TIETOKANTOJEN PERUSTEET

**Tietokanta - mikä se on?**

|  |
| --- |
| Tietokanta on jotakin käyttötarkoitusta varten laadittu kokoelma toisiinsa liittyviä säilytettäviä tietoja. |

Tietokanta suunnitellaan jotain määrättyä käyttöä varten. Sen käyttötarkoitus voi ajan myötä laajeta alunperin suunniteltua laajemmalle. Suunnittelun yhteydessä pitäisi varautua käytön laajenemiseen.

Tietoja säilytetään tietokannassa. Ne pitää saada tietokannasta tarvittaessa tehokkaasti käyttöön ja niitä pitää pystyä ylläpitämään.

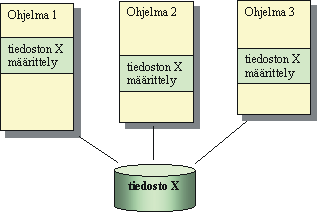
Tietokanta kuvaa jotain sitä hallitsevan yksikön toimintaan liittyvää reaalimaailman ilmiötä ja säilyttää tätä kuvausta.

Sopivalla abstraktiotasolla tarkasteltuna mikä tahansa tietokokoelma on tietokanta. Usein tietokantaan liitetään kuitenkin erityispiirteitä kuten:

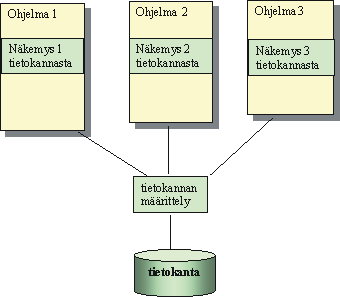
* ohjelmien tietoriippumattomuus
* tietokannan riippumattomuus ohjelmista
* monipuoliset tiedonhakumahdollisuudet
* useat samanaikaiset käyttäjät
* hyvä tietosuoja
* tietojen säilyvyys myös ongelmatilanteissa
* tehokkuus
* suuret tietomäärät
* mutkikkaat tietojen väliset kytkennät.

**Ohjelmien tietoriippumattomuus**

Kun ohjelmassa käsitellään tavanomaista tiedostoa, määritellään tiedoston rakenne, esimerkiksi miten tietueet jakautuvat kenttiin, osana ohjelmaa. Usean ohjelman käyttäessä samaa tiedostoa on tiedoston rakenne määriteltävä kussakin ohjelmassa. Tästä voi aiheutua yhteensopivuusongelma; eri ohjelmilla on erilainen käsitys samoista tiedoista. Joka tapauksessa tämä lisää ohjelmien ylläpitotarvetta. Jos tiedoston rakennetta muutetaan yhdessä ohjelmassa, on sama muutos tehtävä kaikkiin tiedostoa käsitteleviin ohjelmiin, ja ohjelmat on käännettävä sekä asennettava uudelleen. Mahdollisesti on tehtävä myös uudelleenorganisointiohjelma, joka muuttaa tiedoston uuden rakenteen mukaiseksi.



Tietokantoja käytettäessä tietojen kuvaus on erillään ohjelmista, ja jokaisella ohjelmalla voi olla oma näkemyksensä tietokannan rakenteesta. Tämä näkemys saattaa säilyä ennallaan, vaikka tietokannan rakennetta muutettaisiin, joten ohjelmia ei tarvitse muuttaa ellei tietokannan muutos kohdistu juuri niiden käyttämiin tietoihin.



**Tietokannan riippumattomuus ohjelmista**

Tavalliset tiedostot on kiinteästi kytketty niitä käsitteleviin ohjelmiin. Tekstitiedostoja lukuun ottamatta tiedostoja ei pysty käsittelemään ilman sovellusohjelmaa. Kullakin ohjelmointikielellä ja jopa saman kielen eri versioilla saattaa olla oma käsityksensä tiedostojen muodosta. Tietyllä ohjelmointikielellä toteutetulla ohjelmalla laadittua tiedostoa ei välttämättä pysty tulkitsemaan samalla tavoin toisella kielellä tehdyllä ohjelmalla. Tietokannat tarjoavat yleensä liittymät kannan käyttämiseen usean ohjelmointikielen kautta.

**Monipuoliset tiedonhakumahdollisuudet**

Ohjelmointikielissä tiedostoa voi yleensä käsitellä vain

* käymällä sitä läpi peräkkäisjärjestyksessä alusta loppuun tai
* hakemalla tietueita niiden sijaintikohdan perusteella.

Ohjelmointikielet eivät yleensä tarjoa minkäänlaista tietueiden sisältöön perustuvaa hakua. Tietokannoissa monipuolinen sisältöön perustuva käsittely on keskeistä.

**Useat samanaikaiset käyttäjät**

Käyttöjärjestelmät huolehtivat tiedostojen samanaikaisesta käytöstä usein varaamalla koko tiedoston yhdelle käyttäjälle. Toiset käyttäjät eivät pääse muokkaamaan tiedostoa vaikka muutos kohdistuisi aivan eri osaan tiedostoa kuin mitä toinen käyttäjä on muokkaamassa. Jos samanaikaisia käyttäjiä on paljon, tällainen menettely ei onnistu. Tietokantojenkin yhteydessä käsittelykohde joudutaan jossain tilanteessa varaamaan vain tietyn käyttäjän operaatioiden kohteeksi, mutta varattavat kohteet pyritään tekemään mahdollisimman pieniksi ja siten mahdollistamaan suuret samanaikaisten käyttäjien määrät.

**Hyvä tietosuoja**

UNIX-käyttöjärjestelmässä tiedoston luku- ja kirjoitusoikeus voidaan antaa joko pelkästään tiedoston omistajalle, jollekin käyttäjäryhmälle tai kaikille käyttäjille. Tietokantojen tarjoama suojaus on monipuolisempaa ja täsmällisempää. Oikeuden kohde voidaan tarvittaessa rajata huomattavasti pienemmäksi. Käyttäjäryhmiä voi määritellä monipuolisemmin. Tietokannan käytöstä voidaan pitää hyvin tarkkaa operaatiokohtaista käyttökirjanpitoa. Tietokanta voi myös olla kokonaan salakirjoitettuna.

**Tietojen säilyvyys myös ongelmatilanteissa**

Tiedostojen säilyminen varmistetaan ajoittain otettavin varmuuskopioin. Näin menetellään myös tietokantojen kohdalla. Näiden lisäksi on usein tarjolla jatkuva varmistuskopioiden oton välisenä aikana tapahtuva muutosten kirjaus häiriöistä toipumista varten. Järjestelmät pyrkivät takaamaan muutosten pysyvyyden. Erityisesti taataan, ettei jokin operaatio tule vajavaisesti suoritettua siten, että vain osa tarkoitetuista muutoksista toteutuisi.

**Tehokkuus**

Monipuoliset sisältöön perustuvat tiedonhakumahdollisuudet edellyttävät tehokkaita teknisiä rakenteita tiedon tallennukselta ja haulta. Hakuindeksit ja niitä hyödyntävät hakumenetelmät kuuluvat tietokantaohjelmistojen vakiovarusteisiin.

**Suuret tietomäärät**

Useat tietokannat ovat tietomäärältään suuria ja jatkuvasti kasvavia. Tietojen määrän kasvaessa myös käsittelytarpeet saattavat muuttua. Tietokantaa pitää pystyä helposti virittämään, jotta se toimisi tehokkaasti. Tämä pitää pystyä tekemään ilman, että tietokantaa käsitteleviä ohjelmia pitäisi muuttaa.

**Mutkikkaat tietojen väliset kytkennät**

Tietokanta kuvaa jotain reaalimaailman ilmiötä. Reaalimaailmassa asioilla on riippuvuuksia ja maailmassa vallitsee erilaisia lainalaisuuksia. Näiden riippuvuuksien ja lainalaisuuksien täytyy olla voimassa myös reaalimaailmaa kuvaavassa tietokannassa. Erilaisten tietoihin liittyvien tarkistusten tekeminen vie suuren osan ohjelmien koodimäärästä. Jos samoja tietoja voidaan muuttaa usean ohjelman kautta, on samat tarkistukset koodattava toistuvasti. Jos ne unohtuvat jostain ohjelmasta, tietokanta voi korruptoitua. Tavoitteena onkin ollut siirtää sääntöjen valvonta tietokannan vastuulle, jolloin kannan eheys ei olisi niin paljon sidoksissa sovellusohjelmiin. Osittain tämä on tosin johtanut moninkertaisiin tarkistuksiin, sillä järjestelmien käytettävyyssyistä käyttäjien pitäisi saada mahdollisimman nopeasti palautetta virheistä, eikä vasta sitten kun tietoja viedään tietokantaan.

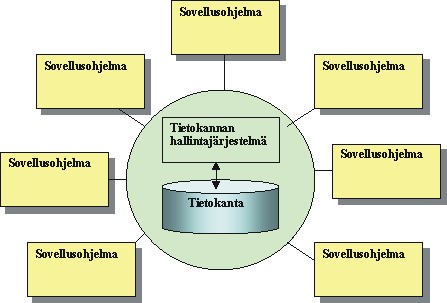
**Milloin tietokantaa kannattaa käyttää**

Tietokannan käyttöä tavanomaisen tiedoston asemasta kannattaa harkita, kun mikä tahansa yllä luetelluista erityispiirteistä näyttäisi toteutuvan sovelluksessa. On esimerkiksi hyvin työlästä ryhtyä itse toteuttamaan kunnollista samanaikaisuuden hallintaa  sovellukseen.

TIETOKANTOJEN PERUSTEET

## Tietokantajärjestelmä

**Tietokantajärjestelmä** (database system, DBS) muodostuu tietokannasta (database), **tietokannan hallintajärjestelmästä** (database management system, DBMS) ja tietokantaa käyttävistä sovellusohjelmista. [Helsingin yliopiston tietojenkäsittelytieteen laitoksen opetushallintojärjestelmä](http://www.cs.helsinki.fi/u/laine/tkp/johdanto/tktlesimerkki.html) on esimerkki tietokantajärjestelmästä.



Sovellusohjelmat voivat kuulua eri sovelluksiin. Ne voivat toimia eri tietokoneissa kuin tietokannan hallintajärjestelmä ja niiden toteutukseen on voitu käyttää useita eri ohjelmointikieliä. Jos sovellusohjelmat toimivat eri tietokoneissa kuin tietokannan hallintajärjestelmä, tarvitaan tietoliikenneohjelmistoja huolehtimaan järjestelmän osien välisestä tietoliikenteestä. Tietokannan hallintajärjestelmän,  tietokannan ja niiden toimintaympäristönä olevan laitteiston yhdistelmää kutsutaan **tietokantapalvelimeksi** (database server).

## Tietokannan hallintajärjestelmä

Tietokannan hallintajärjestelmällä tarkoitetaan yleensä tietokantojen käsittelyyn tarkoitettua yleisohjelmistoa. Tällainen ohjelmisto ei ole tarkoitettu yhden tietyn tietokannan käsittelyyn, vaan sillä voi käsitellä mitä tahansa tietyn tyyppistä tietokantaa. Erilaisia tietokantatyyppejä ovat esimerkiksi relaatiotietokannat, tekstitietokannat, oliotietokannat ja multimediatietokannat. Relaatiotietokannat ja oliotietokannat on tarkoitettu määrämuotoisen datan tallentamiseen. Ne eroavat toisistaan perustana olevan tietomallin suhteen. Relaatiotietokannoissa tietomallina on yksinkertaisiin rakenteisiin perustuva hyvin määritelty relaatiomalli, oliotietokannoissa laajempiin käsitteistöihin perustuvat oliomallit. Oliomalleja on useita ja ne poikkeavat osittain peruskäsitteiltään. Relaatiotietokannat soveltuvat hyvin tilanteisiin, joissa joudutaan käsittelemään suuria määriä lyhytkestoisia tietokantaoperaatioita. Oliotietokannoille puolestaan ovat usein tyypillisiä pitkäkestoiset operaatiot ja mutkikkaat yhteydet tietojen välillä. Suurten operaatiomäärien käsittelyyn ne eivät yleensä sovellu. Teksti- ja multimediatietokannat tallentavat enimmäkseen vapaamuotoista tietoa. Tietomäärät saattavat olla suuria.

Tietokannan hallintajärjestelmän voi karkeasti jakaa

* käyttöliittymiin (interface)
* kuvausjärjestelmään (data definition facility) ja
* tietokantamoottoriin (database engine)

Käyttöliittymät muodostavat tietokannan hallintajärjestelmän ulkoisen rajapinnan.

* Hallintaliittymän  kautta määritellään ja muokataan tietokannan rakenteita, hallinnoidan käyttöoikeuksia, koordinoidaan suojausta ja varmistusta sekä tarkkaillaan ja säädellään tietokannan käyttäytymistä.
* Suorakäyttöliittymän avulla käyttäjät voivat suorittaa tietokantaoperaatioita suoraan ilman sovellusohjelmia.
* Ohjelmaliittymät tarjoavat mahdollisuuden käyttää tietokantaa sovellusohjelmista.

Kuvausjärjestelmä tallentaa kaiken tietokantaan liittyvän määrittelytiedon. Määrittelytietoon eli metatietoon kuuluvat esimerkiksi käyttäjien, oikeuksien ja tietokannan rakenteiden määrittelyt. Kuvausjärjestelmä on keskeinen tietokannan hallintajärjestelmän kannalta. Kaikki tietokannan käsittelyoperaatiot tarvitsevat kuvausjärjestelmään sisältyviä tietoja.

Tietokantamoottorin vastuulla on tietokannan käsittelyoperaatioiden suorittaminen. Moottori voidaan jakaa toiminnallisuutensa perusteella useisiin komponentteihin

* Pääsynvalvonnan (authorization control) tehtävänä on tarkastaa käyttäjien oikeudet operaatioihin. Oikeudet tarkistetaan jokaisen operaation yhteydessä
* Kyselyn optimoijan (query optimizer) tehtävänä on laatia toteutussuunnitelma tietokantaoperaatiolle. Käyttäjien antamat tietokantaoperaatiopyynnöt ovat varsin korkealla abstraktiotasolla. Optimoija kääntää operaatiopyynnön joukoksi alemman abstraktiotason operaatioita ja määrää niille suoritusjärjestyksen.
* Transaktion hallinnan (transaction manager) tehtävänä on valvoa samanaikaisia tietokantaoperaatioita sekä varata ja vapauttaa resursseja, huolehtia operaatioiden päättymisistä ja mahdollisista peruutuksista.
* Eheyden valvonnan (integrity control) tehtävänä on valvoa, että tietokantaan liittyviä sääntöjä ei rikota.
* Suorittaja (command processor) ohjaa operaatiopyynnön käsittelyä.
* Puskurien hallinta (buffer management) vastaa keskusmuistin ja apumuistin välisestä tiedonsiirrosta. Puskurit ovat keskusmuistista tietokannan tiedoille varattuja työtiloja. Puskurien hallinnan tehtävänä on tietää mikä osa tietokannasta on kulloinkin puskureissa ja ovatko tiedot mahdollisesti muuttuneet sen jälkeen kun ne haettiin keskusmuistiin. Se hoitaa tietojen palautukset kantaan ja päättää pitääkö jokin tieto hakea apumuistista.
* Hakumenetelmät (access methods) toteuttavat tietojen hakua kannasta. Menetelmiä on yleensä tarjolla useita peräkkäisestä läpikäynnistä hakuindeksejä hyödyntävään täsmähakuun.

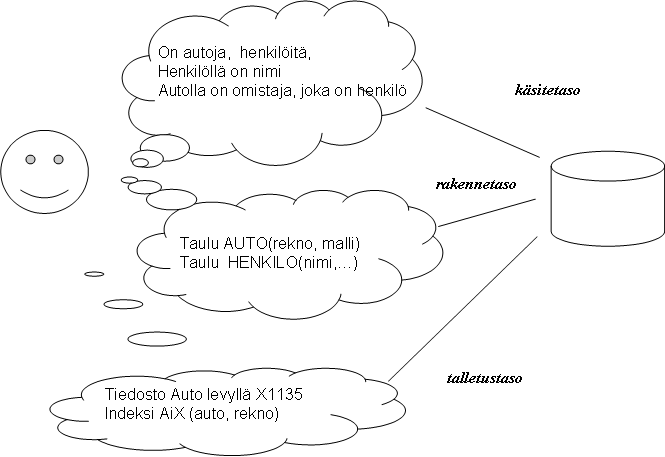
TIETOKANTOJEN PERUSTEET

**Tietomallit**

Tietokantaa tai mitä tahansa tietokokoelmaa ja sen sisältämiä tietoja voidaan tarkastella eri näkökulmista. Perinteinen näkökulmajaottelu perustuu kolmeen abstraktiotasoon:

* käsitetasoon (conceptual level)
* rakennetasoon (logical level, structural level) ja
* talletustasoon (physical level, internal level).

Kuhunkin näkökulmaan liittyen on esitetty lukuisia tietomalleja (data model) eli käsitteistöjä, joihin pohjautuen tietoa kyseisellä abstraktiotasolla kuvataan.

 Kuva 1: Abstraktiotasot ja niiden näkökulma tietokantaan.

**Käsitetaso**

Käsitetason abstraktiossa tietokannasta määritellään sen sisältö: mitä tietoja tietokannassa on ja miten nämä liittyvät toisiinsa. Määrittelyn tulisi olla riippumaton kaikista tietokanta- ja ohjelmointitekniikoista. Tällainen kuvaus saadaan aikaan esimerkiksi laatimalla malli siitä kohdealueesta, mitä tietokannan tiedoilla on tarkoitus kuvata. Tunnetuimpia tietomalleja tietojen kuvaamiseen käsitetasolla ovat ER-malliperheen mallit (Entity-Relationship models, ER-models) ja nykyään oliomallit. Käsitetason tietokantakuvauksia käytetään tietokannan tietosisällön suunnittelussa.

Käsitetasolla tietokannan kuvaamisessa käytettäviä käsitteitä ovat esimerkiksi tietokohteet, tietokohteiden väliset yhteydet ja tietokohteiden ominaisuudet. Kurssin 'Käsiteanalyysi' osiossa tarkastellaan tietojen käsitetason kuvaamista.

**Rakennetaso**

Rakennetason abstraktiossa tietokantaa tarkastellaan lähinnä ohjelmoijan tai tietokannan suorakäyttäjän kannalta: millaisina rakenteina hän tietokannan hahmottaa. Tämän tason tietomalleista tunnetuin on relaatiomalli (the relational model of data), jonka mukaisia tietokantoja tällä kurssilla pääasiassa tarkastellaan. Jo enimmäkseen historiaan vaipuneita 70- ja 80-lukujen tietokannanhallintajärjestelmissä käytettyjä tietomalleja ovat verkkomalli (network model) ja hierakkinen malli (hierarchical model).[EN99] Nykyisin rakennetason mallina ovat yleistyneet myös oliomalli ja XML. Rakennetason kuvauksia tarvitaan tietokantaan kohdistuvassa tiedonhaussa ja tietokantaa käyttävien ohjelmien laadinnassa.

Rakennetason kuvaukset ovat keskeisiä tietokannan hallintajärjestelmän kannalta. Ne ovat järjestelmäriippuvia. Saman tyyppiset järjestelmät voivat käyttää samaa kuvausrakennetta.

**Talletustaso**

Tallennustasolla kiinnostuksen kohteena on se, miten tietokanta on sijoitettu apumuistiin, millaisia hakurakenteita käytetään, miten säädellään tietokantapuskureita, tilanvarauksia, jne. Kuvausvälineistö ja -käsitteistö näiden esittämiseen on järjestelmäkohtaista. Tallennustason kuvauksia tiedoista tarvitaan kun tietokantaa viritetään toimimaan tehokkaasti.

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

# Tietosisällön kuvaus ER-mallilla

Tietokannan (tai minkä tahansa tietokokoelman) käsitetason kuvauksessa määritellään, mitä tietoa tietokokoelmaan sisältyy. Kuvauksessa

* kiinnitetään tietoon liittyvät käsitteet
* määritellään tiedon merkitys ja
* esitetään tietoon liittyvät säännöt.

Tietosisällön kuvaamiseen on esitetty useita tietomalleja. Tunnetuimpia malleja ovat Entity-Relationsip (ER) -malliperheen mallit, joiden kantamallina voidaan pitää [Chenin vuonna 1976 esittämää mallia](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#ref1). Malliperheen malleilla on yhteinen peruskäsitteistö, mutta jonkin verran vaihtelua yksityiskohdissa ja erityisesti kuvauksen esitystekniikassa. Seuraavassa esitellään ER-mallin käsitteet geneerisinä, ei mihinkään tiettyyn mallivariaatioon liittyvinä. Käsitteiden esittelyn jälkeen käsitellään näihin perustuvan kuvauksen graafista esittämistä. Perinteisestä esitystavasta poiketen tällä kurssilla kuvaustekniikkana käytetään UML-luokkakaavio -pohjaista kuvaustekniikkaa.

## Sisällysluettelo

|  |
| --- |
| [ER-mallin käsitteet](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#ermalli) |
| [Tietokohteet](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#tietokohteet) |
| [Ominaisuudet ja attribuutit](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#attribuutit) |
| [Yhteydet](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#yhteydet) |
| [Alisteiset tietokohteet](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#alisteiset) |
| [Tyyppihierarkia](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#tyyppihierarkia) |
| [Kuvaustekniikka](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#kuvaustekniikka) |
| [Tietokohteet ja attribuutit](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#tietokohteet_kuva) |
| [Yhteydet](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#yhteydet_kuva) |
| [Alisteiset kohdetyypit](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#alisteiset_kuva) |
| [Tyyppihierarkia](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#tyyppihierarkia_kuva) |
| [Lähteitä](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#lahteet) |

## ER-mallin käsitteet

### Tietokohteet (entities)

ER-malli määrittelee tietokohteet kohdetyyppien (entity types) avulla. Kohdetyypit (myös yksilötyyppi termiä käytetään) nimeävät asioita, joihin liittyy jotain tietoa. Tällainen kohdetyyppi voisi olla opiskelija, kurssi, kurssikirja, luentokerta tai mikä tahansa konkreettinen tai abstrakti kohde, johon liittyvää tietoa halutaan tallentaa. Mallissa ei kuvata yksittäisiä kohteita, siis esimerkiksi jotain tiettyä opiskelijaa tai jotain tiettyä kurssia, vaan kohteita tarkastellaan tyyppeinä. Kullakin kohdetyypillä on kuvaava nimi. Lisäksi kohdetyyppiin voidaan liittää määrittelyteksti. Yksittäisten tiettyä tyyppiä olevien kohteiden olemassaolo ja ominaisuudet puolestaan esitetään tietokannassa datana.

[sisällysluetteloon](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#sisalto)

### Ominaisuudet ja attribuutit

Jokaiseen tietokohteeseen liittyy ominaisuuksia, jotka halutaan esittää datana. Tällaisia ominaisuuksia mallinnetaan kohdetyyppeihin liittyvillä***attribuuteilla*** (attribute). Kohdetyyppiin opiskelija liittyviä attribuutteja voisivat olla opiskelijanumero, syntymäaika, osoite, opiskelun aloitusvuosi, jne. Kohdetyyppiin kurssi liittyviä attribuutteja voisivat olla kurssinumero, nimi, opintopisteiden määrä ja kurssikuvaus.

Jokaisella attribuutilla on**nimi** (name). Nimen pitäisi mahdollisimman hyvin kuvata attribuutin merkitystä. Tietyn yksittäisen kohteen ominaisuus esitetään datana attribuuttiin liitettävän **arvon** (value) avulla. Attribuutin kelvolliset arvot kuuluvat samaan**arvojoukkoon** (domain). Jokaisella arvojoukon arvolla on merkitys, kun sitä tarkastellaan attribuutin yhteydessä. Tarkastellaan arvojoukkona esimerkiksi pienten kokonaislukujen 0-300 järjestettyä joukkoa. Luvuilla itsessään ei ole mitään merkitystä. Kun näitä lukuja käytetään tietokohteen henkilö attribuutin pituus arvoina, niihin liitetään merkitys ’henkilön pituus senttimetreinä’. Tällöin esimerkiksi arvo 170 esittää, että kuvattava henkilö on 170 cm pitkä. Tietosisällön määrittelyssä esitellään attribuuttien arvojoukot ja kerrotaan yleisesti arvojen merkitykset. Merkitykset voi joutua kertomaan arvokohtaisesti, jos arvot ovat koodattua tietoa.

#### Esimerkki:

Kohdetyyppi: Henkilö

* attribuutti: henkilönumero (henkilön tunnistukseen käytettävä tunnus),   
  arvojoukko: rakenteinen merkkijono, tarkistusmerkillä,  
  yksiarvoinen, pakollinen, tunnistava
* attribuutti: sukupuoli (kuvaa henkilön sukupuolen)  
  arvojoukko: Merkki, N=nainen, M=mies, T=tuntematon  
  yksiarvoinen, valinnainen
* attribuutti: loppututkinto (henkilön suorittama tutkinto)  
  arvojoukko: tutkintokoodisto  
  moniarvoinen, valinnainen

Attribuutit voidaan määritellä yksi- tai moniarvoisiksi. Yksiarvoisella attribuutilla voi olla kunkin kohteen kuvauksessa enintään yksi arvo. Esimerkiksi henkilöllä on vain yksi henkilönumero ja vain yksi sukupuoli. Moniarvoisella attribuutilla voi kohteen kuvauksessa olla useita samanaikaisia arvoja, esimerkiksi henkilöllä voi olla useita loppututkintoja. Attribuutit voidaan myös määritellä pakollisiksi tai valinnaisiksi. Pakollisen attribuutin arvon pitää sisältyä jokaiseen kohdetyyppiä olevan kohteen kuvaukseen. Valinnaisen arvo voi puuttua jostain kuvauksesta. Kun kohteista tallennetaan dataa, täytyy jotenkin pystyä luomaan yhteys tallennetun datan ja sillä kuvatun kohteen välille eli pitää pystyä yksilöimään kuvauksen kohde. Kuvauksissa yksilöinti voi tapahtua vain niihin sisältyvän datan avulla. Niitä attribuutteja, joiden arvojen perusteella kohde voidaan yksilöidä, kutsutaan yksilöiviksi attribuuteiksi. Joskus yksilöintiin riittää yksittäisen attribuutin arvo. Toisinaan tarvitaan useampien attribuuttien arvot. Esimerkiksi henkilöiden kohdalla attribuutti henkilönumero voisi olla yksilöivä attribuutti.

[sisällysluetteloon](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#sisalto)

### Yhteydet (relationships)

Tietokohteet ovat usein jollain tavoin riippuvaisia toisistaan. Esimerkiksi opiskelija ja hänen kurssisuorituksensa kytkeytyvät toisiinsa, ja tieto tällaisen kytkennän olemassaolosta olisi syytä tallentaa tietokantaan. Kytkennän olemassaolo voitaisiin esittää määrittelemällä kurssisuoritus-tyyppisille tietokohteille attribuutti suorittaja, jonka arvona olisi jokin opiskelijan yksilöivä tieto. ER-mallissa tällaisia kytkentöjä pidetään kuitenkin niin tärkeinä, että niille on olemassa oma mallinnusrakenteensa: yhteydet (relationships). Joissakin ER-malliperheen malleissa yhteyksien edellytetään olevan kahdenvälisiä (binäärisiä) eli yhteydessä on kaksi osapuolta. Toisissa sallitaan myös useamman osapuolen yhteydet. Tällä kurssilla rajaudutaan kahdenvälisiin yhteyksiin. Yhteys siis mallintaa tilannetta, jossa kahden kohteen välillä on jokin kytkentä, esimerkiksi:

* henkilö on töissä yrityksessä
* opiskelija on saanut aikaan opintosuorituksen
* kirja kuuluu kurssin oppimateriaaliin.

Yhteydet määritellään osapuolten kohdetyyppien välisinä. Yllä mainitussa ’on töissä’ yhteydessä toinen osapuoli on tyyppiä henkilö ja toinen osapuoli on tyyppiä yritys. Osapuolille voidaan nimetä roolit, jossa ne osallistuvat yhteyteen. ’On töissä’ yhteydessä henkilön rooli olisi työntekijä ja yrityksen rooli työnantaja. ’On töissä’ yhteydessä osapuolet ovat keskenään erityyppisiä, mutta esimerkiksi esimies-alainen suhteessa molemmat osapuolet ovat samaa kohdetyyppiä.

Edellä esitetyn perusteella yhteyksille voidaan antaa **nimi** ja määritellä **tulkinta** sille, mitä yhteyden olemassaolo tarkoittaa. Osapuolille voidaan myös nimetä***roolit***. Yhteyksien määrittelyyn voidaan lisäksi liittää tietosisällön rakennetta kuvaavia ***sääntöjä***. Tärkein sääntötyyppi ovat***osallistumisrajoitteet*** (cardinality constraints). Osallistumisrajoitteilla kuvataan sitä, kuinka monessa samantyyppisessä yhteydessä kukin osapuolityypin kohde voi olla samanaikaisesti. Osallistumisrajoitteella voitaisiin esimerkiksi määritellä, että henkilö voi olla roolissa työntekijä enintään yhdessä samanaikaisessa ’on töissä’ -tyyppisessä yhteydessä, mutta yritys voi olla useassa tällaisessa yhteydessä samanaikaisesti työnantajana. Osallistumisrajoitteet voidaan määritellä antamalla osallistumisille ala- ja yläraja. Alarajoista merkittävimmät ovat nolla (valinnainen yhteys) ja yksi (pakollinen yhteys). Ylärajoina useimmin käytettyjä ovat yksi (funktionaalinen yhteys) ja epämääräisen monta. Esimerkiksi yhteydessä ’opiskelija on saanut aikaan opintosuorituksen’ opiskelijan ei tarvitse olla osapuolena yhdessäkään kytkennässä, mutta hän voi olla epämääräisen monessa. Opintosuoritus sen sijaan voi olla kytkettynä vain tasan yhteen opiskelijaan.

Vaikka osallistumisrajoitteet voidaankin antaa sekä ala- että ylärajojen avulla, luokitellaan yhteyksiä erityisesti tietokannan suunnittelun kannalta usein pelkästään ylärajapohjaisesti. Tällöin saadaan kolme luokkaa: ***yhden suhde yhteen***, ***yhden suhde moneen*** ja ***monen suhde moneen***. Yhden suhde yhteen -luokan yhteyksissä kummallakin osapuolella voi olla enintään yksi samanaikainen partneri. Esimerkki tällaisesta on vaikkapa suomalaisen lainsäädännön mukainen avioliitto. Yhden suhde moneen -yhteyksissä toisella osapuolella voi olla useita partnereita. Yllä mainittu opiskelijan ja opintosuorituksen välinen yhteys on esimerkki tällaisesta. Monen suhde moneen -yhteyksissä molemmilla osapuolilla voi olla monia partnereita, esimerkiksi kurssikirjaa voidaan käyttää monella kurssilla ja kurssilla voi olla monta kurssikirjaa.

ER-malliperheen mallit eroavat toisistaan sen suhteen, voiko yhteyksiin liittää attribuutteja vai onko yhteyden olemassaolo ainoa yhteyksistä kiinnostava asia.

[sisällysluetteloon](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#sisalto)

### Alisteiset tietokohteet

Tietokohteet voivat olla riippumattomia tai alisteisia (weak). Alisteinen kohdetyypin kaikki kohteet ovat aina kytkettyinä johonkin alistavaan kohteeseen. Ne ovat myös koko olemassaolonsa ajan kytkettyinä samaan kohteeseen. Yllä mainittu opintosuoritus on esimerkki alisteisesta kohdetyypistä. Opintosuoritusta ei voi olla olemassa ellei sitä ole kytketty johonkin opiskelijaan. Opintosuoritus on alisteinen myös kurssin suhteen. Alisteisen kohdetyypin alistavaan kytkevää yhteyttä kutsutaan alistavaksi yhteydeksi. Alisteisen kohdetyypin kohteiden yksilöintiin voidaan tarvita tieto siitä, mihin alistavaan kohteeseen ne on kytketty. Alistajan kautta tunnistettavaa kohdetta kutsutaantunnisteriippuvaksi ja yhteyttä tunnistavaksi yhteydeksi.

[sisällysluetteloon](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#sisalto)

### Tyyppihierarkia

ER-malli sallii kohdetyyppien määrittelyn siten, että niillä voi olla yhteisiä ilmentymiä (= tyypin mukaisia kohteita). Voidaan määritellä esimerkiksi kohdetyypit henkilö ja johtaja. Näiden yhteisellä ilmentymällä olisi sekä tyypin henkilö että tyypin johtaja ominaisuuksia ja kumpaankin tyyppiin liitettyjä attribuutteja voisi käyttää kohteen kuvaamiseen. Yhteisiä ilmentymiä jakavien kohdetyyppien osoittamiseksi ER-malli tarjoaa mahdollisuuden määritellä sisältyvyyssuhde (IsA) tyyppien välille. Esimerkiksi tyyppi henkilö sisältää tyypin johtaja. Tällöin kaikki henkilöön liitetyt attribuutit pätevät myös johtajille, mutta johtajille määritellyt eivät päde kaikille henkilöille.

[sisällysluetteloon](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#sisalto)

## Kuvaustekniikka

ER-mallilla laaditun tietosisällön määrittelyn esittämiseen on kirjallisuudessa tarjottu useita erilaisia kuvaustekniikkoja. Näiden ytimenä on ollut graafinen kuvauskieli, jolla esitettyjä kaavioita on täydennetty tekstimääritelmin. Ohjelmistotekniikan kuvaustavoissa on viimeaikoina tapahtunut standardoitumista UML-kuvausstandardien myötä. UML:n luokkakaavio soveltuu varsin suoraviivaisesti ER-tietosisältömääritysten esittämiseen [[Gomik](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#ref2)]. Pieniä mallin puitteissa olevia laajennoksia perustekniikkaan tarvitaan.

