3. Andmebaaside erinevad kontseptid ja arhitektuurid

3.1. Andmebaasid

andmebaas

Kõige elementaarsemas tähenduses mõistetakse andmebaasi all andmekogumit, mis lisaks andmetele sisaldab eneses ka nende andmete struktuuri (koosseisu) kirjelduse - andmetega koos hoitakse ka nende andmete kirjeldust. Võib öelda ka veel nii: andmebaas on kogum andmeid koos neid andmeid (andmete struktuuri) kirjeldavate metaandmetega. Kõige elementaarsemal tasemel sisaldab andmete kirjeldus tabelite kirjeldusi (milledes hoitakse andmeid) ja tabelite vaheliste seoste kirjeldusi..

Tänapäeva andmebaasisüsteemides on andmebaasides lisaks andmete kirjeldustele ka andmete erinevate andmeid käsitlevate protseduurid, nende protseduuride käivitamise reeglid (trigerid (*triggers*) ja kalenderplaanid (*schedulers*))

andmebaasid eksisteerisid palju aega enne arvutite tulekut

Andmebaaside all ei tohi mõista ainult elektroonilisi andmebaase, mis on realiseeritud arvutisüsteemides. Andmebaasid eksisteerisid palju aega enne seda, kui neid sai hakata realiseerima arvutisüsteemides. Suvalised kartoteegid on andmebaasid – olenemata sellest, milline on andmekandja selle kartoteegis (kartoteegi-kaart, kivitahvel, perfolint, perfokaart vms.).

Nii andmebaasis olevate andmete kirjeldus kui ka andmed peavad alati olema tõlgendatavad ühte moodi – nad peavad olema salvestatud kindlaks määratud ülesehitusega füüsilise struktuurina.

andmebaasi juhtimissüsteem - ABJS

Andmebaasi juhtimissüsteem (ABJS – *DBMS*) on programm, mis “oskab” vastavalt andmebaasis olevale kirjeldusele tõlgendada seal olevaid andmeid.

Andmebaasijuhtimissüsteem - andmebaas ja andmete käsitlemise vahendid

Andmebaasijuhtimissüsteem koosnebki kahest osast – andmetest ja nende kirjeldusest (andmebaasist) ning andmete käsitlemise vahendist – andmebaasi juhtimise süsteemist. Konkreetne andmebaasi juhtimissüsteem “tunneb” temaga seotud andmete struktuuri. Igal andmebaasi juhtimissüsteemil on oma füüsiline andmete struktuur, mida oskab käsitleda ainult see ABJS.

On võimalik, et ühte ja sama andmete struktuuri “tunneb” mitu erinevat andmebaasimootorit (Dbase, FoxBase, AboBase), See on aga erand, mille

kohta rohkem näiteid tuua ei olegi vist võimalik.

ABJS poolt tagatavad tegevused:

ABJS funktsioonid

- andmebaaside loomine ja hävitamine (füüsiliste struktuuride tekitamine kettale ja nende hävitamine)
- andmete struktuuride loomine ja muutmine
- andmete lisamine, kustutamine ja uuendamine (muutmine)
- andmete pärimine (andmete otsimine etteantud tingimuse järgi)
- andmete säilimise haldamine (kindluskoopiate tegemine, andmete taastamine)
- kasutajate ja kasutajagruppide õiguste haldamine (andmete lisamise, muutmise ja kasutamise õiguste piiramine)
- protseduuride (andmebaasi salvestatud programmide) loomine, muutmine, hävitamine, käivitustingimuste kirjeldamine, protseduuride automaatkäivitamine, “käsitsi” käivitamine

ABJS liidesed

Selleks, et töödelda andmebaasis olevaid andmeid peab andmeid töötlev programm saatma korraldusi ABJS-le. Selleks peab see programm "tundma" selle konkreetse ABJS suhtlus-liidest. See on kõikidel ABJS-del erinev. Siiski on töötatud välja ka unifitseeritud liideseid, mida tunnevad nii ABJS-d kui ka erinevad programmeerimiskeskkonnad. Microsofti standard on ODBC *(Opeb Database Connectivity).* Java standard on JDBC *(Java Database Connectivity)*. ISO standardiks on OLE DB *(Object Linkind and Embedding for Databases)*.

native link
("ürgne / igiomane" liides)

Andmebaasi enda spetsiifilist liidest nimetatakse inglise keeles *native link* ("ürgne / igiomane" / ainuomane liides)

3.2. Andmebaasi üldine struktuur

Andmebaas loogilisel tasemel - tabelid ja seosed nende vahel

Olenemata sellest, milline on andmebaas, põhineb alati kogu logiline ehitus (kõige alumisel tasemel) andmetabelitel ja nende vahel kirjeldatud seostel - andmeid hoitakse risttabelites ja nende vahele kirjeldatakse seosed.

