9. Sisteminiai duomenų bazių aspektai

9.1. Duomenų saugumas ir kreipimosi ribojimas

Duomenų saugumas duomenų bazių kontekste reiškia apsaugą nuo neleistino duomenų ieškojimo, keitimo bei šalinimo. Duomenų apsaugai reliacinėje DBVS keliami tokie reikalavimai:

- lentelės duomenys gali būti prieinami vieniems vartotojams, bet neprieinami kitiems;
- vieniems lentelės vartotojams galima susipažinti su duomenimis ir juos keisti, o kitiems - tik susipažinti;
- daliai vartotojų gali būti prieinamos ne visų lentelės stulpelių reikšmės;
- daliai vartotojų lentelės duomenis leidžiama pasiekti tik naudojantis taikomosiomis programomis, bet ne interaktyviai.

Už duomenų apsaugą atsako DBVS. Duomenų apsaugos uždaviniui spręsti RDBVS paprastai yra dvi galimybės:

- virtualios lentelės, kuriomis galima "paslėpti" tam tikras lentelės eilutes ir stulpelius;
- kreipimosi į duomenis ribojimo posistemis, kuris leidžia vartotojams, turintiems atitinkamą teisę, apibrėžti konkrečias kitų vartotojų teises.

Virtualios lentelės "paslepia" duomenis tik sąlygiškai. Taip "paslėpus" duomenis vartotojas nepraranda galimybės sužinoti pradinių lentelių vardus ir pasiekti jose esančius duomenis tiesiogiai, apeinant virtualią lentelę. Kreipimosi draudimą realizuoja specialus DBVS teisių valdymo posistemis. Reliacinėse DBVS vartotojų teisės duomenų atžvilgiu yra nustatomos SQL sakiniais. Duomenų apsaugos valdymą nusako trys pagrindinės sąvokos:

- **DB vartotojai**. Visi veiksmai su duomenimis duomenų bazėje yra atliekami tam tikro vartotojo vardu. DBVS atsisako įvykdyti operaciją, jei vartotojas neturi tam teisės.
- **DB objektai** - tai visi DB elementai, kurių atžvilgiu yra nustatomos teisės (privilegijos). Paprastai teisės yra nustatomos ne tik lentelių ir virtualių lentelių atžvilgiu, bet ir visos DB bei taikomųjų programų atžvilgiu.
- **Privilegijos** – tai vartotojų teisės atlikti vienokius ar kitokius veiksmus su DB objektais. Su kiekvienu vartotoju kiekvieno DB objekto atžvilgiu yra susiejama tam tikra aibė veiksmų, kuriuos vartotojas gali atlikti.

Kiekvienam DB vartotojui yra suteikiamas tapatumo požymis – trumpas vardas, vienareikšmiškai nusakantis vartotoją. DB vartotojo vardas nebūtinai reiškia vieną asmenį. Praktikoje vienas ir tas pats vardas gali žymėti grupę asmenų, pavyzdžiui, su vardu *Informatikas* gali būti susiejami visi darbuotojai, turintys informatiko kvalifikaciją. Todėl, kai kuriose DBVS vietoj vartotojo tapatumo požymio (angl. *user-id*) vartojamas tikslesnis teisių tapatumo požymis (angl. *authorization-id*).

SQL standartas apibrėžia, kad duomenų apsauga realizuojama per vartotojo tapatumo požymį, tačiau neapibrėžia, nei kaip vartotojai užregistruojami, nei kaip tapatumo požymis susiejamas su SQL sakiniais. Daugumoje operacijų sistemų yra savi apsaugos posistemiai, kurie ne tik užregistruoja vartotojus, bet ir stebi, kad sistemą vartotų tik turintys tam teisę vartotojai. Daugeliu atvejų visiškai pakanka, kad DBVS nedubliuotų šių funkcijų ir tik suteiktų tam tikras teises DB objektų atžvilgiu. Operacijų sistema atpažįsta vartotoją (pagal įvestą tapatumo požymį bei, kaip įprasta, slaptažodį) ir perduoda jo vardą DBVS. Kai vartotojas bando atlikti kokią nors operaciją su DB, DBVS tikrina ar šis, einamasis vartotojas, turi tam teisę.

Informacija apie vartotojų teises DB valdymo sistemai yra pranešama SQL sakiniais `GRANT` ir `REVOKE`. Sakiniu `GRANT` yra suteikiamos tam tikros teisės vartotojui ar jų grupei nurodyto DB objekto atžvilgiu, o sakiniu `REVOKE` jos yra atšaukiamos. Šių SQL sakinio bendras pavidalas:

GRANT <privilegijų sąrašas> [ON [<objekto tipas>] <objektų sąrašas>]
TO <vartotojų sąrašas> [WITH GRANT OPTION] ,

REVOKE [GRANT OPTION FOR] <privilegijų sąrašas>
[ON [<objekto tipas>] <objektų sąrašas>] FROM <vartotojų sąrašas>
[CASCADE | RESTRICT],

kur <privilegijų sąrašas> - viena ar kelios privilegijos, atskirtos kableliais, arba bazinis žodis ALL PRIVILEGES, reiškiantis visas įmanomas privilegijas; <objekto tipas> - bazinis žodis, nurodantis vieną iš DB objektų tipų (duomenų bazė, lentelė ir pan.); <objektų sąrašas> - DB objektų, kurių atžvilgiu yra suteikiamos teisės, vardai; <vartotojų sąrašas> - vartotojų tapatumo požymių sąrašas arba bazinis žodis PUBLIC, reiškiantis visus vartotojus. Jei teisės yra suteikiamos lentelei (lentelėms), tai objekto tipą (TABLE) galima nenurodyti.

Apžvelgsime pagrindines privilegijų grupes.

Privilegijos lentelėms. SQL1 standarte lentelių ir jų vaizdų (virtualių lentelių) atžvilgiu yra numatytos keturios privilegijos:

- SELECT – leidžiama susipažinti su nurodytos lentelės duomenimis. Vartotojui, turinčiam šią privilegiją (virtualiai) lentelei, leidžiama vykdyti užklausas, kurių FROM frazėje yra nurodyta ta lentelė;
- INSERT – leidžiama įvesti į (virtualią) lentelę naujus duomenis;
- DELETE – leidžiama šalinti duomenis iš (virtualios) lentelės;
- UPDATE – leidžiama keisti (virtualios) lentelės duomenis. Ši teisė gali būti apribota, leidžiant keisti tik kai kurių lentelės stulpelių reikšmes.