[sisällysluetteloon](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#sisalto)

### Tietokohteet ja attribuutit

Tietokohteiden tyypit voidaan esittää UML:n luokkasymbolin avulla ja näihin liittyvät attribuutit luokan attribuutteina. Kuvassa 1 on kolme muotoa tietokohteen kuvaamiseen. Kahdessa näkyy vain kohdetyypin nimi ja kolmannessa avatussa muodossa myös attribuuttilistan paikka. Stereotyyppi <<entity>> voi olla tarpeen, jos kaaviossa esiintyy muitakin luokkasymbolilla kuvattavia asioita kuin kohdetyyppejä.

kohdetyypit laatikkoina

Kuva 1: Kohdetyyppi UML-luokkasymbolina, suppea ja lavea muoto.

Luokkasymbolin attribuutit osassa määritellään kohdetyyppiin liittyvät attribuutit. Yhden attribuuttimäärittelyn rakenne on

[<stereotyyppi>][<näkyvyys>] <nimi> [<moniarvoisuus>] [: <tietotyyppi> ] [ { <säännöt> } ]

*(Tässä määrittelyssä hakasulut rajaavat valinnaisen osan ja < > -sulut syntaktisen elementin)*

Määrittelyn alkuun tulee stereotyypin tunnus. Stereotyyppi kertoo onko kyse erikoistetusta attribuutista. ER-mallin kohdalla tunnistava attribuutti katsotaan erikoistetuksi ja se merkitään stereotyyppitunnuksen <<id>> avulla. UML:n attribuuttimäärittelyssä seuraavana osana on näkyvyyden määrittely. Näkyvyys on ohjelmointitekninen asia, jolla kohdetyyppien yhteydessä ei ole merkitystä ja sen voi jättää määrittelemättä. *(Tässä materiaalissa olevat esimerkkikuvat on laadittu Poseidon-työkalulla, jossa näkyvyyttä ei voinut jättää määrittelemättä. Attribuuttinimien edessä oleva miinusmerki on ohjelman käyttämän oletusarvoisen private-näkyvyyden merkki.)*

Nimi on välttämätön. Sen pitäisi kuvata attribuutin merkitystä. Nimi ei kuitenkaan yleensä riitä attribuutin määrittelyksi, vaan tarvitaan erillinen tekstiselvitys attribuutin merkityksestä ja käytöstä. Selvityksessä voidaan käsitellä arvojen tulkintaa, tarkkuutta, yksikköä, mittaamista, jne.

UML:n attribuutit ovat oletusarvoisesti yksiarvoisia. Moniarvoisuus voidaan kuitenkin määritellä toistomäärän avulla. Toistomäärä annetaan attribuuttinimen perässä hakasulkeissa alarajan ja ylärajan avulla. Jos ala- ja ylärajat ovat samat, riittää hakasulkeissa antaa yksi luku. Esimerkiksi [0..5] tarkoittaa, että attribuutilla voi olla 0-5 arvoa. Merkintä [2] puolestaan tarkoittaa, että attribuutilla on 2 arvoa. Tähti-merkkiä [\*] voidaan käyttää tarkoittamaan 'mahdollisesti monta, mutta täsmällistä ylärajaa ei voida antaa'.

Attribuutin tietotyyppinä kuvataan attribuutin mahdolliset arvot. Tietotyyppinä voi käyttää ennalta määriteltyjä tietotyyppejä (kokonaisluku, merkkijono, yms.) tai määritellä oman sovellusaluekohtaisen tietotyypin. Tällaisia voisivat olla rahasumma, nimi, osoite, puhelinnumero, sähköpostiosoite, prosenttiosuus, jne.

UML:ssä attribuutin arvo voidaan määritellä pysyväksi 'frozen' tai muuttuvaksi 'changeable'. Pysyvä arvo säilyy samana koko kohteen olemassaolon ajan. Oletusarvoisesti attribuutin arvo on muuttuva. UML ei tarjoa valmista määrittelytekniikkaa välttämättömille ja yksilöiville attribuuteille. Nämä ovat kuitenkin tietosisältöä kuvattaessa vähintään yhtä tarpeellisia kuin pysyvyys ja muuttuvuus. UML sisältää kuitenkin laajennusmahdollisuuden, jota hyödyntämällä voimme päättää, että välttämätön attribuutti ilmaistaan sääntömääreellä ‘not null’. Tunnistavat attribuutit voitaisiin määritellä myös antaa sääntöinä, mutta kirjallisuudessa on useammin päädytty käyttämään stereotyyppinä määriteltyä erikoistettua attribuuttia. Tällä kurssilla käytämme stereotyyppiä <<id>>.

Esimerkkejä:

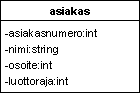
**<<id>>hetu : Henkilötunnus {frozen, not null}**

Attribuutin hetu arvo on tyyppiä Henkilötunnus. Arvo on pysyvä ja välttämätön ja sitä käytetään kohteiden ulkoiseen tunnistamiseen.

**suosikkiruoka [\*] : String**

Attribuutin suosikkiruoka arvona on kokoelma merkkijonoja.

Kuvaesityksessä attribuuteista usein annetaan vain nimi (Kuva 2). Tällöin tarvitaan kuvan täydennykseksi tekstimuotoinen määrittely. Tietokoneavusteisen suunnittelun työkaluohjelmissa on yleensä mahdollisuus valita, miten paljon yksityiskohtia luokkakuvauksessa näytetään.



Kuva 2: Asiakas -luokkaa kuvaava kaaviosymboli (kuvassa attribuuttinimen edessä oleva tavuviiva on näkyvyysmääre, jota ei tarvita, mutta jota ei kuvan laatimiseen käytetyllä UML-kaaviotyökalulla saanut pois)

[sisällysluetteloon](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#sisalto)

### Yhteydet

UML-luokkakaaviossa yhteys kuvataan graafisesti osapuolet yhdistävällä linkillä. Alla olevassa kuvassa (Kuva 3) esitetään yhteys pankkitilin ja asiakkaan välillä. Pankkitili osallistuu yhteyteen roolissa tili ja asiakas roolissa omistaja. Yhteystyypille on annettu nimeksi tilinomistus. Kuvassa näkyvät myös osallistumisrajoitteet. Jokaiseen pankkitiliin pitää liittyä vähintään yksi ja voi liittyä enintään 3 omistajaa. Asiakas ei omista välttämättä yhtään tiliä ja voi omistaa enintään 10 tiliä. Osallistumisrajoitteet luetaan kuvasta siten, että osapuoleen liittyvä osallistumisrajoite luetaan kumppanin puoleisesta päästä yhteysviivaa. Rajoite kertoo samalla montako kumppania kohdetyypin kohteella voi olla.



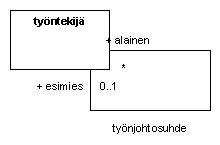
Kuva 3: Tilinomistus yhteys

Yhteydet siis esitetään luokkakaaviossa kytkemällä osapuolten kohdetyypit yhteen viivalla. Viivan päissä ilmoitetaan osallistumisrajoitteet ja roolinimet. Osallistumisrajoite esitetään alaraja .. yläraja –parina. Kuvassa 3 yhteys on nimetty suuntaamattomana. Jos yhteys nimetään suunnatusti, kannattaa nimen yhteydessä antaa lukusuunta (Kuva 4).



Kuva 4: Yhteysnimen lukusuunta: Dokumentti määrittelee ohjelman.

Jos yhteyden molemmat osapuolet ovat samantyyppisiä, kytkeytyy yhteysviivan kumpikin pää samaan kohdetyyppisymboliin. Tällöin roolinimien käyttö on välttämätöntä. Kuvan 5 esimerkissä kuvataan työnjohtosuhdetta. Työtekijällä voi olla enintään yksi esimies ja mahdollisesti useita alaisia.



Kuva 5: Yhteys, jonka osapuolet ovat samaa tyyppiä

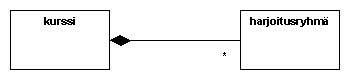
UML-kaaviotekniikassa yhteydet voivat olla suunnattuja. ER-tekniikassa yhteydet ovat kuitenkin aina kaksisuuntaisia.

[sisällysluetteloon](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#sisalto)

### Alisteiset kohdetyypit

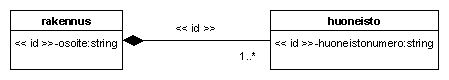
ER-mallin alisteisen kohdetyypin esittämiseen UML-kaaviotekniikka tarjoaa kompositiorakenteen. Alistavan kohteen yhteyskumppani määritellään kompositioksi. Kaavioesityksessä komposition osoittaa yhteyslinkin päässä oleva pieni musta vinoneliö. Komposition jäseneen liittyy viimeisimpien UML määritysten mukaan samat rajoitukset kuin alisteiseen kohteeseen:

* komposition jäsen on olemassaoloriippuva kompositiosta
* komposition jäsen on koko elinkaarensa ajan kytkeytynyt samaan kompositioon



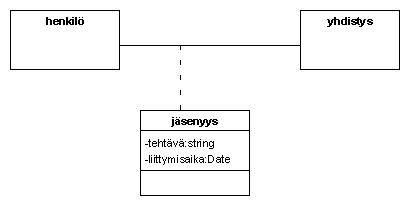
Kuva 6: Alisteinen kohdetyyppi kompositiorakennetta käyttäen.

Alla olevassa kuvassa (kuva 7) huoneisto on alisteinen kohdetyyppi. Jokainen huoneisto on koko olemassaolonsa ajan osa samaa rakennusta. Jos rakennus puretaan, häviää myös huoneisto. UML-perusrakenne ei tarjoa mahdollisuutta kuvata alisteisten kohteiden identifiointia alistavien kohteiden avulla. Esimerkiksi huoneistot identifioidaan tyypillisesti identifioimalla rakennus, johon huoneisto kuuluu. Rakennuksen sisällä huoneiston identifiointiin riittää sisäinen huoneistonumero. Tällainen tilanne on esitetty alla olevassa kuvassa. Tällä kurssilla käytetään stereotyyppiä <<id>>kuvaamaan tunnistavaa yhteyttä. Kuvan 7 esimerkissä huoneiston identifiointiin tarvitaan tieto rakennuksesta, johon huoneisto kuulee, ja lisäksi huoneistonumero -attribuutin arvo.



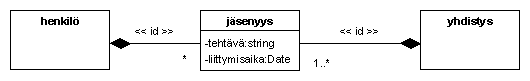
Kuva 7: Alisteisen kohteen identifiointi alistavan yhteyden avulla.

UML-luokkakaaviossa yhteyksiin voi liittää attribuutteja. Tarkastellaan esimerkkinä henkilön jäsenyyttä yhdistyksessä. Yhteyteen liittyvällä attribuutilla ’Tehtävä’ voitaisiin luonnehtia jäsenyyttä - onko jäsen rivijäsen, hallituksessa tai jossain muussa erityisroolissa. Edelleen liittymisaika voitaisiin antaa yhteyden attribuuttina (Kuva 8).



Kuva 8: yhteyteen liittyvät attribuutit

Sama asia voidaan mallintaa myös määrittelemällä uusi kohdetyyppi jäsenyys, johon jäsenyyttä luonnehtivat tekijä ja liittymisaika -attribuutit liitetään (Kuva 9).

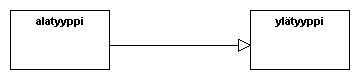


Kuva 9: Yhteys erillisenä tietokohteena, tunnistuu osapuolten kautta

[sisällysluetteloon](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#sisalto)

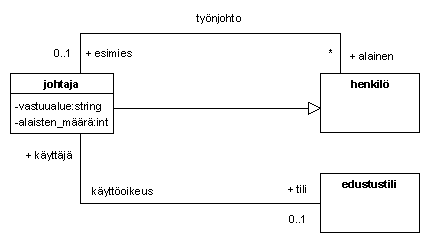
### Tyyppihierarkia

Sisältyvyyssuhde kuvataan UML:ssa yleistys (generalization) -suhteen avulla (Kuva 10).



Kuva 10: Sisältyvyyssuhteen kuvaus

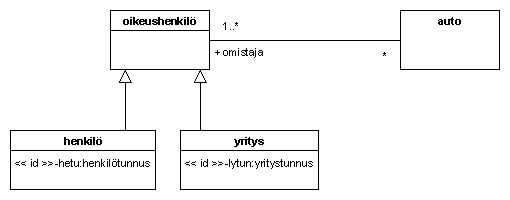
Ala olevassa esimerkissä (Kuva 11) kohdetyypillä johtaja on kohdetyypin henkilö attribuuttien lisäksi attribuutit Vastuualue ja alaisten lukumäärä. lisäksi vain johtajat voivat olla esimiehiä ja heille voidaan antaa edustustilin käyttöoikeus.



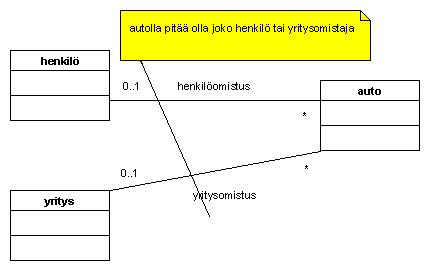
Kuva 11: Johtaja erikoistapaus työntekijästä

Kohdealuetta mallinnettaessa tyyppihierarkian tärkein käyttötilanne on kuvausvaivan säästäminen ja erilaisten sääntöjen täsmällinen ilmaiseminen. Jos usealla kohdetyypillä on samoja attribuutteja ja tyypeille voidaan määritellä yhteinen ylätyyppi, voidaan yhteiset attribuutit liittää ylätyyppiin ja määritellä näin vain kertaalleen.

Sääntöjen täsmällisemmästä esittämisestä voidaan tarkastella esimerkkinä tilannetta, jossa on kuvattava auton omistusta. Auton omistajana voi olla henkilö tai yritys. Jokaisella autolla täytyy olla omistaja. Tyyppihierarkiaa käyttäen asia voidaan esittää kuvan 12 mukaisesti. Tällöin sääntö saadaan esitettyä. Jos asia yritetään esittää ilman tyyppihierarkiaa, päädytään esimerkiksi kuvan 13 mukaiseen malliin. Siinä sääntö ei näy kaaviossa, vaan se on esitettävä erikseen tekstikuvauksena (oheisessa kuvassa puhekuplana), jolloin se jää vähemmälle huomiolle tai sitä ei voida hyödyntää automaattisissa operaatioissa.



Kuva 12: Auton omistukseen liittyvä sääntö - autolla on oltava omistaja.



Kuva 13: Auton omistus kahdella eri yhteydellä, sääntö kommenttina.

[sisällysluetteloon](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#sisalto)

### Lähteitä

Chen P.: The entity-relationship model-toward a unified view of data, ACM Transactions on Database Systems, Vol. 1 , Issue 1 , pages: 9 - 36, 1976

[Davor Comik: Entity-Relationship Modeling with UML, White paper, Rational, IBM, 2003](http://www-128.ibm.com/developerworks/rational/library/content/03July/2500/2785/2785_uml.pdf)

[sisällysluetteloon](http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html#sisalto)

[Harri Laine](mailto:harri.laine@helsinki.fi): *Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

# Käsiteanalyysi

Käsitteellisen tason tietosisältömäärittelyn muodostamisesta käytetään yleisesti nimitystä käsiteanalyysi (conceptual modeling). Tarkoituksena on saada aikaan riittävän tarkka kuva tietosisällöstä, jotta tätä voisi jatkossa käyttää pohjana tietojärjestelmien toimintojen määrittelylle ja toisaalta lähtökohtana tietokannan tai muun tarkasteltavan tietokokoelman, esimerkiksi viestiliikenteen, rakennetason suunnittelulle. Analyysiä joudutaan suorittamaan myös tilanteissa, joissa sovitetaan yhteen eri tietokantoja tai tietokokoelmia.

Tietosisällöstä voidaan laatia eri tarkkuustason kuvauksia. Kuvaus, joka on riittävän tarkka tietojärjestelmän toimintojen karkeaan määrittelyyn, ei ole välttämättä riittävä tietokannan rakennetason suunnittelun pohjaksi. Karkeassa analyysissä riittää tietokohteiden tai jatkoanalyysia vaativien tietokohderyhmien määrittely. Kohteiden välisistä yhteyksiin ei tällöin välttämättä liitetä vielä sääntöjä ja yksi tunnistettu yhteys saattaa edustaa joukkoa yhteyksiä. Attribuutteja ei joko määritellä lainkaan, määritellään vain tärkeimmät tai määritellään vain karkeita tietoryhmiä.

Mallin laatimiseen on monia erilaisia työtapoja. Malli voidaan laatia työryhmässä, jossa on mukana kohdealueen asiantuntijoita ja suunnittelijoita. Kohdealueen tuntemuksensa perusteella asiantuntijat osaavat nimetä tietokohteita. Analyysi on kokonaisvaltaista ja keskeiset kohteet löytyvät suhteellisen vaivattomasti. Työskentelyalustana on esimerkiksi seinätaulu, jonne kohteita kuvaavia muistilappuja kiinnitetään. Yhteyksiä kohteiden välille voidaan merkitä vaikkapa langoilla. Tällainen kaavio muodostaa hyvän perustan suunnitteluryhmän työskentelylle, mutta dokumentointia varten kaavio pitää piirtää puhtaaksi. Toinen lähestymistapa olisi tutustua kehitettävä tietojärjestelmän käyttötapauksiin ja erilaisiin kohdealuetta kuvaaviin dokumentteihin ja laatia niiden pohjalta osamalleja tietosisällöstä. Analyysi tuottaa hienojakoisia osamalleja, jotka yhdistämällä saadaan kokonaiskuva. Tässä lähestymistavassa on työskentelyalustana luonnollista käyttää jotain suunnitteluohjelmistoa. Yhdistämällä nämä lähestymistavat saadaan lähtökohdaksi karkea yleiskuva, jota täydennetään hienojakoisella analyysillä.

## Tietosisällön määrittelyn tehtävät

### 1. Kartoita tietokohde-ehdokkaita.

Laadi luettelo tarkasteltavan ilmiön kannalta keskeisistä kohteista tai ilmiöistä, jotka voisivat tulla kyseeseen tietokohteina. Tällaisia voisivat olla tarkasteltavaan toimintaan osallistujat, toiminnan kohteet, toimintaan liittyvät tapahtumat, materiaalit, tuotteet ja välituotteet ja toiminnalle edellytyksiä luovat asiat..

Kartoituksen pohjana voi käyttää vapaamuotoista tekstikuvausta tarkasteltavasta kohdealueesta. Tästä kuvauksesta alleviivataan tietokohde-ehdokkaita ja kerätään ne luetteloon. Ehdokkaat esiintyvät kuvauksessa usein substantiiveina. Verbit voivat ilmaista yhteyksiä. Alustavaa karsintaa voi tehdä sen perusteella onko asia lainkaan oleellinen mallinnettavan ilmiön kannalta ja liittyykö siihen sellaista informaatiota, joka pitäisi tallentaa tietokantaan tai jota tarvitaan jossakin ajatellussa käsittelytoiminnassa. Vapaamuotoisen tekstikuvauksen vaihtoehtona voi käyttää olemassa olevia raportteja, lomakkeita tai muuta kohdealueeseen liittyvää materiaalia.

### 2. Karsi ehdokkaita

Luetteloon saadut ehdokkaat käydään läpi ja arvioidaan voisiko ehdokas tulla kyseeseen kohdetyyppinä. Arvioinnissa tulisi tarkastella  
• Liittyykö ilmiöön tietosisältöä, joka on välttämätöntä tallentaa.  
• Onko asia riittävän tärkeä kohdealueen kannalta.  
Karsintaa ja ehdokkaiden kartoitusta voidaan joutua tekemään iteratiivisesti (= toistuvasti uudelleen). Ensimmäinen karsintakierros ei välttämättä tuota lopullista tulosta.

### 3. Tunnista kohteiden väliset yhteydet

Yhteyksiä voi etsiä vapaamuotoisesta kuvauksesta (verbit, genetiivit, muut ilmaukset jotka kuvaavat kytkentää). Yhteyksienkin suhteen tulisi miettiä onko yhteys oleellinen tarkasteltavan ilmiön kannalta ja onko tieto yhteyksien olemassaolosta tarpeen tallentaa.

### 4. Täsmennä tietokohdemäärityksiä määrittelemällä attribuutit

Attribuutteja saattaa löytyä vapaamuotoisesta kuvauksesta, mutta yleensä niiden löytäminen edellyttää lisäselvityksiä kohdealueesta, esimerkiksi toiminnan osapuolten haastatteluja. Attribuuttien kohdalla pitäisi myös selvittää, mihin niitä tarvitaan.

### 5. Määrittele yhteyksiin liittyvät osallistumisrajoitteet.

Osallistumisrajoitteiden avulla ilmaistaan rakenteellisia sääntöjä. Ne eivät välttämättä tule esiin vapaamuotoisessa kuvauksessa vaan edellyttävät tarkempaa kohdealueen analysointia.

### 6. Varmista yhteensopivuus järjestelmän toimintoihin.

Tietosisällön analyysia suoritetaan yleensä tietojärjestelmän suunnittelun yhteydessä. Samaan aikaan suunnitellaan myös järjestelmän käyttötapauksia ja järjestelmän toimintoja. Tietosisällön kattavuus on varmistettava selvittämällä onko tietosisältö riittävä järjestelmän käyttötapausten hoitamiseksi. Samoin on syytä varmistaa, ettei tietosisältöä määritellä liian laajaksi ja oteta mukaan tietoa, jota ei missään tarvita.

### 7. Varmista minimaalisuus

Tietosisältöä määriteltäessä sama asia voi tulla mallinnetuksi usealla tavalla. Esimerkiksi jokin tieto voi tulla mallinnetuksi sekä yhteytenä että attribuuttina. Samoin mallissa voi olla samaa asiaa tarkoittavia synonyymejä. Lopullisen mallin pitäisi olla kuitenkin minimaalinen siten, että asia olisi mallinnettu vain yhteen kertaan. Tämän saavuttaminen voi edellyttää useita iteraatiokierroksia.

### Analyysiesimerkki

*Tarkasteltavana ilmiönä on elokuvalipun varaaminen. Lippu oikeuttaa paikkaan tietyssä näytöksessä. Näytöksellä tarkoitetaan elokuvan esittämistä tietyssä teatterissa tiettyyn aikaan. Samaa elokuvaa voidaan esittää monessa teattereissa useina aikoina. Teatterin ohjelmakartta määrittelee, mitä elokuvia missäkin näytöksessä esitetään. Asiakas voi samassa varauksessa varata useita lippuja. Lippuja myydään vain rekisteröityneille asiakkaille. Liput voi tulostaa omalla tulostimella. Lipussa on koneellisesti luettava tarkistusmerkintä. Asiakas voi valita haluamansa paikat vapaiden paikkojen kartasta.*

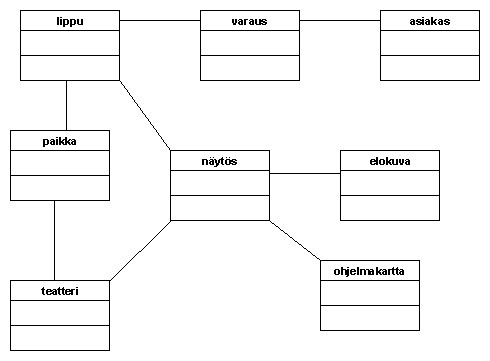
Edellistä kappaletta voidaan pitää ilmiön vapaamuotoisena kuvauksena. Otetaan se jatkokäsittelyyn ja alleviivataan kohde-ehdokkaita.

*Tarkasteltavana ilmiönä on elokuvalipun varaaminen. Lippu oikeuttaa paikkaan tietyssä näytöksessä. Näytöksellä tarkoitetaanelokuvan esittämistä tietyssä teatterissa tiettyyn aikaan. Samaa elokuvaa voidaan esittää monessa teattereissa useina aikoina. Teatterin ohjelmakartta määrittelee, mitä elokuvia missäkin näytöksessä esitetään. Asiakas voi samassa varauksessa varata useita lippuja. Lippuja myydään vain rekisteröityneille asiakkaille. Liput voi tulostaa omalla tulostimella. Lipussa on koneellisesti luettava tarkistusmerkintä. Asiakas voi valita haluamansa paikat vapaiden paikkojen kartasta.*

Saatiin luettelo

|  |  |
| --- | --- |
| **Ehdokas** | **Analyysi** |
| elokuvalippu | Tämä on lipun synonyymi ja karsitaan pois. |
| lippu | Oikeus paikkaan, asia, mahdollinen, onko tietosisältöä? |
| paikka | Istuin teatterissa, varauksen kohde, mahdollinen, tietona tunnus. Tämä voidaan joutua tarkentamaan. |
| näytös | Varauksen kohde, keskeinen, tietona esitysaika |
| elokuva | Näytöksen sisältö, tietoja nimi, yms. |
| teatteri | Isompi paikka |
| asiakas | Varauksen tekijä, oleellinen |
| ohjelmakartta | Ohjelmatarjonta, kytkennät ajan, paikan ja elokuvan välillä. Pilkottava mahdollisesti pienempiin osiin. |
| varaus | lipun tilaus |

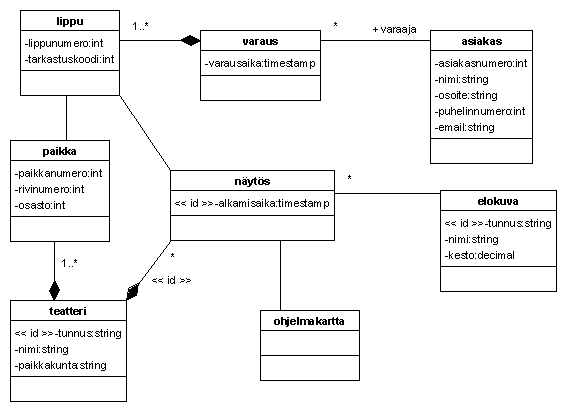
Päätetään alustavasti ottaa mukaan muut ehdokkaat paitsi elokuvalippu, joka on lipun synonyymi. Tarve ottaa lippu mukaan on kyseenalaista. Sen kohtalo jää ratkaistavaksi myöhemmin attribuuttien selvityksen yhteydessä. Samoin jää myöhemmin ratkaistavaksi pilkotaanko ohjelmatarjonta pienemmiksi osiksi. Kuvassa 1 on ensimmäinen luonnos ER-mallista. Yhteyksiä on vasta luonnosteltu, mutta niihin ei vielä liity osallistumisrajoitteita eikä edes nimiä.



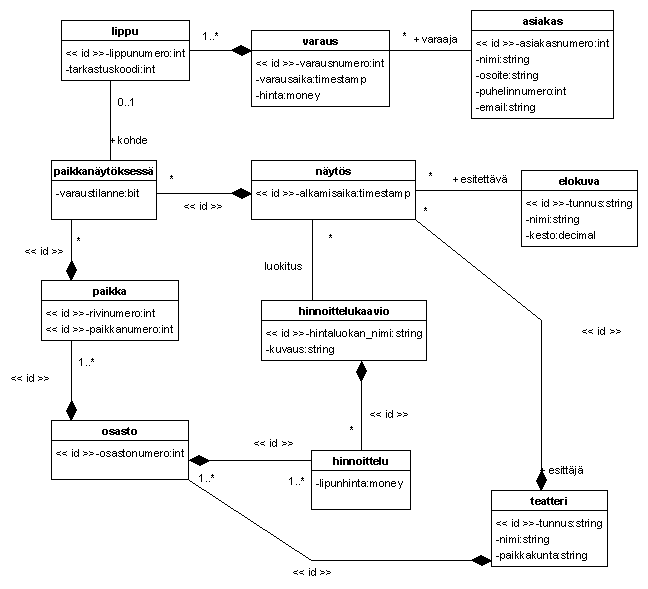
Kuva 1: Alustava käsitemalli

Lisätään kaavioon kohteisiin liittyviä attribuutteja ja täsmennetään yhteyksien määrittelyä. Päädytään kuvan 2 mukaiseen kaavioon. Ohjelmakarttaan ei näytä löytyvän mitään attribuutteja. Tieto elokuvan esityksistä saadaan ilmaistua kohteiden elokuva, näytös ja teatteri välisten yhteyksien avulla. Ohjelmatarjonta on siis tarpeeton. Analyysin suorittajaa jää vaivaamaan joukko ongelmia:

* Varaukseen pitäisi liittää hinta, mutta ei ole tietoa miten se muodostuu. Tätä asiaa pitää lähteä selvittämään haastatteluilla.
* Tietosisältö näyttäisi riittävän lippujen kirjoittamiseen, mutta joitain perustavaa laatua olevia sääntöjä ei kaavioteknisin keinoin kyetä esittämään. Esimerkki tälläisesta säännöstä on, ettei näytöksen paikkaa voi myydä useaan kertaan.



Kuva 2: Ensimmäisen tarkennuskierroksen tulos.

Suunnittelija saa haastatteluissaan selville, että teatterin saman osaston paikat ovat saman hintaluokan näytöksissä kaikki samanhintaisia. Hinta kuitenkin riippuu elokuvasta ja näytöksestä. Eri teattereilla on erilaisia hintoja. Tämä edellyttää, että teatterin rakenne- ja hinnoittelutietoa otetaan mukaan tietokantaan. Suunnittelija tuottaa seuraavan version (kuva 3). Tässä on hintaluokat määritelty tietokohteena hinnoittelukaavio. Käyttöön on otettu tietokohde 'paikkanäytöksessä' jolla kuvataan tietyssä näytöksessä tarjolla olevaa paikkaa. Sääntö paikan moneen kertaan myymisen estämisestä voidaan nyt esittää rakenteellisena sääntönä.

Kuva 3: Toisen tarkennuskierroksen tulos.

[Harri Laine](mailto:harri.laine@helsinki.fi): *Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

# Relaatiotietokannan peruskäsitteet

E. F. Codd esitteli relaatiomallin peruskäsitteet vuonna 1970 artikkelissaan: [E.F.Codd: A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, Vol. 13, No. 6, June 1970, pp. 377-387.](http://www.acm.org/classics/nov95/toc.html) Malli perustana on selkeä matemaattinen käsitteistö. Toisaalta malli on hyvin yksinkertaine ja helposti hahmotettavissa.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sisältö** | |  | | --- | | [Relaatio, arvo ja arvojoukko](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#relaatio) | | [Relaatiokaavio, relaatiokaavion ilmentymä, attribuutti](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#kaavio) | | [Havaintoesitys](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#havainto) | | [Avain ja pääavain](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#avain) | | [Viiteavain ja viite-eheys](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#viiteavain) | |
| **Lisämateriaalia** | |  | | --- | | [*Ramakrisnan&Gehrke*](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/lahteita.html): Luvut 3.1 -3.3 (sivut 57-72) | | [*Elmasri&Navathe*](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/lahteita.html): luvut 5.1 - 5.2 (sivut 125-140) | |

## Relaatio, arvo ja arvojoukko

Relaatiotietokantojen perusajatuksena on tiedon hahmotus matemaattisina relaatioina. Kukin relaatio kuvaa jotakin todellisuuden ilmiötä.

Relaatiomallin mukainen relaatio määritellään seuraavasti

*Olkoon D1, D2, ..., Dn atomisista arvoista koostuvia arvojoukkoja (domain). Relaatio R on joukko monikkoja [tuple, n-tuple] (a1,a2,...,an), joiden ensimmäinen arvo a1 kuuluu joukkoon D1, toinen arvo a2 joukkoon D2 jne. Relaatio on siis ristitulon   
D1  x  D2  x  ...  x  Dn osajoukko.*

### Arvojoukko

Arvojoukko koostuu joukosta arvoja (value). Arvot ovat arvojoukon alkioita. Arvot ovat esimerkiksi nimiä asioille, mittalukuja tai koodeja. Arvojoukko voisi olla vaikkapa

* joukko kokonaislukuja, X= {0,1,2,3}
* joukko kirjainkoodeja, Y= {A,B,N,C, M,E.L}
* joukko sanoja, Z= {kissa, koira, marsu}
* Relaatiotietokannan tapauksessa arvojen oletetaan olevan atomisia eli ne eivät jakaudu pienempiin osiin vaan niitä käsitellään aina kokonaisuuksina.

### Joukko matemaattisena käsitteenä

Matemaattisessa joukossa:

* mikä tahansa arvo esiintyy vain kerran
* arvojen järjestyksellä ei ole merkitystä, joten joukko   
  U={B, A,N,C, E,L,M} on identtinen joukon Y= {A,B,N,C, M,E.L} kanssa

### Ristitulo

Arvojoukkojen K ja P ristitulo:

* on kaikkien sellaisten parien joukko, joissa parin ensimmäinen alkio kuuluu joukkoon K ja toinen alkio joukkoon P, eli matemaattisesti
  + {(x,y) |  x kuuluu joukkoon  K ja y  kuuluu joukkoon P}
* jokaisen K:n alkion parina jokainen P:n alkio
* relaation monikoiden paikkamäärä = relaation aste.   
  Tämä ilmoittaa monenko arvojoukon ristitulosta on kyse
* Merkintätapa:
  + monikot esitetään sulkeissa olevana järjestettynä arvojonona (a,b,c,...,x)
  + jonon arvot erotetaan toisistaan pilkuilla
  + 2-paikkaista monikkoa kutsutaan pariksi

* Esimerkki:

X x Z =

{ (0,kissa), (0,koira), (0,marsu),  
(1,kissa), (1,koira), (1,marsu),  
(2,kissa), (2,koira), (2,marsu),  
(3,kissa), (3,koira), (3,marsu) }

Arvojoukkoihin X ja Z perustuvia relaatioita ovat esimerkiksi

XZ1 = { (0,kissa), (1,koira) } ja  
XZ2 = { (1,marsu), (2,kissa), (3,kissa) } ja  
XZ3 = { } = tyhjä joukko

**Tehtävä** (linkki tehtävään)

[[http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/top.gif](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#top) Alkuun](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#top)

## Relaatiokaavio, relaatiokaavion ilmentymä, attribuutti

Jokaiseen relaatioon liittyy jokin tulkinta monikoissa olevien arvojen tarkoitukselle. Tietyssä paikassa olevalla arvolla on tietty merkitys. Esimerkiksi relaation

XZ1 = { (0,kissa), (1,koira) }

2-paikkaisissa monikoissa ensimmäinen alkio voisi ilmaista eläimen tunnuksen ja jälkimmäinen eläimen lajin jolloin relaation monikko esittäisi eläimen olemassaoloa.

