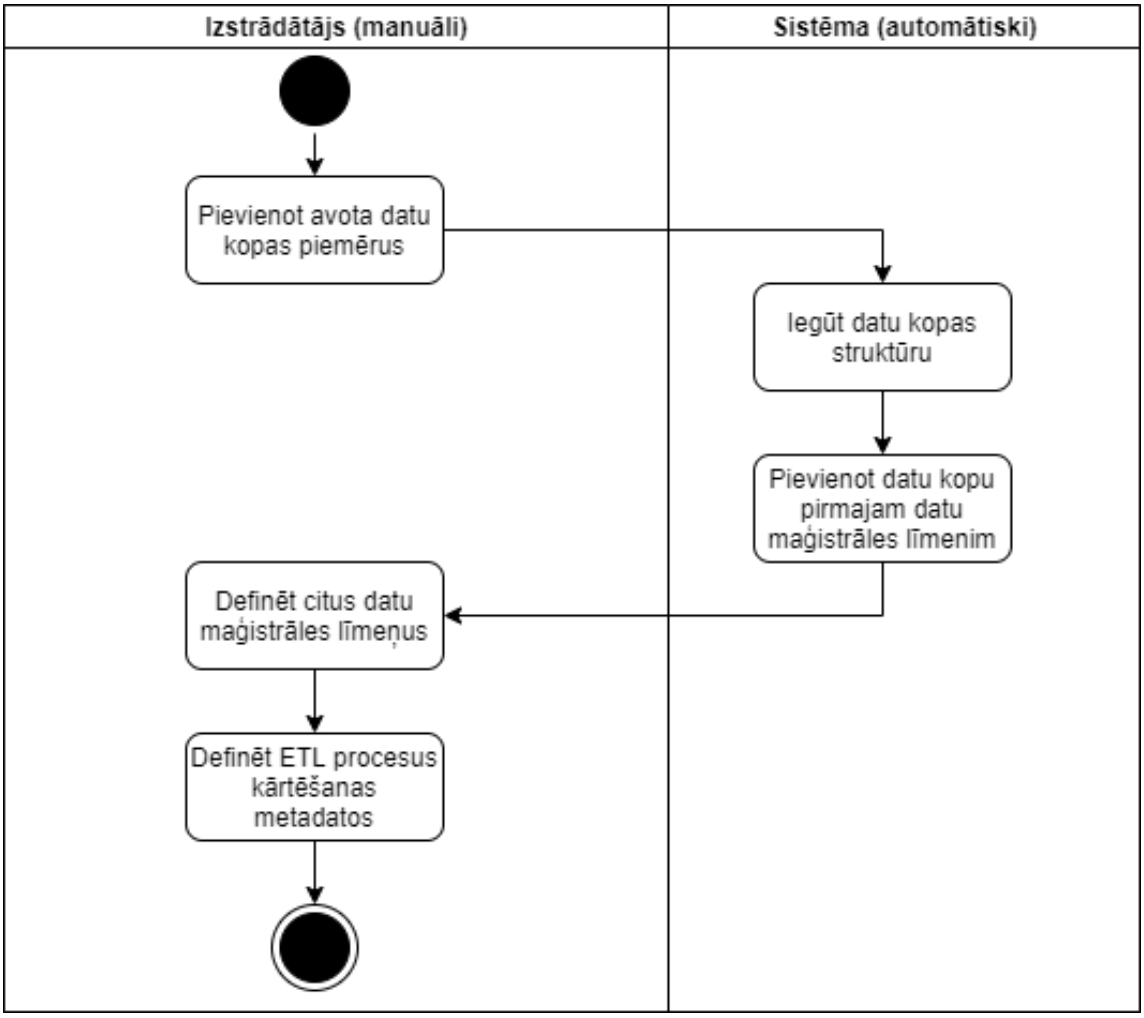


3.6. att. **Datu maģistrāles līmeņa pievienošanas peldceliņu diagramma**

Attēlā 3.6. iespējams apskatīt pilnu datu maģistrāles līmeņa pievienošanas scenāriju. Kā redzams, lielākā daļa darbību ir veicamas manuāli no izstrādātāja puses – nepieciešams aprakstīt jauno struktūru metadatos, kā arī definēt atbilstošos ETL procesus. Taču pēc tam, ja datu maģistrāles līmeņa aizpildīšanai nepieciešams jauns datu avots, sistēma automātiski spēj iegūt pievienotās datu kopas piemēra struktūru un pievienot to izveidotajam datu maģistrāles līmenim.