Pavyzdžiui,

GRANT SELECT ON TABLE *Vykdytojai* TO *Jonas* ,
GRANT INSERT ON TABLE *Vykdymas* TO PUBLIC,
GRANT SELECT, UPDATE, DELETE ON TABLE *Vykdytojai* TO *Jonas*,
GRANT ALL PRIVILEGES ON *Vykdytojai, Projektai, Vykdymas* TO *Informatikas,*
GRANT UPDATE (*Valandos*) ON TABLE *Vykdymas* TO PUBLIC,
REVOKE ALL PRIVILEGES ON TABLE *Vykdymas* FROM PUBLIC.

Šiuolaikinėse komercinėse DBVS šis privilegijų sąrašas yra iš esmės papildytas. Paminėsime tik keletą papildomų privilegijų, vartojamų DBVS DB2:

- REFERENCES – leidžiama kurti lentelėms išorinius raktus, imant pagrindu nurodytą sakinyje lentelę;
- ALTER – leidžiama keisti lentelės struktūrą, pavyzdžiui, papildyti ją naujais stulpeliais.

SQL standarte nėra numatytas privilegijų dalijimas atskiroms lentelės eilutėms. Jei to reikia, tai problema nesunkiai sprendžiama naudojantis virtualia lentele. Tarkime, įstaigoje, atitinkančioje DB *Darbai*, eiliniai darbuotojai gali peržiūrėti duomenis tik apie save. Jeigu Jonaičio DB vartotojo vardas yra *Jonaitis*, tai uždavinį jo atžvilgiu galima išspręsti tokiais SQL sakiniais:

CREATE VIEW *Vykdytojas_Jonaitis*
AS SELECT * FROM *Vykdytojai* WHERE *Pavardė* = ‘Jonaitis’,
CREATE VIEW *Vykdo_Jonaitis* AS SELECT * FROM *Vykdymas*
WHERE *Vykdytojas* = (SELECT *Nr* FROM *Vykdytojai* WHERE *Pavardė* = ‘Jonaitis’),
REVOKE ALL PRIVILEGES ON *Vykdytojai, Vykdymas* FROM *Jonaitis*,
GRANT SELECT ON *Jonaitis, Vykdo_Jonaitis* TO *Jonaitis*.

Privilegijos duomenų bazėms. Šiai grupei priklausančių privilegijų sąrašas nėra standartizuotas. Paminėsime tik keletą DBVS DB2 vartojamų privilegijų:

- `CONNECT` – teisė dirbti su DB, tiksliau vykdyti sakinį `CONNECT`;
- `CREATETAB` – leidžiama kurti naujas lenteles;
- `BINDADD` – teisė kurti jungtinius vykdymo planus (vykdyti utilitą `BIND`);
- `DBADM` – DB administratoriaus privilegija, suteikianti teisę atlikti bet kokius veiksmus su DB.

Pavyzdžiui,

`GRANT CONNECT ON DATABASE` *Darbai* `TO PUBLIC`,
`GRANT DBADM ON DATABASE` *Darbai* `TO` *Informatikas*.

Privilegijos vykdymo planams. Daugelyje DBVS yra numatyta privilegijos vykdomiesiems planams, kuriuos sukuria SQL preprocesorius tiesiogiai arba kartu su DBVS procedūra `BIND`. DBVS DB2 yra keletas šios grupės privilegijų, tarp kurių dvi pagrindinės yra:

- `BIND` – teisė sukurti planą. Vartotojas, turintis tokią privilegiją, gali vykdyti procedūrą `BIND`;
- `EXECUTE` – teisė vykdyti planą. Vykdant vartotojui SQL sakinį iš programos, teisės įvykdyti operaciją DB objekto atžvilgiu nepakanka - jis dar turi turėti teisę atlikti tai per vykdymo planą.

Pavyzdžiui,

`GRANT BIND ON PACKAGE` *Prog1* `TO` *Informatikas*,
`GRANT EXECUTE ON PACKAGE` *Prog1* `TO PUBLIC`.

Vartotojas, sukūręs DB objektą, pvz., lentelę, tampa to objekto savininku. Vartotojas, sukūręs objektą, savaime įgyja visas privilegijas jo atžvilgiu ir gali skirstyti jas kitiems vartotojams. Suteikus vartotojui tam tikrą teisę, šis netampa pilnateisu objekto šeimininku – jis negali skirstyti privilegijų to objekto atžvilgiu kitiems vartotojmas. Kartais toks apribojimas nėra patogus. Jei vartotojui suteikiama privilegija su parinktimi `WITH GRANT OPTION`, šis savo nuožiūra galės "perleisti" jam suteiktą privilegiją kitiems vartotojams. Pastebėjus, kad vartotojas "nepateisina lūkesčių", "švaistydamasis" privilegijomis, sakiniu `REVOKE` su fraze `GRANT OPTION FOR` galima atimti iš jo teisę skirstyti privilegijas.

Laikantis standarto SQL2, atimant iš vartotojo privilegijas, reikia nurodyti vieną iš pasirinkčių: `CASCADE` ar `RESTRICT.` Jei privilegija atimama su parinktimi `CASCADE`, tai ji atimama ir iš visų kitų vartotojų, kurie šią privilegiją gavo iš sakinyje `REVOKE` nurodyto vartotojo. Jei privilegijos atėmimo sakinyje `REVOKE` nurodyta `RESTRICT`, tai privilegijos nepavyks atimti, jei sakinyje nurodytas vartotojas jau yra suteikęs privilegiją kam nors kitam.

Suprantama, vartotojui neleidžiama suteikti nei sau, nei kitiems vartotojams privilegijų, aukštesnių negu jam pačiam buvo suteikta.