Jos kaksi relaatiota perustuu samoihin arvojoukkoihin ja niihin liittyvät arvojen tulkinnat ovat samat, on kyseessä saman relaatiokaavion (relation schema) kaksi ilmentymää (instance), esimerkiksi XZ1 ja XZ2 ovat saman relaatiokaavion ilmentymiä. Relaatiokaavio puolestaan on ilmentymiensä määrittely ja ilmentymät ovat määrittelynsä mukaisia.

Relaatiokaaviolle annetaan sen ilmentymien tarkoitusta kuvaava nimi, esimerkiksi ilmentymät XZ1 ja XZ2 voisivat olla kaavion 'Eläin' ilmentymiä. Alkioiden paikoille relaation monikoissa (ensimmäinen, toinen, jne) annetaan paikassa olevan alkion merkitystä kuvastava nimi. Siten kaavion 'Eläin' kohdalla voitaisiin monikoiden ensimmäistä alkiota kutsua *eläintunnukseksi* ja toista *lajiksi*. Alkion paikalle annettua nimeä kutsutaan*attribuutiksi*. Kaavio voidaan nyt esittää yksinkertaisessa muodossa

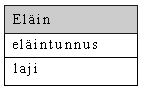
*Eläin(eläintunnus, laji)*

eli määritellään 2-paikkaisia relaatioita, joiden monikoissa ensimmäisessä paikassa on eläintunnus ja jälkimmäisessä laji. Täydellisemmässä kaavioesityksessä

*Eläin(eläintunnus:X, laji:Z)*

on mukaan otettu arvojoukot kertomaan, että *eläintunnus*-arvo otetaan arvojoukosta X ja *laji*-arvo arvojoukosta Y.

Kaavio voidaan esittää myös kuvana

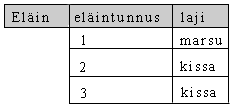


Relaatiokaavioon liittyvät arvojoukot tulisi valita sellaisiksi, että niiden arvot kattavat kaikki mahdolliset reaalimaailman tilanteet, joita relaatiokaavion ilmentymien pitäisi kyetä esittämään. Yllä arvojoukkoon X pitäisi kuulua kaikki mahdolliset eläintunnukseksi kelpaavat arvot ja arvojoukoon Z kaikki mahdolliset lajinimet.

[[http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/top.gif](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#top) Alkuun](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#top)

## Havaintoesitys

Relaatio voidaan esittää havainnollisesti taulukkona, esimerkiksi kaavion *Eläin(eläintunnus, laji)* ilmentymänä relaatio XZ2 = { (1,marsu), (2,kissa), (3,kissa) } esitettäisiin muodossa:

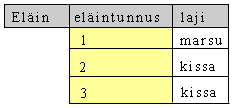


* Otsakerivinä on relaatiokaavio.
* Taulukon (vaaka)rivi esittää relaation monikon.
* (Pysty)sarakkeessa olevat arvot muodostavat osajoukon kyseiseen paikkaan liitetystä arvojoukosta. Otsakerivillä sarakkeessa on paikan nimeävä attribuutti.

[[http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/top.gif](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#top) Alkuun](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#top)

## Avain ja pääavain

Koska relaatio on matemaattinen joukko, ovat kaikki relaation monikot keskenään erilaisia. Monikot kyetään erottamaan toisistaan eli yksilöimään sisältönsä perusteella. Erotteluun ei välttämättä tarvita kaikkia monikon arvoja. Esimerkiksi seuraavassa relaatiossa riittää monikoiden (rivien) yksilöintiin *eläintunnus*-paikassa (sarakkeessa) oleva arvo.



**Avain**

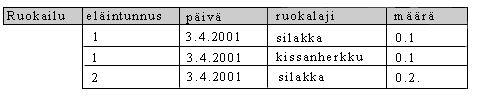
muodostuu niistä monikkojen paikoista, joissa olevien arvojen avulla relaation monikot voidaan yksilöidä missä tahansa kyseisen relaatiokaavion ilmentymässä eikä yksilöinti ole mahdollista jättämällä jokin paikoista pois

* Monikot eivät erottaudu toisistaan vain jossakin relaation ilmentymässä vaan kaikissa mahdollisissa ilmentymissä
* Avaimen määrittely relaatiokaaviolle on sääntö, joka rajoittaa sitä millaisia ilmentymiä kaaviolla voi olla. Sääntö perustuu relaatiolla kuvattavaan todellisuuteen.

Kun määritellään attribuutin '*eläintunnus*' nimeämä paikka kaavion *Eläin* mukaisten relaatioiden avaimeksi asetetaan kaavion *Eläin* ilmentymille sääntö:

* Paikassa *'eläintunnus*' on jokaisessa monikossa eri arvo. Tällä säännöllä ilmaistaan, että kullakin eläimellä on yksikäsitteinen tunnus.

Aina ei ole löydettävissä vain yhtä paikkaa, jossa oleva arvo riittäisi yksilöintiin vaan tarvitaan useammassa paikassa olevia arvoja. Seuraavassa relaatiossa kuvataan eläinten nauttimia ruoka-aineita.



Relaatiosta ei löydy yhtä paikkaa (saraketta), jonka arvojen perusteella rivit olisi yksilöitävissä. Tämän ilmentymän perusteella paikat

* *ruokalaji* ja *määrä* yhdessä, kuten myös
* *eläintunnus* ja *ruokalaji* yhdessä muodostavat yksilöivän yhdistelmän.

Yksilöivätkö nämä yhdistelmät relaation monikot kaikissa Ruokailu -ilmentymissä?

* *Ruokalaji, määrä* -yhdistelmän määrittely avaimeksi tarkoittaa, että tietyn määrän tiettyä ruokalajia, esim. 100g silakkaa, voi syödä vain yksi eläin kertaalleen. Tämä ei vastanne todellisuutta.
* *Eläintunnus, ruokalaji* -yhdistelmä avaimena suosisi monipuolista ruokailua edellyttämällä, ettei eläin voi syödä samaa ruokalajia kuin yhtenä päivänä. Tämäkään ei vastaa todellisuutta.

Tässä tapauksessa ainoa todellisuutta heijastava avainmääritys  on yhdistelmä

*eläintunnus, päivä, ruokalaji,*

joka tarkoittaa sitä, että tietystä ruokalajista tehdään kullekin eläimelle vain yksi kirjaus päivää kohden. Tämän relaatiokaavion kohdalta on syytä  todeta, että tätä ei voi käyttää kirjaamaan jokaista erillistä syöntikertaa. Eläinhän voisi syödä kahdella erillisellä syöntikerralla saman määrän samaa ruokalajia ja tällöin relaatioon tulisi kaksi samanlaista riviä, mikä ei määritelmän mukaan ole mahdollista. Jos erilliset syöntikerrat halutaan kirjata, täytyy relaatiokaavioon ottaa lisätietona mukaan vaikkapa kellonaika ja ottaa se myös avaimeen.

**Tyhjäarvo**

Tyhjäarvo (null) on erikoisarvo, jota käytetään kun

* ei tiedetä mikä todellinen arvo paikkaan kuuluu tai
* paikassa ei voida käyttää mitään todellista arvoa.

Avaimeen kuuluviksi määritellyissä paikoissa ei voi olla tyhjäarvoja.

Tarkastellaan relaatiokaaviota

*Henkilö(Nimi, Osoite, Kännykkänumero, ...)*

*Nimi* ei yksinään riitä yksilöintiin. *Kännykkänumero*, olettaen, että kännykät ovat henkilökohtaisia, riittäisi, mutta kaikilla ei ole kännykkää, joten kännykkänumeroa ei voi käyttää edes avaimen osana. *Nimi* ja *osoite* yhdistelmä saattaisi riittää yksilöintiin, jos osoitteet ovat riittävän tarkkoja ja kaikki osoitteet tiedetään. *Henkilötunnus* tietenkin riittäisi yksilöintiin, jos henkilöiden joukossa ei ole ulkomaalaisia, joilta puuttuu tunnus, ja henkilötunnuksen ottamiselle relaatioon ei ole muita esteitä kuten tietosuoja. Käytännössä tällaisessa tilanteessa turvaudutaan usein keinotekoiseen automaatisesti tuotettuun avaimeen (surrogate, keinoavain), joka tässä tapauksessa olisi vaikkapa *henkilönumero*.

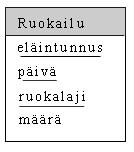
**Pääavain (primary key)**

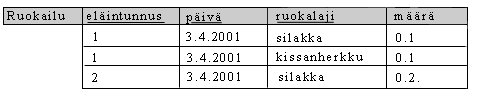
Usein on mahdollista löytää monta sellaista paikkayhdistelmää, joissa olevia arvoja voisi käyttää relaation monikoiden yksilöintiin.  Näiden mahdollisten avainten joukosta on valittava yksi erityisasemaan pääavaimeksi eli ensisijaiseksi tavaksi viitata relaatiokaavion ilmentymien monikoihin.

**Pääavaimen merkitseminen relaatiokaavioon**

Pääavain merkitään relaatiokaavioon alleviivaamalla kaikki pääavaimeen kuuluvat attribuutit:

Ruokailu( eläintunnus, päivä, ruokalaji, määrä)





[[http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/top.gif](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#top) Alkuun](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#top)

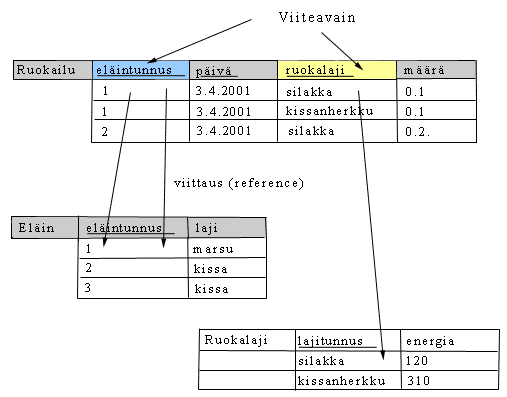
## Viiteavain ja viite-eheys

Relaatiotietokanta muodostuu useasta relaatiosta. Nämä eivät ole toisistaan riippumattomia vaan relaation monikoiden välillä on kytkentöjä. Monikot kytketään toisiinsa sisällyttämällä yhteen monikkoon viittaus (reference) toiseen monikkoon. Relaation monikoilla ei ole järjestysnumeroa tai muuta ulkoista tunnistetta. Niinpä viittaukset täytyy tehdä monikon sisältöön perustuvaa yksilöintitietoa eli pääavainta käyttäen.

**Viiteavain (foreign key)**

Viiteavaimen muodostavat ne relaatiokaavion mukaisten monikkojen paikat (sarakkeet), joita käytetään viittauksen esittämiseen

Viiteavaimeen kuuluu yhtä monta paikkaa kun on viitatun kohteen avaimessa.



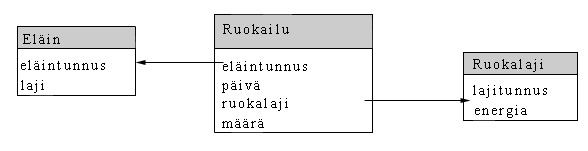
Samaan monikkoon voi kohdistua useita viittauksia.

Monikko voi sisältää useita viittauksia. Yhdessä arvopaikassa voi kuitenkin olla vain yksi arvo. Yllä olevan kuvan Ruokailu-relaatiossa on kaksi viiteavainta. Eläintunnus viittaa *Eläin*-monikkoon ja *ruokalaji* viittaa *Ruokalaji*-monikkoon.

Tällä kurssilla viiteavain esitetään relaatiokaaviossa siten, että viiteavaimena käytettävän attribuutin perään liitetään nuoli ja viitatun relaatiokaavion nimi. Kuvaesityksessä viiteavain-attribuutista  piirretään nuoli viitattuun kaavioon.

Esimerkki:

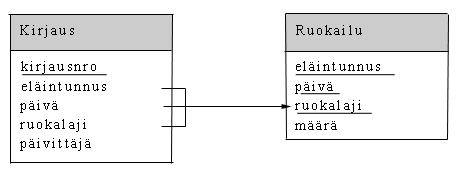
Eläin( eläintunnus, laji)  
Ruokalaji( lajitunnus, energia)  
Ruokailu( eläintunnus->Eläin, päivä, ruokalaji ->Ruokalaji, määrä)



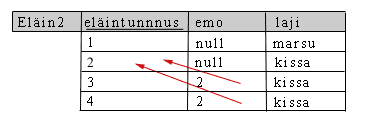
Viittauksessa tulisi aina käyttää kohteena olevan kaavion pääavainta. Kuvassa nuoli lähtee siitä paikasta (attribuutista), joka toimii viiteavaimena, mutta kärjen ei tarvitse osua kohteen pääavaimeen.

Jos viittauksen kohteen pääavain on monipaikkainen, tarvitaan myös viiteavaimeen useita paikkoja. Monipaikkainen viittaus Kirjaus-monikoista Ruokailu-monikoihin esitetään seuraavasti

Kirjaus( kirjausnro, (eläintunnus, päivä, ruokalaji) ->Ruokailu, päivittäjä, ...)

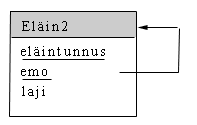
 

Viittaus voi kohdistua myös monikkoon, joka on samassa relaatiossa kuin viittaava monikko. Esimerkiksi



Kaaviossa tämä esitetään

Eläin2( eläintunnus, emo ->Eläin2, laji )



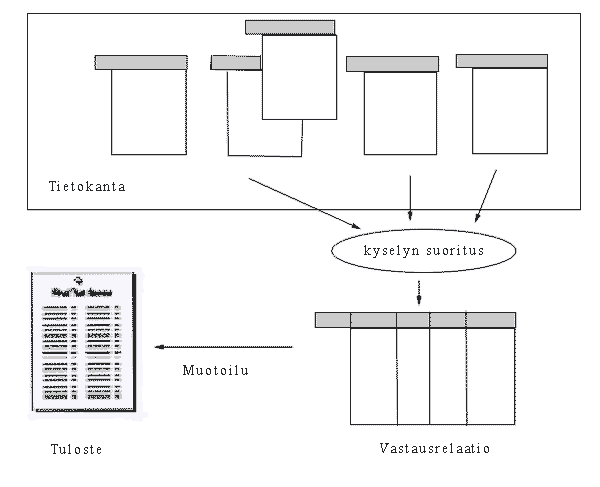
[[http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/top.gif](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#top) Alkuun](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/rakenne.html#top)

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

## Tietotarve, kysely, vastaus

Tietokannat sisältävät reaalimaailman tilaa kuvaavaa tietoa. Tietokannan käyttäjät haluavat tietokannan avulla tyydyttää tietotarpeitaan. Käytännössä tietotarve tyydytetään yleensä antamalla käyttäjälle jokin tietokannan perusteella tuotettu tuloste. Tuloste voi olla muotoiltu tyylikkääksi, mutta sen varsinainen tietosisältö voidaan useissa tapauksissa hahmottaa relaationa samoin kun tietokannan tiedotkin. Tietotarpeen tyydyttäminen olisi tällöin tulosteen perustana olevan relaation tuottamista tietokannassa olevien relaatioiden perusteella.



Tulosteen pohjana olevaa relaatiota kutsutaan vastausrelaatioksi ja sen tuottamista kyselyn suorittamiseksi.

Tietokannan ylläpidossa relaatio korvataan uudella relaatiolla. Uusi relaatio pohjautuu yleensä olemassaoleviin relaatioihin. Sen muodostaminen olemassaolevien relaatioiden avulla vastaa kyselyn vastausrelaation tuottamista.

Relaatiotietokannan käsittelyä varten määriteltiin alunperin kokoelma operaatioita, joita tietokannan relaatioihin soveltamalla saatiin tuloksena vastausrelaatio. Tätä operaatiokokoelmaa kutsutaan **relaatioalgebraksi**.

Perinteisessä algebrassa on määritelty joukko laskentaoperaatioita, esimerkiksi yhteenlasku, vähennyslasku ja kertolasku, joita yksittäisiin lukuihin soveltamalla saadaan laskutoimituksen tuloksena jokin luku. Matematiikan joukko-opissa yksittäisten lukujen asemasta operaatioiden osapuolina ovat matemaattiset joukot. Myös operaatioiden tulokset ovat matemaattisia joukkoja. Relaatiotietokannan relaatiot ovat erikoistapauksia matemaatisista joukoista. Relaatioalgebran ytimen muodostavatkin joukko-opin operaatiot. Näiden lisäksi relaatioalgebra sisältää muutamia lisäoperaatioita, jotka liittyvät tietojen eristämiseen, poimintaan ja yhdistelyyn.

Relaatioalgebran perusoperaatiot ovat yksinkertaisia yhteen tai kahteen relaatioon kohdistuvia operaatioita. Näiden tuottamia tuloksia voidaan kuitenkin käyttää operaatioiden osapuolina ja operaatioiden yhteisvaikutus määrää lopputuloksen.

[Takaisin](javascript:history.back(-1))

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

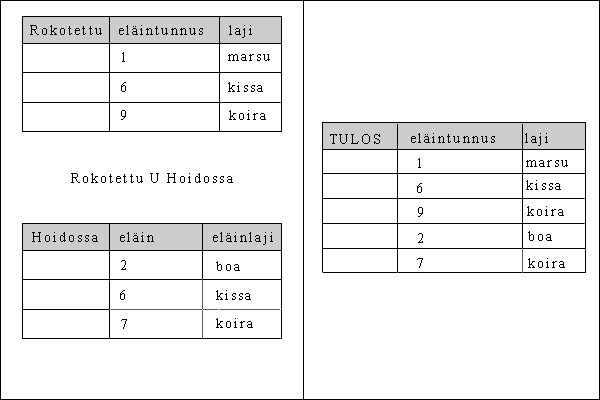
### Relaatioalgebran operaatiot - Yhdiste (Union)

Relaatioiden R ja S yhdiste on niiden monikoiden t joukko, jotka kuuluvat relaatioon R tai relaatioon S, eli

[Katso matemaattiset merkinnät](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/ra_popup.html)

Yhdisteen tulosrelaation sarakkeet nimetään ensimmäisen osapuolen mukaisesti. Relaatioalgebrassa yhdiste on mahdollinen vain[samarakenteisten](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/samarak_popup.html) relaatioiden välillä.

Yhdisteen tulosrelaation koko (monikoiden määrä) on vähintään sama kuin suuremman osapuolen koko ja enintään osapuolten kokojen summa.

Esimerkki:  


[Katso yhdisteen muodostaminen kuvasarjana](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/yhdiste_1.html)

[Takaisin](javascript:history.back(-1))

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

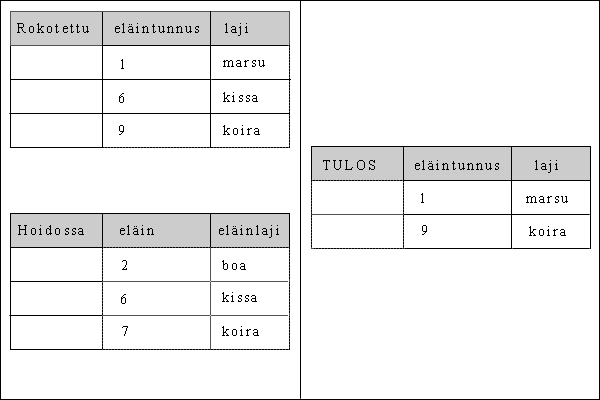
### Relaatioalgebran operaatiot - Erotus (Difference)

Relaation S erotukseen R:stä kuuluvat ne R:n monikot, jotka eivät kuulu S:ään, eli

[Katso matemaattiset merkinnät](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/ra_popup.html)

Relaatioalgebrassa erotus on mahdollinen vain [samarakenteisten](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/samarak_popup.html) relaatioiden välillä. Erotuksen tulosrelaation sarakkeet nimetään ensimmäisen osapuolen mukaisesti.

Erotuksen tulosrelaatio voi olla tyhjä. Enimmillään se on kooltaan sama kuin relaation R koko.

Esimerkki: Rokotettu - Hoidossa  


[Katso erotuksen muodostaminen kuvasarjana](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/erotus1.html)

[Takaisin](javascript:history.back(-1))

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

### Relaatioalgebran operaatiot - Leikkaus (Intersection)

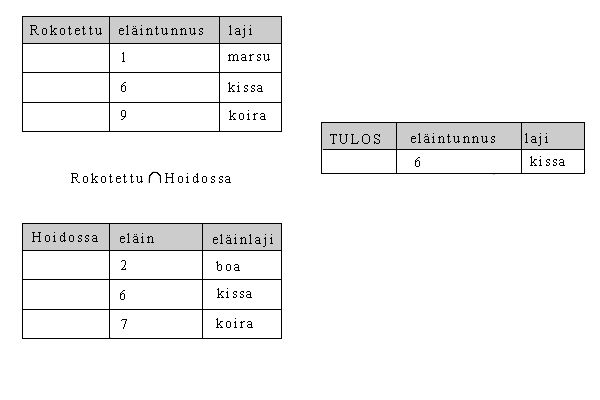
Relaatioiden R ja S leikkaus on niiden monikoiden t joukko, jotka kuuluvat sekä relaatioon R että relaatioon S, eli

[Katso matemaattiset merkinnät](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/ra_popup.html)

Relaatioalgebrassa leikkaus on mahdollinen vain [samarakenteisten](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/samarak_popup.html) relaatioiden välillä. Leikkauksen tulosrelaation sarakkeet nimetään ensimmäisen osapuolen mukaisesti. Leikkaus ei siinä mielessä ole perusoperaatio, että se voidaan esittää erotuksen avulla:

[Katso matemaattiset merkinnät](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/ra_popup.html)

Leikkauksen tulosrelaatio voi olla tyhjä, jos relaatioissa ei ole yhteisiä monikoita. Sen maksimikoko on pienemmän osapuolen koko.

Esimerkki:  


[Katso leikkauksen muodostaminen kuvasarjana](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/leikkaus1.html)

[Takaisin](javascript:history.back(-1))

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

### Relaatioalgebran operaatiot - Ristitulo (cross product, Cartesian product)

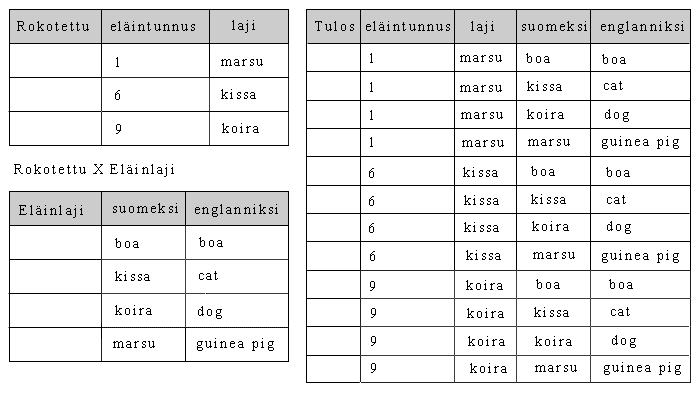
Ristitulo-operaatiota käsiteltiin jo relaatiomallin peruskäsitteiden yhteydessä.

Ristitulon R  x  S tulosrelaatio koostuu kaikista sellaisista monikoista, jotka on muodostettavissa kokoamalla yhdeksi monikoksi arvot monikkopareista, joissa parin monikoista ensimmäinen kuuluu relaatioon R ja toinen relaatioon S.

[http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/ra_3_5.gif](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/ra_popup.html)

Ristitulon tulosrelaation attribuuttien lukumäärä on operaation osapuolten attribuuttimäärien summa. Relaation monikoiden lukumäärä on osapuolten monikoiden lukumäärien tulo.

Tulosrelaation attribuutit nimetään osapuolten mukaisesti. Jos jokin attribuutti, esimerkiksi NIMI, esiintyy kummassakin osapuolessa, attribuuttinimen alkuun liitetään tulosrelaatiossa relaation nimi tarkenteeksi, esim. R.NIMI.

Esimerkki:  


[Katso ristitulon muodostaminen kuvasarjana](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/ristitulo1.html)

[Takaisin](javascript:history.back(-1))

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

### Relaatioalgebran operaatiot - Projektio (projection)

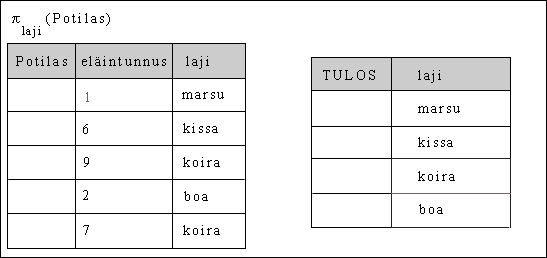
Projektion (sarakearvojen poiminnan) tulosrelaatio saadaan poimimalla annetun relaation monikoissa nimetyissä sarakkeissa esiintyvät arvoyhdistelmät tulosrelaation monikoiksi.

[Katso matemaattiset merkinnät](javascript:popUp('ra_popup.html'))

A1…An ovat sarakenimiä. x[A] tarkoittaa attribuutin A arvoa monikossa x.

Vaikka kaikki relaation monikot ovatkin keskenään erilaisia saattaa sama arvoyhdistelmä esiintyä poimittavissa sarakkeissa usealla relaation rivillä. Tulosrelaatioon tällainen yhdistelmä otetaan kuitenkin mukaan vain yhteen kertaan. Projektion tulosrelaation monikoiden määrä on täten pienempi tai yhtäsuuri kuin lähtörelaation monikoiden määrä. Jos lähtörelaation avain sisältyy poimittavien attribuuttien joukkoon on tulosrelaation rivimäärä sama kuin lähtörelaation rivimäärä.

Projektio on erityisesti relaatioalgebraa varten kehitetty operaatio.

Esimerkki:  


[>Katso projektion muodostaminen kuvasarjana](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/projektio1.html)

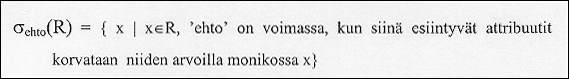
[Takaisin](javascript:history.back(-1))

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

### Relaatioalgebran operaatiot - Valinta (selection)

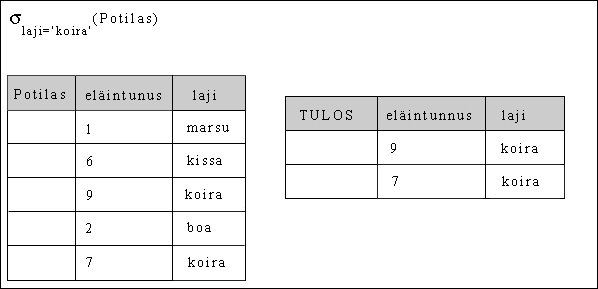
Projektiossa eristettiin relaatiosta sarakearvoja. Valinnassa eristetään tulosrelaatioon annetun valintaehdon täyttävät monikot, eli monikot joiden kohdalla valintaehdon arvo on 'tosi' (true).

[](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/ra_popup.html)

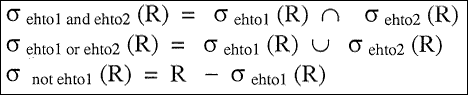
Ehdossa vertailtavina voivat olla attribuutit ja vakiot. Vertailuoperaattoreina tulevat kyseeseen http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/yhtasuuri.gif (yhtäsuuruus), http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/erisuuruus.gif (erisuuruus), http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/pienempi.gif (pienempi kuin), http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/suurempi.gif (suurempi kuin), http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/pienempi_tai.gif (pienempi tai yhtäsuuri kuin) ja http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/suurempi_tai.gif (suurempi tai yhtäsuuri kuin).

Relaatioalgebrassa vertailu on yksinkertaisten arvojen vertailua ja voidaan olettaa vertailuoperaatioiden käyttäytyvän samoin kuin ohjelmointikielissä. Tyhjäarvo on erikoistapaus, joka käyttäytyy toisin. Sen vertaaminen mihin tahansa arvoon, myös toiseen tyhjäarvoon, tuottaa tuloksen 'tuntematon' (unknown). Tuntematemattomien vertailutulosten käsittelyä varten on määritelty erityinen kolmiarvoinen logiikka. Tätä tarkastellaan kurssilla myöhemmin SQL-kyselyjen yhteydessä.

Valinnan tulosrelaatio voi olla tyhjä, jos yksikään monikko ei täytä valintaehtoa. Enimmillään valinnan tulosrelaation koko voi olla sama kuin lähtörelaation koko.

Esimerkki:  


Valintaehtona voidaan käyttää yksinkertaisen alkeisehdon asemasta myös loogista lauseketta, sillä loogiset operaatiot voidaan palauttaa aiemmin esillä olleiksi joukko-opin operaatioiksi seuraavasti:



[Katso valinnan muodostaminen kuvasarjana](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/valinta1.html)

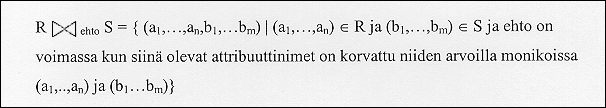
[Takaisin](javascript:history.back(-1))

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

**Relaatioalgebran operaatiot - Liitos (join)**

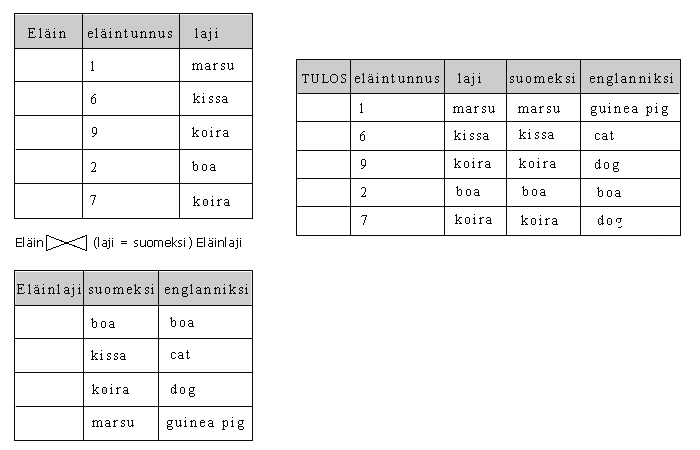
Relaatiotietokannoissa yhteen loogiseen kokonaisuuteen kuuluvia tietoja voidaan joutua esittämään useiden eri relaatioissa olevien monikoiden avulla. Tietoja tarvitaan kuitenkin usein yhdessä. Kootun monikon muodostaminen erillisistä yhteenkuuluvista palasista onkin yleisimpiä käytännössä esiintyviä tehtäviä. Relaatioalgebrassa tätä operaatiota kutsutaan liitokseksi. Liitoksessa jonkin ehdon perusteella yhteenkuuluvista monikoista muodostetaan monikkopareja. Monikkoparin alkiot kootaan sitten yhdeksi monikoksi.

[](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/ra_popup.html)

Liitos ei ole perusoperaatio siinä mielessä, että se voidaan esittää muiden relaatioalgebran operaatioiden avulla.

[Katso matemaattiset merkinnät](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/ra_popup.html)

Liitokseen liittyvässä ehdosssa (liitosehdossa) verrataan relaation R jotain attribuuttia relaation S johonkin attribuuttiin. Vertailuoperaattorina voi olla mikä tahansa valinnan yhteydessä mainittu operaattori. Yleisemmin kyseessä on yhtäsuuruusvertailu, jolloin liitosta kutsutaan yhtäsuuruusliitokseksi (equijoin).

Esimerkki:  


[Katso yhtäsuuruusliitoksen muodostaminen kuvasarjana](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/liitos1.html).

**Miksi liitos on oma operaationsa?**

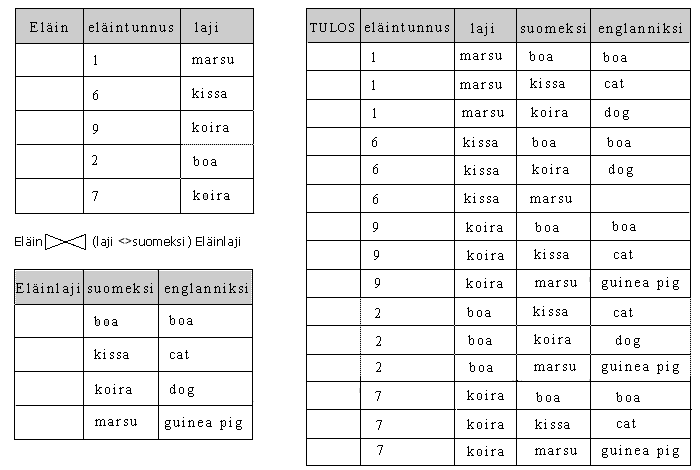
Liitos voidaan hoitaa ristitulon ja valinnan avulla. Miksi tarvitaan siis erillinen operaatio?

Tähän on löydettävissä kaksi syytä

* liitoksen yleisyys ja
* tehokkuusnäkökohdat

Hyvin suuressa osassa tietokantakyselyistä on yhdistettävä eri monikoihin kuuluvia tietoja ja tähän tarvitaan liitosta. Liitos on useimmissa tapauksissa toteutettavissa huomattavasti tehokkaammin kuin muodostamalla ensin ristitulo ja valitsemalla sieltä liitosehdon täyttävät monikot. Ajatellaan tapausta, jossa halutaan laatia luettelo opiskelijoiden suorittamien opintoviikkojen lukumääristä. Oletetaan, että käytettävissä ovat relaatiot OPISKELIJA ja SUORITUS. Olkoon OPISKELIJA relaatiossa 40 000 monikkoa ja SUORITUS relaatiossa 500 000. Ristitulon tulosrelaation koko olisi tällöin 20 000 000 000 monikkoa. Jos järjestelmä pystyisi tuottamaan 1000 monikkoa sekunnissa, niin ristitulon muodostamiseen kuluisi aikaa yli 5500 tuntia. Yhtäsuuruusliitos on kuitenkin tällaisessa tilanteessa toteutettavissa huomattavasti nopeammin, vajaassa parissa tunnissa.

Erisuurus operaation käyttöä liitosehdossa on syytä välttää. Tulos on harvoin sitä mitä halutaan. Seuraavassa esimerkki erisuuruuteen perustuvasta liitoksesta. Ajatuksena kyselyssä voisi olla yritys löytää eläinlajit, joille ei löydy lajinimen käännöstä suomesta englanniksi. Tuloksena saadaan kuitenkin kuhunkin eläinlajiin kytkettyä kaikkkien muiden lajinimien käännökset paitsi lajin itsensä.