3.3.2. Datu avota pievienošana

Lai pievienotu jaunu datu avotu, izstrādātājam nepieciešams sagatavot datu kopu piemērus, lai būtu iespējams iegūt to struktūru, kā arī definēt datu maģistrāles līmeņus, kur tiks glabāti transformētie dati.

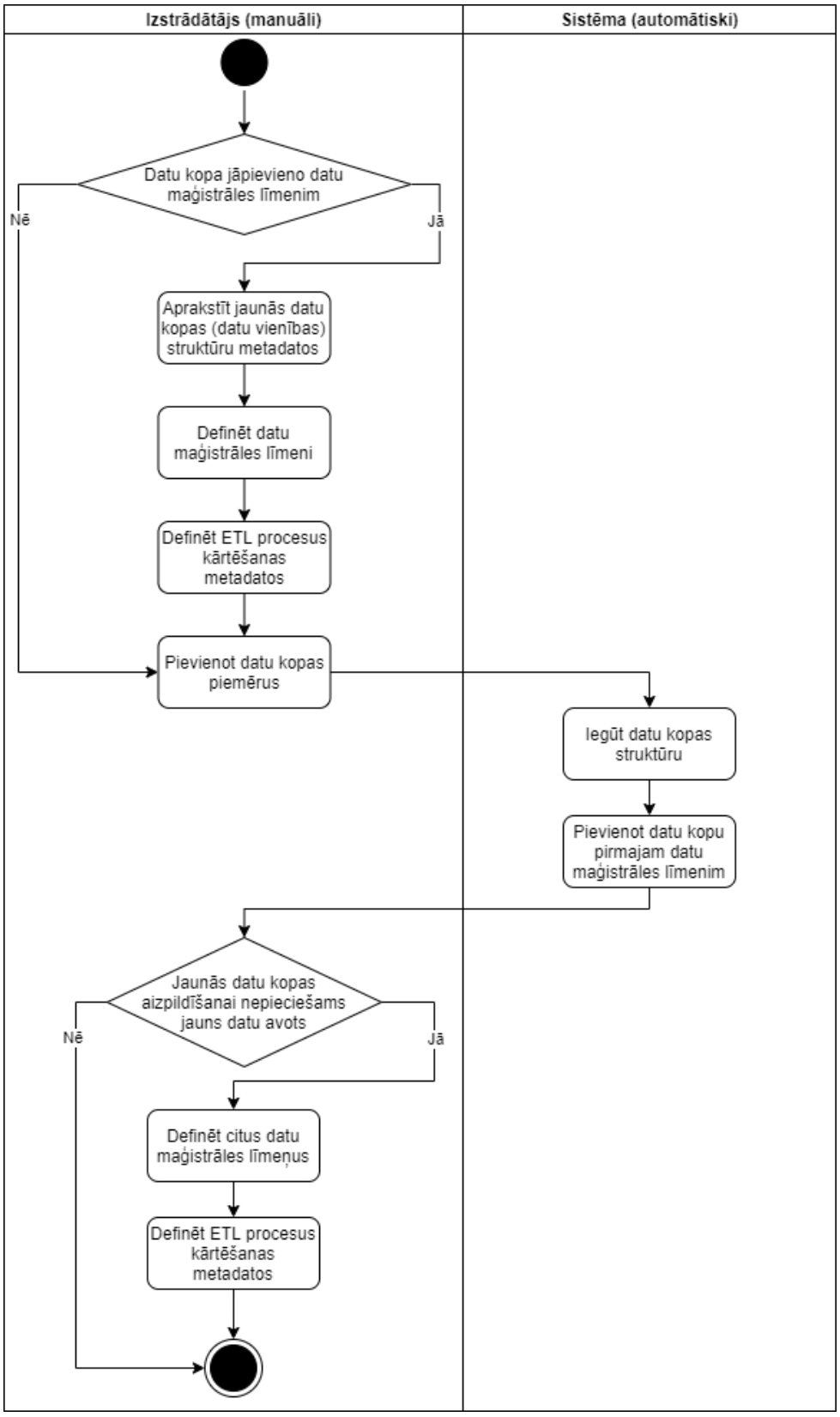


3.7. att. **Datu avota pievienošanas peldceliņu diagramma**

Kā redzams attēlā 3.7., vispirms izstrādātājs manuāli pievieno datu avota datu kopas piemērus un pēc tam uzreiz iespējams definēt citus datu maģistrāles līmeņus un ETL procesus. Datu kopas struktūru no pievienotajiem datu kopas piemēriem iespējams iegūt automātiski, kā rezultātā arī datu kopas pievienošana pirmajam datu maģistrāles līmenim ir automātiska.

3.3.3. Datu kopas pievienošana