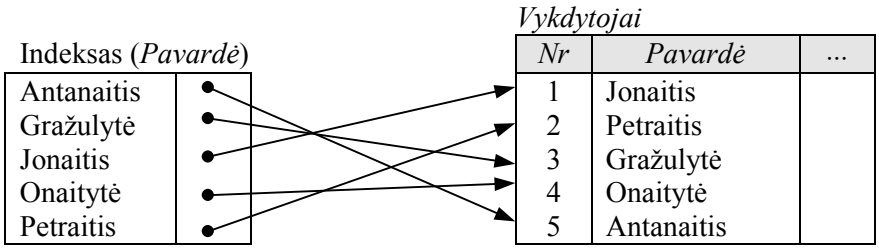
9.2. Indeksai

Lentelės duomenų **indeksas** reliacinėse DBVS yra duomenų paieškos palengvinimo priemonė. Panašiai kaip knygos dalykinė rodyklė palengvina reikiamų puslapių paiešką knygoje, indeksavimas pagreitina informacijos išrinkimą iš duomenų bazės. Duomenų bazėje indeksas atlieka loginių nuorodų į fizinę duomenų vietą vaidmenį.

Tarp indekso ir knygos dalykinės rodyklės yra ir esminių skirtumų. Skaitytojas pats nusprendžia, ar naudotis dalykine rodykle reikiamai informacijai rasti. DB vartotojas tik sprendžia, ar jam sukurti indeksą. Ieškant duomenų, DBVS sprendžia pati, ar naudoti paieškai indeksą. Vartotojas neturi jokių galimybių įtakoti DBVS sprendimo – tai jau yra DBVS dispozicijoje. Indeksas skiriasi nuo dalykinės rodyklės ir tuo, kad DB lentelė gana dažnai turi keletą indeksų, o knygoje paprastai tėra viena dalykinė rodyklė. Panašiai kaip knygoje gali nebūti dalykinės rodyklės, DB lentelė gali neturėti nė vieno indekso.

Indeksas - tai surūšiuota reikšmių ir nuorodų į reikšmes atitinkamas lentelės eilutes aibė. Indeksas apibrėžiamas vienam ar keletui lentelės stulpelių. Indeksas nėra lentelės duomenų dalis - tai atskiras DB objektas. Sukūrus indeksą, DBVS automatiškai sukuria atitinkamą duomenų struktūrą ir stebi, kad indeksas, besikeičiant lentelės duomenims, nuolat atspindėtų jos duomenis ir kad išliktų išrūšiuotas pagal apibrėžtų stulpelių reikšmes.

Indeksai yra labai patogūs todėl, kad duomenų paieška išrūšiuotame masyve yra daug lengvesnė negu neišrūšiuotame. Bet nereikia manyti, kad pakanka saugoti lentelėje išrūšiuotus duomenis. Lentelės duomenys gali būti išrūšiuoti tik pagal vieną kriterijų (tam tikrų stulpelių reikšmes). Jei duomenų masyvas išrūšiuotas pagal vieną kriterijų, tai jis dažnai nėra išrūšiuotas pagal kitą. Kadangi, duomenų paieška lentelėje gali vykti pagal įvairius kriterijus, tai, nors lentelė ir išrūšiuota pagal kurį nors kriterijų, efektyviai paieškai pagal kitus kriterijus reikia papildomų indeksų. Tarkime, lentelė *Vykdytojai* yra išrūšiuota pagal stulpelio *Nr* reikšmes. Sukurkime indeksą dar ir stulpelio *Pavardė* reikšmėms:



DBVS, vykdydama užklausą lentelei *Vykdytojai* pagal nurodytą *Nr*, gali pasinaudoti lentelės savybe, kad jos eilutės yra išrūšiuotos pagal šio stulpelio reikšmes, o vykdant užklausą pagal pavardę gali naudoti indeksą.

Kadangi duomenų indeksas yra fizinė sąvoka, o ne loginė, tai SQL standarte nėra galimybės kurti indeksus. Tačiau daugumoje komercinių DBVS tokia galimybė yra. Kai kuriose iš jų indekso galimybės yra ypač išplėtotos.

Indeksas dar reikalingas ir dėl loginės lentelės rakto vartojimo. Prisiminkime, kad lentelės kiekvienos eilutės rakto reikšmių rinkinys turi būti unikalus lentelėje. Vadinasi, kiekvieną kartą, kai į lentelę įterpiama nauja eilutė, DBVS turi tikrinti ar nebus pažeistas rakto vientisumas. O tai reiškia, kad DBVS turi patikrinti, ar tuo metu lentelėje dar nėra eilutės su nurodyta `INSERT` sakinyje rakto reikšme. Šią procedūrą nesunku realizuoti tik turint išrūšiuotą rakto reikšmių masyvą. Jei lentelė turi tik vieną raktą (pirminį), tai DBVS gali laikyti lentelę išrūšiuotą pagal to rakto reikšmes. Tačiau jei lentelė turi keletą raktų, tai reikalingas papildomas indeksas.

Indeksai yra naudojami:

- siekiant padidinti duomenų paieškos efektyvumą, kadangi paieška išrūšiuotame duomenų masyve yra kur kas lengvesnė negu neišrūšiuotame;
- siekiant užtikrinti atitinkamų stulpelių reikšmių unikalumą lentelėje, kadangi tik išrūšiuotame duomenų masyve galima greitai patikrinti, ar atnaujinant duomenis nebus pažeistas reikšmių unikalumas.

Daugelyje DBVS yra sakinys

`CREATE [UNIQUE] INDEX` <indekso vardas> `ON` <lentelės vardas>(<stulpelių vardai>).

Sukurkime, pavyzdžiui, keletą indeksų lentelei *Vykdytojai*:

`CREATE UNIQUE INDEX` *IndeksasPavardei* `ON` *Vykdytojai*(*Pavardė*) ,
`CREATE INDEX` *IndeksasKvalifikacijai* `ON` *Vykdytojai*(*Kvalifikacija*).