tabelid -> veerud ja read

rida = kirje

veerg = kirje omadus

Tabelid on risttabelid, mis koosnevad veergudest ja ridadest. Igas veerus on semantiliselt ja füüsiliselt struktuurilt ühesugused andmed (eesnimi, perekonna nimi, sünniaeg, kogus, hind jms). See tähendab, et igas tabeli veerus on sama tähendusega andmed ja seega on ka nende andmete

formaat ühes veerus samasugune. Elementaarse andmekomplekti (kirje) moodustavad ühes tabeli reas olevad andmed (ühe inimese andmed, ühe auto andmed, ühe maja andmed, ühe ürituse andmed jne.)

Lihtsaim andmebaas on üks üksik tabel, millel on üks veerg.

lihtsaim andmebaas

Andmebaasis võib olla (ja tavaliselt ongi) ka mitu tabelit, millede vahele ei ole kirjeldatud ühtegi seost. See tähendab seda, et tabelite vaheliste seoste olemasolu andmebaasis ei ole oluline. Samas peab aga andmebaasis olema vähemalt üks tabel.

andmebaasis ei pea olema seoseid vaid neid võib seal olla

Kui kahe tabeli vahel on seos, peab mõlemas tabelis olema vähemalt kaks veergu – muidu ei ole asjal “ei sisu ega vormi”. Üks nendest väljadest, kummaski tabelis, on sisuline andmeväli ja teine on väli, mille väärtuse kaudu kaks tabelit seotakse. Kaks kirjet erinevates tabelites on omavahel seotud siis, kui nende väljade väärtused, mille kaudu seoseid kirjeldatakse on mõlemas kirjes sama väärtusega (väärtused on võrdsed).

tabeleid seostatakse veergudes olevate väärtuste abil

Näited.

NÄIDE

Iskud:

ISIKU_NIMI	SYNNIAEG	SUGU	AADRESS
Rabaja Ruts	15.10.1960	1	Pähkli 2, Assmalla, EST
Lahke Laima	21.12.1972	2	Mahtra 21-123, Tallinn, Harjumaa, EST
Tiino Bura	13.07.1867	3	Spagetti 232-12, Milano, ITA
Julm Juhan	22.03.1657	1	Tsarskoje Selo 1, st. Petersburg, RUS

Sugude loend:

SUGU	NIMETUS
1	mees
2	naine
3	määramata

Siin on näha, kuidas mõlemas tabelis oleva veeru SUGU kaudu on võimalik neid kahte tabelit seostada - kui isikute tabelis on näha ainult soo tähis, siis selle tähise järgi on võimalik sugude loendist üles otsida ka soo verbaalne nimetus - need kaks tabelit on omavahel seotud väljade SUGU kaudu. Seose väljade nimed ei pea olema samad. Küll aga peab olema sama seose väljade semantika (tähendus) ja andmetüüp.

seos moodustub kahe tabeli sama tähendusega veeru kaudu, mille väärtused on samad

3.3. Andmebaasi protseduurid

AB protseduuri keeled

Nagu juba korra mainitud ei hoita tänapäeva andmebaasides mitte ainult andmeid (koos kirjeldusega). Vaid sinna saab salvestada ka protseduure (programme) andmete käsitlemiseks. Igal andmebaasil on oma keel nende protseduuride kirjutamiseks. Nii Näiteks on Oracle protseduuri keeleks PL/SQL ja SqlBase protseduuri keeleks SAL *(SQL Application Language)*. Samas on siiski kaks andmebaasisüsteemi, mille protseduuri keel on sama nimega - TransactSQL Nendeks andmebaasisüsteemideks on SyBase ja MS SQLServer. See on tingitud nende ühistest juurtest. SyBase versiooni 3 ajal ostis Microsoft selle lähtetekstid koos arendusõigusega teise brändi all. Sellest tingituna on ka protseduuri keeled sarnased - erinevused on ainult mõningates detailides.

Oracle ja PL/SQL
SQLBase ja SAL
SQLServer ja TransactSQL
SyBase ja TransactSQL

Protseduuri keeles kirjutatakse andmebaasi tavaliselt andmekontrolli algoritmid, automaattransaktsioonide genereerimise reeglid, andmete muutuste jälgimise protseduurid jms.