Datu kopas pievienošanas scenārijs lielā mērā ir atkarīgs no tā, vai izstrādātājs datu kopu vēlas pievienot datu maģistrāles līmenim vai esošam datu avotam.



3.8. att. **Datu kopas pievienošanas peldceliņu diagramma**

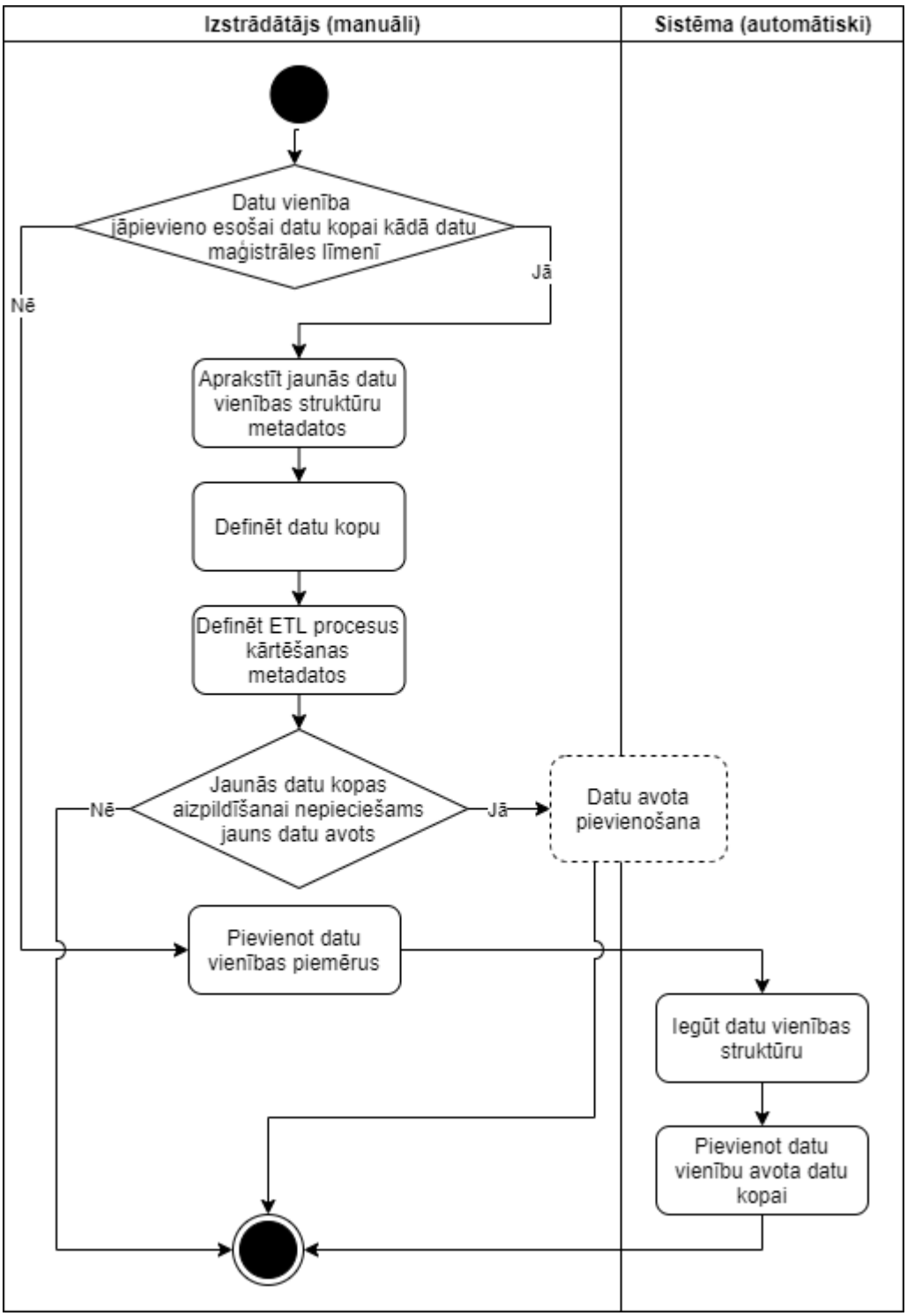
Pievienojot jaunu datu kopu (skat. att. 3.8.), jau sākumā tiek veikta pārbaude, kādai datu struktūrai jauno datu kopu nepieciešams pievienot. Ja datu kopa jāpievieno esošam datu avotam, tad adaptācijas process ir gandrīz pilnībā automātisks – izstrādātājam nepieciešams tikai pievienot datu kopu piemērus struktūras noskaidrošanai. Pretējā gadījumā izstrādātājam jādefinē datu maģistrāles līmenis un ETL procesi, kas atbildīgi par datu kopas datu vienību apstrādi.