Šiame pavyzdyje *IndeksasPavardei* ir *IndeksasKvalifikacijai* yra indeksų pavadinimai. Bazinio žodžio `UNIQUE` įtraukimas į indekso *IndeksasPavardei* apibrėžimą garantuoja, kad bus užtikrinama stulpelio *Pavardė* reikšmių unikalumas lentelėje *Vykdytojai*. Tokiu būdu užtikrinamas rakto *Pavardė* vientisumas. Priminsime, kad lentelės raktų vientisumo užtikrinimą jau aptarėme ankstesniame skyriuje.

Daugelis DBVS leidžia apibrėžti **sudėtinius indeksus** – indeksus, kuriuos sudaro keletas lentelės stulpelių. Sudėtiniai indeksai vartojami sudėtinių raktų vientisumui užtikrinti arba kai užklausos yra dažniau vykdomos pagal kelis logiškai susijusius stulpelius, negu pagal kiekvieną iš jų atskirai. Tarus, kad lentelėje *Vykdytojai* yra stulpelis *Vardas*, vietoj indekso *IndeksasPavardei* būtų galima apibrėžti tokį indeksą:

`CREATE UNIQUE INDEX` *IndeksasAsmeniui* `ON` *Vykdytojai*(*Pavardė, Vardas*) .

Indeksas negali būti sukurtas virtualiai lentelei, tačiau vykdant užklausas virtualiai lentelei, DBVS atsižvelgia į bazinių lentelių indeksus.

Užklausose indeksai niekuomet nenurodomi. DBVS pati nusprendžia, ar užklausos vykdymui naudoti indeksus, ir jei taip, tai kokius konkrečiai ir kaip juos panaudoti.

Nors indeksas turi daug gerų savybių, jais negalima piktnaudžiauti, kadangi:

- indeksui reikia kompiuterio atminties (kartai indeksui gali prireikia daugiau vietos atmintyje nei pačios lentelės duomenims);
- indeksui (jo išrūšiavimui) užtikrinti reikalingas procesoriaus laikas. Didelis lentelės indeksų kiekis gali neigiamai atsiliepti lentelės duomenų atnaujinimo laikui.

Sukuriant indeksą nurodomas jo vardas, kuris dar gali būti pavartotas vieninteliu atveju – pašalinant indeksą SQL sakiniu:

`DROP INDEX` <indekso vardas> .

Pašalinus indeksą, jo duomenų struktūra yra sunaikinama ir kompiuterio atmintis yra atlaisvinama.

9.3. Bendro duomenų naudojimo problemos

DB yra bendrai naudojama sistema. Tai reiškia, kad vienu metu gali būti vykdomos kelios transakcijos. Keliems vartotojams (programoms) dirbant tuo pačiu metu, DBVS ne tik turi užtikrinti duomenų atkūrimą, kai atšaukiami transakcijoje padaryti duomenų pakeitimai, bet ir garantuoti, kad vartotojai netrukdytų vienas kitam. Kiekvienas vartotojas turi jaustis taip, tartum jis tik vienas dirbtų su DB. Todėl DBVS derina vartotojų darbą.

Kad suprastume, kaip transakcijos vykdomos kartu ir kokius uždavinius sistemai tenka spręsti, apžvelgsime problemas, kurios kyla, kai transakcijos nederinamos. Tokių problemų yra gana daug, tačiau daugelį jų galima priskirti kuriai nors vienai tipiniai. Aptarsime dvi tipines situacijas, kai dviem transakcijoms dirbant su tais pačiais duomenimis, gali būti gauti neteisingi rezultatai, nors kiekviena iš jų, atskirai paėmus, dirba teisingai.

Vieną iš dviejų kartu veikiančių transakcijų pažymėkime *A*, o kitą - *B*. Bendrai naudojamą konkrečios lentelės eilutę pažymėkime *R*. Fraze `FETCH` *R* žymėsime eilutės *R* nuskaitymą į atmintį, o `UPDATE` *R* – tos eilutės reikšmių atnaujinimą (keitimą). Laiko momentus, kuriais kreipiamasi į DB, žymėsime t_1, t_2, t_3,…. Be to, $\forall\ i, j$, kuriems teisinga nelygybė $i < j$, yra teisinga ir $t_i < t_j$.

Pakeistų duomenų praradimo problema. Būseną, kurioje susidaro ši problema, galima pavaizduoti taip:

Transakcija *A*	Laikas	Transakcija *B*
-	|	-
`FETCH` *R*	t_1	-
-	|	-
-	t_2	`FETCH` *R*
-	|	-
`UPDATE` *R*	t_3	-
-	|	-
-	t_4	`UPDATE` *R*
-	↓	-

Abi transakcijos *A* (laiko momentu t_1) ir *B* (laiko momentu t_2) perskaito eilutės *R* reikšmes, kurias vėliau abi transakcijos pakeičia (atitinkamai, laiko momentais t_3 ir t_4). Laiko momentu

t_4 transakcija A praranda atnaujintus duomenis, kadangi jos pakeistus duomenis pakeičia transakcija B. Transakcija B keičia eilutę R atsižvelgdama tik į jos pačios perskaitytas reikšmes, bet ne į transakcijos A priskirtas naująsias reikšmes. Taip transakcijoje A atliktas duomenų keitimas netenka prasmės.

Ši problema kyla kiekvieną kartą, kai dvi transakcijos nuskaito iš DB tuos pačius duomenis, kuriuos jos abi ir keičia. Nesprendžiant šios problemos, duomenų bazėje atsispindėtų tik paskutinis duomenų keitimas.

Priklausymo nuo neužfiksuoto pakeitimo problema. Ši problema atsiranda, kai nuskaitoma eilutė, kurią anksčiau jau keitė kita transakcija, tačiau tas pakeitimas dar nebuvo galutinai įtvirtintas. Kadangi ta kita transakcija dar nepatvirtino pakeitimo, tai gali atsitikti taip, kad jis ir nebus patvirtintas, tiksliau, bus panaikintas. Pavaizduokime tokią būseną:

Transakcija A	Laikas	Transakcija B
-	|	-
-	t_1	UPDATE R
-	|	-
FETCH R	t_2	-
-	|	-
-	t_3	ROLLBACK
-	↓	-

Laiko momentu t_2 transakcija A nuskaito duomenis, kurie prieš tai (momentu t_1) buvo pakeisti transakcijos B. Nuo momento t_2 transakcijos A tolimesnio darbo teisingumas priklauso nuo to, kaip pasibaigs transakcija B, patvirtindama pakeitimą ar anuliuodama jį. Jei transakcija B atsisako pakeitimo (t_3), tai nuo to momento transakcija A naudoja neteisingus duomenis.