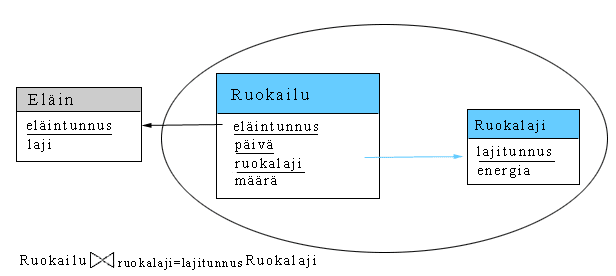
Esimerkki:  


[Katso erisuuruusliitoksen muodostuminen kuvasarjana](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/nonequijoin1.html).

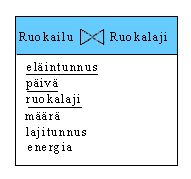
Kuten ristitulossa myös liitoksessa operaation osapuolissa saattaa olla samannimisiä attribuutteja. Muodostuvan relaation sarakenimien yksikäsitteisyys saadaan aikaan tarkentimilla. Tarkentimena käytetään relaatiokaavion nimeä, jos osapuolina ovat eri relaatiot, tai osapuolen järjestysnumeroa, jos liitos on saman relaation monikoiden välinen.

Liitosoperaation tulosrelaatio voi olla tyhjä, jos yksikään ristitulon monikkopari ei täytä liitosehtoa. Maksimissaan tulosrelaatio on yhtä suuri kuin ristitulon tulosrelaatio. Tällöin kaikkien parien on täytettävä liitosehto.

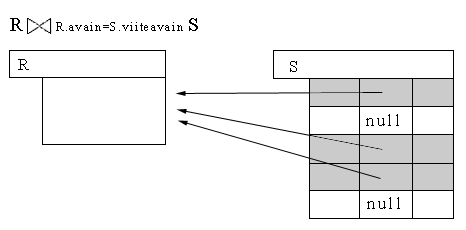
Tyypillisin liitoksen käyttötilanne on viiteavaimeen perustuva liitos. Tällaisessa liitoksessa liitosehtona on viiteavaimen ja viitatun relaation avaimen yhtäsuuruus. Tuloksena saadaan yhdistelmämonikkoja, joissa viittaavaan monikkoon on liitetty viittauksen kohteena olevan monikon tiedot. Esimerkkimme Ruokailu -relaation tapauksessa voisimme kytkeä kuhunkin Ruokailu-monikkoon tiedot ruokalajin energiamäärästä



tuottaa tuloksenaan seuraavan kaavion mukaisia monikoita



Tässä tapauksessa tulosrelaation monikoiden määrä on sama kuin relaation Ruokailu monikoiden määrä. Tämä siitä syystä, että attribuutti*'ruokalaji'* sisältyy relaation Ruokailu avaimeen eikä siten voi sisältää tyhjäarvoja. Koska ruokalaji on viiteavain sen jokaiselle arvolle löytyy pari relaatiosta Ruokalaji. Tällaisia voi löytyä vain yksi kutakin ruokailu-monikkoa kohti. Yleisesti viiteavaimeen perustuvan liitoksen tulosrelaation koko on sama kuin viiteavaimen sisältävän relaation sellaisten monikoiden määrä, joissa viiteavaimen arvo on jokin muu kuin tyhjäarvo.



Tuloksen koko on niiden S:n monikoiden määrä, joissa viiteavain-attribuutilla ei ole tyhjäarvoa.

[Takaisin](javascript:history.back(-1))

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

### Relaatioalgebran operaatiot - Ulkoliitos (outer join)

Liitoksella

Ruokailu http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/rusetti2.gif ruokalaji=lajitunnus Ruokalaji

saadaan tietoja eri ruokalajien käytöstä. Kaikista nautituista ruokajajeista saadaan tiedot. Kuitenkin voi olla olemassa myös sellaisia ruokalajeja, joita ei ole lainkaan nautittu. Tällaisille ei löydy liitosehdon täyttävää paria vaan ne jäävät parittomiksi eivätkä siten näy lainkaan edellisen liitoksen tulosrelaatiossa.

Parittomiksi jäävät ruokalajit voidaan selvittää kyselyllä

1.lajitunnus, energia (  
     Ruokalaji http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/rusetti2.gif1.lajitunnus=2.lajitunnus  
          (lajitunnusRuokalaji - ruokalajiRuokailu))

Jos halutaan tuottaa raportti eri ruokalajien käytöstä, tarvitaan raporttiin tiedot sekä käytetyistä ruokalajeista että käyttämättömistä. Tällaista tarkoitusta varten on olemassa ulkoliitos-operaatio. Sen ytimenä on tavallinen liitos-operaatio, jonka tulokseen lisätään yhdisteellä parittomaksi jääneet monikot tyhjäarvoin täydennettyinä. Esimerkin tapauksessa tämä voidaan esittää vaikkapa ns. vasemmanpuoleisena ulkoliitoksena (left outer join)

Ruokalaji http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/ulkoliitos_vas.gif lajitunnus=ruokalaji Ruokailu

Normaalin yhtäsuuruusliitoksen tulokseen lisätään parittomiksi jääneet Ruokalaji-monikot täydennettyinä siten, että yhdistemonikon Ruokailu-attribuuttien arvoina on tyhjäarvot. eli vasemmanpuoleisesta operandista otetaan mukaan kaikki monikot vaikkei niille löytyisikään paria. Vastaavasti oikeanpuoleisessa ulkoliitoksessa (merkitään R http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/ulkoliitos_oik.gif S) oikeanpuoleisesta operandista otetaan tulokseen kaikki monikot. Molemminpuoleisessa ulkoliitoksessa (merkitään R http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/ulkoliitos_mol.gif S) kummankin osapuolen parittomat otetaan mukaan tyhjäarvoparin kanssa.

[Takaisin](javascript:history.back(-1))

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

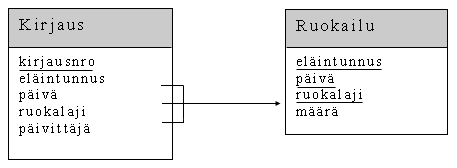
TIETOKANTOJEN PERUSTEET

### Relaatioalgebran operaatiot - Luonnollinen liitos (natural join)

Yhtäsuuruusliitoksessa liitosehtona on kahden attribuutin arvojen yhtäsuuruus. Tällaisen liitoksen tuloksena syntyvissä monikoissa sama arvo esiintyy kahdessa paikassa. ja relaatiota taulukkona tarkasteltaessa siinä on kaksi samasisältöistä saraketta. Relaation tietosisällön kannalta toinen näistä on tarpeeton. Luonnollisessa liitoksessa tällaiset toistuvat sarakkeet poistetaan. Luonnollisessa liitoksessa ei myöskään tarvitse eksplisiittisesti kirjoittaa liitosehtoa vaan se määräytyy automaattisesti attribuuttinimien perusteella. Jos sama attribuuttinimi esiintyy kummassakin liitettävässä relaatiossa, otetaan liitosehtoon vaatimus siitä, että näillä on sama arvo liitettävissä monikoissa

R\*S=  
{R.a1,...,Ram,R.b1,...,R.bk,S.c1,...,S.cn (Rhttp://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/rusetti2.gifR.b1=S.b1 and ... and R.bk=S.bk S))  
  
kun attribuutit a1,...,am ovat sellaisia R:n attribuutteja, jotka eivät esiinny S:ssä,  ja c1,...,cn sellaisia S:n attribuutteja, jotka eivät esiinny R:ssä ja  b1,...,bk attribuutteja, jotka esiintyvät sekä R:ssä että S:ssä.

Esimerkiksi relaatioiden Kirjaus ja Ruokailu



luonnollinen liitos Kirjaus\*Ruokailu tuottaa kaavion

tulos(kirjausnro, eläintunnus, päivä, ruokalaji, päivittäjä, määrä)

mukaisen relaation ja vastaa operaatiota

Kirjaus.kirjausnro, Kirjaus.eläintunnus, Kirjaus.päivä, Kirjaus.ruokalaji, Kirjaus.päivittäjä, Ruokailu.määrä

(Kirjaus http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/rusetti2.gif Kirjaus.eläintunnus=Ruokailu.eläintunnus and  
     Kirjaus.päivä=Ruokailu.päivä and  
     Kirjaus.ruokalaji=Ruokailu.ruokalaji  
Ruokailu)

Luonnollinen liitos on varsin kätevä operaatio, mutta edellyttää että samannimiset attribuutit ovat vertailukelpoisia myös merkitykseltään. Esimerkiksi tapauksessa

Kirjailija(tunnus, nimi, osoite,…)  
Teos(tunnus, nimi, ...)

relaatioiden Kirjailija ja Teos luonnollinen liitos ei suinkaan kytke teokseen sen tekijää vaan tuottaa tuloksenaan tyhjän tai merkitykseltään epämääräisen relaation, sillä liitosehtona vaaditaan, että kirjailijan tunnus on sama kuin teoksen tunnus ja kirjailijan nimi on sama kuin teoksen nimi.

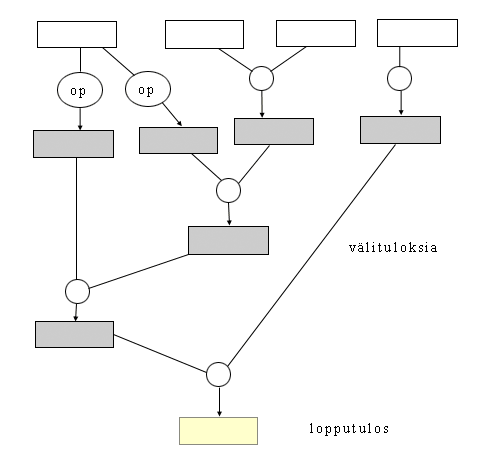
[Takaisin](javascript:history.back(-1))

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

**Kyselyn laatiminen relaatioalgebralla**

Kyselyn laatiminen lähtee tietotarpeesta. Tehtävänä on löytää operaatiot, joilla tietokannan relaatioista saadaan muodostettua tietotarpeen tyydyttävä vastausrelaatio. Koska relaatioalgebran perusoperaatiot ovat hyvin yksinkertaisia joudutaan vastausrelaation muodostamiseksi tuottamaan välituloksia ja käyttämään näitä sitten operaatioiden osapuolina tuottamaan uusia välituloksia. Kyselyn koostuminen operaatioista ja välituloksista voidaan esittää myös suoritusrakenteena alla olevan kuvan tapaan. Kuvassa suorakaiteet ovat relaatioita (joko alkuperäisiä tai tulosrelaatioita) ja ympyrät operaatioita. Ympyrän sisällä on operaation nimi ja viereen voidaan kirjoittaa ehto tai muu operaatiossa tarvittava tarkennos.



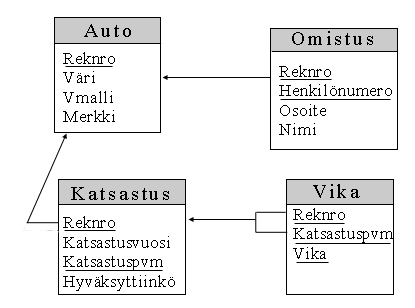
Useimpien relaatioalgebran operaatioiden kohdalla suoritusjärjestyksellä ei ole lopputuloksen kannalta merkitystä, joten sama tulos voidaan saada eri tavoin välituloksia muodostamalla ja yhdistelemällä. Käytännön tietokannoissa kyselyn suoritus tapahtuu  pitkälti relaatioalgebran operaatioiden kaltaisten operaatioiden avulla. Operaatioiden suoritusjärjestyksen ja tarvittavat välitulokset päättää tällöin kyselyn optimoija (query optimizer). Suoritusjärjestyksellä ja tarvittavilla välituloksilla on merkitystä kyselyn tehokkuuden kannalta. Kyselyn optimoijat pyrkivät järjestämään operaatiot siten, että kyselyn kokonaissuoritus olisi mahdollisimman tehokas. Käytännössä tämä yleensä tarkoittaa sitä, että pyritään mahdollisimman nopeasti pienentämään välituloksia.

Kyselyn muodostuksessa on

* ensiksi valittava, mitä relaatioita tarvitaan vastauksen aikaansaamiseen ja
* seuraavaksi määriteltävä tarvittavat operaatiot ja välitulokset

Tarkastellaan esimerkkitietokantaa

Auto(Reknro, Väri, Vmalli, Merkki)  
Omistus(Reknro -> Auto, Henkilönumero, Osoite, Nimi),  
Katsastus(Reknro -> Auto, Katsastusvuosi, KatsastusPvm, Hyväksyttiinkö),  
Vika((Reknro, KatsastusPvm)-> Katsastus, Vika)



**Esimerkki 1:**

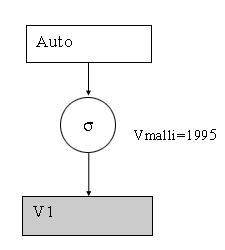
*Tietotarve: Tiedot vuosimallia 1995 olevista autoista, joista on löytynyt vikoja vuoden 1999 katsastuksissa*

Päättelemme, että tulokseen tarvitaan kaikki autoon liittyvät perustiedot. Nämä löytyvät relaatiosta Auto. Vastauksen sisältö muodostuu siis vain tämän relaation sisältämistä tiedoista. Tämän relaation tietosisällön perusteella tulokseen otettavia monikoita voidaan karsia attribuutiin Vmalli perusteella

Välitulos

**V1=select** Vmalli=1995 **(Auto)**

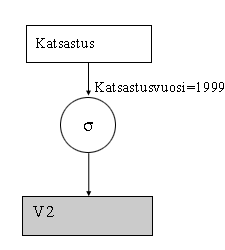
sisältää tiedot vuosimallia 1995 olevista autoista.



Tietotarpeen kohteena olevat autot on pitänyt katsastaa vuodelle 1999. Tiedot vuodelle1999 katsastetuista autoista löytyvät relaatiosta Katsastus. Välitulos

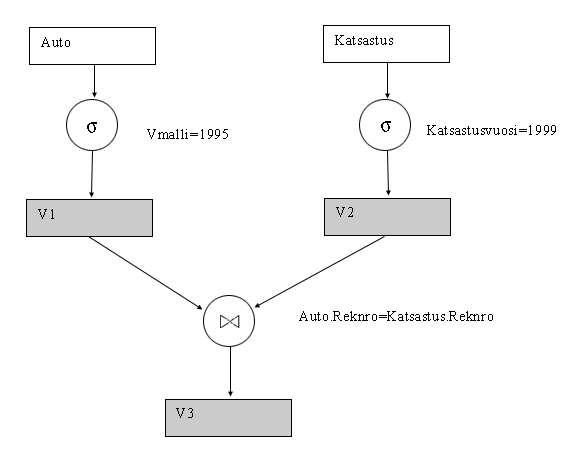
**V2=**select Katsastusvuosi=1999 **(Katsastus)**

antaa kaikki vuoden 1999 katsastuksiin liittyvät monikot.



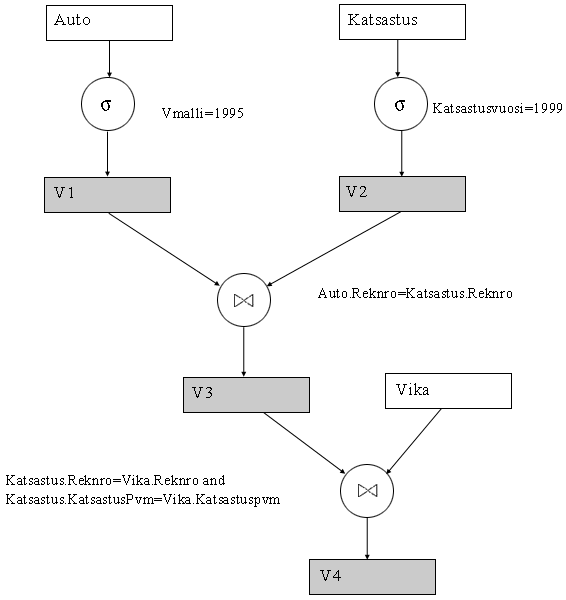
Katsastukset ja autot saadaan kytkettyä rekisterinumeroon perustuvan liitoksen avulla. Tehdään liitos määriteltyjen välitulosten V1 ja V2 välille. Näin saadaan uusi välitulos:

**V3=**select Vmalli=1995 **(Auto)**   
     join Auto.Reknro=Katsastus.Reknro **select**Katsastusvuosi=1999 **(Katsastus)**



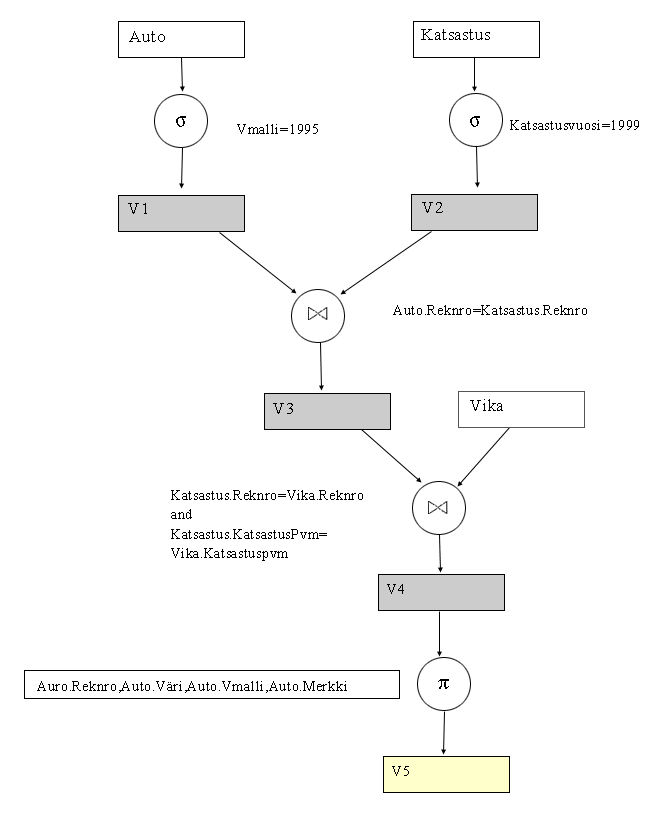
Yllä olevassa välituloksessa V3 ovat mukana vain ne vuosimallin 1995 autot, joille on tehty vuoden 1999 katsastus, eli liitos on edelleen karsinut Auto-monikoita. Liitos-operaation karsintavaikutusta voidaan käyttää karsimaan myös sellaiset autot, joista ei löytynyt vikoja. Tehdään siis viiteavaimeen perustuva liitos väli-tuloksen V3 ja relaation Vika välille

V4= (select Vmalli=1995 (Auto)  
     http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/rusetti2.gifAuto.Reknro=Katsastus.Reknro **select** Katsastusvuosi=1999 **(Katsastus))**  
     http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/rusetti2.gifKatsastus.Reknro=Vika.Reknro and Katsastus.KatsastusPvm=Vika.KatsastusPvm  
     **Vika**



Koska viattomiiin autoihin ei liity yhtään monikkoa Vika relaatiossa, ei niille löydy paria liitoksessa, ja jäljelle jäävät vain vialliset. Välituloksen V4 monikoissa on kuitenkin mukana paitsi autojen myös sekä relaation Katsastus että relaation Vika tiedot. Sama auto esiintyy tuloksessa niin monta kertaa kuin siitä on kirjattu vikoja. Projisoimalla pelkästään Auto-tiedot saadaan haluttu vastaus.

**V5= project**Auto.Reknro, Auto.Väri, Auto.Vmalli, Auto.Merkki  
**((selectVmalli=1995 (Auto)**   
     http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/rusetti2.gifAuto.Reknro=Katsastus.Reknro select Katsastusvuosi=1999 **(Katsastus))**  
     http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/rusetti2.gifKatsastus.Reknro=Vika.Reknro and Katsastus.KatsastusPvm=Vika.KatsastusPvm  
**Vika**)

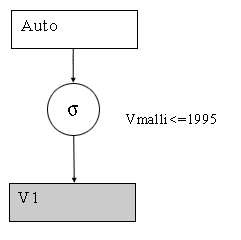


Dataa tulokseen tulee  varsinaisesti relaatiosta Auto, mutta relaatiot Katsastus sekä Vika ovat mukana karsimassa valittavia monikoita.

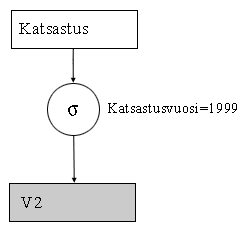
Esimerkki 2:

*Tietotarve: Omistajan nimi ja osoite vuosimallia 1995 tai sitä vanhemmista autoista, joiden vuoden 1999 katsastusta ei ole vielä tehty*

Tässä kyselyssä lähtökohtana on relaatio Auto. Halutaan tietoja tietyn vuosimallin autoista. Kiinnostavat vuosimallia 1995 tai sitä vanhemmat autot saadaan selville kyselyllä  
**V1= select** Vmalli <= 1995 **(Auto)**

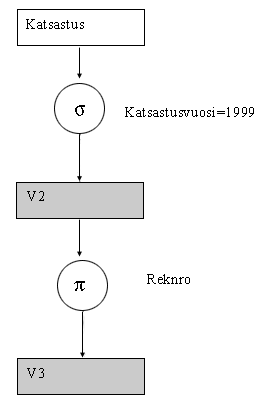


Vuoden 1999 katsastukset saadaan selville kyselyllä  
**V2= select** Katsastusvuosi=1999 **(Katsastus)**



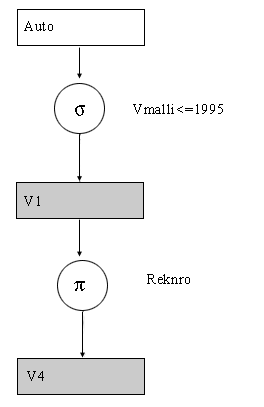
Katsastamattomien autojen selville saaminen onkin jo hankalampaa. Katsastamaton tarkoittaa sitä, että autoon liittyvää monikkoa ei ole relaatiossa katsastus. Autot, joille on tehty vuoden 1999 katsastus saadaan selville eristämällä rekisterinumerot välituloksesta V2.

**V3= project**Reknro **(select** Katsastusvuosi=1999 **(Katsastus) )**



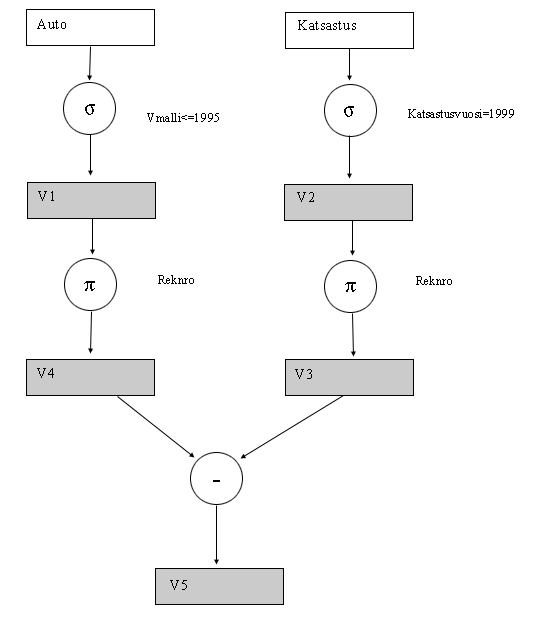
Katsastamattomat autot ovat katsastettujen autojen vastakohta (komplementti), eli kun kaikkien autojen joukosta poistetaan katsastetut autot, niin  jäljelle jäävät katsastamattomat. Jotta tätä pystyttäisiin käyttämään hyväksi tarvitaan kaikkien vuoden 1995 ja sitä vanhempien autojen rekisterinumerot. Nämä saadaan eristämällä rekisterinumero välituloksesta V1

**V4= project**Reknro **(V1) = project**Reknro (**select** Vmalli <= 1995 **(Auto))**



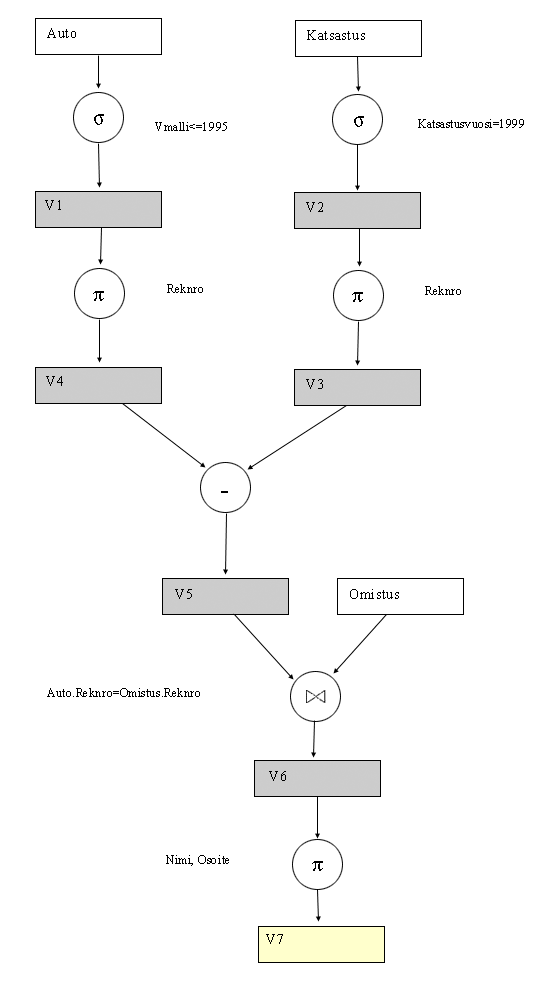
Vuodelle 1999 katsastamattomien autojen rekisterinumerot saadaan nyt muodostamalla välitulosten V4 ja V3 erotus.

**V5= project**Reknro (**select** Vmalli <= 1995 **(Auto)) - project**Reknro **(select** Katsastusvuosi=1999 **(Katsastus))**



Autojen omistajat saadaan selville Omistaja relaation ja välituloksen V5 liitoksella. Tästä poimitaan projektiolla nimi ja osoite, jolloin kyselyn lopullinen muoto on

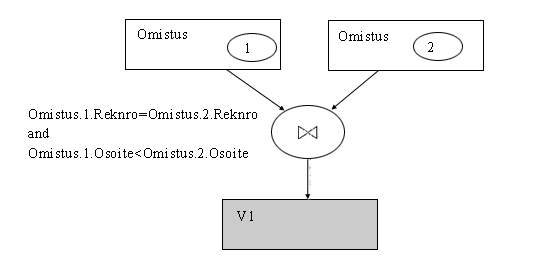
V6= **project**Nimi, Osoite (  
( **project**Reknro (select Vmalli <= 1995 **(Auto)) - project**Reknro (select Katsastusvuosi=1999 **(Katsastus)) )**  
     http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/rusetti2.gifAuto.Reknro=Omistus.Reknro **Omistus)**



**Esimerkki 3**

*Nimet ja osoitteet henkilöpareista, jotka omistavat yhteisen auton, mutta asuvat eri osoitteissa.*

Henkilöparin muodostamiseksi on paritettava omistus monikoita. Paritus tehdään liitos-operaatiolla. Liitosehtona on yhteisen auton omistus, eli parin muodostavissa omistus-monikoissa täytyy olla sama rekisterinumero. Edelleen vaadittiin, että parin jäsenillä on eri osoite.



Kyselyssä on osoitteiden erilaisuutta testattu 'pienempi kuin' -operaatiolla. Tämä varmistaa samalla sen, että sama pari ei tule kahdessa eri järjestyksessä, kuten kävisi,  jos vertailuoperaationa olisi erisuuruus.

Tästä on vielä karsittava nimet ja osoitteet

V2(Nimi1, Osoite1, Nimi2, Osoite2) =  
projectOmistus{1}.Nimi, Omistus{1}.Osoite, Omistus{2}.Nimi, Omistus{2}.Osoite (  
Omistus{1} http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/kuvat/rusetti2.gif  
          Omistus{1}.Reknro=Omistus{2.}Reknro and  
          Omistus{1}.Osoite<Omistus{2}.Osoite   
Omistus{2})

Jos autolla AAA-111 on kolme omistajaa Liisa, Pekka ja Kalle, jotka kaikki asuvat eri osoitteissa saadaan tulokseksi tähän autoon liittyen kolme monikkoa eli henkilöpareja (Liisa, Pekka), (Liisa, Kalle) ja (Pekka, Kalle) vastaavat monikot.

**Esimerkki 4**

*Autot, joissa on esiintynyt vähintään yksi sama vika sekä vuoden 1999 että vuoden 2000 katsastuksessa. Mikä tämä vika on?*

Tulosta rajaavia katsastus-monikoita **ei ole** syytä hakea lausekkeella

selectkatsastusvuosi=1999 and katsastusvuosi=2000(Katsastus)

sillä tämä lauseke tuottaa tyhjän tulosrelaation, koska paikassa katsastusvuosi ei missään monikossa voi olla kuin yksi arvo.

Valitaan siis kummankin vuoden katsastukset erikseen  
V1= selectkatsastusvuosi=1999(Katsastus) ja  
V2= selectkatsastusvuosi=2000(Katsastus)

Viat on kytkettävissä katsastuksiin viiteavaimeen perustuvalla liitoksella

V3= Vika \* V1 = Vika \* selectkatsastusvuosi=1999(Katsastus)  
V4= Vika \* V2 = Vika \* selectkatsastusvuosi=2000(Katsastus)

Näistä voidaan eristää rekisterinumero ja vikakuvaus.

V5 = projectReknro, vika(V3)   
      = projectReknro, vika(Vika \* selectkatsastusvuosi=1999(Katsastus))

V6 = projectReknro, vika(V4)    = project>Reknro, vika(Vika \* selectkatsastusvuosi=2000(Katsastus)

Yhteiset viat löytyvät nyt leikkausjoukkona

V7 = V5  V6   
   = projectReknro, vika(Vika \* select katsastusvuosi=1999(Katsastus)    
     projectReknro, vika(Vika \* select katsastusvuosi=2000(Katsastus)

Vaikka sama relaatiokaavio esiintyykin useasti, ei tässä tarvita esiintymänumeroita, vaan lauseke on yksiselitteisesti tulkittavissa.

[Takaisin](javascript:history.back(-1))

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

# Relaatiotietokannan suunnittelu

|  |  |  |
| --- | --- | --- |
| [**Oheismateriaali:**](http://www.cs.helsinki.fi/u/laine/tkp/lahteita.html) | |  | | --- | | Ramakrishnan&Gehrke: luku 3.5, sivut 75-86.  Elmasri&Navathe: luku 7, sivut 191-206. | |

Relaatiotietokannan suunnittelussa lähdetään liikkeelle siitä, että tietokannan tietosisältö on kartoitettu. Jos kartoitus on tehty käsitetason tietomalliksi, tehdään suunnittelussa muunnos käsitetason mallista relaatiomalliin. Muunnos voidaan tehdä varsin suoraviivaisesti, jopa algoritmisesti. Suoraviivaiset muunnokset eivät kuitenkaan ole aina käyttökelpoisia ja joissain tilanteissa voi olla tarpeen miettiä muitakin ratkaisuja.

## Relaatioiden johtaminen ER-kaaviosta

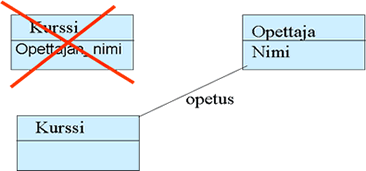
### Tietosisältömallin laatuvaatimukset

Jotta relaatiot pystytään suunnittelemaan tietosisältömallin pohjalta, täytyy tietosisältömallin täyttää joitain laatuvaatimuksia:

* Mallin on oltava yksityiskohtainen eli tietosisältöanalyysi on viety niin pitkälle, että kaikki yksityiskohdat ovat mukana ja kaikki tietoihin liittyvät säännöt on kartoitettu. Tietoryhmät ja muut kokonaisuudet, joita saattaa esiintyä karkean tason tietosisältöanalyysissä, on pilkottu osiin.
* Tietokohteiden väliset yhteydet ovat näkyvissä eikä yhteyksiä ole piilotettu attribuuteiksi. Kuvassa 1 tietokohteiden välinen yhteys on piilotettu attribuutiksi. Kuvassa 2 se on mallinnettu näkyvänä.

piilotettu yhteys opettajan nimi attribuutin kautta

Kuva 1: Yhteys piilotettu opettajan\_nimi attribuuttiin



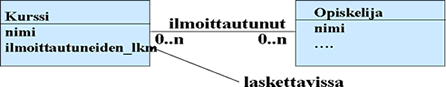
Kuva 2: Yhteys mallinnettu näkyvästi

* Kukin asia on mallinnettu vain yhdellä tavalla. Kuvassa 3 on tilauksen ja asiakkaan välinen yhteys kuvattu sekä yhteytenä että attribuuttina. Edellisen kohdan perusteella attribuutti on turha.



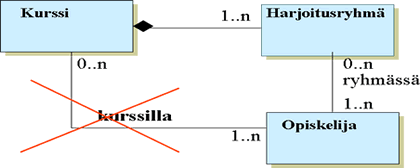
Kuva 3: Asia mallinnettu kahteen kertaan

* Johdettavissa tai pääteltävissä olevat tiedot on karsittu tai merkitty pääteltävissä oleviksi. Attribuutin arvo voi olla pääteltävissä samoin myös yhteyden olemassaolo. Kuvassa 4 ilmoittautuneiden lukumäärä on laskettavissa. Sen karsimista voisi harkita. Toisaalta sen ottaminen mukaan relaatioihin voi nopeuttaa tiedonhakua merkittävästi.



Kuva 4: Laskettavissa oleva attribuutti

Kuvassa 5 kurssin ja opiskelijan välinen yhteys on pääteltävissä ja kannattaisi karsia.