3.3.4. Metadatu īpašības pievienošana

Metadatu īpašības pievienošana ir pilnībā manuāls process, tāpēc vienīgā scenārija darbība tiek norādīta kā informācija izstrādātājam, ka viņam nepieciešams izlemt par tālāko pievienotās īpašības izmantošanu. Pievienotās metadatu īpašības tiek glabātas tabulā *MetadataProperty*.

3.3.5. Datu vienības pievienošana

Datu vienības pievienošanas scenārijs pilnībā atkarīgs no tā, vai izstrādātājs datu vienību vēlas pievienot esošai datu kopai, kas atrodas kādā datu maģistrāles līmenī (izņemot pirmo datu maģistrāles līmeni) vai datu avota datu kopai.



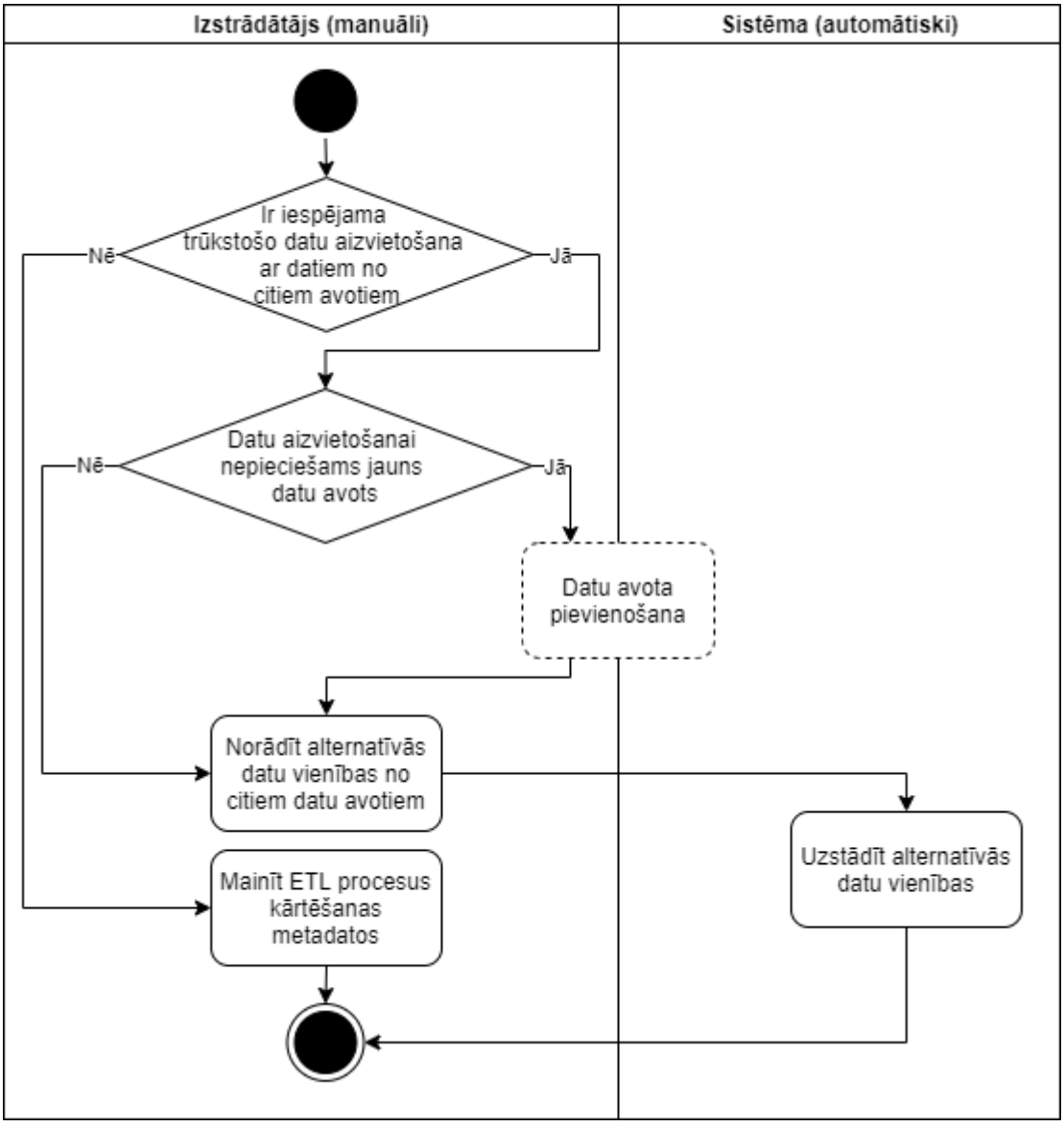
3.9. att. **Datu vienības pievienošanas peldceliņu diagramma**