Dar blogiau kai momentu t_2 transakcija A ne skaito duomenis, o keičia juos:

Transakcija A	Laikas	Transakcija B
-	|	-
-	t_1	UPDATE R
-	|	-
UPDATE R	t_2	-
-	|	-
-	t_3	ROLLBACK
-	↓	-

Tokiu atveju, nuo laiko momento t_2 transakcijos A teisingumas priklauso nuo neužfiksuoto pakeitimo, o laiko mometu t_3 ji praranda dar ir savo pakeitimą, nes tuomet transakcija B atkuria eilutės R reikšmes, kurios buvo transakcijos B pradžioje (iki momento t_1).

9.4. Duomenų atribojimas

Bendro duomenų naudojimo problemos sprendžiamos duomenų atribojimu (užrakinimu) (angl. *locking*). Pagrindinė duomenų atribojimo idėja yra labai paprasta - jei transakcijai reikia, kad DB objektas (pvz., lentelės eilutė ar kelios eilutės) nebūtų pakeistas tam tikrą laikotarpį, tai ji užrakina tą objektą. Užrakintas objektas yra atskiriamas nuo kitų transakcijų įtakos, t.y. jis tampa neprieinamas kitoms transakcijoms. Transakcija yra priversta laukti, jei ji nori pasiekti (nuskaityti ar modifikuoti) objektą, kurį užrakino kita transakcija. Todėl, netgi paprasčiausio SQL sakinio, kuriuo norima perskaityti vieną lentelės eilutę, vykdymas gali užtrukti ilgai, jei tą eilutę yra užrakinusi kita transakcija. Kad nesukelti didelių nesklandumų, duomenys užrakinėjami gana lanksčiai.

Komercinėse DBVS paprastai naudojami du duomenų atribojimo užraktai:

- X (angl. *eXclusive lock*) – visiškas užrakinimas;
- S (angl. *Shared lock*) – dalinis užrakinimas.

Kartais X ir S duomenų atribojimo užraktai vadinami atitinkamai **rašymo ir skaitymo užraktais**. Kai kuriose komercinėse DBVS yra daugiau duomenų atribojimo užraktų, tačiau pagrindas yra toks pat.

Duomenų atribojimą pasiaiškinsime lentelės eilutės atveju, nors jis taikomas ir kitiems DB objektams, pvz., užklausos rezultatui bei lentelei. Duomenų atribojimą galima nusakyti tokiomis taisyklėmis:

- Jei transakcija *A* visiškai užrakina (užraktu X) eilutę *R*, tai transakcija *B*, norinti perskaityti ar pakeisti tą eilutę, yra priversta laukti kol transakcija *A* atrakins tą eilutę.
- Jei transakcija *A* iš dalies užrakina (užraktu S) eilutę *R*, tai:
  - jei transakcija *B* nori visiškai užrakinti (užraktu X) tą pačią eilutę *R*, tai *B* priverčiama lauki, kol *A* neatrakins eilutę *R*;
  - jei transakcija *B* nori tik iš dalies užrakinti (užraktu S) eilutę *R*, tai šis noras išpildomas, t.y. ir transakcijai *B* leidžiama užrakinti eilutę *R* užraktu S.

Šias taisykles galima pavaizduoti užraktų suderinamumo matrica:

	X	S	-
X	Ne	Ne	Taip
S	Ne	Taip	Taip
-	Taip	Taip	Taip

Pirmajame iš kairės stulpelyje yra surašyti visi įmanomi transakcijos *A* duomenų atribojimo užraktai. Minusu pažymėtas atribojimo nebuvimas. Viršutinėje eilutėje yra išvardyti duomenų atribojimo užraktai, kuriais kita transakcija *B* nori užrakinti tuos pačius duomenis kaip ir transakcija *A*. Vidiniuose matricos langeliuose pažymėta, ar transakcijos *B* prašymai yra patenkinami (Taip), ar nepatenkinami (Ne). Nesunku suprasti, kodėl ši matrica yra simetriška.

Prašymai užrakinti lentelės eilutę paprastai yra netiesioginiai. Kai transakcija nori perskaityti eilutę (pvz., įvykdyti sakinį FETCH), ji automatiškai "prašo" leidimo užrakinti tą eilutę užraktu S, o kai nori keisti eilutę (įvykdyti sakinį UPDATE) - "prašo" užrakinti eilutę užraktu X. Eilutė yra užrakinama, o SQL sakinys įvykdomas, kai tik transakcija "sulaukia" iš DBVS leidimo.

Transakcija, užrakinusi eilutę užraktu X, paprastai išlaiko šią eilutę užrakintą iki transakcijos pabaigos. Užraktu S užrakinta eilutė gana dažnai atrakinama nepasibaigus transakcijai. Tačiau duomenų atribojimo laikas priklauso ne tik nuo užrakto, bet ir nuo išorinių veiksnių, kuriuos aptarsime vėliau. Bet kuriuo atveju, pačiai transakcijai nereikia rūpintis atrakinimu - tai atliekama automatiškai. Pasibaigus transakcijai visi jos užrakinti objektai atrakinami.

9.5. Bendro duomenų naudojimo problemų sprendimas

Įvedus duomenų atribojimą, prarasto duomenų pakeitimo ir priklausymo nuo neužfiksuoto pakeitimo problemos išnyksta. Dabar **pakeistų duomenų praradimo** padėtyje įvykiai klostysis taip:

Transakcija *A*	Laikas	Transakcija *B*
-	|	-
FETCH *R* (prašoma ir užrakinama užraktu S)	t_1	-
-	|	-
-	t_2	FETCH *R* (prašoma ir užrakinama užraktu S)
-	|	-
UPDATE *R* (prašoma užrakinti užraktu X)	t_3	-
Laukiama	|	-
Laukiama	t_4	UPDATE *R* (prašoma užrakinti užraktu X)
Laukiama	↓	Laukiama

Atribojant duomenis, vienoje transakcijoje pakeisti duomenys nebus anuliuojami kitoje. Tačiau, spręsdami šią problemą, mes susidūrėme su kita problema - **aklavietė**. Nuo laiko momento t_4, abi transakcijos lauks viena kitos. Kaip sprendžiama aklavietės problema, aptarsime vėliau.