AB protseduuride aktiveerimine

Protseduure saab käivitada mitut moodi. Protseduuri aktiveerib:

- tegevus andmebaasi tabeliga (andmete lisamine, kustutamine, uuendamine tabelis - protseduuri käivitamiseks võib lihtsalt kirjutada konkreetse toimingu toimumist jälgiva "valvuri" (trigeri) ja vastava toimingu toimumisel (kui triger tunneb ära, et toimus toiming, mille toimumist ta "valvab) käivitatakse trigeris määratud protseduur. Näiteks andmete lisamisel konkreetsesse tabelisse käivitatakse protseduur, mis kontrollib, kas kõik vajalikud andmed olid lisatud kirjes olemas, kas kõik väärtused jäävad lubatud piiridesse ning

- triger

- “manuaalne” korraldus so. keegi lihtsalt annab andmebaasile korralduse käivitada mingi protseduur. Protseduuri väljakutse võib olla kirjutatud mingisse programmi, kuid protseduuri võib "käsurealt" käivitada ka kasutaja (näiteks administraator).

- manuaalne korraldus

- teine protseduur st. üks protseduur võib olla teise protseduuri alamprotseduur nii nagu mistahes programmeerimiskeeles on võimalik kogu protseduur jagada erinevateks alamprogrammideks.

- teine protseduur

- kalenderplaan. Mitmetes andmebaasisüsteemides (mitte kõigis= on võimalik öelda andmebaasimootorile ette, millisel kuupäeva ja millisel

- kalenderplaan

kellaajal või siis millise regulaarsusega on vaja mingisuguseid protseduure käivitada

- andmebaasisüsteem. Korraldus protseduuri käivitamiseks võib tulla ka andmebaasisüsteemi mootorilt eneselt. Selliselt käivitatakse protseduurid, mis on kirjeldatud käivituma mingite andmebaasi mootori situatsioonide, toimingute või teadete peale.

- Andmebaasisüsteem

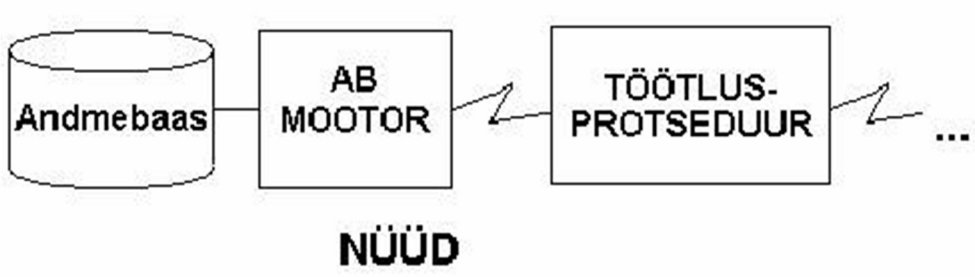
Andmebaasi protseduur ei ole kuidagi jäigalt seotud ainult ühe tabeliga. Protseduur on tegelikult tavaline programm, mille sisu sõltub kasutaja vajadustest

AB protseduur on tavaline programm, mis on salvestatud andmebaasi

3.4. Andmebaasi mootor. Ajalooline ja klient / teenindaja arhitektuur

Kaasaaegsetel andmebaasisüsteemidel on “mootor” (andmebaasi server) – andmebaasi juures asuv programm, millele korralduste andmisega opereeritakse andmebaasiga. Selleks, et andmebaasiga midagi teha tuleb andmebaasi mootorile saata korraldus ja andmed. Korralduse tulemusena saadab andmebaasi mootor vastuse sellises formaadis, mida küsiti. Vastuseks võib olla ka veateade, kui korraldus esitati vales vormingus (süntaksi viga) või sisaldas korraldus selliseid tabelite ja veergude nimesid, mida andmebaasis pole (sisuline viga). Tulemuseks võib olla ka tühi vastus. Seda juhul, kui andmebaasist küsiti midagi sellist, mida seal pole.

andmebaasi mootor



klient / teenindaja arhitektuur

Sellist arhitektuuri nimetatakse klient/teenindaja (*Client/Server)* arhitektuuriks. Selle kohapeal, kus joonisel on "TÖÖTLUSPROTSEDUUR" võib seista nii pakett-töötluse programm, aplikatsioonserver või siis mõni paks klient. Kui nüüd "töötlusprotseduur" soovib andmebaasiga vahetada informatsiooni, siis valmistatakse seal ette korraldus (tavaliselt SQL-keelne korraldus) ja need andmed, mis tuleb, koos korraldusega serverisse saata. Korraldus saadetakse täitmiseks koos andmetega andmebaasi mootorisse. Viimane töötleb korralduse ja salvestab andmed andmebaasi või siis teeb nende andmete alusel päringu andmebaasi.

Näited:

NÄIDE

1) Andmete lisamiseks andmebaasi saadetakse andmebaasi mootorile INSERT-korraldus koos nende andmetega, mida andmebaasi lisatakse. Andmebaasi mootor kontrollib lause ja veakohal tagastab veakoodi. Kontrolli eduka läbimise järel üritab andebaasi mootor lisada, koos INSERT-lausega saadetud andmed sellesse tabelisse, mis oli näidatud INSERT-lauses. Kui see ei õnnestu (näiteks kettal polnud ruumi), siis tagastatakse korralduse esitajale veakood. Kui kõik õnnestus, siis tagastatakse veakood 0.