Pievienojot jaunu datu kopu (skat. att. 3.9.), jau sākumā tiek veikta pārbaude, kādai datu struktūrai jauno datu kopu nepieciešams pievienot. Ja datu vienība jāpievieno esošai datu kopai kādā datu maģistrāles līmenī (izņemot pirmo datu maģistrāles līmeni), izstrādātājam nepieciešams pievienot papildus informāciju un izdarīt lēmumus par pievienojamo datu vienību un ar to saistīto struktūru. Citādi process ir gandrīz automātisks – nepieciešams tikai

pievienot datu vienības piemērus. Ar raustīto līniju apzīmēts datu avota pievienošanas scenārijs (skat. nodaļu 3.3.2.).

3.3.6. Datu avota dzēšana

Lai dzēstu datu avotu, izstrādātājam nepieciešams izvērtēt, vai dzēšamā avota datu kopas datu vienības iespējams aizvietot ar datiem no citiem avotiem.



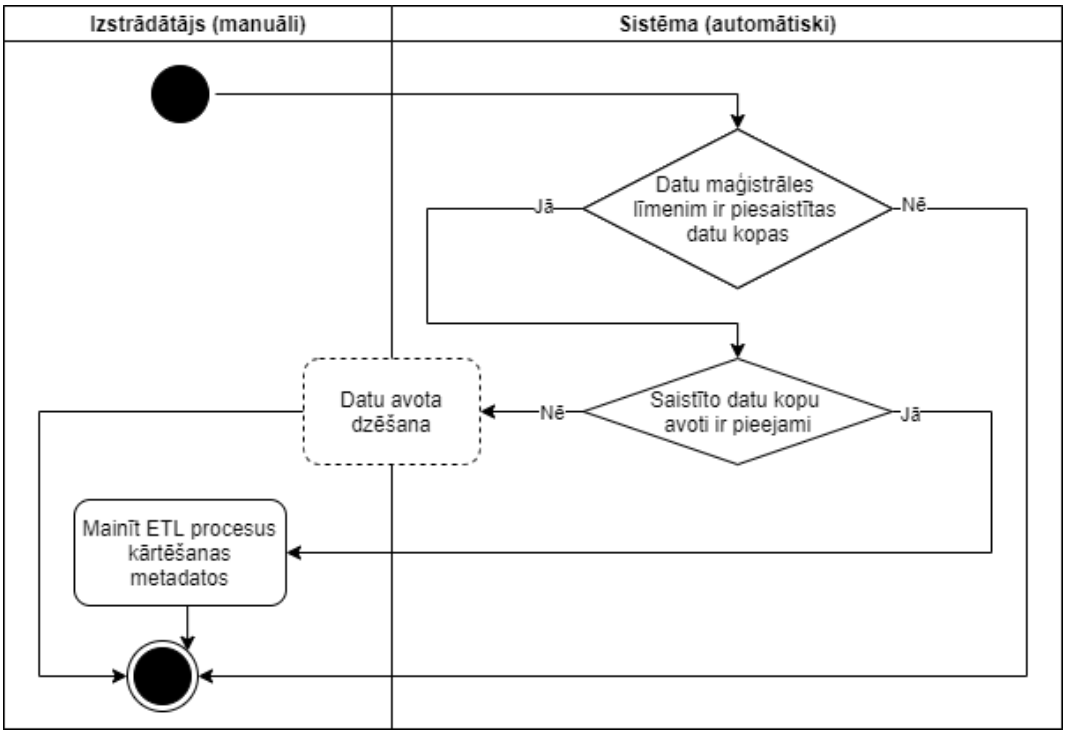
3.10. att. **Datu avota dzēšanas peldceliņu diagramma**