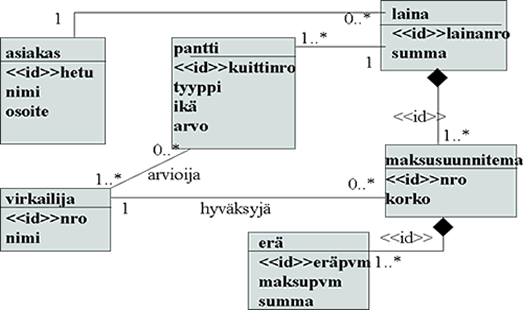


Kuva 5: Pääteltävissä oleva yhteys

* Yhteydet on määritelty oikeiden osapuolten välille. Tämän toteaminen edellyttää useamman yhteyden analysointia rinnakkain ja voi olla vaikeasti havaittavissa. Saattaa olla, että virheellinen yhteys havaitaan vasta muunnoksen jälkeen.

## Muunnos ER-mallista relaatiomalliin

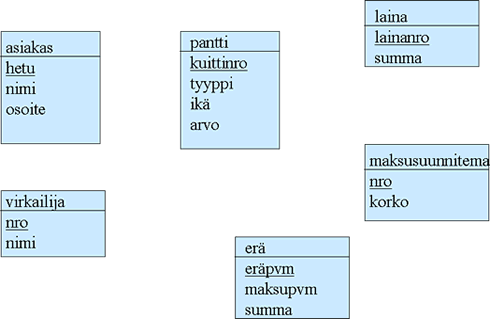
Seuraavassa annetetaan joukko suoraviivaisia muunnossääntöjä. Esimerkkinä sääntöjen toiminnasta käytetään kuvan 6 panttilainaamosovellusta.



Kuva 6: Panttilainaamokannan tietosisältö

### Muunnossäännöt

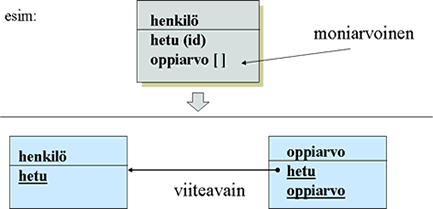
1. Kutakin tietokohdetyyppiä vastaa samanniminen taulu
2. Kullakin tietokohteen yksiarvoisella attribuutilla on samanniminen vastinsarake tietokohdetta vastaavassa taulussa. Tunnistavia attribuutteja vastaavat sarakkeet sisältyvät taulun pääavaimeen.



Kuva 7: Panttilainaamokannan taulut sääntöjen 1 ja 2 soveltamisen jälkeen

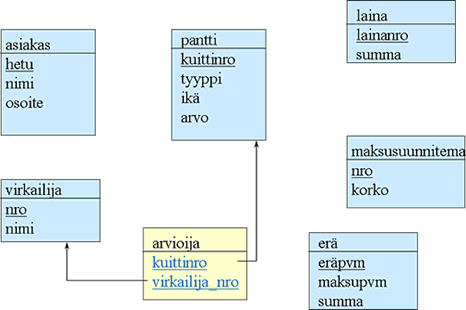
1. Kutakin moniarvoista attribuuttia vastaa taulu, jonka sarakkeina ovat viiteavain attribuutin sisältävää tietokohdetta vastaavaan tauluun ja sarake attribuutin arvoa varten. Taulun kaikki sarakkeet kuuluvat taulun pääavaimeen.

Panttilainaamoesimerkissä ei ole moniarvoisia attribuutteja.



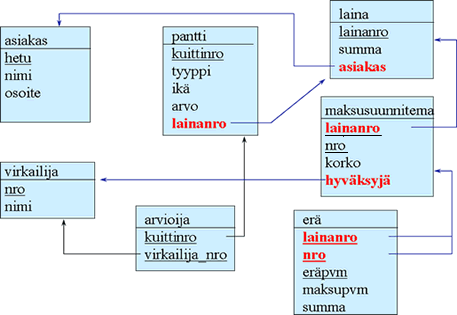
Kuva 8: Moniarvoisen attribuutin muunnos

1. Kutakin 'monen suhde moneen' -tyyppistä yhteyttä vastaa taulu, jonka nimenä on yhteyden nimi ja jonka sarakkeina ovat yhteyden osapuolia vastaaviin tauluihin osoittavat viiteavaimet. Taulun kaikki sarakkeet kuuluvat taulun pääavaimeen.



Kuva 9: Panttilainaamon kanta kun moneen suhde moneen -yhteydet on muunnettu

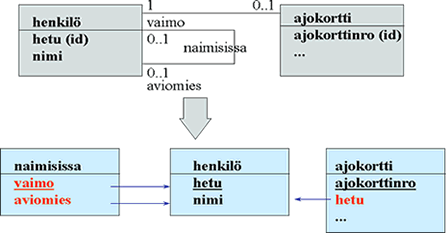
1. 'Yhden suhde moneen' -tyyppiset yhteydet toteutetaan ensisijaisesti lisäämällä yhteyskumppaniin osoittava viiteavain sitä osapuolta vastaavaan tauluun, jolla on korkeintaan yksi kumppani. Jos kyseinen osapuoli on alisteinen ja yhteyden kautta tunnistettava, niin lisätty viiteavain kuuluu taulun avaimeen.



Kuva 10: Yhden suhde moneen -yhteydet muunnettu

1. 'Yhden suhde moneen' -tyyppiset yhteydet voidaan toteuttaa myös omina tauluinaan säännön 4 mukaisesti.
2. 'Yhden suhde yhteen' -tyyppisille yhteyksille on vaihtoehtoisia toteutustapoja.
   * Jos yhteys on molemmille osapuolille valinnainen, sille kannattaa perustaa oma taulu säännön 4 mukaisesti.
   * Jos yhteys on toiselle osapuolelle pakollinen ja toiselle valinnainen, lisätään yhteyden toteuttava viiteavain sen osapuolen vastintauluun, jolle yhteys on pakollinen.
   * Jos yhteys on molemmille osapuolelle pakollinen, lisätään viiteavain toisen osapuolen tauluun. Myös taulujen yhdistämistä voi harkita.

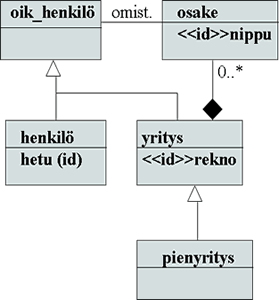
Panttilainaamoesimerkissä ei ole yhden suhde yhteen yhteyksiä.



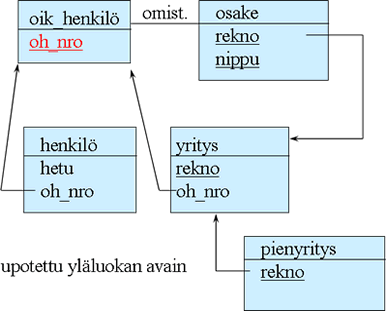
Kuva 11: Yhden suhde yhteen -yhteyksien muunnosesimerkki

1. Sisältyvyyssuhteen toteuttamiseen on useita vaihtoehtoja:
   * Alityypin tauluun voidaan lisätä ylityyppiä vastaavaan tauluun osoittava viiteavain.
   * Alityypille ei toteuteta omaa taulua vaan tähän tauluun muiden sääntöjen perusteella tulevat sarakkeet lisätäänkin ylityypin tauluun.

Esimerkkinä sisältyvyyssuhteiden muunnoksesta tarkastellaan osakkeiden omistusta:



Kuva 12: Osakkeiden omistukseen liittyvää tietosisältöä.



Kuva 13: Muunnettu yleistyshierarkia (muunnoksessa käytetty keinoavainta)

1. Edellisissä säännöissä tauluille on muodostunut luonnolliset pääavaimet tunnistavien attribuuttien ja yhteyksien perusteella. Jos pääavain muodostuu kovin moniosaiseksi ja sitä käytetään runsaasti viittauksissa, voidaan harkita keinoavaimen (surrogate) käyttöä. Keinoavaimet säästävät tilaa ja pysyvät muuttumattomina mutta hidastavat monia tietokantaoperaatioita, joten niiden käyttöönottoa on syytä tarkoin harkita.

Jos lähtökohtana käytettävä käsitetason tietomalli täyttää laatuvaatimukset, tuottavat yllä kuvatut muunnossäännöt toimivan relaatiotietokantarakenteen. Relaatiotietokannoille on määritelty myös tietojen välisiin riippuvuuksiin ja ns. normaalimuotoihin pohjautuva suunnittelutekniikka. Sen periaatteita voidaan käyttää muunnoksen tuloksena syntyneiden taulujen laadun varmistamiseen. Erityisesti kannattaa tarkastella tauluja, joihin muunnos on tuonut useita viiteavaimia.

[Harri Laine](mailto:harri.laine@helsinki.fi): *Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

# Relaatiotietokannan suunnittelu riippuvuuksiin perustuen

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sisältö:** | |  | | --- | | [Eroon toistuvasta tiedosta](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/suunnittelu.html#yleista) | | [Taulurakenteiden suunnittelu](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/suunnittelu.html#Taulurakenteidensuunnittelu) | | [Funktionaaliset riippuvuudet](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/suunnittelu.html#funktionaalisetriippuvuudet) | | [Avaimen määrääminen](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/suunnittelu.html#avain) | | [Riippuvuuksien päättely](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/suunnittelu.html#Riippuvuuksienpäättely) | | [Boyce-Codd normaalimuoto](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/suunnittelu.html#Boyce-Codd) | | [Attribuuttien uudelleenryhmitys Boyce-Codd normaalimuodossa oleviksi kaavioiksi](http://www.cs.helsinki.fi/u/laine/tkp/relaatiomalli/suunnittelu.html#bc-muotoon) | |
| [**Oheismateriaali:**](http://www.cs.helsinki.fi/u/laine/tkp/lahteita.html) | |  | | --- | | Laine: Luku 7 (sivut 134-140)  Ramakrishnan&Gehrke: luvut 19.1-19.4, sivut 605-615  Elmasri&Navathe: luku 10, sivut 293-331. | |

## Eroon toistuvasta tiedosta

Relaatiotietokantojen suunnittelun pääperiaatteena on **tiedon toiston välttäminen**. Kukin tieto pitäisi tallentaa vain kertaalleen. Tiedon toistumisesta aiheutuvia ongelmia ovat

* toistuva tieto vie tarpeettomasti tilaa
* ylläpito muodostuu hankalaksi ja
* ylläpito-operaatioilla voi olla odottamattomia sivuvaikutuksia.

Tarkastellaan pientä esimerkkitaulua

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Työ | ttNro | nimi | s\_aika | oNro | oNimi | osSijainti |
| 10 | M.Seppä | 1.3.59 | 3 | myynti | Helsinki |
| 20 | D.Leivo | 4.5.40 | 3 | myynti | Helsinki |
| 30 | S.Koivu | 8.6.66 | 4 | hallinto | Espoo |
| 40 | B.Oja | 2.4.65 | 4 | hallinto | Espoo |
| 50 | O.Itä | 1.2.55 | 6 | tuotanto | Espoo |

Tässä taulussa osastojen 3 ja 4 nimi sekä sijaintipaikka on tallennettu kahteen kertaan. Jos osasto3 muuttaisi Helsingistä Espooseen, pitäisi muutos tehdä kahdelle taulun riville eli ylläpito muodostuu työläämmäksi. Tietenkin näin pienessä taulussa toiston tuomat ongelmat ovat varsin vähäisiä, mutta suuremmissa ne vastaavasti moninkertaistuvat. Hankalampi ongelma on se, että jos työntekijä O.Itä päättäisi lähteä yrityksestä ja hänen rivinsä poistettaisiin, poistuisi samalla tieto siitä, että tuotanto-osastokin on olemassa. Uutta osastoa ei voida myöskään lisätä, ellei sinne heti sijoiteta jotain työntekijää.

Rakenteeltaan parempaan tietokantaan päästään suunnittelemalla taulurakenne paremmin tilanteeseen sopivaksi. Yllä olevan esimerkin tapauksessa toistosta päästään eroon seuraavilla tauluilla

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Työntekijä | ttNro | nimi | s\_aika | oNro |
| 10 | M.Seppä | 1.3.59 | 3 |
| 20 | D.Leivo | 4.5.40 | 3 |
| 30 | S.Koivu | 8.6.66 | 4 |
| 40 | B.Oja | 2.4.65 | 4 |
| 50 | O.Itä | 1.2.55 | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| Osasto | oNro | oNimi | osSijainti |
| 3 | myynti | Helsinki |
| 4 | hallinto | Espoo |
| 6 | tuotanto | Espoo |

Nyt osastoja voi lisätä ilman, että niihin sijoitetaan työntekijää. Työntekijän poisto ei hävitä osastoa missään tilanteessa ja osaston muuttaessa riittää tehdä muutos yhdelle riville.

Tiedon toiston välttäminen on hyvä asia tietokannan tilantarpeen ja ylläpidon kannalta. Kyselyjen tehokkuutta toisteinen tieto voi kuitenkin parantaa.

## Taulurakenteiden suunnittelu

Tällä kurssilla käsitellään vain taulurakenteiden suunnittelua ja sitäkin johdantoluonteisesti siten, että vain pieni mutta keskeinen osa relaatiomalliin liittyvästä suunnitteluteoriasta tulee katetuksi.

Relaatiotietokantojen suunnitteluteoriassa relaatiokaavion hyvyys määritellään tietojen välisiin riippuvuuksiin perustuen. Riippuvuustyyppejä on määritelty useita, mutta tärkeimpiä niistä ovat **funktionaaliset riippuvuudet** (functional dependency).

### Funktionaaliset riippuvuudet

Funktionaalisella riippuvuudella tarkoitetaan sitä, että **sarakkeen arvon avulla pystytään selvittämään yksikäsitteinen arvo sarakkeesta funtionaalisesti riippuvalle sarakkeelle**. Esimerkiksi henkilötunnuksen perusteella saadaan selville yksikäsitteinen nimi, syntymäaika, osoite, yms. Sama muodollisemmin:

|  |
| --- |
| Attribuutti B on funktionaalisesti riippuva attribuutista A (A määrää funktionaalisesti B:n, merkitään A->B), jos ja vain jos kaikissa relaatiokaavion R ilmentymissä kuvaus A:n arvojoukolta B:n arvojoukolle on funktionaalinen.  Kuvaus A:n arvojoukolta B:n arvojoukolle on funktionaalinen, jos jokainen A:n arvo kuvautuu yhdelle B:n arvolle eli, jos riveillä r ja s attribuutilla A on sama arvo (r.A=s.A), niin näillä riveillä täytyy myös B-attribuuteilla olla keskenään sama arvo (r.B=s.B). |

Riippuvuudessa A->B kutsutaan attribuuttia A määrääjäksi. A voi olla yksittäinen attribuutti tai kokoelma attribuutteja. Kun riippuvuuksia määritellään, pyritään määrääjiksi löytämään minimaaliset attribuuttiyhdistelmät. Määrääjä on minimaalinen, jos siinä ei ole turhia attribuutteja. Esimerkiksi riippuvuudessa  Ax->B attribuutti x on turha, mikäli  A->B. Relaatiokaavioon liittyvät riippuvuudet voidaan aina esittää siten, että kussakin riippuvuudessa oikealla puolella, siis riippuvana, on vain yksi attribuutti.

### Avaimen määrääminen

Relaation avain määritellään riippuvuuksien avulla seuraavasti

|  |
| --- |
| Sarakeyhdistelmä A on relaatiokaavion R **avain**, jos se määrää funktionaalisesti kaikki relaatiokaavion attribuutit eikä ole olemassa sellaista A:n osajoukkoa, jolla olisi tämä sama ominaisuus. |

Funtionaalisten riippuvuuksien perusteella voidaan täten selvittää, mitkä sarakkeet muodostavat relaation avaimen. Avaimen saa selville seuraavalla algoritmilla:

Ota avainehdokkaaksi K kaikki attribuutit, jotka eivät esiinny riippuvuuksissa oikealla puolella eivätkä ole siis riippuvia muista. Kunnes K määrää funktionaalisesti kaikki kaavion attribuutit, yhdistä K:hon määrääjä sellaisesta riippuvuudesta, jossa määräytyvä attribuutti ei kuulu K:hon. Kun attribuuttijoukkoa K:ta ei enää tarvitse täydentää, on avain löytynyt.

Yllä olevassa voidaan riippuvuuksia soveltaa eri järjestyksissä ja näin saadaan vaihtoehtoisia avaimia.

### Riippuvuuksien päättely

Annetusta joukosta riippuvuuksia voidaan päätellä uusia riippuvuuksia. Päättely perustuu kolmeen päättelysääntöön, ns. Armstrongin aksiomiin

|  |  |
| --- | --- |
| refleksiivisyys | Jos XY, niin X->Y   eli X määrää funktionaalisesti kaikki osajoukkonsa, Tällaisia riippuvuuksia kutsutaan triviaaleiksi eikä niitä kirjata kartoituksessa. |
| laajennettavuus | Jos X->Y niin XZ->YZ   eli voimassaolevan  riippuvuuden kummallekin puolelle voidaan lisätä sama attribuuttiyhdistelmä ja saadaan uusi riippuvuus |
| transitiivisuus | Jos X->Y ja Y->Z, niin  X->Z   eli funktionaaliset riippuvuudet ovat transitiivisia |

Mille tahansa riippuvuusjoukolle voidaan määritellä minimaalinen joukko riippuvuuksia, josta on pääteltävissä samat riippuvuudet kuin annetusta riippuvuusjoukosta. Kartoitettaessa relaatiokaavioon liittyviä riippuvuuksia on tavoitteena löytää minimaalinen riippuvuusjoukko. Minimaalisessa riippuvuusjoukossa ei ole riippuvuuksia, jotka voidaan päätellä muista riippuvuuksista. On olemassa algoritmeja, jotka laskevat annetulle riippuvuusjoukolle minimaalisen riippuvuusjoukon.

Riippuvuudet heijastavat tietokannan kohdealueella vallitsevia lainalaisuuksia. Niitä ei voi määrätä tarkastelemalla vain tiettyä kaavion ilmentymää vaan niitä määrättäessä pitää ajatella tilannetta missä tahansa kaavion ilmentymässä.

Tarkastellaan esimerkkikaaviota Ostos ja siihen liittyviä mahdollisia riippuvuuksia. Alla olevassa taulukossa punertavalla värillä merkityt vaihtoehdot hylätään. Jäljelle jäävien riippuvuuksien joukko on minimaalinen.

|  |  |
| --- | --- |
| Ostos(tuotenumero, tuotenimi, listahinta, ostajan\_nimi, alennusprosentti, maksettuhinta,ostopäivä) | |
| tuotenumero->tuotenimi | Tuotenumeroon liittyy yksi tuotenimi. OK |
| tuotenumero->listahinta | Tuotteella on vain yksi listahinta. OK |
| tuotenumero->ostajan\_nimi | Tuotteella on vain yksi ostaja. EI |
| tuotenumero->alennusprosentti | Alennusprosentti on tuotekohtainen.  EHKÄ, oletetaaan kuitenkin, että näin ei ole |
| tuotenumero->maksettuhinta | Tuotteella on vain yksi maksettu hinta. EI, riippuvuus olisi voimassa, jos alennusprosentti olisi tuotekohtainen |
| tuotenumero->ostopäivä | Tuotetta voi ostaa vain yhtenä päivänä. EI |
| tuotenimi->tuotenumero | Tuotenimi on yksikäsitteinen. EHKÄ,  Oletetaan, ettei tämä ole voimassa, jolloin kaaviossa ei ole mielekkäitä riippuvuuksia, joissa tuotenimi olisi määrääjä |
| ostajan\_nimi->tuotenumero | Ostaja ostaa vain yhtä tuotetta. EI |
| ostajan\_nimi->alennusprosentti | Alennusprosentti on ostajakohtainen. EHKÄ, oletetaan näin olevan |
| alennusprosentti-> jotain | Alennusprosentin perusteella ei saada selville jonkin muun sarakkeen arvo. EI |
| listahinta-> jotain | Listahinnan perusteella saadaan selville |
| maksettuhinta-> jotain | Maksetun hinnan perusteella saadaan selville jonkin muun sarakkeen arvo. EI |
| listahinta,alennusprosentti-> maksettuhinta | Maksettu hinta on laskettavissa listahinnan ja alennusprosentin avulla. OK |

### Boyce-Codd normaalimuoto

Tietojen yhteenkuuluvuus voidaan määritellä riippuvuuksien avulla. Relaatiomalliin liittyvässä suunnitteluteoriassa on määritelty useita ns. normaalimuotoja, joihin kuhunkin liittyy erilaiset säännöt. Tällä kurssilla tarkastellaan normaalimuodoista vain Boyce-Codd normaalimuotoa. Normaalimuoto määritellään seuraavasti:

|  |
| --- |
| Relaatiokaavio on Boyce-Codd -normaalimuodossa, jos ja vain jos relaatiokaavioon ei liity yhtään sellaista funktionaalista riippuvuutta, jossa määrääjä ei sisältäisi relaation avainta. |

Kun tietokannan relaatiot ovat kaikki Boyce-Codd normaalimuodossa, **ei tietokannassa ole lainkaan toisteista tietoa.** Boyce-Codd normaalimuoto ei kuitenkaan ole aina saavutettavissa. Hieman lievemmässä kolmannessa normaalimuodossa sallitaan Boyce-Codd normaalimuodossa sallittujen riippuvuuksien lisäksi sellaiset riippuvuudet, joissa riippuva attribuutti kuuluu johonkin avaimeen. Kaavion jako tällaisiin osakaavioihin on aina saavutettavissa. Esimerkiksi kaavio

Osoite(Postinumero,Kaupunki,Katuosoite), jossa riippuvuudet ovat

Postinumero->Kaupunki ja Kaupunki, Katuosoite -> Postinumero

ei ole Boyce-Codd -normaalimuodossa sillä avaimet tässä kaaviossa ovat {Postinumero,Katuosoite} ja {Kaupunki,Katuosoite} eikä Postinumero sisällä näistä kumpaakaan. Sen sijaan riippuvuuden Postinumero->Kaupunki riippuva attribuutti Kaupunki kuuluu avaimeen {Kaupunki,Katuosoite}, joten relaatiokaavio on kolmannessa normaalimuodossa.

Boyce-Codd normaalimuodon tarkistamiseksi on siis ensiksi määriteltävä relaation avaimet ja sen jälkeen tarkistettava rikkooko jokin relaatiokaavioon liittyvä riippuvuus Boyce-Codd normaalimuodon ehdon.

Kaavion Ostos riippuvuuksissa oikealla puolella eivät esiinny *tuotenumero*, *ostajan\_nimi*eikä *ostopäivä*. Siispä K on aluksi {*tuotenumero,ostajan\_nimi ja ostopäivä*}. K:sta riippuvien attribuuttien joukko sisältää refleksiivisyyssäännön mukaan attribuutit itsensä sekä transitiivisuuden perusteella kaikki sellaiset attribuutit, jotka riippuvat näistä. K:sta riippuvia ovat siis tuotenumerosta riippuvat *tuotenimi*ja*listahinta*sekä ostajan\_nimestä riippuva *alennusprosentti*. *Listahinta*ja *alennusprosentti*puolestaan määräävät yhdessä *maksuhinnan*eli kaikki relaatiokaavion attribuutit kuuluvat K:n perusteella määräytyviin attribuutteihin. K on siis tämän kaavion ainoa avain, koska siihen ei tarvinnut lisätä mitään.

Yksikään kaavioon Ostos liittyvistä riippuvuuksista ei sisällä avainta K, joten kaavio ei ole Boyce-Codd normaalimuodossa. Attribuutit on siis uudelleenjärjesteltävä sellaisiksi relaatioiksi, joissa Boyce-Codd normaalimuodon vaatimukset ovat voimassa. Seuraavat relaatiot täyttävät ehdon

1. Tuote( tuotenumero , tuotenimi, listahinta)
2. Ostaja( ostajan\_nimi , alennusprosentti)
3. Alennus( listahinta, alennusprosentti ,maksuhinta)
4. Ostos( tuotenumero,ostajan\_nimi,ostopäivä )

Näistä kolmas (Alennus) on laskettavissa oleva funktio eikä sitä ole tarpeen tallentaa tietokantaan.

### Attribuuttien uudelleenryhmitys Boyce-Codd normaalimuodossa oleviksi kaavioiksi

Boyce-Codd normaalimuodossa oleva tietokantakaavio voidaan määrätä minimaalisen riippuvuusjoukon perusteella seuraavasti:

1. Ryhmittele riippuvuudet ryhmiin yhteisen määrääjän perusteella
2. Muodosta kutakin ryhmää kohden relaatiokaavio, johon otat kaikki ryhmän riippuvuuksissa esiintyvät attribuutit
3. Jos relaatiokaavion avain ei sisälly muodostettuihin relaatioihin, muodosta siihen kuuluvista attribuuteista oma relaatio.
4. Karsi mahdolliset saman asian toisteiset esitykset.
5. Anna muodostuneille relaatiokaavioille nimet. Jos niille löytyy nimet ja luonnollinen merkitys jakoa voi pitää onnistuneena.

Kaavion Ostos uudelleenjärjestelyn tuloksena syntyneistä kaavioista a, b ja c määräytyvät kohdan 2 perusteella ja d kohdan 3 perusteella. Niille löytyi myös luonnolliset nimet.

**Esimerkki:**

Aiemmin oli esillä ongelmallinen taulu Työ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Työ | ttNro | nimi | s\_aika | oNro | oNimi | osSijainti |
| 10 | M.Seppä | 1.3.59 | 3 | myynti | Helsinki |
| 20 | D.Leivo | 4.5.40 | 3 | myynti | Helsinki |
| 30 | S.Koivu | 8.6.66 | 4 | hallinto | Espoo |
| 40 | B.Oja | 2.4.65 | 4 | hallinto | Espoo |
| 50 | O.Itä | 1.2.55 | 6 | tuotanto | Espoo |

Tähän tauluun liittyen voidaan löytää riippuvuudet

ttNro ->  nimi,

ttNro -> s\_aika

ttNro -> oNro

oNro -> oNimi

oNro -> osSijainti

Riippuvuuksien perusteella taulun avain on sarake *ttNro*. Taulu ei ole Boyce-Codd normaalimuodossa. Siinä esiintyy siis toistoa, kuten aiemmin todettiin. Kun riippuvuudet ryhmitellään yhteisen määrääjän perusteella, saadaan ryhmät:

ttNro ->  nimi,

ttNro -> s\_aika

ttNro -> oNro

ja

oNro -> oNimi

oNro -> osSijainti.

Näiden perusteella määräytyvät kaaviot

kaavio\_1 (ttNro,  nimi, s\_aika, oNro) ja

kaavio\_2 (oNro, oNimi, osSijainti).

Kun kaavioille annetaan uudet nimet, päädytään aiemmin esillä olleisiin tauluihin

Työntekijä (ttNro,  nimi, s\_aika, oNro) ja

Osasto(oNro, oNimi, osSijainti).

**Esimerkki:**

Opintosuoritusote sisältää seuraavat tiedot

* Opiskelijan\_henkilötunnus
* Opiskelijanumero
* OpiskelijanNimi
* OpiskelijanOsoite
* OpiskelijanKirjoilletuloPvm
* PääOpinto-oikeudenTiedekunta
* PääOpinto-oikeudenKoulutusohjelma
* PääOpinto-oikeudenPääaine
* SuorituksenKoodi
* Suorituspäivä
* Arvosana
* SuorituksenLaajuus
* KurssinNimi
* SuorituksenLaitos

Oletetaan, että kaikki opiskelijoiden suoritusotteisiin tarvittavat tiedot talletettaisiin yhteen relaatioon (ns. universaalirelaatio-oletus). Määritellään tässä relaatiossa voimassa olevat funktionaaliset riippuvuudet

|  |  |
| --- | --- |
| Opiskelijan\_henkilötunnus -> Opiskelijanumero | Henkilöllä voi olla vain yksi opiskelijanumero |
| Opiskelijanumero -> Opiskelijan\_henkilötunnus | Opiskelijanumero henkilökohtainen |
| Opiskelijanumero -> OpiskelijanNimi | Opiskelijanumeroon liittyy yksi nimi (1) |
| Opiskelijanumero -> OpiskelijanOsoite | Opiskelijanumeroon liittyy yksi osoite (1) |
| Opiskelijanumero -> OpiskelijanKirjoilletuloPvm | Opiskelijanumeroon liittyy yksi kirjoilletulopäivä (1) |
| Opiskelijanumero -> PääOpinto-oikeudenKoulutusohjelma | Opiskelijalla on pääopinto-oikeus yhteen koulutusohjelmaan (1) |
| Opiskelijanumero ->  PääOpinto-oikeudenTiedekunta | Opiskelijalla on pääopintooikeus yhteen tiedekuntaan. Tämä on pääteltävissä. |
| Opiskelijanumero -> PääOpinto-oikeudenPääaine | Opiskelijalla on pääopistoikeus yhteen pääaineeseen. Tämä on pääteltävissä |
| PääOpinto-oikeudenKoulutusohjelma-> PääOpinto-oikeudenTiedekunta | Koulutusohjelma kuuluu yhteen tiedekuntaan |
| PääOpinto-oikeudenKoulutusohjelma-> PääOpinto-oikeudenPääaine | Koulutusohjelmalla on yksi pääaine |
| SuorituksenKoodi -> KurssinNimi | Kurssikoodiin liittyy yksi kurssin nimi |
| SuorituksenKoodi -> SuorituksenLaitos | Kurssikoodit ovat laitoskohtaisia |
| Opiskelijanumero, SuorituksenKoodi -> Suorituspäivä | Kurssista kirjataan kullekin vain yksi suorituskerta (1) |
| Opiskelijanumero, SuorituksenKoodi -> Arvosana | Kurssista kirjataan kullekin vain yksi arvosana (1) |
| Opiskelijanumero, SuorituksenKoodi -> SuorituksenLaajuus | Kurssin voi suorittaa erikokoisena |

(1) Myös riippuvuudet joissa henkilötunnus määrää näissä riippuvuuksissa oikealla olevan attribuutin ovat voimassa. Tässä on valittu nämä keskeisimmiksi joten henkilötunnukseen perustuvat riippuvuudet jäävät pääteltäviksi

Kun punaisella merkityt karsitaan, jää jäljelle minimaalinen joukko riippuvuuksia. Näiden perusteella saadaan relaatiolle avaimet {Opiskelijanumero, SuorituksenKoodi} ja {Opiskelijan\_henkilötunnus, SuorituksenKoodi}.

Relaatio ei ole Boyce-Codd normaalimuodossa, koska esimerkiksi riippuvuudessa 'SuorituksenKoodi  -> SuorituksenLaitos' määrääjä ei sisällä kumpaakaan avaimista.

Tehdään riippuvuuksien ryhmittely yhteisen määrääjän perustella ja muodostetaan relaatiokaaviot ryhmiin kuuluvista attribuuteista. Tuloksena saadaan kaaviot

|  |  |
| --- | --- |
| Kaavio 1 | Opiskelijanumero |
| Opiskelijan\_henkilötunnus |
| OpiskelijanNimi |
| OpiskelijanOsoite |
| OpiskelijanKirjoilletuloPvm |
| PääOpinto-oikeudenKoulutusohjelma |
| Kaavio 2 | Opiskelijanumero |
| Opiskelijan\_henkilötunnus |
| Kaavio 3 | PääOpinto-oikeudenKoulutusohjelma |
| PääOpinto-oikeudenTiedekunta |
| PääOpinto-oikeudenPääaine |
| Kaavio 4 | SuorituksenKoodi |
| KurssinNimi |
| SuorituksenLaitos |
| Kaavio 5 | Opiskelijanumero |
| SuorituksenKoodi |
| Suorituspäivä |
| Arvosana |
| SuorituksenLaajuus |

Saatuja kaavioita tarkastelemalla havaitaan, että Kaavio 2 on projektio Kaavioista 1, joten se voidaan toisteisena jättää pois. Sarakkeet voidaan myös nimetä uudelleen. Seuraavassa taulukossa taulut on nimetty ja muutama sarake uudelleennimetty. Taulukkoon on merkitty myös kaavioiden pääavaimet. Nämä kaaviot ovat kaikki Boyce-Codd normaalimuodossa.

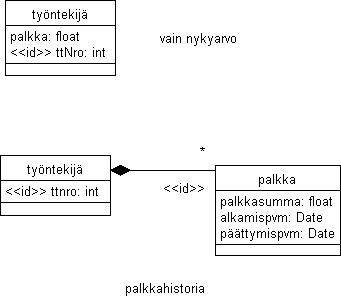
|  |  |
| --- | --- |
| Opiskelija | Opiskelijanumero |
| Henkilötunnus |
| Nimi |
| Osoite |
| KirjoilletuloPvm |
| PääOpinto-oikeudenKoulutusohjelma |
| Koulutusohjelma | Koulutusohjelma |
| Tiedekunta |
| Pääaine |
| Kurssi | KurssiKoodi |
| KurssinNimi |
| VastuuLaitos |
| Suoritus | Opiskelijanumero |
| KurssiKoodi |
| Suorituspäivä |
| Arvosana |
| SuorituksenLaajuus |

[Harri Laine](mailto:harri.laine@helsinki.fi): *Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

# Tietoja kuvaavat tiedot ja tietosisällön laajennettavuus käsitemallissa

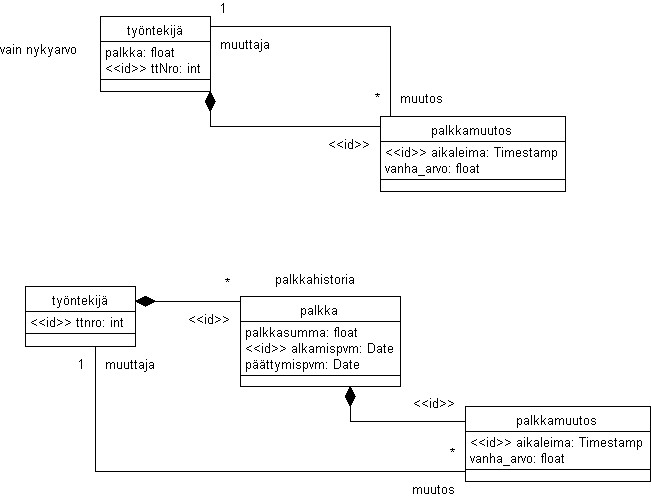
Tietosisältöä kuvaavat tietomallit ovat rakenteiltaan yksinkertaisia. Käytännön tietokannoissa saattaa tulla esiin mutkikkaita tilanteita, joiden mallintaminen on yksinkertaisella mallilla hankalaa. Tällaisia tilanteita ovat esimerkiksi historiatiedot, tietojen käyttöä tai tietoja muuten luonnehtivat yksilötason metatiedot (metatieto on tietoa tiedosta, esimerkiksi tiedon luotettavuus) sekä ratkaisut, joissa pyritään yleistyksiin ja tietosisällön laajennettavuuteen.