Pažiūrėkime, kaip klostysis įvykiai situacijoje, kuri sukėlė vienos transakcijos **priklausymą nuo neužfiksuoto kitos transakcijos duomenų pakeitimo**. Pirmuoju atveju, kai transakcija *A* perskaito transakcijos *B* pakeistus duomenis, turime:

Transakcija *A*	Laikas	Transakcija *B*
-	|	-
-	t_1	UPDATE *R* (prašoma ir užrakinama užraktu X)
-	|	-
FETCH *R* (prašoma užrakinti užraktu S)	t_2	-
Laukiama	|	-
Laukiama	t_3	ROLLBACK (atrakinama)
Nuskaitoma (užrakinama užraktu S)	t_4	-
-	↓	-

Panašiai kaip ir ankstesnėje situacijoje, momentu t_2 transakcijai *A* neleidžiama perskaityti eilutės *R*. Transakcija *A* yra priversta laukti, kol transakcija *B* atrakins šią eilutę. Momentu t_3 transakcija *B* baigiasi ir transakcija *A* gali pratęsti darbą. Momentu t_4 transakcija *A* perskaito tas eilutės *R* reikšmes, kurios buvo prieš transakcijai *B* keičiant jas (momentu t_1). Abi transakcijos gali tęsti pradėtą darbą.

Antruoju atveju, kai transakcija *A* keičia duomenis, kurie jau buvo pakeisti kitos transakcijos *B*, turėsime labai panašią situaciją:

Transakcija *A*	Laikas	Transakcija *B*
-	|	-
-	t_1	UPDATE *R* (prašoma ir užrakinama užraktu X)
-	|	-
UPDATE *R* (prašoma užrakinti užraktu X)	t_2	-
Laukiama	|	-
Laukiama	t_3	ROLLBACK (atrakinama)
Atnaujinama (užrakinama užraktu X)	t_4	-
-	↓	-

Taigi, duomenų atribojimas, kuris yra gana paprastai tvarkomas, neleidžia susidaryti klaidoms kai tuo pačiu metu su tais pačiais duomenimis dirba kelios transakcijos.

9.6. Aklavietės ir jų likvidavimas

Duomenų atribojimas užtikrina, kad duomenys nebus apdoroti klaidingai dėl to, kad jais tuo pat metu naudojasi kelios transakcijos. Tačiau taip išsprendžiamos ne visos problemos. Duomenų bazės objekto (pvz., lentelės eilutės) užrakinimas gali tapti aklavietės priežastimi. **Aklavietė** - tai būsena, kai dvi ar daugiau transakcijų tuo pačiu metu laukia, kol kita iš jų atrakins duomenis, kad pratęsti darbą.

Ankstesniame skyrelyje mes jau susidūrėme su aklaviete. Būseną, kurioje susidaro aklavietė tarp dviejų transakcijų *A* ir *B*, galima išreikšti taip:

Transakcija *A*	Laikas	Transakcija *B*
-	|	-
R_1 užrakinama S arba X užraktu	t_1	-
-	|	-
-	t_2	R_2 užrakinama S arba X užraktu
-	|	-
Prašoma užrakinti R_2 X užraktu	t_3	-
Laukiama	|	-
Laukiama	t_4	Prašoma užrakinti R_1 X užraktu
Laukiama	↓	Laukiama

Šiame pavyzdyje R_1 ir R_2 žymi du konkrečius DB objektus, nebūtinai eilutes. Tai gali būti ir eilučių grupės ar visos lentelės. Čia į aklavietę patenka dvi transakcijos. Teoriškai į aklavietę tuo pačiu metu gali patekti ir daugiau transakcijų. Tačiau aklavietės, kurios apimtų daugiau negu dvi transakcijas, praktiškai nėra sutinkamos.

Svarbu, kad DBVS aptiktų aklavietes ir padėtų transakcijoms išeiti iš jų. DBVS realizuodama duomenų atribojimą tvarko **transakcijų**, laukiančių kitų transakcijų pabaigos, **žurnalą**. Tam, kad aptikti aklavietę, pakanka rasti ciklą **transakcijų būsenų grafe**, kuriame vaizduojamos transakcijų tarpusavio priklausomybės. Aptikusi aklavietę, DBVS vieną iš transakcijų išrenka **auka** ir, priklausomai nuo konkrečių aplinkybių:

- išrinktą transakciją nutraukia ir atkuria duomenis, buvusius transakcijos pradžioje;
- nutraukia SQL sakinį, kuriam įvykdyti transakcija laukė galimybės užrakinti DB objektą, anksčiau jau užrakintą kitos transakcijos.

Kiekvienu atveju, DBVS praneša vartotojui (programai) apie panaudotas priemones konkrečiu pranešimu apie klaidą. Priklausomai nuo klaidos kodo, vartotojas gali pakartotinai įvykdyti tik paskutinį SQL sakinį arba pakartoti visą transakciją nuo pradžios. Kai kurios komercinės DBVS imasi pačios pakartoti veiksmą.

9.7. Duomenų atribojimo tvarkymas

Jau minėjome, kad paprastai duomenų atribojimas tvarkomas automatiškai. Papildomai daugelyje komercinių DBVS vartotojas pats gali tvarkyti duomenų atribojimą. Dažnai yra dvi duomenų atribojimo tiesioginio tvarkymo priemonės:

- **lentelės užrakinimas**;
- **transakcijos atribojimo lygio** (angl. *isolation level*) priskyrimas transakcijai.