Datu avota dzēšanas procesā (skat. att. 3.10.) ietilpst arī datu avota pievienošana (attēlā atzīmēta ar raustīto līniju, skat. nodaļu 3.3.2.), ja ir iespējama datu avota aizvietošana ar datiem no citiem avotiem un nepieciešams jauns datu avots. Norādot alternatīvās datu

vienības no citiem datu avotiem, tiek izveidots ieraksts tabulā *ChangeAdaptationAdditionalData*.

3.3.7. Datu maģistrāles līmeņa dzēšana

Datu maģistrāles līmeņa dzēšana ir pavisam vienkāršs process, ja datu maģistrāles līmenis nav saistīts ar nevienu datu kopu. Šādā gadījumā nekādi adaptācijas procesi nav nepieciešami. Tomēr šādi gadījumi ir salīdzinoši reti un funkcionējošam datu maģistrāles līmenim lielākoties jau ir piesaistītas kādas datu kopas.



3.11. att. **Datu maģistrāles līmeņa dzēšanas peldceliņu diagramma**

Datu maģistrāles līmeņa dzēšanas procesā (skat. att. 3.11.) iesaistīti arī automātiskie nosacījumi, kuru pārbaudei nav nepieciešama izstrādātāja iejaukšanās – automātiski iespējams pārbaudīt, vai datu maģistrāles līmenim piesaistītas kādas datu kopas, kā arī vai šīs datu kopas ir pieejamas. Ar raustīto līniju apzīmēta datu avota dzēšana (skat. nodaļu 3.3.6.).

REZULTĀTI

Darba gaitā izpildīti visi sākotnēji izvirzītie darba uzdevumi. Vispirms izanalizēta pieejamā teorētiskā literatūra par datu noliktavām, to arhitektūru un veidiem, kā arī tajās izmantotajiem ETL procesiem. Datu noliktavu pielietojuma kontekstā izpētīta un apkopota informācija par lielajiem datiem un to svarīgākajām raksturiezīmēm.

Datu avotu evolūcijas kontekstā pētīta un analizēta esoša sistēma, kurā iekļauti algoritmi dažādu evolūcijas rezultātā radušos izmaiņu atklāšanai. Darbā aprakstīta sistēmas arhitektūra, īpašu uzmanību pievēršot tajā iekļautajai adaptācijas komponentei, kas izvirzīta par pamatu neviendabīgu integrētu datu avotu evolūcijas apstrādes mehānisma izstrādei.

Balstoties uz esošās sistēmas datu bāzes struktūras, kā arī pastāvošo metadatu glabāšanas mehānismu, izstrādāts risinājums, kas paredzēts datu avotu evolūcijas rezultātā radušos izmaiņu adaptācijai sistēmas metadatos. Izstrādāta datu bāzes struktūra izmaiņu adaptācijas procesā radušos metadatu glabāšanai, kā arī datu atlases vaicājumi un procedūras pašas izmaiņu adaptācijas veikšanai. Reāli notikušām datu avotu izmaiņām sastādīti pilni izmaiņu adaptācijas scenāriji, kas sastāv gan no automātiski izpildāmas funkcionalitātes (procedūru formā), gan manuāli veicamiem norādījumiem (veicamās darbības apraksta formā). Izstrādātais risinājums ir universāls, tāpēc tas izmantojams arī nākotnē jaunu izmaiņu veidu adaptācijas scenāriju glabāšanai un adaptēšanai sistēmas metadatos.