Historiatiedoilla tarkoitetaan tilannetta, jossa tietoon liittyy voimassaoloaika, tai halutaan tallettaa vanhat arvot tiedon muuttumisen yhteydessä. Esimerkiksi työntekijän kuukausipalkka voi muuttua ja vanhakin tieto halutaan säilyttää. Jos työntekijän palkkatiedosta halutaan tallentaa vain nykyarvo, voidaan palkka määritellä työntekijän attribuutiksi. Jos palkasta halutaan tietää myös milloin se oli voimassa, ei yksittäinen attribuutti enää riitä, vaan pitää löytää jokin muu ratkaisu. Yksi mahdollisuus on mallintaa palkkatieto omana tietokohteenaan (kuva 1). Tässä ratkaisussa palkkasumma on tietokohteen yksi attribuutti ja voimassaoloajan alkamis- ja päättymispäivät erillisiä attribuutteja. Samaan työntekijään voi liittyä useita palkkatieto-kohteita.



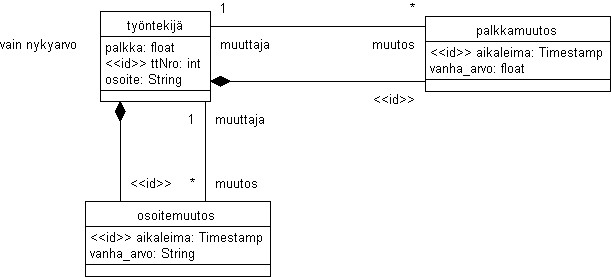
Kuva 1: Palkka nykyarvona ja palkkatieto tietokohteena.

Usein on tarpeen tallentaa myös tietoa tietojen käytöstä. Tämäkin on eräänlaista historiaa. Kuvan 2 ylemmässä esimerkissä on mallinnettu muutoshistoria palkkatiedolle tilanteessa, jossa työntekijätiedoissa on vain yksi palkka-attribuutti. Kuvan alemmassa esimerkissä muutoshistoria on liitetty erillisiin historiaa sisältäviin palkkatietokohteisiin. Joissain tilanteissa ylemmän esimerkin ratkaisumallia voi käyttää vaihtoehtona kuvan 1 mukaiselle historiatiedon mallinnukselle.



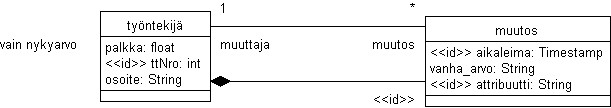
Kuva 2: Tietoon liittyvä muunnoshistoria.

Tietokohteeseen voi liityä useita tietoja, joiden yhteydessä on tarpeen tallentaa muunnoshistoria. Kutakin seurattavaa kohdetta varten voitaisiin määritellä kuvan 3 mukaisesti omat muunnoshistoriatietonsa. Tämä voi johtaa hyvin suureen määrään tietokohteita ja hankaloittaa tietosisällön hahmottamista ja ehkä lopulta myös tietokannan käsittelyä.



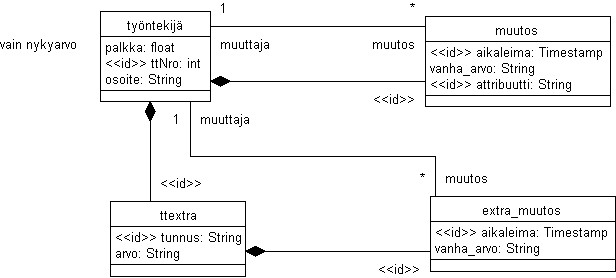
Kuva 3: Kaksi muunnoshistoriaa.

Vaihtoehtoisena tapana on hahmottaa muunnoshistoria geneerisesti (kuva 4). Tässä mallissa selvitään yhdellä tietokohdetyypillä 'muunnos', jota voisi käyttää kaikkien työntekijään kohdistuvien muunnoshistorioiden esittämiseen. Ratkaisussa muuttuvalle tietokohteelle on annettu nimi (attribuutti). Jos kyse olisi palkan muutoksesta olisi tämän nimen arvona 'palkka' ja vastaavasti osoitetiedon muuttuessa arvona olisi 'osoite'. Tiedon vanha-arvo joudutaan tällaisessa tilanteessa esittämään yleistettyä tietotyyppiä (merkkijono) käyttäen . Ratkaisua voisi vieläkin yleistää siten, että eri kohteiden tietoihin liittyvät muunnoshistoriatkin koottaisiin yhteen. Tämä edellyttäisi muutoksia kohteiden identifiointiin.



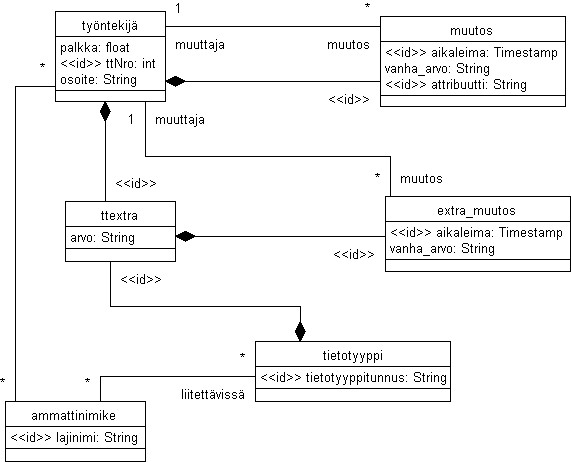
Kuva 4: Yleistetty muunnoshistoria.

Geneerisiä tietoja voidaan käyttää myös tilanteissa, joissa käyttäjälle annetaan mahdollisuus laajentaa tietotietosisältöä omien tarpeidensa mukaan. Kuvan 5 esimerkissä työntekijään voidaan liittää ennalta määrittelemättömiä lisätietoja ja näihin edelleen muutoshistoria. Lisätiedot mallinnetaan omina geneerisinä tietokohteinaan. Lisätiedon merkitys määräytyy sille annetun tunnuksen perusteella.



Kuva 5: Geneerinen lisätieto mahdollistaa tietosisällön laajennuksen.

Edellä olevassa esimerkissä ei ole mitenkään rajoitettu sitä millaisia lisätietotunnuksia saa käyttää. Riittää, että tunnukset ovat yksikäsitteisiä työntekijäkohtaisesti. Kuvassa 6 mallia on laajennettu siten, että lisätiedot voidaan määritellä. Lisätieto määritellään 'tietotyyppi' tietokohteena. Kuvassa tuohon tietotyyppiin on liitetty vain attribuutti tyyppitunnus, mutta siihen voisi liittyä erilaisia luonnehdintoja tietotyypistä ja sen yhteydessä käytettävistä arvoista, esimerkiksi ovatko arvot numeerisia vai aakkostietoa. Kuvan esimerkissä on myös mallinnettu tilanne, jossa työntekijöitä on eri ammateissa ja ammattinimikekohtaisesti voidaan määritellä mitkä lisätiedot ovat mahdollisia.



Kuva 6: Geneerinen määritelty lisätieto

Harri Laine 1.10.2008

TIETOKANTOJEN PERUSTEET

# SQL:n historia

SQL-kieli kehitettiin alunperin IBM:n San Josen tutkimuslaboratoriossa, System R nimisen relaatiotietokannanhallintajärjestelmän prototyyppiin liittyvän tutkimusprojektin yhteydessä 1970-luvun alkupuolella.

Ensimmäiset SQL-kieleen pohjautuvat tietokannan hallintajärjestelmät tulivat kaupallisesti saataville 1970-luvun lopulla. Samaan aikaan ilmestyi myös muihin käsittelykieliin perustuvia relaatietietokannan hallintajärjestelmiä. Tunnetuin vaihtoehtokieli SQL:lle oli Ingres-järjestelmän QUEL-kieli.

Standardoinnin kohteeksi valittiin SQL. Standardointia hoitivat sekä amerikkalainen ANSI että kansainvälinen ISO.

1986 hyväksyttiin ensimmäinen SQL-standardi (SQL:n ydin noin 40s + moduulikieli, upotus ohjelmointikieliin, yhteensä noin 100s)

1989 valmistui laajennos, joka sisälsi muun muassa viiteavainten määrittelymahdollisuuden (noin 20 sivua).

1992 hyväksyttiin uusi versio SQL-92 (käytetty myös nimitystä SQL2), joka sisälsi merkittäviä laajennoksia kieleen. Standardin laajuus oli noin 600 sivua + kuvaustietostandardi, kaikkiaan lähes 1000 sivua. SQL:lle määriteltiin kolme **laajuustasoa:**

* perustaso, joka sisälsi lähinnä vanhan SQL-ytimen
* välitaso, joka sisälsi uusia tietotyyppejä, operaatioita ja rakenteita
* täysi SQL, joka edelleen lisäsi tietotyyppejä ja rakenteita

Yhden massiivisen standardin asemesta päätettiin jatkaa jatkokehitystä osina.

1995 hyväksyttiin SQL/CLI kutsurajapinta (call level interface) eli ohjelmien kautta tapahtuvan käytön rajapintamäärittely. Standardi määrittelee ODBC rajapinnan.

1996 hyväksytty SQL/PSM (persistent stored modules) on tietokantaproseduurien määrittelykieli.

1999 hyväksyttiin uusi versio SQL-99 ( käytetty myös nimitystä SQL3). Standardi jakautuu viiteen osaan

* Framework (johdanto)
* Foundation (ydin)
* CLI (call level interface)
* PSM (persistent stored modules)
* Bindings (ohjelmointikieliliittännät).

Yhteiskooltaan standardi on noin 2100 sivua. Se sisältää runsaasti laajennoksia vuoden 1992 standardiin nähden, esimerkiksi uusia tietotyyppejä, automaattisesti käynnistyvät tietokantaproseduurit, oliopiirteitä, laajemmat eheystarkistukset, rekursiiviset kyselyt, tietovarastotoimintoja.

Kielelle ei enää ole määritelty laajuustasoja, vaan on määritelty ydin, johon voidaan liittää lisäpiirteitä pakkauksina.

Tällä kurssilla käsitellään vain vähän SQL-99 laajennoksia.

2000 SQL/OBL (SQLJ) SQL:n upotus Java-ohjelmaan.

Tekeillä on ainakin multimedialaajennoksiin liittyvä standardi ja tietovarastopiirteitä täydentävä standardi.

## Tietokantatuotteiden markkinatilanne

Tietokantatuotteiden markkinat olivat vuonna 2000 kooltaan noin 8.8 miljardia USD ja vuosikasvu 10 prosenttia. Relaatiotietokantatuotteiden osuus tästä on yli 80 prosenttia ja kasvoi vuonna 2000 15 prosenttia. Seuraavassa taulukossa näkyy lisenssien määrään perustuvien markkinaosuuksien kehitys vuosina 1999-2000.

|  |  |  |
| --- | --- | --- |
| **Tuottaja** | **Osuus 2000** | **Osuus 1999** |
| Oracle | 33.8 | 31.4 |
| IBM | 30.1 | 29.9 |
| Microsoft | 14.9 | 13.1 |
| Sybase | 3.2 | 3.3 |
| Informix | 3.0 | 5.0 |
| Muut | 15.0 | 17.3 |

Worldwide Database Management Systems Software New License Revenue,Market Share Estimates for 2000, Company 2000 Market Share (Percent) 1999 Market Share (Percent), Source: Gartner Dataquest (May 2001)

Unix ympäristössä Oraclen markkinaosuus oli 66.2 prosenttia, IBM:n 14.4 ja Informix:n 6.7 prosenttia. Sensijaan Windows NT ympäristössä Microsoft nousi niukasti ohi Oraclen 38 prosentin osuudellaan Oraclen 37.3 prosentia vastaan. Kolmantena näillä markkinoilla oli IBM, jonka osuus oli 14.4 ja kasvu nopeinta.

IBM:n tietokannan hallintajärjestelmä on nimeltään DB2. Microsoftilla on kaksi relaatiotietokannan hallintajärjestelmää; kevyt Access ja palvelinkäyttöön tarkoitettu SQL Server. Suomalaisista relaatiotietokannan hallintajärjestelmistä tunnetuin on Solid. Freeware- tai shareware-pohjalla saatavia relaatiotietokannan hallintajärjestelmiä ovat esimerkiksi ruotsalainen MySQL ([http://www.mysql.com](http://www.mysql.com/)) ja Berkleyn yliopistosta lähtöisin oleva PostgreSQL ([http://prostgresql.matrix.fi/](http://prostgesql.matrix.fi/))

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

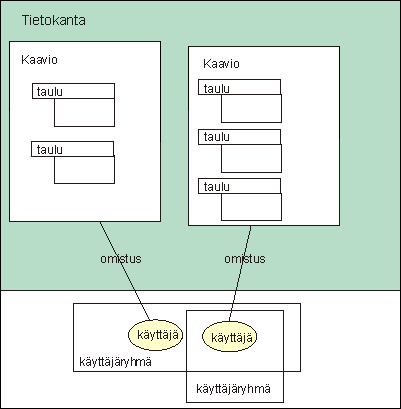
# SQL-tietokanta

SQL-tietokanta muodostuu yhden tai useamman kaavion ( *schema*) määrittelemistä tauluista ( *table*). Taulu vastaa relaatiomallin relaatiota, mutta

* sallii etenkin kyselyiden tuloksissa samanlaisen rivin toistumisen useaan kertaan
* kyseessä ei ole siis matemaattinen relaatio vaan ns. monijoukko (multiset)

Taulu muodostuu riveistä ( *row*). Rivit vastaavat relaatiomallin monikkoja. Taulussa on vähintään yksi sarake (column). Sarakkeille on annettava nimet. Jokaiseen sarakkeeseen liittyy arvojoukko (domain). Arvojoukot perustuvat SQL:n perustietotyyppeihin. SQL-92 standardin mukaisesti arvot ovat atomisia, kuten relaatiomallissa. Sen sijaan SQL-99 sallii myös rakenteiset arvot eli sellaiset arvot, jotka voivat jakautua nimettyihin osiin. SQL-99 mahdollistaa myös omien tietotyyppien määrittelyn.

Tauluja on kahden tyyppisiä: perustauluja (base table) ja johdettuja tauluja (derived tables, views). Perustaulut ovat aidosti olemassa apumuistille tallennettuina. Johdetut taulut määritellään kyselyjen avulla. Niitä voidaan tietokantaoperaatioissa käyttää miltei perustaulujen tapaan.



Kullakin kaaviolla on omistaja (owner), joka omistaa myös kaavion määrittelemät taulut. Omistaja identifioidaan käyttäjätunnuksella (user account). Käyttäjätunnus voidaan liittää useaan käyttäjäryhmään. Sekä käyttäjälle että käyttäjäryhmälle voidaan myöntää erilaisia oikeuksia tietokantaan liittyen.

Omistajalla on kaikki oikeudet omistamiinsa tauluihin liittyen. Oletusarvoisesti muilla käyttäjillä (superkäyttäjiä lukuunottamatta) ei ole pääsyä käyttäjän tauluihin. Omistaja voi kuitenkin antaa muille käyttäjille tai käyttäjäryhmille oikeuksia tauluihinsa.

Tauluihin voi liittyä erilaisia eheysehtoja (constraint). Nämä voivat koskea yksittäisiä taulun rivejä, koko taulua tai jopa useita tauluja. Sellaisia tietokantaan kohdistuvia muutoksia, jotka rikkoisivat eheysehtoja, ei hyväksytä.

## SQL-kielen kirjoitusasu

SQL-kielen kirjoitusasu on vapaa. Tämä tarkoittaa sitä, että kielen avainsanat, kaavioiden, taulujen ja sarakkeiden nimet voidaan kirjoittaa joko suur- tai pienaakkosilla tai sekamerkein.

Esimerkiksi 'select nimi, Osoite from henkilo ' on merkitykseltään sama kuin 'SELECT NIMI, OSOITE from HENKILO '.

Perinteisesti kaikki viitaukset taulujen tietosisältöön on kuitenkin pitänyt kirjoittaa talletusasun mukaisina. Täten

    select osoite from henkilo where nimi like 'Kalle%'

tuottaa eri tuloksen kuin

    select osoite from henkilo where nimi like 'KALLE%'

Joissain järjestelmissä kirjoitusasukäyttäytymistä voidaan säädellä asetusparametrein.

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

# SQL tiedonmäärittelykielenä

|  |  |
| --- | --- |
| **Sisältö** | [Kaavioiden määrittely ja poisto](http://www.cs.helsinki.fi/u/laine/tkp/sql/rakenne.html#kaavio)  [Arvojoukkojen määrittely ja poisto](http://www.cs.helsinki.fi/u/laine/tkp/sql/rakenne.html#arvojoukko)  [Perustietotyypit](http://www.cs.helsinki.fi/u/laine/tkp/sql/tietotyypit.html)  [Taulujen määrittely, muutos ja poisto](http://www.cs.helsinki.fi/u/laine/tkp/sql/rakenne.html#taulu)  [Sarakkeiden määrittely](http://www.cs.helsinki.fi/u/laine/tkp/sql/rakenne.html#sarake)  [Eheysehtojen määrittely, muutos ja poisto](http://www.cs.helsinki.fi/u/laine/tkp/sql/rakenne.html#eheysehto)  [Tietokantaproseduurien määrittely muutos ja poisto](http://www.cs.helsinki.fi/u/laine/tkp/sql/rakenne.html#proseduuri) |
| **Oheismateriaalia**: | * [Elmasri&Navathe](http://www.cs.helsinki.fi/u/laine/tkp/lahteita.html): luvut 8.1-8.3, sivut 207-217 * [Ramakrishnan&Gehrke](http://www.cs.helsinki.fi/u/laine/tkp/lahteita.html): luvut 3.1.1-3.3, sivut 62-71 * [Laine](http://www.cs.helsinki.fi/u/laine/tkp/lahteita.html): luku 5.3: SQL-määrittelykielenä |

## Kaavion määrittely

Kaavion määrittelyn alle voidaan koota samaan hallinnolliseen kokonaisuuteen kuuluvia tietokantakohteita, esimerkiksi kaikki oppilaitoksen ilmoittautumisjärjestelmään liittyvät kohteet. Kaavio määritellään 'create schema' -lauseella

|  |  |
| --- | --- |
| [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | create schema *schema\_name*   authorization *user\_name* |
| esimerkki | create schema opintokanta authorization laine |

Koko kaaviosta pääsee eroon lauseella 'drop schema'. Jos kaaviossa on kohteita, operaatio epäonnistuu ellei lauseessa ole mukana cascade avainsanaa.

|  |  |
| --- | --- |
| [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | drop schema *schema\_name*[cascade] |
| esimerkki | drop schema opintokanta cascade |

## Arvojoukkojen määrittely ja poisto

Nimettyjä arvojoukkoja on mahdollista määritellä järjestelmän [perustietotyyppeihin](http://www.cs.helsinki.fi/u/laine/tkp/sql/tietotyypit.html) pohjautuen. Määrittelyyn liittyy tällöin jokin sallittuja arvoja rajoittava ehto. Ehtolauseketta käsitellään tarkemmin kyselyjen yhteydessä. Kaikki samaan perusjoukkoon kuuluvat arvot ovat vertailtavissa keskenään.

|  |  |
| --- | --- |
| [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | create domain *domain\_name* [as] *data\_type* [defaul *default\_value*] [[ constraint *constraint\_name*] check (value *condition*) ] |
| esimerkkejä | create domain kuukausi as integer    default 1    check (value between 1 and 12);  create domain arkipaiva varchar(11)    constraint paivat check     (value in ('maanatai', 'tiistai', 'keskiviikko',      'torstai', 'perjantai','lauantai')); |

Ensimmäisessä esimerkissä oletusarvona on arvo 1 ja arvon on oltava välillä 1-12. Jälkimmäisessä ei ole määritelty oletusarvoa, joten se on tyhjäarvo NULL. Vain luetellut arvot kelpaavat tämän arvojoukon avulla määriteltäviin sarakkeisiin.

Arvojoukko poistetaan ja sitä voidaan muuttaa lauseilla

|  |  |
| --- | --- |
| [muodot](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | drop domain *domain\_name*;  alter domain  *domain\_name*drop default;  alter domain  *domain\_name*     set default *default\_value ;*  alter domain  *domain\_name*   drop constraint  *constraint\_name ;*  alter domain  *domain\_name*add    constraint *constraint\_name* check (value *condition*); |
| esimerkkejä | drop domain kuukausi;  alter domain arkipaiva drop constraint paivat;  alter domain arkipaiva add constraint paivat  check (value in ('maanatai', 'tiistai', 'keskiviikko','torstai', 'perjantai')); |

## Taulun määrittely

Tietokannan perustaulut määritellään create table -lauseilla

|  |  |
| --- | --- |
| [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | create table *tablename*    ( *column definition 1*[, **...**]        [, *constraint 1*[, **…**]] ) |
| esimerkki | create table opintojakso (    kurssikoodi varchar(15) **not null primary key**,    opintoviikot decimal(3,1) **not null**,    nimi\_suomeksi varchar(50) **not null**,    nimi\_engl varchar(50)); |

Tauluun voi kuulua useita sarakkkeita. Sarakkeiden enimmäismäärä on toteutuskohtaista, usein 256. [Sarakemäärittelyt](http://www.cs.helsinki.fi/u/laine/tkp/sql/rakenne.html#sarake) ovat osa taulun määrittelyä. Tauluun voi liittyä myös useita taulukohtaisia eheysehtoja.

Taulun ja sarakkeen nimi voi standardin mukaan olla 128 merkin pituinen. Nimi ei voi alkaa numerolla. Nimessä ovat sallittuja aakkosmerkit a-z, numerot ja eräät erikoismerkit. Jos nimi suljetaan lainausmerkkihin, sallitaan nimessä muitakin merkkejä esimerkiksi välilyönti (esim. "työhuoneen numero").

Katso laajempi [esimerkki opetustietojärjestelmän tietokantakaaviosta](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/kanta1.html).

Taulun määrittelyn voi poistaa ja samalla tuhota taulun drop table -lauseella. Drop table ei onnistu, jos tauluun

|  |  |
| --- | --- |
| [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | drop table *tablename* |
| esimerkki | drop table opintojakso |

Taulun rakennetta muutetaan alter table -lauseella. Lauseella voi lisätä tauluun uusia sarakkeita, muuttaa olemassaolevien sarakkeiden määrityksiä (esimerkiksi arvojen pituusmääreitä), poistaa sarakkeita (ei onnistu kaikissa toteutuksissa), lisätä ja poistaa eheysehtoja.

|  |  |
| --- | --- |
| [muodot](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | alter table *tablename*add (    [*column definition 1*[ *,***...**]    [*constraint definition1*[ *,***...**]   );  alter table *tablename*modify (  *column definition 1*[, **...**]);  alter table *tablename*drop column *column\_name ;*  alter table *tablename*drop constraint    *constraint\_name ;* |
| esimerkki | alter table opintojakso add (kesto smallint);  alter table opintojakso modify (nimi\_engl varchar(80));  alter table kurssi drop constraint fk\_kurssi\_oj ; |

## Sarakkeiden määrittely

Sarakkeen määrittely sisältyy osana taulun määrittelyyn. Sarakkeelle on määriteltävä tietotyyppi joko käyttäen SQL:n [perustietotyyppejä](http://www.cs.helsinki.fi/u/laine/tkp/sql/tietotyypit.html) tai[nimettyjä arvojoukkoja](http://www.cs.helsinki.fi/u/laine/tkp/sql/rakenne.html#arvojoukko).

|  |  |
| --- | --- |
| [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | *column\_name*{ *data\_type*| *domain\_name*}    [default *default\_value*]   [ *column constraint 1*[, **...**]] |
| esimerkki | kurssikoodi varchar(15) **not null primary key**, |

Sarakkeelle voidaan määritellä oletusarvo, joka annetaan uudelle riville siinä tapauksessa, että riviä lisättäessä ei kyseiselle sarakkeelle ole annettu mitään arvoa. Sarakemäärittelyyn voidaan liittää myös sarakekohtaisia eheysehtoja. Puhtaasti sarakekohtainen eheysehto on arvon pakollisuusvaatimus (not null). Tämä estää tyhjäarvot sarakkeessa. Jos sarake yksinään muodostaa taulun avaimen voidaan avainmäärittely (primary key) antaa sarakemäärittelyn yhteydessä. Tämä samoin kuin viiteavainmäärittelykin voidaan antaa myös sarakemäärittelyjen jälkeen taulun määrittelyn lopussa. Eheysehtona voidaan antaa myös samanlainen check-määre kuin [arvojoukon määrittelyssä.](http://www.cs.helsinki.fi/u/laine/tkp/sql/rakenne.html#arvojoukko)

## Eheysehdon määrittely, muutos ja poisto

Tauluun voidaan liittää taulukohtaisia eheysehtoja. Tärkeimmät näistä ovat avainmäärittely ja viiteavainmäärittely.

#### Avainmäärittely

Jos taulun avain muodostuu useasta sarakkeesta ei avainta voi määritellä sarakemäärittelyn yhteydessä vaan tarvitaan erillinen avainmäärittely. Eheysehdoille kannattaa antaa nimi, vaikka se ei ole välttämätöntä, sillä silloin ehdot voi helposti poistaa tarvittaessa.

|  |  |
| --- | --- |
| [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | [constraint *constraint\_name*] primary key (  *column\_name1*[, **...**] ) |
| esimerkki | constraint pk\_kurssi    primary key (kurssikoodi, lukuvuosi, lukukausi,kurssinumero) |

#### Viiteavainmäärittely

Yksisarakkeinen viiteavain voidaan määritellä sarakemäärittelyn osana. Monisarakkeiset viiteavaimet on määriteltävä erillisellä viiteavainmäärittelyllä.

|  |  |
| --- | --- |
| [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | [constraint *constraint\_name*]  foreign key      ( *column\_name1*[,  *...*] )      references *tablename*[( *foreign\_column\_name*[, **...**] )]     [on *event1 action1*[ , *event 2 action 2*[, **...**] ] |
| esimerkki | constraint fk\_kurssi\_oj foreign key    (kurssikoodi) references opintojakso    on delete cascade, on update restrict |

Ylläolevassa esimerkissä viiteavain kurssikoodi viittaa taulun opintojakso pääavaimeen. Jos viitattu rivi poistetaan poistetaan myös kaikki sellaiset rivit jotka viittaavat siihen. Viitatun rivin pääavainta ei voi vaihtaa.

Viiteavainmääritykseen voidaan liittää toimenpideosa, jossa kuvataan mitä tehdään sinä tapauksessa että jokin ylläpito-operaatio rikkoo viite-eheyden. Oletusarvoisesti viite-eheyttä rikkovia operaatioita ei sallita. Poistojen ja muutosten yhteydessä viite-eheys on kuitenkin palautettavissa korjaustoimenpitein.

ON DELETE -tilanteessa ollaan poistamassa viitattua riviä. Toimintamahdollisuudet ovat:

* Ei hyväksytä poistoa (RESTRICT)
* korvataan viitteet tyhjäarvoilla (SET NULL)
* korvataan viitteet oletusarvolla (SET DEFAULT)
* poistetaan kaikki viittaavat rivit (CASCADE)

ON UPDATE -tilanteessa on muutettu viitatun rivin avainta. Toimintamahdollisuudet ovat:

* Ei hyväksytä muutosta (RESTRICT)
* korvataan viitteet tyhjäarvoilla (SET NULL)
* korvataan viitteet oletusarvolla (SET DEFAULT)
* muutetaan kaikki viittaavat viiteavaimet vastaavasti (CASCADE)

## Tietokantaproseduurien määrittely, muutos ja poisto

Tietokantaproseduurit (persistent stored modules) ovat tietokantaan 'tallennettuja' valmiiksi käännettyjä ohjelmia tietokannan käsittelyyn. Tietokantaproseduurit voidaan jakaa kahteen tyyppiin

* kutsusta käynnistyviin proseduureihin ja funktioihin ja
* automaattisesti käynnistyviin (trigger).

Näiden ohjelmointia varten on määritelty oma Ada-ohjelmointikieltä muistuttava tietokantaohjelmointikieli. Oraclessa kielen nimenä on PL/SQL. Tietokantaproseduurit tarjoavat useissa tapauksissa tehokkaan tavan paketoida tietokantaoperaatioita kokonaisuudeksi ja kätkevät tietokannan rakenteen käyttäjiltä. Käyttäjien ei tarvitse tietää yksityiskohtia tietokannasta. Heidän tarvitsee tietää vain tietokantaoproseduurien tarjoama rajapinta tietokannan käyttöön.

Automaattisesti käynnistyvät tietokantaproseduurit 'triggerit' voidaan määritellä käynnistymään kun tietokantaan on kohdistettu jotain operaatioita, esimerkiksi lisätty uusi rivi johonkin tauluun. Esimerkiksi viite-eheyden valvonta suoritetaan tietokannanhallintajärjestelmissä usein automaattisesti käynnistyvien viiteavainmäärittelyn perusteella generoitujen proseduurien avulla.

Alla on PL/SQL:llä kirjoitettu esimerkki siitä, miltä tietokantafunktio näyttää. Funktio hakee parametreina määritettyn kurssin opettajan nimen muodossa 'Etunimi Sukunimi' ja palauttaa tyhjän merkkijonon, jos kurssilla ei ole luennoijaa. Proseduureissa käytettävä kieli ei kuulu tämän kurssin asiasisältöön.

|  |
| --- |
| create function hae\_kurssin\_luennoija (  in kurssikoodi varchar, in lukuvuosi integer,  in lukukausi char, in kurssinumero integer)  is  cursor opehaku is  select etunimi||' '||sukunimi  from opetustehtava ot, tyo, opettaja  where ot.kurssikoodi=tyo.kurssikoodi and  ot.lukuvuosi=tyo.lukuvuosi and  ot.lukukausi=tyo.lukukausi and  ot.kurssinumero=tyo.kurssinumero and  ot.opetusnumero=tyo.opetusnumero and  ot.tyyppi='LUENTO' and  tyo.opettaja=opettaja.opetunnus;  nimi varchar(80);  begin  open cursor opehaku;  fetch opehaku into nimi;  if opehaku%notfound then  nimi:=''; -- palautetaan tyhjä merkkijono  end if;  close opehaku;  return nimi;  end; |

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

## SQL:n perustietotyypit

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sisältö** | |  | | --- | | [Tekstitieto](http://www.cs.helsinki.fi/u/laine/tkp/sql/tietotyypit.html#tekstitieto) | | [Binääritieto](http://www.cs.helsinki.fi/u/laine/tkp/sql/tietotyypit.html#binäärinen) | | [Tarkka ja epätarkka numeerinen tieto](http://www.cs.helsinki.fi/u/laine/tkp/sql/tietotyypit.html#numeerinen) | | [Aikatieto](http://www.cs.helsinki.fi/u/laine/tkp/sql/tietotyypit.html#aikatieto) | | [Totuusarvo](http://www.cs.helsinki.fi/u/laine/tkp/sql/tietotyypit.html#totuusarvo) | |

SQL-99 tarjoaa yllä esitettyjen tietotyyppien lisäksi taulukkotiedot, olioviitteet ja käyttäjän omat tietotyypit. Tässä keskitytään kuitenkin SQL-92 tietotyyppeihin.

### Tekstitieto

Tekstitiedon esittämiseen on tarjolla kuusi tietotyyppiä:

|  |  |
| --- | --- |
| Tietotyyppi | SQL-määrittelyn [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) |
| kiinteäpituinen teksti | character[( *size*)]  char[(*size*)] |
| vaihtuvapituinen teksti | character varying [(*size*)]  varchar[(*size*)] |
| pitkä teksti | character large object (*size*)  CLOB (*size*) |
| kiinteäpituinen kansallismerkistöön perustuva teksti (unicode-koodattu?) | national character [(*size*)]  nchar [(*size*)] |
| vaihtuvapituinen kansallismerkistöön perustuva teksti (unicode-koodattu?) | national character varying [(*size*)]  nchar varying [( *size*)] |
| pitkä kansallismerkistöön perustuva teksti (unicode-koodattu) | national large character object ( *size*)  nclob (*size*) |

Koko (size) ilmaisee joko merkkijonon pituuden tai maksimipituuden merkkeinä. Puuttuva koko tarkoittaa yhden merkin mittaista tekstitietoa. Pituuksien ylärajat ovat järjestelmäkohtaisia. Esimerkiksi Oracle 8 järjestelmässä pitkää tekstiä on käytettävä, jos merkkijonon pituus ylittää 4000 merkkiä. Kiinteäpituisen tekstin maksimipituus Oraclessa on 2000 merkkiä. Pitkiin teksteihin saattaa liittyjä järjestelmäkohtaisesti käyttörajoitteita esimerkiksi arvojen vertailussa. Joissain järjestelmissä vertailukäyttökin onnistuu. Kansallismerkistö (käytännössä unicode) vaikuttaa järjestämiseen ja vertailuihin. On syytä huomata, että 'normaalimerkistöllä' ja kansallismerkistöllä toteutettujen merkkijonojen vertailu ei suoraan onnistu. Kiinteäpituisille tiedoille varataan tilaa pituuden mukaan, vaihtuvapituisille kulloisenkin tarpeen mukaan.

Tekstitietovakiot esitetään yksinkertaisiin lainausmerkkeihin suljettuina:

|  |  |  |
| --- | --- | --- |
| henkilötunnus | char(11) | '101010-1234' |
| osoite | varchar(80) | 'Koulukatu 6, Helsinki' |
| graduteksti | clob(240K) | 'Johdanto\nTutkielmassa...' |

### Binäärinen tieto

Binäärisen tiedon, esimerkiksi kuvien, esittämiseen on tarjolla kolme tietotyyppiä

|  |  |
| --- | --- |
| Tietotyyppi | SQL-määrittelyn [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) |
| kiinteäpituinen bittivektori | bit[(*size*)] |
| vaihtuvapituinen bittivektori | bit varying [(*size*)] |
| iso binääritieto | binary large object (*size*)  BLOB (*size*) |

Bittivektoreiden koko ilmaistaan bitteinä, ison binääritiedon koko 8-bitin tavuina. Ylärajat vaihtelevat toteutuskohtaisesti.