Jei transakcijoje yra daug kartų kreipiamasi į konkrečią lentelę, tai laiko ir kitų resursų sąnaudos, kurios susidaro dėl daugelio nedidelių lentelės sričių užrakinimo ir atrakinimo, gali būti santykinai didelės. Tokiais atvejais yra geriau iš karto užrakinti visą lentelę ir atrakinti ją tik apdorojus visus lentelės duomenis. Visos lentelės užrakinimas, lyginant jį su eilutės užrakinimu, turi keletą privalumų:

- nėra sąnaudų, reikalingų kiekvienai atskirai eilutei užrakinti ir atrakinti;
- nėra pavojaus, kad, pradėjus duomenų apdorojimą, veiksmai užsitęs dėl to, kad dalį eilučių užrakino kita transakcija;
- nėra pavojaus, kad, apdorojus dalį eilučių, transakcija pateks į aklavietę ir reikės vėl pradėti viską iš naujo.

Lentelės užrakinimas turi ir gana didelį trūkumą - visos kitos transakcijos turės laukti lentelės atrakinimo. Todėl, patartina lentelę užrakinti tik programose, nes interaktyvios transakcijos tęsiasi daug ilgiau. Lentelė užrakinama SQL sakiniu:

`LOCK TABLE` <lentelės vardas> `IN` [`SHARE` | `EXCLUSIVE`] `MODE`.

Vykdant šį sakinį su parametru `EXCLUSIVE`, yra prašoma DBVS visiškai užrakinti lentelę, t.y. užrakinti X užraktu. Taip užrakinus lentelę, jokia kita transakcija negalės lentelės duomenų nei perskaityti, nei pakeisti. Visiškas užrakinimas yra naudingas, kai reikia

atnaujinti daugumą ar net visas lentelės eilutes. Parametru SHARE nurodoma užrakinti lentelę S užraktu. Taip atribojus lentelę, kitos transakcijos galės be suvaržymų skaityti lentelės duomenis, bet negalės jų keisti. Šis režimas naudojamas norint sudaryti momentinį lentelės duomenų vaizdą, pavyzdžiui, suformuoti ataskaitą. Įvydžius sakinį

LOCK TABLE *Vykdymas* IN SHARE MODE,

niekas kitas negalės keisti lentelės *Vykdymas* duomenų tol, kol nesibaigs šį sakinį įvykdžiusi transakcija.

9.8. Transakcijų atribojimas

Pagal transakcijos apibrėžimą jokia kita, kartu veikianti transakcija negali daryti įtakos duomenims, su kuriais ši transakcija dirba. Jei transakcijoje yra nuskaitomi bei apdorojami duomenys, ir po to vėl jie nuskaitomi, tai transakcija gali būti tikra, kad perskaityti duomenys bus tapatūs anksčiau perskaitytiems, žinoma, jei tik pati transakcija jų nepakeitė. Pakartotinis nepakeistos eilutės perskaitymas transakcijoje yra pats aukščiausias transakcijos (programos) atribojimo nuo kitų transakcijų lygis.

Visiškas transakcijos atribojimas nuo kitų transakcijų reikalauja daug sąnaudų, nes kiekviena perskaityta eilutė turi būti užrakinta užraktu S. Eilutės išlieka užrakintomis iki transakcijos pabaigos. Dažnai DBVS galėtų duomenų atribojimą (užrakinimą-atrakinimą) tvarkyti daug geriau, jei tik ji žinotų, kas transakcijoje daroma su duomenimis. Todėl su kiekviena transakcija yra susiejamas jos atribojimo lygis. Transakcijos atribojimo lygis išreiškia kompromisą tarp visiško transakcijų atribojimo ir efektyvaus jų darbo.

Transakcijos atribojimo lygis - tai:

- laipsnis, kuriuo lentelių eilutės, perskaitytos ar pakeistos transakcijoje, yra prieinamos kitoms transakcijoms;
- laipsnis, kuriuo kitų transakcijų perskaitytos ar pakeistos eilutės yra prieinamos šioje transakcijoje.

Standarte SQL2 yra apibrėžti keturi transakcijos atribojimo lygiai.

- **Visiško atribojimo** (SERIALIZABLE) lygis. Tai pats griežčiausias transakcijos atribojimo lygis, užtikrinantis, kad:
  - bet kuri transakcijoje perskaityta eilutė nebus pakeista jokios kitos transakcijos kol duotoji transakcija nesibaigs;
  - jokia eilutė, pakeista kitoje transakcijoje nebus perskaityta duotojoje transakcijoje, kol pakeitusi eilutę transakcija nepasibaigs.

  Šis lygis visiškai atriboja transakciją nuo kitų transakcijų poveikio. Jis užtikrina, kad vienoje transakcijoje, pakartotinai perskaitant duomenis (vykdant SELECT sakinį), kiekvieną kartą bus perskaityti tie patys duomenys, jei tik duotoji transakcija pati nekeičia jų. Kitoms transakcijoms nebus leista ne tik keisti ir pašalinti šioje transakcijoje perskaitytas eilutes, bet ir įterpti naujas eilutes, kurios galėtų patekti į perskaitytų duomenų sritį. Šis lygis yra priimamas pagal nutylėjimą, kadangi jis atitinka pagrindinį principą – transakcijos neturi įtakoti viena kitai.
- **Pakartotinio duomenų skaitymo** (REPEATABLE READ) lygis. Šis lygis yra labai panašus į SERIALIZABLE lygį. Transakcijoje nėra prieinami jokie kitų transakcijų neužfiksuoti duomenų pakeitimai. Transakcijai vieną kartą perskaičius duomenis, visos kitos transakcijos negalės jų nei keisti, nei šalinti, kaip ir SERIALIZABLE atveju, bet joms bus leidžiama įterpti naujas eilutes, kurios gali patekti ir į duotosios transakcijos perskaitytų duomenų sritį. Todėl pakartotinai skaitant duomenis toje pačioje transakcijoje gali būti perskaitytos ir papildomos eilutės, įterptos ir užfiksuotos (COMMIT sakiniu) kitų transakcijų. Tokios naujos eilutės, atsirandančios pakartotinai skaitant duomenis vienoje transakcijoje ir įterptos kitos transakcijos, vadinamos “vaiduoklėmis” (angl. *phantom*). Jeigu transakcijoje nenumatyta keletą kartų vykdyti tą pačią užklausą, tai galima vartoti šį lygį nerizikuojant duomenų vientisumu.