2) Andmete pärimiseks andmebaasist saadetakse andmebaasi mootorile SELECT-korraldus, mis sisaldab loendi nendest andmeväljadest mida soovitakse saada ja tingimuse, mille alusel andmed andmebaasist välja valitakse. Andmebaasi mootor kontrollib lause ja veakohal tagastab veakoodi. Kontrolli eduka läbimise järel teostab andmebaasi mootor päringu andmebaasi ja moodustab küsitud andmetest loendi. Kui see ei õnnestu (näiteks hoidis keegi teine andmeid lukus), siis tagastatakse korralduse esitajale veakood. Kui kõik õnnestus, siis tagastatakse veakood 0. Nüüd võib mootorile saata teateid järgmise kirje saamiseks. Esimese küsimise peale saadab server esimese rea (paneb andmed puhvrisse ja saadab "küsija" puhvrisse), teise järel järgmise rea jne. Kui ridu enam pole vastab server veateatega EOF (end of file).

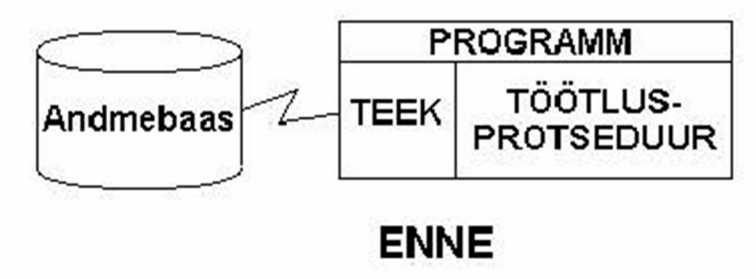
süsteemsed korraldused

Sama skeemi kohaselt saadetakse andmebaasi ka administreerimiskäsud: kasutajate lisamine, kasutajate õiguste muutmine, tabelite lisamine, seoste kirjeldamine, andmebaasi mootori töörežiimide muutmine jne. Need käsud tagastavad üldjuhul ainult veakoodi. Ka see, et viga ei tekkinud on veateade. Seda lihtsalt enamike andmebaasisüsteemide korral koodiga 0 (null).

ajalooline arhitektuur

Andmebaasisüsteemid ei ole muidugi alati olnud sellise arhitektuuriga. Esimestes andmebaasisüsteemides teostati andmebaasi käsitlust moodulite abil, mis lingiti selle programmi külge, millega käsitleti andmebaasi. Moodulid lingiti programmide külge teekide kaupa. Vajaliku operatsiooni teostamiseks (lisamine, pärimine, tabeli loomine jms.) kutsuti välja vajalik süsteemne alamprogramm ja anti sellele ette

parameetrite väärtused, mis määrasid selle, mida moodul andmebaasiga tegi.



ENNE

andmebaasi ja töötlusprogrammi paiknemine

Kuigi joonisel on näidatud programmi ja andmebaasi vahel kaug-lingi sümbol, paiknesid programm ja andmebaas suure tõenäosusega ühes arvutis ja korraga oli ühel andmebaasil üks kasutaja. Andmete samaaegne ühiskasutus oli suureks probleemiks, kuna andmete lukud kirjutati andmebaasi ja kui ühendus luku pannud kasutaja ja andmevaasi vahel kadus jäigi lukk "igaveseks" peale, kuni süsteemi administraator selle kõrvaldas. Kasutaja ise ei saanud enam teha ka siis, kui ta uuesti andmebaasi sisse logis. Seda selle pärast, et nüüd oli tal teine sessiooni võti ja eelmise sessiooni võtmega pandud lukud olid tema jaoks "võõrad".

andmete lukustamise probleemid

3.5. Klassifikatsioon struktuuri järgi

Struktuuri järgi klassifitseerimisel eristatakse andmebaase seoste lubatud struktuuri järgi. Sellist klassifikatsiooni kasutati peamiselt andmebaaside arengu algetappidel. Tänapäeval on peaaegu kõik enam levinud andmebaasid võrk-struktuuriga.