Bittivektoriliteraalit voidaan esittää bitteinä (B) tai heksadesimaalimerkkeinä (X).

|  |  |  |
| --- | --- | --- |
| pyhapaivakartta | bit(365) | B'10010000000...' |
| pienigifkuva | blob(100K) | X'A09FF00A11FFF....' |

### Numeerinen tieto

Tarkan numeerisen tiedon esittämiseen on tarjolla 4 tietotyyppiä

|  |  |
| --- | --- |
| Tietotyyppi | SQL-määrittelyn [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) |
| pieni kokonaisluku (2 tavua) | smallint |
| kokonaisluku (4 tavua) | integer, int |
| luku | numeric [(*precision*[,*scale*])] |
| desimaaliluku | decimal [( *precision*[,*scale*])]  dec [(*precision*[,*scale*])] |

Lukuarvoihin liitettävistä kokotiedoista *precision*ilmaisee luvun numeropositioiden kokonaismäärän ja *scale*desimaaliosan pituuden. *Scale*ei voi olla suurempi kuin *precision*. Pituuksien puuttuessa käytetään järjestelmäkohtaisia oletusarvoja.

Liukulukuina esitettäville epätarkoille numeerisille tiedoille on kolme tietotyyppiä, jotka kaikki perustuvat mantissan ja eksponentin esittämiseen omina luvun osinaan.

|  |  |
| --- | --- |
| Tietotyyppi | SQL-määrittelyn [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) |
| määriteltävä liukuluku | float (*precision*) |
| reaaliluku | real |
| kaksoistarkkuuden liukuluku | double precision |

Määriteltävän liukuluvun tarkkuus on määriteltävissä toteutuskohtaisissa puitteissa. Reaaliluvun maksimitarkkuus on tkhj:n toimittajan päätettävissä samoin kaksoistarkkuuden liukuluvun. Kaksoistarkkuuden liukuluku ei saa olla vähemmän tarkka kuin tavallinen liukuluku.

Numeeriset literaalit esitetään ohjelmointikielistä tutulla notaatiolla. Desimaalierottimena on piste.

|  |  |  |
| --- | --- | --- |
| pikkuraha, alle 10000 | decimal(6,2) | 320.15 |
| generoitu tunnusluku | numeric(20) | 123456 |
| värähtelytiheys | double precision | 123E2 |

### Aikatieto

Aikatiedon esittämiseen on tarjolla neljä tietotyyppiä:

|  |  |
| --- | --- |
| Tietotyyppi | SQL-määrittelyn [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) |
| päiväys | date |
| kellonaika | time[(*precision*)] |
| aikaleima | timestamp[(*precision*)] |
| aikaväli | interval |

Päiväys sisältää vain päivämäärän. Aikaleima sisältää sekä kellonajan että päiväyksen. Kellonajan tarkkuus (precision) ilmoittaa monellako desimaalilla sekunnit ilmoitetaan. Kellonajassa oletusarvona on nolla (0) eli kellonaika kirjataan ilman sekunnin osia. Aikaleimassa oletusarvona on 6 eli miljoonasosasekunnin tarkkuus.

Ajoilla voi suorittaa laskentaa. Oletetaan, että to\_day olisi tämän päivän päiväys. Tällöin to\_day + interval 1 days antaisi huomisen päiväyksen ja'31.12.2010' - to\_day ilmoittaisi aikaeron vuoden 2010 lopusta tähän päivään. Aikaero ilmaistaan SQL-standardin mukaan interval tyyppisenä tietona, jossa on erikseen vuodet, kuukaudet, päiväys , jne. Näiden muuntamiseen eri yksiköiksi löytyy muunnosfunktioita. Olkoon nownykyhetken aikaleima, tällöin now+ interval 6 hours on 6 tuntia nykyhetkestä. Aikatiedon vertailu perustuu ajalliseen edeltävyyteen, joten '1.2.2002' on suurempi kuin '31.1.2002'.

|  |
| --- |
| Huom.: Oracle:ssa tietotyyppi *DATE* on perinteisesti vastannut aikaleimaa (timestamp) kiinteällä tarkkuudella 0. Aikalaskennassa on aikavälinä voinut käyttää desimaalilukua, jonka kokonaisosa kertoo päivät ja desimaaliosa päivän osat, esimerkiksi 0.5 olisi 12 tuntia. |

Standardin mukainen päiväysliteraalin esitystapa on DATE 'VVVV-KK-PP'. Kellonajan esitystapa on TIME 'HH:MI:SS.FFFFFF' ja aikaleiman TIMESTAMP 'VVVV-KK-PP HH:MI:SS.FFFFFF'. Useissa järjestelmissä on kuitenkin käytössä tästä poikkeavia päiväyksen esitysmuotoja ja muoto saattaa riippua järjestelmän asennuksen kieliasetuksista, joten suomalainenkin päiväysmuoto voi kelvata.

|  |  |  |
| --- | --- | --- |
| syntymäpäivä | date | DATE '1998-10-30' |
| f1 kierrosaika - tuhannesosan tarkkuudella | time(3) | TIME 1:24.301 |
| kirjausaika | timestamp(0) | TIMESTAMP '2001-04-15 13:10:12' |

### Totuusarvo

Totuusarvojen esittämisen on tarjolla tietotyyppi boolean.

|  |  |  |
| --- | --- | --- |
| tutkinto\_suoritettu | boolean | TRUE |

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

**SQL-lauseiden muotokuvauksissa käytettävät merkinnät**

* Punaisella kursiivilla merkityt ovat  elementtejä, joiden tilalle tulee elementin nimen mukainen arvo. Esimerkiksi *taulunimi* tai *table\_name*korvautuu jonkin taulun nimellä.
* Mustalla tekstillä korostettuina tai normaalifontilla kirjoitetut ovat SQL-kielen avainsanoja, esimerkiksi '**create table'.**Pilkut ovat SQL-kieleen kuuluvia erottimia. Normaalit kaarisulut ovat myös SQL-kielen symboleja.
* Hakasulkeiden sisällä ilmaistaan valinnainen lauseen osa, esim. [ *valinnainen osa* ]. Hakasulkeita ei SQL:ssä esiinny.
* Aaltosulkeissa luetellaan vaitoehtoja, jotka erotetaan toisistaan pystyviivalla, esimerkiksi
* { *vaihtoehto 1* | *vaihtoehto 2* | *vaihtoehto 3* }. Aaltosulkeita ei SQL:ssä esiinny.
* Kolme pistettä '**...**' kuvaa edellä esitellyn elementin toistumista. Jos pisteitä edeltää pilkku, erotellaan listan elementit pilkulla.
* Joissakin lauseissa esiintyy lauseen lopussa puolipiste. Tätä käytetään tyhjän rivin lisäksi lauseiden erottamiseen silloin, kun samassa yhteydessä esitellään useita lauseita. SQL:ää käytettäessä lauseen lopettavan puolipisteen tarpeellisuus riippuu suoritusympäristöstä. Useat komentotulkit edellyttävät sitä. Sen sijaan esimerkiksi JDBC-liittymän kautta lauseita suoritettaessa niiden loppuun ei tule puolipistettä.

TIETOKANTOJEN PERUSTEET

## Opetustietojärjestelmän tietokanta

Linkeistä näet taulun sisällön esimerkkikannassa.

create table [Opintojakso](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/opintojakso_taulu.html)(

kurssikoodi varchar(15) not null primary key,

opintoviikot decimal(3,1) not null,

nimi\_suomeksi varchar(50),

nimi\_engl varchar(50)

);

create table [Kurssi](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/kurssi_taulu.html) (

kurssikoodi varchar(15) not null,

lukuvuosi numeric(4) not null,

lukukausi char not null,

kurssinumero smallint not null,

alkamis\_pvm date,

paattymis\_pvm date,

constraint pk\_kurssi primary key

(kurssikoodi, lukuvuosi, lukukausi, kurssinumero),

constraint fk\_kurssi foreign key (kurssikoodi)

references Opintojakso

on update cascade

);

create table [Opetustehtava](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/opetustehtava_taulu.html)(

kurssikoodi varchar(15) not null,

lukuvuosi numeric(4) not null,

lukukausi char not null,

kurssinumero smallint not null,

opetusnumero smallint not null,

tyyppi varchar(20) not null,

tunteja smallint,

alkamis\_pvm date,

paattymis\_pvm date,

constraint pk\_ot

primary key (kurssikoodi, lukuvuosi,

lukukausi, kurssinumero, opetusnumero),

constraint fk\_ot

foreign key (kurssikoodi, lukuvuosi,

lukukausi, kurssinumero)

references Kurssi

on delete cascade

on update cascade

);

create table [Opiskelija](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/opiskelija_taulu.html)(

opiskelijanumero numeric(10) not null,

henkilotunnus char(11),

sukunimi varchar(60) not null,

etunimet varchar(60) not null,

osoite varchar(120),

sahkoposti varchar(120),

viim\_havainto\_pvm date not null,

constraint pk\_opiskelija

primary key (opiskelijanumero)

);

create table [Ilmoittautuminen](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/ilmoittautuminen_taulu.html) (

opiskelijanumero numeric(10) not null,

kurssikoodi varchar(15) not null,

lukuvuosi numeric(4) not null,

lukukausi char not null,

kurssinumero smallint not null,

opetusnumero smallint not null,

ilm\_aika timestamp not null,

perumisaika timestamp,

constraint pk\_ilmo

primary key (opiskelijanumero, ilm\_aika),

constraint fk\_ilmo\_opetus

foreign key

(kurssikoodi, lukuvuosi, lukukausi,

kurssinumero, opetusnumero)

references Opetustehtava

on delete restrict,

constraint fk\_ilmo\_opiskelija

foreign key (opiskelijanumero)

references opiskelija

on delete restrict

);

create table [Opettaja](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/opettaja_taulu.html) (

opetunnus varchar(8) not null,

sukunimi varchar(60) not null,

etunimet varchar(60) not null,

kotiosoite varchar(120),

tyohuone varchar(60),

tyopuhelin varchar(30),

sahkoposti varchar(120),

kotipuhelin varchar(30)

constraint pk\_opettaja

primary key (opetunnus)

);

create table [Tyo](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/tyo_taulu.html)(

kurssikoodi varchar(15) not null,

lukuvuosi numeric(4) not null,

lukukausi char not null,

kurssinumero smallint not null,

opetusnumero smallint not null,

opettaja varchar(8),

tunnit smallint,

constraint pk\_tyo

primary key (kurssikoodi, lukuvuosi,

lukukausi, kurssinumero,

opetusnumero, opettaja),

constraint fk\_tyonkohde

foreign key

(kurssikoodi, lukuvuosi, lukukausi,

kurssinumero, opetusnumero)

references Opetustehtava

on delete cascade,

constraint fk\_tyontekija

foreign key (opettaja)

references opettaja

);

TIETOKANTOJEN PERUSTEET

**SQL-kyselyn rakenne**

|  |  |
| --- | --- |
| **Oheismateriaalia** | * [Elmasri&Navathe](http://www.cs.helsinki.fi/u/laine/tkp/lahteita.html): luku 8.4.1, sivut 218-224 * [Ramakrishnan&Gehrke](http://www.cs.helsinki.fi/u/laine/tkp/lahteita.html): luvut 5.1-5.2 sivut 130-141 * [Laine](http://www.cs.helsinki.fi/u/laine/tkp/lahteita.html): luku 5.4 alku (sivu 45-46) |

SQL kysely esittää tietokantaan kohdistuvan tietotarvemäärityksen. Kyselyn suorituksen tuloksena saadaan **tulostaulu**. Tulostaulu on rakenteeltaan samanlainen kuin  mikä tahansa tietokannan taulu lukuun ottamatta sitä, että

* kyselyn tulostaululla ei ole nimeä
* kyselyn tulostaulu on olemassa vain hetkellisesti ja katoaa kun se on läpikäyty
* kyselyn tulostaulu voi olla järjestetty

SQL-kysely määrittelee tulostaulunsa täysin. Tulostaululla ei siis ole erillistä taulumäärittelyä.

SQL kysely muodostuu seuraavista osista:

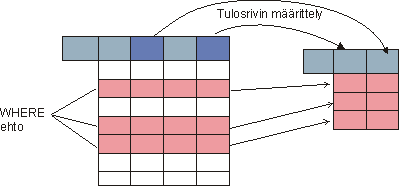
|  |  |
| --- | --- |
| **select***tulosrivin määrittely* | määrittelee sekä tulosrivin rakenteen että datan, joiden perusteella rivit muodostetaan. **Välttämätön.** |
| **from***taulukokoelma* | nimeää kyselyssä käytettävät taulut (vähintään ne taulut joiden sarakearvoja tulee tulosriveille). **Välttämätön.** |
| **where**  *valinta ja liitosehdot* | Rajaa tulosrivien muodostukseen käytettät rivit. Tarvitaan mikäli kyselyyn liittyy kohderivien valintaa tai taulujen välisiä liitoksia. **Valinnainen.** |
| **group by***ryhmitysmääreet* | Määrittelee tulosrivien mahdollisen ryhmittelyn. **Valinnainen.** |
| **having***ryhmien valintaehdot* | Rajaa tulosrivien muodostukseen käytettäviä ryhmiä.**Valinnainen.** |
| **order by** *järjestysmääreet* | Määrittelee tulosrivien järjestyksen. **Valinnainen.** |

**Esimerkki**

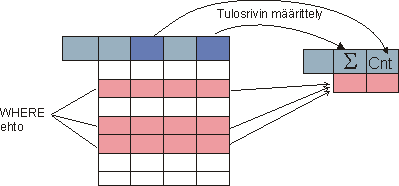
Seuraavalla kyselyllä haetaan taulusta opiskelija sukunimi, etunimet ja sähköpostiosoite kultakin sellaiselta riviltä, jossa viimeisin kirjattu käyttöaika (viim\_havainto\_pvm) on ennen vuotta 1999 ja osoitteessa esiintyy sana Helsinki. Kutakin tällaista opiskelijariviä kohden muodostetaan yksi vastausrivi.

|  |
| --- |
| select sukunimi, etunimet, sahkoposti  from opiskelija  where viim\_havainto\_pvm < date '1999-01-01' and  (osoite like '%Helsinki%')  order by sukunimi, etunimet; |

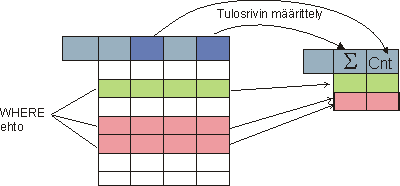
Yksinkertaisissa ryhmittämättömissä kyselyissä muodostetaan yksi tulosrivi jokaista where-osassa annetut ehdot tayttävää riviä tai riviyhdistelmää kohti. Tältä riviltä poimitaan tulosriviin niiden sarakkeiden arvot, jotka on nimetty tulosrivin määrittelyssä (kuva alla).



SQL:ssä on mahdollista tehdä myös yhteenvetokyselyitä. Koko aineistoon perustuvissa yhteenvedoissa muodostetaan vain yksi tulosrivi, jonne lasketaan tulosrivin määrittelyssä annetut yhteenvetotiedot (kuva alla).



Yhteenvetoja on mahdollista tuottaa myös ryhmitellystä aineistosta. Tällöin muodostetaan yksi tulosrivi kutakin ryhmää kohti (kuva alla)



[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

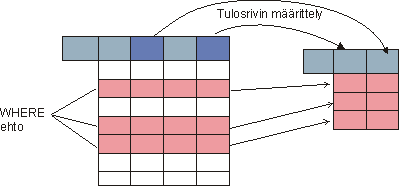
TIETOKANTOJEN PERUSTEET

# Yksinkertaiset kyselyt

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sisältö** | |  | | --- | | [Johdanto](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#yleista) | | [Toistuvat rivit ja niiden karsinta](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#tuplat) | | [Tulosrivin määrittelyn ja ehtolausekkeiden elementit](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#tulosrivi) | | [Sarakenimet](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#sarakenimet) | | [Vakiot](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#vakiot) | | [Aritmeettiset lausekkeet](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#lausekkeet) | | [Merkkijonolausekkeet](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#merkkijonolausekkeet) | | [Skalaarifunktiot](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#funktiot) | | [Tulosrivin määrittelyn erityiskysymyksiä](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#tulosrivi) | | [Sarakkeiden nimeäminen](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#uudelleennimeäminen) | | [Taulun kaikki sarakkeet kerralla](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#tähti) | | [Kyselyn ehto-osa](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#ehto_osa) | | [Kahden arvon keskinäinen vertailu](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#kaksiarvoa) | | [Arvon vertaaminen joukkoon arvoja](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#arvo_ja_joukko) | | [Maskivertailu](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#maski) | | [Arvon puuttuminen](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#olemassaolo) | | [Tulostaulun järjestäminen](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#järjestäminen) | | [Joukko-opin operaatiot osakyselyjen tulostauluille](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#joukkoopin) | |
| [**Oheismateriaalia**](http://www.cs.helsinki.fi/u/laine/tkp/lahteita.html) | * Elmasri&Navathe: luku 8.4., sivut 218-228 * Ramakrishnan&Gehrke: luvut 5.2, 5.3, 5.6, sivut 138-143,162-164 * Laine: luvut 5.4.1-5.4.6, 5.4.9, sivut 46-57, 66-67 |  |

## Johdanto

Yksinkertaisissa ryhmittämättömissä kyselyissä muodostetaan yksi tulosrivi jokaista where-osassa annetut ehdot täyttävää riviä tai riviyhdistelmää kohti. Tältä riviltä poimitaan tulosriviin niiden sarakkeiden arvot, jotka on nimetty tulosrivin määrittelyssä (kuva alla).



Jos kysely kohdistuu vain yhteen tauluun vastaa where-ehdon perusteella tapahtuva rivien poiminta relaatioalgebran  valinta-operaatiota. Mikäli tulosrivin määrittelyssä pelkästään luetellaan poimittavien sarakkeiden nimet, on tulos miltei sama kuin relaatioalgebran projektio-operaatiossa. Erona on se, että **SQL-kyselyn vastauksesta ei automaattisesti karsita pois toistuvia rivejä**kuten tapahtuu projektiossa. Tulostauluun voi siten tulla sama rivi useaan kertaan.

### Toistuvat rivit ja niiden karsinta

Esimerkiksi kysely

|  |
| --- |
| select sukunimi from opiskelija |

antaa kunkin sukunimen niin moneen kertaan kuin nimi esiintyy taulussa opiskelija.

Relaatioalgebran projektiota vastaava toistuvien rivien karsinta saadaan aikaan lisäämällä tulosrivin määrittelyn alkuun (heti select:in jälkeen) avainsana **DISTINCT.**

Näin muokattu kysely

|  |
| --- |
| select **distinct**sukunimi from opiskelija |

antaa kunkin sukunimen vain kertaalleen.

|  |
| --- |
| **Pohdintatehtävä:  Mieti miksi SQL ei automaattisesti karsi toistuvia rivejä**.  Katso [SELITYS](http://www.cs.helsinki.fi/u/laine/tkp/sql/tuplat.html) |

## Tulosrivin määrittelyn ja ehtolausekkeiden elementit

Tulosrivin määrittelyssä ja where-osan ehdoissa voidaan käyttää samoja peruselementtejä:

* sarakenimiä viittaamaan sarakkeisiin
* vakioita
* skalaarifunktioita ja yhteenvetofunktioita
* ylläolevista muodostettuja aritmeettisia- tai merkkijonolausekkeita

Sarakenimi tulosrivin määrittelyssä tarkoittaa sitä, että vastaavan sarakkeen arvo otetaan mukaan tulosriville. Vakio tulosrivin määrittelyssä tarkoittaa sitä, että jokaiselle tulosriville tulee tämä vakio kyseiseen paikkaan. Skalaarifunktiota käytettäessä saadaan tulosriville kyseisen funktion arvo ja lauseketta käytettäessä lausekkeen arvo. Where-osan ehdoissa vertailujen osapuolet määräytyvät vastaavasti.

### Sarakenimet

Sarakenimen kirjoitusasulla ei ole merkitystä. Sarakenimeen **voidaan** liittää alkuun tarkenne, joka kertoo, minkä taulun sarakkeesta on kyse. Tarkenne muodostuu seuraavasti

|  |  |
| --- | --- |
| [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | [*tietokannan tunnus***.**][*kaavion tunnus***.**] *taulun nimi* |
| esimerkki | testi.laine.opiskelija |

### Vakiot

Vakioiden esitystapa riippuu tietotyypistä. Asiaa on käsitelty [perustietotyyppien yhteydessä](http://www.cs.helsinki.fi/u/laine/tkp/sql/tietotyypit.html). Tyhjäarvoon viitataan alvolla NULL (tai null).

### Aritmeettiset lausekkeet

SQL:n aritmeettiset lausekkeet ovat samanlaiset kuin ohjelmointikielissä yleensä. Käytettävissä ovat operaatiot + (yhteenlasku),- (erotus), \* (kertolasku) ja / (jako). Sulkeita voi käyttää laskentajärjetyksen määräämiseksi. Jos aritmeettisen lausekkeen jollakin elementillä on tyhjäarvo (null), niin koko lausekkeen arvoksi muodostuu tyhjäarvo.

|  |  |
| --- | --- |
| esimerkki | 2.5\*(pituus-100)/paino |

Aikatiedoilla voi tehdä yhteen ja vähennyslaskua, esimerkiksi päiväystä, kellonaikaa ja aikaleimaa voidaan kasvattaa tai vähentää kellonaikojen sekä aikaleimojen erotuksella tuottaa aikavälejä. Seuraavassa esimerkissä on käytetty päiväykselle standardin mukaista. Ensimmäisessä esimerkissä päiväykseen lisätään 10 päivää. Toisessa esimerkissä lisätään puoli päivää eli 12 tuntia. Kolmannessa esimerkissä lasketaan aikaleimojen erotus ja muunnetaan

|  |  |
| --- | --- |
| esimerkki | date '2001-04-12' + interval 10 day   = date '2001-04-22'    Oraclessa käy myös date '2001-04-12' + 10  timestamp '2001-04-12 8:45' + interval 12 hour = timestamp '2001-04-12 20:45'  '2001-04-12 16:20' - '2001-04-12 14:00' = interval '2:20' hour to minute  (Oraclessa date tyypeillä laskettaessa tuloksena olisi 0,0972222, eli 140 min /(24\*60) min)) |

### Merkkijonolausekkeet

Merkkijonoille voi tehdä yhteenliittämisoperaation (katenaation). Operaatiosymboli on ||. Seuraavassa esimerkissä sukunimen perään liitetään pilkun ja välilyöntimerkin jälkeen etunimet:

|  |  |
| --- | --- |
| esimerkki | Sukunimi || ', ' || Etunimet |

Useimmissa SQL-toteutuksissa tyhjäarvon osallistuminen yhteenliittämiseen tuottaa tuloksena tyhjäarvon. Oraclessa tyhjäarvon ja jonkin todellisen arvon yhteenliittäminen tuottaa kuitenkin tuon todellisen arvon.

Myös binääritiedolle voi tehdä katenaation.

### Skalaarifunktiot

SQL tarjoaa joukon skalaarifunktioita arvojen käsittelyyn. Standardin tarjoama funktiojoukko on ollut ja on edelleenkin varsin suppea vaikkakin kasvussa. Esimerkiksi vuoden -86 standardissa ei ollut lainkaan skalaarifunktioita. Tästä johtuen tietokannanhallintajärjestelmien toimittajat ovat päätyneet hieman toisistaan poikkeaviin funktioihin. Tästä syystä skalaarifunktioiden käyttö tuo mukanaan lähes aina siirrettävyysongelmia.

Alla olevassa taulukossa on muutamia keskeisiä funktioita sekä standardin että Oraclen vanhalla esitystavalla (Huom.: Oraclen uusimmissa versioissa toimivat myös standardin mukaiset funktiot). Funktioiden parametreina voi käyttää sarakeniminiä, vakioita tai lausekkeita. Oraclessa on merkittävästi enemmän funktioita kuin standardissa.

|  |  |  |
| --- | --- | --- |
| Standardi | Oracle | Merkitys |
| abs( *x*) | abs( *x*) | x:n itseisarvo |
| mod( *x*,y ) | mod( *x*,*y*) | jakojäännös laskusta x/y |
| length(*m*) | length( *m*) | merkkijonon m pituus |
| position( *p*in *m*) | instr( *m*,p ) | merkkijonon p sijaintikohta merkkijonossa m. Huom. SQL:ssä merkkijonon ensimmäisen merkin positio on 1. |
| lower( *m*) | lower(*m*) | merkkijono m pikkukirjaimin |
| upper( *m*) | upper( *m*) | merkkijono m isoin kirjaimin |
| substring( *m*from *b*for *s*) | substr( *m*, *b*, *s*) | m:n kohdasta b alkava ja s merkin pituinen osamerkkijono |
| cast( *x*as *type*) | to\_char( *x*, *format*)  to\_number( *x*, *format*)  to\_date( *x*, *format*) | x:n tietotyypin muunnos |
| case *x* when *y1*then *z1*...  [else *e*]  end | decode( *x*,*y1*,*z1*,... *e*) | valinta x:n arvon perusteella |
| nullif( *x*, *y*) | decode( *x*, *y*,NULL, *x*) | jos x=y niin tyhjä muuten x |
| coalesce( *x*, *y*) | nvl( *x*, *y*) | jos x on tyhjä niin y muuten x |
| extract( *fn*from *t*) | to\_char( *t*, *format*) | aikatiedon t komponentin fn eristys |
| current\_date  current\_time  current\_timestamp | sysdate | nykyhetki |

Seuraavassa esimerkissä haetaan sekä standardin että Oraclen mukaisesti taulusta opiskelija kultakin riviltä sukunimi, josta varmistetaan, että nimi tulee tulokseen isolla alkukirjaimella, sekä tieto hänen viimeisimmästä aktiviteettiajankohdastaan muotoiltuna suomalaiseen päiväysasuun

|  |  |
| --- | --- |
| Standardi | select upper(substring(sukunimi from 1 for 1))||  lower(substring(sukunimi from 2 for 60)),  cast(extract(DAY,viim\_havainto\_pvm),VARCHAR)||'.'||  cast(extract(MONTH,viim\_havainto\_pvm),VARCHAR)||'.'||  cast(extract(YEAR,viim\_havainto\_pvm),VARCHAR)  from opiskelija; |
| Oracle | select initcap(sukunimi),  to\_char(viim\_havainto\_pvm,'DD.MM.YYYY')  from opiskelija; |

SQL-kielessä on tarjolla myös yhteenvetofunktioita. Näitä voi kayttää tulostietomäärittelyssä. Yhteenvetofunktioita käsitellään yhteenvetokyselyjen yhteydessä.

## Tulosrivin määrittelyn erityiskysymyksiä

### Sarakkeiden nimeäminen

Tulostietomäärittelyssä oleva elementti määrittelee paitsi tulostaulun sarakkeen sisällön myös sarakkeen nimen.  Jos elementtinä on sarakenimi tulee sama nimi  tulostaulun sarakkeen nimeksi. Jos elementti on jotain muuta, nimen määräytymisperusteet vaihtelevat järjestelmäkohtaisesti. Oraclessa nimeksi tulee tällöin elementin teksti, esimerkiksi ylläolevan esimerkin tapauksessa tulossarakkeet olisivat nimeltään

* INITCAP(SUKUNIMI) ja
* TO\_CHAR(VIIM\_HAVAINTO\_PVM,'DD.MM.YYYY')

Tällaiset nimet eivät ole kovin käyttökelpoisia. Elementtiin voidaankin liittää uudelleennimeämismääre.

|  |  |
| --- | --- |
| [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | *elementti*[AS] *uusi\_nimi* |
| esimerkki | select initcap(sukunimi) as nimi,  to\_char(viim\_havainto\_pvm,'DD.MM.YYYY') havaintoaika  from opiskelija |

Ylläolevan esimerkin mukaisesti tulostaulun sarakkeiden nimet olisivat

* NIMI ja
* HAVAINTOAIKA

Uudelleennimeäminen on välttämätöntä, jos tulostauluun tulisi muutoin useita samannimisiä sarakkeita

### Taulun kaikki sarakkeet kerralla

Lyhennemerkintää \* (tähti) voidaan käyttää kun tulostauluun halutaan lähtötaulun tai tauluyhdistelmän kaikki sarakkeet. Sarakkeet tulevat tällöin tulostauluun samannimisinä kun ne ovat lähtötauluissa. Tähden eteen voidaan liittää tarkenne kertomaan minkä taulun sarakkeet halutaan mukaan. Seuraava esimerkki hakee koko opiskelija-taulun.

|  |  |
| --- | --- |
| esimerkki | select \* from opiskelija |

## Kyselyn ehto-osa

Kyselyn ehto-osassa ilmaistaan valinta- tai liitosehdot. Ehto-osa where avainsanan jälkeen on rakenteeltaan looginen lauseke, jossa alkeisehtoja voidaan yhdistää loogisilla AND ja OR operaatioilla. Lausekkeissa voidaan käyttää sulkeita määräämään operaatioiden evaluointijärjestys. Ilman sulkeita AND evaluoidaan ennen OR:ia. Negaation saa aikaan NOT-operaatiolla.

Alkeisehtoja on neljää tyyppiä

* kahden arvon vertaamista keskenään
* arvon vertaamista joukkoon arvoja
* arvon vertaamista maskiin
* arvon olemassaolon tutkimista

### Kahden arvon keskinäinen vertailu

Kahden arvon vertailussa käyvät ohjelmointikielistä tutut vertailuoperaatiot: < (pienempi kuin), > (suurempi kuin), = (yhtäsuuri kuin), => (suurempi tai yhtäsuuri kuin), <= (pienempi tai yhtäsuuri kuin) ja != tai <>  (erisuuri kuin). Numerotiedon kohdalla vertailuperustana on luvun suuruusarvo. Merkkitiedon kohdalla joko merkkikoodiarvot tai kansallismerkistöön perustuvien tietotyyppien yhteydessä merkeille kiinnitetty kansallinen järjestys. Aikatietojen järjestyksen määrää ajallinen edeltävyys. SQL osaa jossain määrin tehdä automaattisia tyyppimuunnoksia vertailussa. Esimerkiksi sopivan muotoinen merkkijono voidaan muuttaa päiväykseksi tai numeeriseksi ennen arvon vertaamista päiväykseen tai numeroon.

Tyhjäarvon (NULL) vertaaminen mihin tahansa arvoon, myös toiseen tyhjäarvoon, johtaa tulokseen 'UNKNOWN'. Tämä totuusarvo arvo käyttäytyy loogisissa lausekkeissa seuraavien taulukoiden mukaisesti:

|  |  |  |  |
| --- | --- | --- | --- |
| **AND** | **true** | **false** | **unknown** |
| **true** | true | false | unknown |
| **false** | false | false | false |
| **unknown** | unknown | false | unknown |

|  |  |  |  |
| --- | --- | --- | --- |
| **OR** | **true** | **false** | **unknown** |
| **true** | true | true | true |
| **false** | true | false | unknown |
| **unknown** | true | unknown | unknown |

|  |  |
| --- | --- |
| **NOT** |  |
| **true** | false |
| **false** | true |
| **unknown** | unknown |

### Arvon vertaaminen joukkoon arvoja

Arvon vertaamisen joukoon arvoja voidaan tehdä

* arvovälivertailuna (operaatio BETWEEN ... AND ...)
* joukoon kuulumisen testauksena (operaatiot IN ja NOT IN) tai
* käyttämällä kahden arvon vertailuoperaatioita joukkosovittimen kanssa.

Arvojoukko esitetään luettelemalla joukkoon kuuluvat arvot sulkeiden sisällä pilkulla erotettuna.

|  |  |
| --- | --- |
| joukon esittäminen | ( alkio1, alkio2, ...., alkioN) |

 Joukko voidaan määritellä myös ns. alikyselyn avulla. Tätä määrittelytapaa käsitellään enemmän [alikyselyjen](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely4.html#alikysely)yhteydessä.

Joukkosovittimella kuvataan miten joukon alkioiden on täytettävä annettu vertailuehto. Sovitin SOME edellyttää, että ainakin yksi joukon jäsenistä täyttää ehdon. Sovitin ALL edellyttää, että kaikki joukon jäsenet täyttävät ehdon.

Oletetaan, että sarakkeessa TYYPPI esiintyvät arvot 1,2,3,4,5 ja 6. Seuraavat ehdot ovat tosia kun sarakkeen tyyppi arvo on 3, 4 tai 5.

|  |  |
| --- | --- |
| esimerkki | tyyppi between 3 and 5  tyyppi in (3,4,5)  tyyppi not in (1,2,6)  tyyppi = some (3,4,5)  tyyppi <> all (1,2,6) |

### Maskivertailu

Maskivertailua voi käyttää vain merkkijonojen yhteydessä. Maskissa prosentti-merkki (%) korvaa minkä tahansa merkkijonon ja alaviiva-merkki (\_) minkä tahansa yksittäisen merkin.

|  |  |
| --- | --- |
| esimerkki | sukunimi like 'Vir\_\_'  *'Vir' alkuinen 5 merkkinen*  sukunimi like 'V%nen'  *'V'-alkuinen ja 'nen'loppuinen* |

### Arvon olemassaolo

Arvon olemassaololla tarkoitetaan sitä, että sarakkeessa on jokin muu arvo kuin tyhjäarvo NULL. Olemassaoloa testataan ehtolausekkeella '*sarake*is not NULL' ja tyhjäarvon esiintymistä vastaavasti ehdolla ' *sarake*is NULL'.   
Seuraavassa esimerkissä haetaan tiedot opiskelijoista, joiden osoite ei ole tiedossa.

|  |  |
| --- | --- |
| esimerkki | select \* from opiskelija  where osoite is NULL |

## Tulostaulun järjestäminen

Järjestetty tulos on käyttäjän kannalta järjestämätöntä helpommin hahmotettavissa ja soveltuu paremmin tiedon etsintään. SQL:ssä tietokannan taulujen järjestys on määrittelemätön, mutta kyselyn tulos voidaan järjestää. Järjestyksen määrittely perustuu samoihin elementteihin, joita voidaan käyttää tulostietojen määrittelyyn. Järjestys määritellään liittämällä kyselyyn järjestyksen määrittelyosa (ORDER BY).