SECINĀJUMI

Datu noliktavu jēdziens attīstījies līdz ar lielu datu plūsmu parādīšanos tehnoloģiju vidē. Šie dati tika uzkrāti, bet arvien vairāk tika novērotas dažādas "zirnekļa tīkla" vides, kuras bija grūti pārvaldīt. Lai strukturizētu iegūto informāciju, stratēģisku lēmumu pieņemšanai nepieciešamā informācija tika glabāta atsevišķā datu noliktavā, kuras aizpildīšanai izmantoti ETL procesi. Drīz vien, līdz ar esošo iespēju efektīvi un ātri uzkrāt un pārvaldīt datus, tika ieviests lielo datu jēdziens. Taču, ņemot vērā lielo datu apjomu, dažādību un ātrumu, izkristalizējušās neviendabīgu integrētu datu avotu evolūcijas problēmas.

Bakalaura darba ietvaros izpētīta esoša datu avotu evolūcijas sistēma. Šī sistēma iekļauj datu ieguvi no dažādiem avotiem, kā arī ETL procesus datu pārveidošanai vienotā struktūrā, lai tos būtu iespējams ievietot datu noliktavā. Sistēmas darbība balstīta uz metadatiem, kas tiek glabāti par katru saņemto informācijas vienību un tās transformāciju integrācijai datu noliktavā.

Lai nodrošinātu neviendabīgu integrētu datu avotu izmaiņu adaptāciju sistēmas metadatos, izstrādāts risinājums, kas papildina esošo metadatu glabāšanas shēmu ar papildus datu bāzes struktūru un funkcionalitāti. Minētā struktūra nodrošina adaptācijas scenāriju glabāšanu, kā arī adaptācijas procesa monitoringu, taču funkcionalitāte - adaptācijas soļu izpildi. Izmaiņu adaptācijas scenāriji izstrādāti reāli notikušām izmaiņām.

Darbā izvirzītais mērķis ir sasniegts - atrasts risinājums, kā apstrādāt neviendabīgu integrētu datu avotu evolūcijas rezultātā radušās izmaiņas un adaptēt tās sistēmā. Mērķis sasniegts, veicot izvirzīto uzdevumu izpildi.

Darbā sasniegtais rezultāts izmantojams gan kā esošās sistēmas papildinājums, gan kā universāls konceptuāls piemērs citu līdzīgu sistēmu papildināšanai ar šādu izmaiņu adaptācijas komponenti. Turpmāk plānots attīstīt sistēmu, iekļaujot visu iespējmo datu avotu izmaiņu veidu adaptācijas scenāriju realizāciju.

IZMANTOTĀ LITERATŪRA UN AVOTI

1. Akadēmiskā terminu datubāze "***AkadTerm***". [Tiešsaiste] – [Pārbaudīts 08.05.2020]. Pieejams: http://termini.lza.lv/term.php

2. **Alfredo Cuzzocrea, Laura Puglisi**, "*Encyclopedia of Information Science and Technology, Third Edition*", IGI Global (2015)

3. **Amber Lee Dennis**, "*The Data Warehouse: From the Past to the Present*" (2017). [Tiešsaiste] – [Pārbaudīts 27.04.2020]
   Pieejams: https://www.dataversity.net/data-warehouse-past-present/

4. "*Big Data Now: 2012 Edition*", O'Reilly Media, Inc. (2012)

5. **Wayne W. Eckerson**, "The Role of Big Data and Data Warehousing in the Modern Analytics Ecosystem", Eckerson Group (2018)

6. **Lars George,** "*HBase: The Definitive Guide*", O'Reilly Media, Inc. (2011)

7. **Judith Hurwitz, Alan Nugent, Dr.Fern Halper, Marcia Kaufman**, "*Big Data For Dummies*", John Wiley & Sons, Inc. (2013)

8. **William Inmon**, "*Building the Data Warehouse, Third Edition*", John Wiley & Sons, Inc. (2002) .