3.5.1. Võrkstruktuuriga andmebaasid

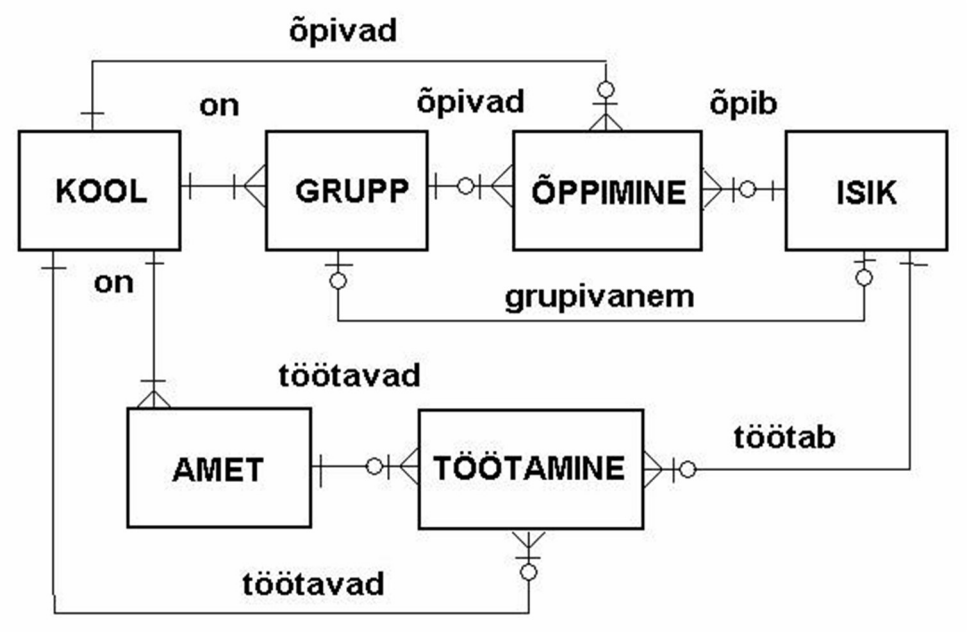
seoste arv kahe tabeli vahel pole loogiliselt piiratud

Võrkstruktuuriga andmemudelites ei ole seoste moodustamine tabelite vahel kuidagi reglementeeritud. Kahe tabeli vahele võib kirjeldada nii palju seoseid kui see ainult püstitatud ülesande poolt vajalikuks osutub. Iga kahe tabeli vahele saab/võib teha “suvalise” hulga seoseid, mille ainsaks piiriks on andmebaasi mootori poolt seatud füüsilised piirid (palju on seoste tekitamiseks aadressvälju võimalik teha, kui palju on lubatud ühte tabelisse veergusid teha).

füüsilised piirid võivad olla tingitud konkreetsest andmebaasi mootorist

Näide

NÄIDE



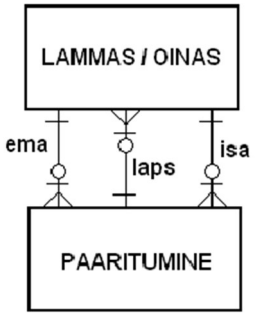
Joonisel on toodud lihtsustatud andmemudel, mis kirjeldab inimeste seoseid koolidega. Meil on koolide loend (olem KOOL). Selles loendis on talletatud kõik meid huvitavad koolid. Isikute loend olenemata sellest, kas nad on tudengid, õppejõud või töötajad on olemis ISIK. See kas ISIK on tudeng või õppejõud/töötaja sõltub sellest, kas ta on kooliga seotud läbi õpperühma (olem GRUPP) ja õppimise (olem ÕPPIMINE so. õppeleping) või läbi ameti (olem AMET) ja töötamise (olem TÖÖTAMINE so. tööleping).

kahe tabeli vahel "kõik" seosed samas suunas

Üks **soovituslik** piirang seoste moodustamisel siiski on - kahe tabeli vahele on soovitav teha üks-mitmeseid seoseid ainult ühes suunas st. kõik seosed kahe tabeli vahel peaksid olema sama suunalised. Siiski ei ole see reegel 100%-line. On olemas loogilisi konstruktsioone, kus üks suhetest võib olla teist pidi.

Näide

NÄIDE



Paaritumiseks on vaja lammast (ema) ja oinast (isa). Samas sünnib samast paaritumisest enamikel juhtudel mitu talle (laps), kes tulevikus saavad ka emadeks ja isadeks (kui neid enne ära ei sööda :) ja kes loomulikult on ka lambad ja oinad ning seega tuleb registreerida kohe pärast sündi olemis LAMMAS/OINAS ja siduda selle paaritamisega, mille tulemusena nad sündisid.

3.5.2. Puu-kujulised andmebaasid

Puu-kujulises andmemudelis saab iga tabel tabel olla ainult ühes üks-mitmeses seoses, kus ta on suhte alluv (suhte “mitu” poolel). Kuna puu hargneb seose üks-mitmeses suunas, siis see ongi puu-kujulise mudeli kõige rangemaks piiranguks.