- **Užfiksuotų duomenų skaitymo** (`READ COMMITTED`) lygis. Šio lygio transakcijose yra užtikrinama, kad jokia eilutė, pakeista kitoje transakcijoje, nebus prieinama duotojoje transakcijoje, kol pakeitusi eilutę transakcija nepasibaigs. Tačiau užfiksuoti kitų transakcijų rezultatai yra "matomi" šioje transakcijoje. Taip atribotoje transakcijoje, pakartotinai įvykdžius užklausą, gali būti aptinkamos eilutės, kurias pakeitė kitos transakcijos. Tai reiškia, kad transakcijoje perskaitytos eilutės neišlieka užrakintomis iki transakcijos pabaigos. Perskaityta eilutė atrakinama, kai tik užklausos rezultato žymuo pasislenka prie kitos eilutės. Transakcijoje pakeistos ir užraktu `X` užrakintos eilutės išlieka tokiomis iki transakcijos pabaigos. Jei programoje nereikia tas pačias eilutes peržiūrėti keletą kartų ir jei nėra skaičiuojami suminiai rodikliai, tai šio atribojimo lygio pakanka, kad užtikrinti duomenų neprieštaringumą. Šio lygio transakcijos vykdomos greičiau negu aukštesnių atribojimo lygių transakcijos.
- **Neužfiksuotų duomenų skaitymo** (`READ UNCOMMITTED`) lygis. Tai pats žemiausias transakcijos atribojimo lygis. Taip atribotoje transakcijoje perskaitytą lentelės eilutę, kitos transakcijos gali ne tik perskaityti, bet ir keisti. Bet kuri eilutė, pakeista kitoje transakcijoje, gali būti perskaityta šioje transakcijoje, net jei tas pakeitimas neužfiksuotas, t.y. net nepasibaigus eilutę keitusiai transakcijai. Tačiau ir šio lygio transakcijose yra užtikrinama, kad nebus prarandami pakeisti duomenys. Kitose transakcijose pakeistus ir neužfiksuotus duomenis duotojoje transakcijoje galima tik perskaityti, bet ne keisti, kaip ir kitos transakcijos negali keisti šioje transakcijoje pakeistų duomenų kol pakeitimai nebus užfiksuoti. Kadangi šio lygio transakcijoje galima perskaityti neužfiksuotus duomenis, tai jo taikymas yra labai ribotas. Jis yra tinkamas tik tuomet, kai vartotojui yra priimtini ir ne visiškai teisingi duomenys. Jei svarbu tik "pamatyti" duomenis, nesirūpinat jų teisingumu, tai šis atribojimo lygis gali labai pagreitinti rezultato gavimą.

Pateiksime transakcijos atribojimo lygių suvestinę:

Atribojimo lygis	Neužfiksuotų duomenų perskaitymas	Pasikeitimai pakartotiniame perskaityme	Eilutės "vaiduoklės"
`SERIALIZABLE`	Negalimas	Negalimi	Negalimos
`REPEATABLE READ`	Negalimas	Negalimi	Galimos
`READ COMMITTED`	Negalimas	Galimi	Galimos
`READ UNCOMMITTED`	Galimas	Galimi	Galimos

Atribojimo lygį galima nustatyti pačioje transakcijoje SQL sakiniu:

`SET TRANSACTION ISOLATION LEVEL` <atribojimo lygis> [<apdorojimo rūšis>] .

Sakinyje nurodoma vienas iš keturių atribojimo lygių bei duomenų apdorojimo rūšis: duomenų skaitymas (`READ ONLY`) ar ir duomenų keitimas (`READ WRITE`). DBVS naudoja šiuos transakcijos požymius savo veiksmams optimizuoti. Pagal nutylėjimą, yra priimamas `SERIALIZABLE` lygis. Jei nurodytas `READ UNCOMMITTED` lygis, tai, pagal nutylėjimą, priimama `READ ONLY` veiksmų rūšis ir keisti duomenis transakcijoje neleidžiama. Visais kitais atvejais, pagal nutylėjimą, priimamas `READ WRITE` duomenų apdorojimas. Jei šis SQL sakinys yra naudojamas, tai jis turi būti pirmuoju transakcijoje.

Transakcijos atribojimo lygis pradėtas vartoti dar prieš standartizuojant šią sąvoką. Todėl lygių pavadinimuose atsirado painiava. Sistemoje DB2, pavyzdžiui, `REPEATABLE READ` vadinamas lygis, kuris standarte vadinamas `SERIALIZABLE`. Bet to, šioje sistemoje atribojimo lygis yra ne transakcijos požymis, bet transakcijų grupės, tiksliau, jungtinio vykdomojo plano požymis, kuris priskiriamas visoms plano transakcijoms. Atribojimo lygis vykdomajam planui yra priskiriamas jį sukūrus, t.y. SQL preprocesoriui apdorojus vartotojo sudarytą programinį modulį arba procedūra `BIND` pagal ryšio failą. Vėliau atribojimo lygį galima keisti.

Esant keliems transakcijos atribojimo lygiams neišvengiamai kyla klausimas: kaip jį pasirinkti? Sunku pateikti tikslias taisykles, leidžiančias formaliai išspręsti šią problemą.

Atribojimo lygio parinkimas yra gana atsakingas uždavinys. Parinkus per griežtą atribojimo lygį, dėl duomenų užrakinimo gali nepagrįstai sumažėti sistemos efektyvumas bei padidėti tikimybė patekti į aklavietę. Parinkus per žemą transakcijos atribjimo lygį, gali atsirasti šalutiniai poveikai (pvz., perskaitomi neužfiksuoti duomenys), kurie daro įtaką neteisingam programos rezultatui ar net pažeidžia duomenų neprieštaringumą.

Pateiksime paprastas euristikas, galinčias padėti pasirinkti reikiamą izoliavimo lygį:

Duomenų apdorojimo rūšis	Aukštas duomenų stabilumas reikalingas	Aukštas duomenų stabilumas nebūtinas
Skaitymo ir rašymo transakcijos	`REPEATABLE READ`	`READ COMMITTED`
Tik skaitymo transakcijos	`SERIALIZABLE`	`READ UNCOMMITTED`