9. **William Inmon, Derek Strauss, Genia Neushloss**, "*The Architecture for the Next Generation of Data Warehousing*", Morgan Kaufmann Publishers (2010).

10. **Ahmed Kabiri, Dalila Chiadmi**, "*Survey on ETL Processes*", Journal of Theoretical and Applied Information Technology (2013)

11. **Sean Kelly**, "*Data Warehousing in Action*", John Wiley & Sons, Inc. (1997)

12. **Krish Krishnan**, "*Data Warehousing in the Age of Big Data*", Morgan Kaufmann Publishers (2013)

13. **Felix Naumann**, "*Managing ETL processes*", Hasso Plattner Institute (2009)

14. *Oracle9i Data Warehousing Guide, Release 2.* [Tiešsaiste] – [Pārbaudīts 27.04.2020]. Pieejams: https://docs.oracle.com/cd/B10500_01/server.920/a96520/concept.htm#50413

15. **Sweety Patel**, "*What is Data Warehouse?*", Department of Computer Science, Fairleigh Dickinson University (2012). [Tiešsaiste] – [pārbaudīts 27.04.2020] Pieejams: https://www.omicsonline.org/open-access/what-is-data-warehouse-2277-1891-1000117.php?aid=12878

16. **Paulraj Ponniah**, "*Data warehousing Fundamentals for IT Professionals, Second Edition*", John Wiley & Sons, Inc. (2010)

17. **Erhard Rahm, Hong Hai Do**, "*Data cleaning: Problems and Current Approaches*", Bulletin of the Technical Committee on Data Engineering (2000)

18. **Stefano Rizzi, Matteo Golfarelli**, "*Data Warehouse Design: Modern Principles and Methodologies*", McGraw Hill Professional (2009).

19. **Leo Willyanto Santoso, Yulia**, "*Data Warehouse with Big Data Technology for Higher Education*", 4th Information Systems International Conference (2017)

20. **K. Shvachko, H. Kuang, S. Radia, R. Chansler**, "*The Hadoop Distributed File System*", IEEE 26th Symposium on Mass Storage Systems and Technologies (2010)

21. **Darja Solodovnikova, Laila Niedrite**, "*Change Discovery in Heterogeneous Data Sources of a Data Warehouse*", 14th International Baltic Conference on Databases and Information Systems (2020)

22. **Darja Solodovnikova, Laila Niedrite**, "*Towards a Data Warehouse Architecture for Managing Big Data Evolution*", International Conference on Data Science, E-learning and Information Systems (2018)

23. **Darja Solodovnikova, Laila Niedrite, Aivars Niedritis**, "*On Metadata Support for Integrating Evolving Heterogeneous Data Sources*", European Conference on Advances in Databases and Information Systems (2019)

24. **Tomislav Šubić, Patrizia Poščić, Danijela Jakšić**, "*Big Data in Data Warehouses*", Department of Informatics, University of Rijeka (2015)

25. **Nasser Thabet, Tariq Rahim Soomro**, "*Big Data Challenges*", Journal of Computer Engineering & Information Technology (2015)

26. **A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang., S. Antony, H. Liu, R. Murthy**, "*Hive – a petabyte scale data warehouse using Hadoop*", International Conference on Data Engineering (2010)

27. **Phani Vivekanand Kandalam**, "*Data Warehousing Modernization: Big Data Technology Implementation*", St. Cloud State University (2016)

28. **Lidong Wang**, "*Heterogeneous Data and Big Data Analytics*", Automatic Control and Information Sciences (2017). [Tiešsaiste] – [pārbaudīts 27.04.2020]
Pieejams: http://pubs.sciepub.com/acis/3/1/3/

Bakalaura darbs „Neviendabīgu integrētu datu avotu evolūcijas apstrāde” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Lauma Svilpe _____

Rekomendēju/nerekomendēju darbu aizstāvēšanai *(nederīgo svītro vadītājs)*

Vadītājs: asociētā profesore Dr.dat. Darja Solodovņikova _____ __.06.2020.

Recenzents: profesors Dr.dat. Ģirts Karnītis

Darbs iesniegts Datorikas fakultātē __.06.2020.

Dekāna pilnvarotā persona: vecākā metodiķe Ārija Sproģe _____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

____.06.2020. prot. Nr. _____

Komisijas sekretārs(-e): _____