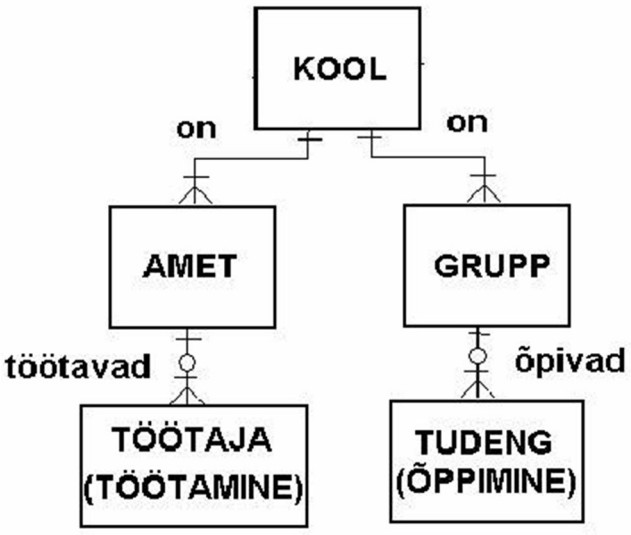
iga tabel saab olla ainult ühes alluvussuhtes

Tabelil võib olla teiste tabelitega kuitahes palju seoseid, kus ta on suhte valdaja (suhte “üks” poolel). Siin ei ole suhete arv piiratud mitte millegi muuga kui konkreetse andmebaasisüsteemi füüsiliste piiridega.

iga tabel saa olla kui tahes paljudes suhetes ülemusena

Näide

NÄIDE



Kasutame näites sama probleemi püstitust nagu jaotises 3.5.1. Nagu näha on siin võrreldes jaotises 3.5.1 toodud skeemiga näha olulisi muudatusi. Esiteks on isikud nüüd jaotatud kahe olemi vahel - töötajad ja tudengid on lahku viidud. Samas ei sisalda olemid TÖÖTAJA ja TUDENG enam "puhtaid" isiku andmeid vaid sinna on lisatud ka töölepingu ja õppelepingu andmed. Skeem on üldse palju raskepärasem ja halvemini käsitletav.

hierarhiline skeem on oluliselt "raskepärasem" ja halvemini käsitletav

3.6. Klassifikatsioon seoste moodustamise viisi järgi

Sõltuvalt sellest, milliseid füüsilisi (andme-)struktuure kasutatakse tabelite omavaheliseks seostamiseks, jagatakse andmebaasid kaheks grupiks:

- navigatsioonilisteks andmebaasideks,
- relatsioonilisteks andmebaasideks.

andmebaasisüsteemide (!) klassifikatsioon seoste moodustamise viisi järgi

Kui struktuuri järgi (vt. p. 3.5) saab jagada nii andmemudeleid kui ka andmebaase kas võrk-struktuuriga või hierarhilisteks andmebaasideks / andmemudeliteks, siis seoste moodustamise viisi järgi saab jaotada ainul andmebaasisüsteeme. Seda selle pärast, et tegemist on füüsilise klassifikatsiooniga - kuidas on seoste tegemine tehniliselt andmebaasisüsteemis lahendatud.

füüsiline klassifikatsioon

Ettehaaravalt võib juba öelda, et tänapäeval kasutuses olevad andmebaasisüsteemid on enamuses relatsioonilised. Samas esimesed andmebaasisüsteemid olid navigatsioonilised. Vaatleme neid mudeleid ajaloolises järjestuses.

valdav osa täna kasutatavaid andmebaasisüsteeme on relatsioonilised

3.6.1 Navigatsioonilised andmebaasid

Navigatsioonilistes andmebaasides koosneb iga tabeli iga kirje kahest osast:: kirje päisest ja andmeblokist. Kirje päises talletatakse informatsiooni kirje pikkuse, staatuse ja selle kirjega seotud teiste kirjete kohta. Andmeblokis talletatakse need andmed, mille talletamiseks andmebaas on loodud. Kirje päis on tehniline instrument, mida kasutajale kunagi ei näidata, kuid mille tähtsus antud mudeli puhul on väga oluline.

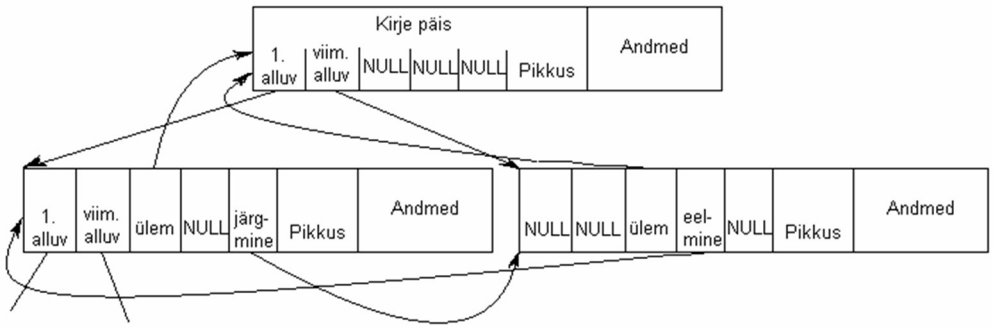
kirje koosneb kahest osast:

- **päis**
- **andmeblokk**

Kirje päises on iga seose jaoks üks aadressväli, millesse seose moodustamisel kirjutatakse ülemus-kirje aadress või siis seotud kirjete ahela esimese või viimase kirje aadress seotud tabelis.

kirje päises on aadressiväljad, mis viitavad seotud kirjetele

Siin peab suhet valdavas kirjes (suhte poolel ”üks”) olema iga seose jaoks aadresside paar “1.alluv – viimane alluv”. Suhte alluva kirje päises (suhte poolel “mitu”) peab olema iga suhte jaoks üks aadress “ülem” ja aadresside paar “eelmine – järgmine”:



kirje päise struktuur

Joonisel on toodud üks ülemus-kirje (ühest tabelist) koos kahe alluvkirjega (teisest tabelist). Tabelite vahel on üks üks-mitmene suhe. Siin on kirje päises järgmised väljad:

- esimese alluva aadress
- viimase alluva aadress
- viit ülemusele
- viit eelmisele kirjele
- viit järgmisele kirjele
- andmebloki pikkus

kirje päise struktuuri laiendused

Siia võib lisada ka tunnuse, kas kirje on loogiliselt kustutatud või mitte. See võimaldaks kirjeid baasist "kustutada" ilma neid tegelikult kustutamata. Kirjete töötluskeskkond (ABJS) peab lihtsalt oskama vahele jätta neid kirjeid, millel on märgitud tunnus "kustutatud".

erinevad andmebaasisüsteemid - erinevad kirje päise struktuurid

Erinevad andmebaasisüsteemid lisavadki kirje päisesse erinevaid juhtandmete talletamiseks ettenähtud välju Aga kõik see väljub vaadeldavast kontekstist ja seega siin kohal me seda detailsemalt ei käsitle.

kirjete ahelad

Nüüd aga kirje päises olevatest aadressiväljadest lähemalt. On aru saadav, et üks-mitmest seost ei ole võimalik teha selliselt, et igale alluvale kirjele viitamise jaoks tehtaks ülemus-kirjesse (üks-mitmese seose "üks"-poolel olevasse kirjesse) eraldi aadressiväli. Kuna me ei tea palju neid alluvaid tuleb (ja neid võib palju tulla) siis muutuks kirje päis ebaloomulikult pikaks, kohmamaks ja tehniliselt realiseerimatuks. Samuti ei piisa sellest, et teha ainult alluvatele kirjete päisesse väli, mis viitab ülemus-kirjele. See tagaks küll ülemuskirjete leidmise alluvate kirjete poolelt aga ülemus kirje

viit esimesele alluvale

viit viimasele alluvale

viit järgmisele kirjele

viit eelmisele kirjele

poolt alluvate kirjete leidmine on võimatu. Selle olukorra lahendamiseks moodustatakse alluvatest kirjetest ülemus-kirjete alla ahelad. Ülemus-kirjesse tehakse kaks aadressvälja, millest esimene viitab alluvatest kirjetest moodustatud ahela esimesele kirjele ja teine väli ahela viimasele kirjele. Alluvatel kirjetel on aga edasi-tagasi viidad (viit eelmisele kirjele, viit järgmisele kirjele), mis võimaldavad läbida kirjete ahelat nii edaspidi kui ka tagurpidi.

3.6.2. Relatsioonilised andmebaasid

tabelite sidumiseks peab mõlemal seose tabelil olema sama semantikaga veerg

Relatsioonilistes andmebaasides teostatakse seosed erinevate tabelite vahel andmeväljade väärtuste kaudu. Selleks, et kahe tabeli kirjeid saaks siduda, peab mõlemas tabelis olema sama semantikaga väli. Kui nüüd kahes tabelis asuvatel kirjetel on sama väärtus ongi kaks kirjet omavahel seotud.

üks-mitmese seose "üks" poolses otsas on seose välja väärtus unikaalne, teisel pool mitte

Üks-mitmese seose "üks"-poolses otsas olevas tabelis peab seose veerg olema unikaalse väärtusega. st. tabeli selles veerus ei tohi ükski väärtus korduda. Seose mitmeses otsas on mitme sama väärtuse ilmnemine seose veerus aga täiesti lubatav aj võiks öelda, et isegi normaalne olukord, sest kuidas muidu saame me ühe kirjega siduda mitu kirjet.

seotud on kirjed, mille seose väljade väärtused on võrdsed

Omavahel seotud on need kirjed, milledel seost moodustavate veergude paari väärtused on mõlemas seotud kirjes samad (võrdsed)..

Näide

NÄIDE

Olgu meil vaja pidada registrit autode ja nende omanike kohta. Seejuures on ühel autol küll korraga üks omanik, aga aja jooksul on autol kindlasti mitu omanikku. Meie soov on näha omanike ajalugu. Teeme selleks kaks tabelit AUTO ja OMANIK. Esimeses neist hoiame autode andmeid ja teises omanike andmeid. Kuna tabel auto on nende kahe tabeli vahelises seoses ülemus (üks-mitmese seose "üks"-poolses otsas), siis peab tal olema unikaalseid väärtuseid sisaldav veerg. Meie näites on selleks auto kere number, mis ka tegelikkuses on üle kõikide autode unikaalne.

Tabelis omanik kirjutame iga omaniku andmete juurde ka selle auto kere numbri, mille omanik konkreetne isik on. All olevast tabelist on näha, et autodel, kere numbriga GL54S91 ja SS23232, ei ole meie baasis kunagi omanikku kirjeldatud. Autol, kerenumbriga XE22378, on meie baasis

registreeritud üks ja ainus omanik, kes on selle auto omanik ka praegu (lõpetamise kuupäev on määramata). Autol, kere numbriga US45342, on enne praegust omanikku (Mait Malakas) olnud veel kaks omanikku.

Teisest küljest, vaadates tabelist OMANIK kere numbrit, saame me selle järgi tabelist AUTO leida auto margi ja väljalaske aasta. Nii näiteks on Sass Sammal oleva auto kere number XE22378 ja selle järgi näeme tabelist AUTO, et tegemist on 2001 aastal toodetud BMW-ga.

AUTO

KERE_NR.	MARK	VL_AASTA
XE22378	BMW	2001
US45342	ZIZ	1972
GL54S91	VAZ07	1985
SS23232	VOLVO	2000

OMANIK

KERE NR.	OMANIK	ALGUS	LÕPP
XE22378	Sass Sammal	10.07.2001	
US45342	Juss Jublakas	12.09.1972	01.01.1982
US45342	Jass Jalakas	02.01.1982	15.01.1999
US45342	Mait Malakas	16.01.1999	

Sellisel reaalsete andmeelementide väärtuste abil kirjete seostamisel on üks viga. Kui selle andmeelemendi väärtus peaks muutuma, siis lisaks sellele, et see väärtus tuleb ära muuta seose ülemus-tabelis, tuleb see ära muuta ka kõigis seostatud tabelite kirjetes, mis selle võtmega kirjega oli seotud. Mida rohkemate tabelitega on see tabel seotud, seda suurem on vajalike uuenduste arv .iga võtme muutuse korral. See võib aga hakata häirima süsteemi tööd.

unikaalse võtmena on otstarbekas kasutada domeen-võtit so. genereeritud väärtusega ID-d

Selle vältimiseks kasutatakse genereeritud võtmeid nn. surrogaatvõtmeid. Neid võtmeid nimetatakse ka ID-deks (identiteet / identity). ID on tavaliselt järjenumber. Iga uue kirje lisamisel vaadatakse, mis on seni suurim ID, liidetakse sellele 1 ja omistatakse saadud väärtus loodava kirje ID veergu. esimese kirje lisamisel on see tavaliselt 1 (nulli välditakse hea tavana), teise kirje korral 2 jne. Selline ID ei oma mingit tähendust ja seega pole teda

surrogaatvõti
ID (identiteet / *identity*)

kunagi vaja muuta. Hea tava ütleb, et seda ei tohigi teha. Tähtsam heast tavast on aga teadmine, et ID väärtuse muutmine ei oma mõtet ja selle muutmine võib tekitada probleeme, mis muidu ei tekkiks.

Näide

NÄIDE

ID on surrogaatvõti

teeme tabelisse AUTO juurde uue veeru (ID) ja genereerime sinna igale kirjele unikaalse ID. Kui nüüd tabelisse OMANIK teha samasugune veerg, siis saame seda kasutada autode omanike andmete sidumiseks autode andmetega.

AUTO

ID	KERE_NR.	MARK	VL_AASTA
1	XE22378	BMW	2001
2	US45342	ZIZ	1972
3	GL54S91	VAZ07	1985
4	SS23232	VOLVO	2000

OMANIK

ID	KERE NR.	OMANIK	ALGUS	LÕPP
1	XE22378	Sass Sammal	10.07.2001	
2	US45342	Juss Jublakas	12.09.1972	01.01.1982
2	US45342	Jass Jalakas	02.01.1982	15.01.1999
2	US45342	Mait Malakas	16.01.1999	