Järjestys määritellään luettelemalla järjestyksen määräävät elementit järjestyksessä siten, että luettelon alussa on ensisijainen järjestyksen määrääjä, sitten toissijainen, jne. Toissijainen järjestyksen määrääjä määrää järjestyksen niiden rivien välillä, joilla ensisijaisella järjestyksen määrääjällä on sama arvo.

Järjestys voidaan määritellä kunkin järjestysksen määrääjän mukaan nousevaksi (ascending tai asc, oletusarvo) tai laskevaksi (descending tai desc). Seuraavassa esimerkissä haetaan tietoja opiskelijoista, joista viimeinen havainto on vuoden 1999 jälkeen. Opiskelijarivit järjestetään ensisijaisesti laskevaan järjesteykseen viimeisen havaintoajan mukaan, eli ensiksi tulevat ne, joista on vanhimmat havainnot. Toissijaisena järjestyksen määrääjänä on sukunimi ja kolmannella sijalla etunimet.

|  |  |
| --- | --- |
| esimerkki | select \*  from opiskelija  where viim\_havainto\_pvm>= date '2000-01-01'  order by viim\_havainto\_pvm desc,  sukunimi, etunimet |

Suorita kysely ja katso sen tulostaulua. Järjestys ei ole kovin ilmeinen. Sarakkeiden uudelleenjärjestely tuo järjestyksen paremmin esiin. Tässä kyselyssä on myös käytetty Oraclen to\_char funktiota poimimaan aikaleimasta vain päiväys.

|  |  |
| --- | --- |
| esimerkki | select to\_CHAR(VIIM\_HAVAINTO\_PVM,'DD.MM.YYYY') PVM,  SUKUNIMI,ETUNIMET, OPISKELIJANUMERO AS OPN,  HENKILOTUNNUS AS HETU, OSOITE, SAHKOPOSTI  from opiskelija  where viim\_havainto\_pvm>= date '2000-01-01'  order by viim\_havainto\_pvm desc,  sukunimi, etunimet |

Järjestyksen määrittelyosa on ainoa tapa varmistaa tulosrivien järjestys ja kyselyn toimiminen samalla tavoin esim. tietokannan hallintajärjestelmän versionvaihdon yhteydessä. Jotkin kyselyrakenteet saattavat aiheuttaa tulosrivien järjestämisen useimmissa tietokannan hallintajärjestelmissä. Tällaisia ovat mm. DISTINCT määreen sisällyttäminen tulostietoluetteloon ja GROUP BY osan sisällyttäminen kyselyyn. Tietokannan hallintajärjestelmän optimoijat saattavat kuitenkin muuttua järjestelmän versionvaihdon yhteydessä ja voisi käydä niin, että aina toistuvia rivejä eliminoitaessa uusi versio ei järjestäisikään rivejä kaikissa tapauksissa.

## Joukko-opin operaatiot osakyselyjen tulostauluille

Kyselyjen tulostauluja voidaan käyttää osapuolina joukko-opin operaatioissa. SQL:ssä nämä saadaan aikaan operaatioilla UNION (yhdiste), INTERSECT (leikkaus) ja EXCEPT (erotus, Oraclessa erotusoperaatio on MINUS). Kuten relaatioalgebrassakin täytyy näiden operaatioiden osapuolten olla samarakenteisia. Tulostaulun sarakkeiden nimet määräytyvät ensimmäisen osapuolen perusteella. Kyselyyn voidaan koota useita joukko-opin operaatioilla yhdistettäviä osakyselyjä. Sulkeita voidaan käyttää laskentajärjestyksen määräämiseen.  Joukko-opin operaatioilla yhdistettävät kyselyt voivat sisältää kaikki kyselyn osat lukuuunottamatta ORDER BY -osaa. ORDER BY -osia kyselyssä voi olla vain yksi ja se määrittelee järjestyksen yhdistetylle tulokselle käyttäen ensimmäisen osapuolen sarakenimiä.

SQL:n yhdisteoperaatio poikkeaa SQL:n normaalista toistuvat rivit säilyttävästä käsittelyperiaatteesta ja poistaa toistuvat rivit.

Seuraavassa esimerkissä haetaan sekä opettajien että opiskelijoiden nimet ja osoitteet. Järjestysmääre vaikuttaa koko aineistoon ei vain yhteen osaan. Tulostaulun sarakkeet ovat *nimi*ja *osoite*.

|  |  |
| --- | --- |
| esimerkki | select sukunimi||' '||etunimet nimi, osoite  from opiskelija  **union**  select sukunimi||' '||etunimet nimi, kotiosoite  from opettaja  order by nimi |

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

## Miksi SQL ei automaattisesti karsi toistuvia rivejä

Käytännössä esiintyy usein tilanteita, joissa on välttämätöntä saada arvot käsittelyyn täsmälleen niin monta kertaa kun ne esiintyvät taulussa. Esimerkkinä voisi tarkastella vaikkapa työntekija-taulua, jossa on sarake kuukausipalkka. Taulun perusteella haluttaisiin laskea kuukaudessa yrityksen työntekijöille maksettava kokonaispalkkasumma. Jos sarakkeen kuukausipalkka arvot haetaan projektiolla, saadaan kukin eri kokoinen palkka vain yhteen kertaan. Näistä arvoista laskettava summa ei suinkaan olisi kokonaispalkkasumma vaan normaalisti paljon sitä pienempi, koska useilla työntekijöillä voi olla sama palkka.

TIETOKANTOJEN PERUSTEET

**Liitoskyselyt**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sisältö** | |  | | --- | | [Useita tauluja kyselyn FROM-osassa](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely3.html#useita) | | [Liitosehdon täydellisyys](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely3.html#taydellisyys) | | [Taulujen tilapäinen uudelleennimeäminen](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely3.html#uudelleennimeaminen) | | [Vaihtoehtoinen liitosnotaatio ja ulkoliitos](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely3.html#ulkoliitos) | | [Kyselyjen tehokkuudesta](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely3.html#tehokkuudesta) | |
| [**Oheismateriaalia**](http://www.cs.helsinki.fi/u/laine/tkp/lahteita.html) | * Elmasri&Navathe: luku 8.4., sivut 218-228 * Ramakrishnan&Gehrke: luvut 5.2, 5.3, 5.6, sivut 138-143,162-164 * Laine: luvut 5.4.7, sivut 57-62, 66-67 |  |

**Useita tauluja kyselyn FROM-osassa**

Kun kyselyn FROM-osassa annetaan useita tauluja, on tuloksena näiden taulujen **ristitulo**. Liitos (eli rivien yhdistely ehdon perusteella) saadaan aikaan antamalla liitosehdot kyselyn WHERE-osassa.

Seuraavassa esimerkissä haetaan kevään 2001 'Ohjelmointi 1' kursseille ilmoittautuneiden opiskelijoiden nimet. Tätä varten on tehtävä liitos  taulujen ilmoittautuminen, opintojakso ja opiskelija välillä. Ilmoittautumiset saadaan selville taulusta ilmoittautuminen ja opiskelijoiden nimet taulusta opiskelija.Taulu opintojakso tarvitaan, jotta saataisiin selville kurssin 'Tietokantojen perusteet' kurssikoodi. Taulua kurssi ei tarvita.

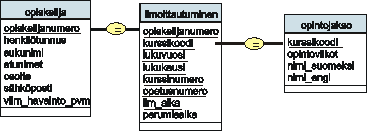
Esimerkki L1 (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select distinct sukunimi, etunimet  from opiskelija, opintojakso, ilmoittautuminen  where  ilmoittautuminen.opiskelijanumero=  opiskelija.opiskelijanumero and  ilmoittautuminen.kurssikoodi=  opintojakso.kurssikoodi and  opintojakso.nimi\_suomeksi=  'Ohjelmointi 1' and  ilmoittautuminen.lukuvuosi=2001 and  ilmoittautuminen.lukukausi='K' and  ilmoittautuminen.perumisaika is null  order by sukunimi, etunimet |

Liitosehdot esimerkissä ovat

*ilmoittautuminen.opiskelijanumero= opiskelija.opiskelijanumero*ja   
*ilmoittautuminen.kurssinumero=opintojakso.kurssinumero*.

Muut ehdot ovat valintaehtoja. Määreellä DISTINCT varmistetaan, että opiskelijan nimi tulee tulokseen vain kertaalleen vaikka hän olisi ilmoittautunut usealle saman opintojakson kurssille. Seuraavassa kuvassa näkyvät taulujen väliset liitosehdot.



Tässä kyselyssä ei tulostauluun oteta mitään tietoja tauluista opintojakso ja ilmoittautuminen. Ne osallistuvat liitokseen vain siksi, että liitoksen avulla saadaan rajattua taulusta opiskelija poimittavia arvoja. Kyselyn FROM osassa voi siis olla tauluja, joista ei oteta mitään tietoja tulostauluun. Kaikki sellaiset taulut, joista otetaan tietoja tulostauluun on lueteltava kyselyn FROM-osassa.

Taulujen järjestyksellä kyselyn FROM-osassa ei ole merkitystä kyselyn vastauksen kannalta. Samoin WHERE-osan ehdot voi antaa missä järjestyksessä tahansa. Näillä kummallakin saattaa kuitenkin järjestelmäkohtaisesti olla merkitystä kyselyn suoritusaikaan, joten erityisesti isoja tauluja käsiteltäessä on syytä selvittää onko taulujen tai ehtojen järjestyksellä merkitystä ja millainen merkitys niillä on.

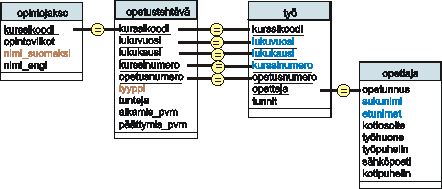
**Liitosehdon täydellisyys**

Koska liitosehdot karsivat riviyhdistelmiä ristitulosta, on oleellista, että liitosehdot ovat täydellisinä mukana WHERE-osassa. Muuten tulokseen tulee ylimääräisiä rivejä. Seuraavassa esimerkissä taulujen liittämiseen tarvitaan moniosaiset liitosehdot. Kyselyllä selvitetään 'Ohjelmointi 1' kurssin luentoja pitäneiden opettajien nimet.

Esimerkki L2. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select tyo.lukuvuosi,tyo.lukukausi,tyo.kurssinumero,  sukunimi, etunimet  from opettaja, opintojakso, opetustehtava, tyo  where  opettaja.opetunnus=tyo.opettaja and  tyo.kurssikoodi= opetustehtava.kurssikoodi and  tyo.lukukausi=opetustehtava.lukukausi and  tyo.lukuvuosi=opetustehtava.lukuvuosi and  tyo.kurssinumero= opetustehtava.kurssinumero and  tyo.opetusnumero= opetustehtava.opetusnumero and  opetustehtava.kurssikoodi=opintojakso.kurssikoodi and  opetustehtava.tyyppi='luento' and  opintojakso.nimi\_suomeksi=  'Ohjelmointi 1'  order by tyo.lukuvuosi,tyo.lukukausi,tyo.kurssinumero |

*Opetustehtava*ja *tyo*taulut liitetään vertaamalla taulussa *tyo*olevaa viiteavainta taulun *opetustehtava*pääavaimeen. Taulujen kytkeytymista havainnollista seuraava kuva



Koska taulun avain on moniosainen (5 saraketta) tarvitaan liitosehtoon viisi alkeisehtoa. Minkä tahansa näistä puuttuminen vääristäisi tuloksen. Jos esimerkiksi ehto

*tyo.opetusnumero= opetustehtava.opetusnumero*

puuttuisi, kytkettäisiin luennointitehtävään kaikki kyseisen kurssin opettajat. Jos taas ehto

*tyo.lukukausi=opetustehtava.lukukausi*

puuttuisi, kytkettäisiin tietyn lukukauden luennointitehtäviin myös eri lukukausina saman numeroista tehtävää hoitavat opettajat. Kokeile miten ehtojen poisottaminen vaikuttaa tulokseen.

Edellisessä esimerkissä samanniminen sarake esiintyy useassa taulussa, joten nimi on tarkennettava sekä tulostaulun määrittelyssä että ehdoissa.

Yleisin taulujen liitostapa on kytkeä taulut yhteen **vertaamalla toisessa taulussa olevaa viiteavainta viitatun taulun pääavaimeen**.

**Taulujen tilapäinen uudelleennimeäminen**

Taulu voidaan kyselyn sisäisesti nimetä uudelleen antamalla sille **viitenimi**(correlation name). Standardin mukaan Viitenimi annetaan kyselyn FROM-osassa taulunimen perässä valinnaisen avainsanan AS jälkeen. Oraclessa avainsanaa 'AS' ei tässä tilanteessa sallita. Jos taululle annetaan viitenimi on kaikki viittaukset tauluun tehtävä käyttäen tätä nimeä. Viitenimeä voi käyttää vaikkapa kirjoitustyön määrän vähentämiseen, esimerkki alla:

Esimerkki L3. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

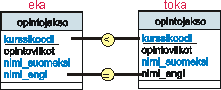
|  |
| --- |
| select tyo.lukuvuosi,tyo.lukukausi,tyo.kurssinumero,  sukunimi, etunimet  from opettaja, opintojakso oj,  opetustehtava ot, tyo  where  opettaja.opetunnus=tyo.opettaja and  tyo.kurssikoodi= ot.kurssikoodi and  tyo.lukukausi=ot.lukukausi and  tyo.lukuvuosi=ot.lukuvuosi and  tyo.kurssinumero= ot.kurssinumero and  tyo.opetusnumero= ot.opetusnumero and  ot.kurssikoodi=oj.kurssikoodi and  ot.tyyppi='luento' and  oj.nimi\_suomeksi=  'Ohjelmointi 1'  order by tyo.lukuvuosi,tyo.lukukausi,tyo.kurssinumero |

Taululle on välttämättä annettava viitenimi, mikäli sama taulu esiintyy from osaan useaan kertaan. Viitenimi erottelee tällöin taulun esiintymät. Seuraavassa esimerkissä yritetään löytää opintojaksopareja, joille on annettu sama englanninkielinen nimi, mutta kyseessä on eri kurssi.

Esimerkki L4. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select eka.nimi\_engl as eng, eka.kurssikoodi as koodi1,  eka.nimi\_suomeksi as nimi1,  toka.kurssikoodi as koodi2, toka.nimi\_suomeksi as nimi2  from opintojakso eka, opintojakso toka  where eka.nimi\_engl=toka.nimi\_engl and  eka.kurssikoodi<toka.kurssikoodi  order by eng |

Seuraava kuva havainnollistaa taulujen käyttöä kyselyssä.

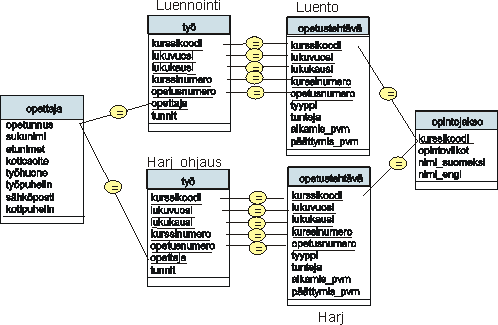


Jos yhdelle tulosriville halutaan saada tietoja usealta saman lähtötaulun riviltä, on lähtötaulu otettava FROM-osaan niin monta kertaa kuin kytkettäviä rivejä on. Jos vaikkapa halutaan tietoja kolmelta riviltä on sama taulu annettava kolmesti. Samoin on meneteltävä, jos saman taulun rivit on muuten kytkettävä yhteen tulosrivin aikaansaamiseksi. Edellä oli esimerkki tällaisesta kytkennästä. Seuraavassa esimerkissä halutaan liitosoperaatioita käyttäen saada selville opettajat, jotka sekä luennoivat että ohjaavat harjoituksia samoilla kevään 2001 kursseilla. Yksinkertaisempia tapoja tämän kyselyn esittämiseen käsitellään myöhemmin.

Esimerkki L5. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select  distinct sukunimi, etunimet, nimi\_suomeksi  from opettaja, opintojakso oj,  opetustehtava luento,  opetustehtava harj,  tyo luennointi, tyo harj\_ohjaus  where  opettaja.opetunnus=luennointi.opettaja and  opettaja.opetunnus=harj\_ohjaus.opettaja and  luennointi.kurssikoodi= luento.kurssikoodi and  luennointi.lukukausi=luento.lukukausi and  luennointi.lukuvuosi=luento.lukuvuosi and  luennointi.kurssinumero= luento.kurssinumero and  luennointi.opetusnumero= luento.opetusnumero and  luento.kurssikoodi=oj.kurssikoodi and  luento.tyyppi='luento' and  harj\_ohjaus.kurssikoodi=harj.kurssikoodi and  harj\_ohjaus.lukukausi= harj.lukukausi and  harj\_ohjaus.lukuvuosi= harj.lukuvuosi and  harj\_ohjaus.kurssinumero= harj.kurssinumero and  harj\_ohjaus.opetusnumero= harj.opetusnumero and  harj.kurssikoodi=oj.kurssikoodi and  harj.tyyppi='harjoitus' and  luento.kurssikoodi=harj.kurssikoodi and  luento.lukukausi=harj.lukukausi and  luento.lukuvuosi=harj.lukuvuosi and  luento.kurssinumero= harj.kurssinumero and  luennointi.lukuvuosi=2001 and  luennointi.lukukausi='K' |

Esimerkin kysely sisältää varsin monimutkaiset liitosehdot. Alla oleva kuva havainnollistaa kyselyä.



**Vaihtoehtoinen liitosnotaatio ja ulkoliitos**

SQL-92 standardissa esiteltiin vaihtoehtoinen tapa liitoksen määrittelyyn. Tämä esitystapa mahdollistaa myös ulkoliitoksen esittämisen. Esitystavassa kyselyn FROM-osaan voidaan ottaa liitostulos.

Esimerkki L6. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |  |
| --- | --- |
| [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | *taulu1 liitostapa*JOIN *taulu2*ON *liitosehdot* |
| esimerkki | kurssi INNER JOIN opintojakso  ON kurssi.kurssikoodi=opintojakso.kurssikoodi |

Liitostapoja on määritelty neljä:

* inner: normaaliliitos
* left outer: vasemmanpuoleinen ulkoliitos
* right outer: oikeanpuoleinen ulkoliitos
* full outer: molemminpuoleinen ulkoliitos

Alla esimerkki ulkoliitoksesta. Esimerkissä laaditaan luettelo Opintojaksojen opetuksesta eri lukukausina. Tulokseen halutaan myös sellaiset opintojaksot, joita ei ole opetettu.

Esimerkki L7. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select opintojakso.nimi\_suomeksi nimi, opintojakso.kurssikoodi koodi,  kurssi.lukuvuosi, kurssi.lukukausi  from opintojakso left outer join kurssi  on kurssi.kurssikoodi=opintojakso.kurssikoodi  order by opintojakso.nimi\_suomeksi |

**Kyselyjen tehokkuudesta**

Valinnat ja erityisesti liitokset ovat keskeisesti kyselyjen tehokkuuteen vaikuttava tekijä.  Liitosoperaatiossa riville on löydettävä ehdon täyttävät parit joko samasta tai jostain toisesta taulusta. Liitoksen muodostamiseen on käytössä useita strategioita, joiden valinta riippuu siitä minkälaisia teknisen tason rakenteita tauluilla on ja minkä kokoisia taulut ovat.

Rivien hakua taulusta jonkin sarakkeen perusteella voidaan tehostaa määrittelemällä kyseiseen sarakkeeseen perustuva hakuindeksi (index). Vertailukohdaksi käy kirjan avainsanojen hakemisto. Avainsarakkeeseen perustuva hakuindeksi nopeuttaa avaimen perusteella tapahtuvia hakuja merkittävästi. Jos esimerkiksi taulussa olisi 10000 riviä, niin jonkin satunnaisen rivin haku ilman hakuindeksiä vaatisi, että käydään läpi keskimäärin puolet taulusta. Usein yhteen levymuistin lohkoon mahtuu useita taulun rivejä. Oletetaan, että tässä esimerkkitilanteessa lohkoon mahtuisi 10 riviä. Keskimääräinen hakuaika olisi tällöin ilman hakuindeksiä noin 500 levyhaun vaatima aika. Hakuindeksiä käytettäessä päästään tämän kokoisella taululla noin kolmeen (3) levyhakuun. Avaimen perusteella tapahtuvat haut ovat tarpeen esimerkiksi tilanteissa, joissa liitos tehdään avainta ja siihen viittaavaa viiteavainta vertaamalla. Avaimeen perustuva haku joudutaan tekemään myös aina lisäyksen yhteydessä avaimen yksikäsitteisyyden varmistamiseksi. Jotkin järjestelmät, esim. Oracle, tekevätkin automaattisesti hakuindeksin taulun avaimen perusteella tarvitsematta mitään muita määrityksiä. Hakuindeksi voi perustua yhteen tai useampaan sarakkeeseen.

Hakuindeksien määrittelyn syntaksi ei sisälly SQL-standardiin. Yleinen indeksin määrittely on kuitenkin rakenteeltaan seuraava

|  |  |
| --- | --- |
| [muoto](http://www.cs.helsinki.fi/u/laine/tkp/sql/muoto.html) | create [unique] index *index\_name*on *table\_name*( *column\_name 1*[, **...**]) |
| esimerkki | create unique index op\_index on opettaja(opetunnus) |

Unique-määre hakuindeksin määrittelyssä tarkoittaa sitä, että sama indeksoitavan sarakkeen arvo ei voi esiintyä taulussa kuin yhdellä rivillä. Esimerkiksi avain täyttää tämän ehdon.

Hakuindeksit eivät varsinaisesti kuulu tämän kurssin sisältöön, mutta niiden määrittelyyn on syytä perehtyä, jos ollaan tekemisissä isojen taulujen kanssa. Hakuindeksin käyttö ei näy mitenkään kyselyssä. Sen hyväksikäyttö on kyselyn optimoijan tehtävä, mutta indeksien määrittely ja perustaminen on tietokannan suunnittelijan vastuulla.

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

# Alikyselyt

|  |  |  |
| --- | --- | --- |
| [**Oheismateriaalia**](http://www.cs.helsinki.fi/u/laine/tkp/lahteita.html) | * Elmasri&Navathe: luku 8.5.1-8.5.6, sivut 229-238 * Ramakrishnan&Gehrke: luvut 5.4, sivut 141-149 * Laine: luvut 5.4.8, sivut 63-66 |  |

Alikyselyllä (subquery) tarkoitetaan kyselyyn upotettua toista kyselyä. SQL:ssä alikyselyjä on mahdollista käyttää kyselyn WHERE-osassa ja SQL-92 versiosta lähtien myös kyselyn FROM-osassa.

WHERE-osassa alikyselyt ovat varsin hyödyllisiä. Alikysely tuottaa tuloksenaan taulun (=joukon rivejä). SQL:n ehtolausekkeissa on mahdollista[verrata arvoa joukkoon arvoja](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely2.html#arvo_ja_joukko). Alikyselyä voidaan käyttää määrittelemään tällaisen vertailun joukko-osapuolen arvot. Seuraavassa esimerkissä haetaan niiden opiskelijoiden nimet, jotka eivät ole ilmoittautuneet millekään syksyn 2001 kurssille. Alikyselyllä haetaan niiden opiskelijoiden tunnukset, jotka ovat ilmoittautuneet jollekin kevään 2001 kurssille ja valintaehtona kyselyssä on, että valittavien opiskelijoiden tunnukset eivät kuulu tähän arvojen joukkoon.

Esimerkki AK1. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select sukunimi, etunimet  from opiskelija  where  opiskelijanumero not in  (select opiskelijanumero  from ilmoittautuminen  where lukuvuosi=2001 and  lukukausi='K')  order by sukunimi, etunimet |

Kun arvoa verrataan vakiona annettuun joukkoon tulevat vertailuoperaatioina kyseeseen vain yhtäsuuruus ja erisuuruus. Alikyselyllä haettuun arvojoukkoon arvoa voidaan verrata myös muita operaatioita käyttäen. Seuraavassa esimerkissä selvitetään, minkä kurssin opintoviikkomäärä on suurin.

Esimerkki AK2. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select nimi\_suomeksi,opintoviikot  from opintojakso  where opintoviikot>=  all (select opintoviikot  from opintojakso) |

Tässä kyselyssä alikyselyllä haetaan kaikkien kurssien opintoviikkomäärät. Valintaehdossa vaaditaan, että valittavan kurssin opintoviikkomäärän on oltava suurempi tai yhtäsuuri kuin jokainen alikyselyn tuottama arvo. Suurimmalle opintoviikkomäärälle tämä ehto on voimassa, muille ei, koska niiden kohdalla löytyy jokin suurempi.

Alikyselyä ja IN-operaatiota voidaan usein käyttää liitosoperaation asemasta. IN- ja NOT IN vertailun yhteydessä voidaan yksittäisen arvon vertaamisen lisäksi verrata myös arvojonoa monisarakkeisen alikyselyn tulokseen. Arvojono annetaan tällöin sulkeiden rajaamana ja alikyselyn tulostaulussa on oltava sama määrä sarakkeita kuin arvojonossa on arvoja. Esimerkisi kevään 2001 harjoitukset, joihin ei ole ilmoittautut ketään, saadaan seuraavalla kyselyllä:

Esimerkki AK3. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select \* from opetustehtava  where tyyppi='harjoitus' and  lukuvuosi=2001 and lukukausi='K' and  (kurssikoodi, kurssinumero, opetusnumero) not in  (select kurssikoodi, kurssinumero, opetusnumero  from ilmoittautuminen  where lukuvuosi=2001 and  lukukausi='K') |

Kyselyssä, valitaan ne kevään 2001 harjoitusryhmät, joissa esiintyvää *(kurssikoodi, kurssinumero, opetusnumero)* -yhdistelmää ei löydy kevään 2001 ilmoittautumisten joukosta. Alikyselyllä muodostetaan siis komplementtijoukko 'harjoitusryhmät, joihin on ilmoittauduttu', ja vastaus on perusjoukon ja komplementin erotus, joka saadan 'NOT IN'-operaatiolla. Tämä on hyvin käyttökelpoinen kyselyrakenne tilanteissa, joissa pitäisi selvittää, että jotain ei ole tapahtunut.

### Kytketyt alikyselyt

Edelliset alikyselyt ovat olleet riippumattomia ulommasta kyselystä. Periaatteessa alikyselyn on voinut suorittaa ensin ja käyttää sen tulosta vakion kaltaisesti ulommassa kyselyssä. SQL-lauseen WHERE-osassa on mahdollista esittää myös alikyselyitä, jotka kytkeytyvät ulompaan kyselyyn. Tällöin alikyselyn tulos lasketaan erikseen kutakin ulomman kyselyn vastausriviehdokasta kohti. Seuraavassa esimerkissä halutaan saada selville syksyn 2001 kursseille ilmoittautuneet opiskelijat, jotka ovat ilmoittautuneet saman opintojakson kurssille myös keväällä 2001 eivätkä ole peruneet ilmoittautumistaan. Myös kurssin koodi halutaan tulokseen.

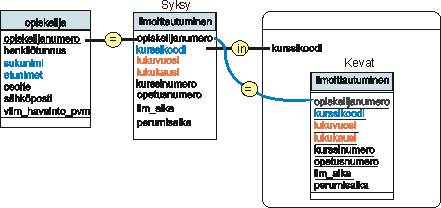
Esimerkki AK4. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select sukunimi, etunimet, kevat.kurssikoodi  from opiskelija, ilmoittautuminen syksy  where  opiskelija.opiskelijanumero=syksy.opiskelijanumero and  syksy.lukuvuosi=2001 and  syksy.lukukausi='S' and  syksy.kurssikoodi in  (select kevat.kurssikoodi  from ilmoittautuminen kevat  where kevat.lukukausi='K' and  kevat.lukuvuosi=2001 and  kevat.perumisaika is null and  kevat.opiskelijanumero=opiskelija.opiskelijanumero) |

Tässä esimerkissä kytkentä tapahtuu ehdon

*kevat.opiskelijanumero=opiskelija.opiskelijanumero*

avulla. Täten alikyselyn tulos lasketaan uudelleen jokaista käsiteltävää opiskelijaa kohti. Seuraava kuva havainnollistaa kyselyä.

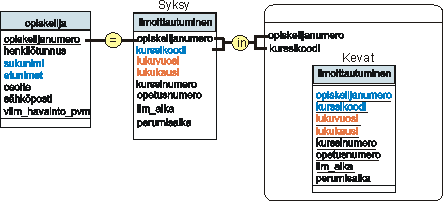


Tämän kyselyn voi esittää myös riippumattoman alikyselyn avulla:

Esimerkki AK5. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select sukunimi, etunimet, syksy.kurssikoodi  from opiskelija, ilmoittautuminen syksy  where  opiskelija.opiskelijanumero=syksy.opiskelijanumero and  syksy.lukuvuosi=2001 and  syksy.lukukausi='S' and  (syksy.kurssikoodi,syksy.opiskelijanumero) in  (select kevat.kurssikoodi, kevat.opiskelijanumero  from ilmoittautuminen kevat  where kevat.lukukausi='K' and  kevat.lukuvuosi=2001 and  kevat.perumisaika is null) |

Seuraava kuva havainnollistaa kytkentöjä.



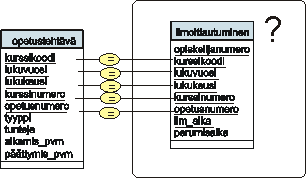
### Tuottaa alikysely tyhjän tuloksen?

Sitä tuottaako alikysely tulosrivejä vaiko tyhjän tuloksen voi testata käyttäen predikaatteja EXISTS ja NOT EXISTS. Näistä edellinen on tosi, jos alikysely tuottaa vähintään yhden tulosrivin ja jälkimmäinen on tosi, jos alikysely tuottaa tyhjän tuloksen. Alikyselyn tyhjyyden testaaminen on mielekästä vain kytkettyjen alikyselyiden yhteydessä, sillä riippumattomien alikyselyiden tapauksessa, alikyselyn tulos on sama kaikkien ulomman kyselyn rivien kohdalla. Seuraavassa esimerkissä selvitetään jälleen kevään 2001 harjoitukset, joihin ei ole ilmoittautut ketään.

Esimerkki AK6. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select \* from opetustehtava  where tyyppi='harjoitus' and  lukuvuosi=2001 and lukukausi='K' and  not exists  (select \*  from ilmoittautuminen  where kurssikoodi=opetustehtava.kurssikoodi and  kurssinumero=opetustehtava.kurssinumero and  opetusnumero=opetustehtava.kurssinumero and  lukuvuosi=opetustehtava.lukuvuosi and  lukukausi=opetustehtava.lukukausi) |

Seuraava kuva havainnollistaa kyselyssä köytettyjä kytkentöjä.



Alikyselyn käytöstä kyselyn FROM-osassa on esimerkkejä yhteenvetokyselyjen yhteydessä.

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

TIETOKANTOJEN PERUSTEET

# Yhteenvetokyselyt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sisältö** | |  | | --- | | [Yhteenvetofunktiot](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely5.html#yhteenvetofunktiot) | | [Koko aineistoon perustuvat yhteenvedot](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely5.html#Kokoaineisto) | | [Ryhmiteltyyn aineistoon perustuvat yhteenvedot](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely5.html#ryhmittelty) | | [Ryhmien valitseminen](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely5.html#ryhmienvalitseminen) | |
| [**Oheismateriaalia**](http://www.cs.helsinki.fi/u/laine/tkp/lahteita.html) | * Elmasri&Navathe: luku 8.5.7-8.5.9, sivut 238-245 * Ramakrishnan&Gehrke: luvut 5.5, sivut 151-161 * Laine: luvut 5.4.10, sivut 67-78 |  |

### Yhteenvetofunktiot

SQL:ssä on mahdollista tehdä yhteenvetokyselyitä koko vastausaineistoon perustuen tai ryhmiteltyyn vastausaineistoon perustuen. Vastausaineistolla tarkoitetaan niitä rivejä, joista tulosrivit muodostettaisiin ellei yhteenvetotietoja laskettaisi, eli WHERE-osan ehdot täyttäviä riviyhdistelmiä.

Yhteenvetotietojen laskemiseksi SQL tarjoaa joukon yhteenvetofunktioita (taulukko alla). SQL-99 standardissa on tullut lisää tilastollisia yhteenvetofunktioita, esim. keskihajonta.

|  |  |
| --- | --- |
| AVG( *lauseke*) | numeerisen lausekkeen keskiarvo |
| SUM( *lauseke*) | numeerisen lausekkeen summa |
| MIN( *lauseke*) | lausekkeen minimiarvo |
| MAX( *lauseke*) | lausekkeen maksimiarvo |
| COUNT( *lauseke*) | lausekkeen arvojen lukumäärä |
| COUNT(\*) | vastausaineiston rivien lukumäärä |

Yhteenvetofunktioiden argumenttina voi olla lauseke. Summa ja keskiarvo edellyttävät numeerista lauseketta. Lausekkeen arvo lasketaan normaalisti jokaiselta vastausaineiston riviltä ja saatu tulos otetaan mukaan yhteenvedon laskemiseen. Tyhjäarvot (NULL) eivät osallistu laskentaan. Täten esimerkiksi COUNT antaa sarakkeessa olevien todellisten arvojen lukumäärän.

Argumentin eteen voidaan liittää määre DISTINCT. Tällöin sama arvo otetaan yhteenvedon laskennassa huomioon vain yhteen kertaan. Tällä on käyttöä lähinnä COUNT-funktion yhteydessä, jolloin tulokseksi saadaan sarakkeessa olevien erilaisten arvojen lukumäärä. Tarkastellaan seuraavaa taulua ESIMERKKI.

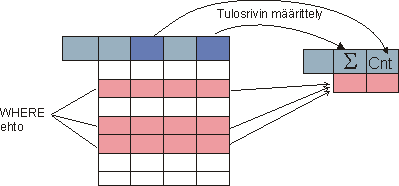
|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | D |
| 3 | ABC | NULL | 5 |
| 4 | NULL | NULL | 1 |
| 4 | DEF | 5 | 5 |
| 4 | DEH | 8 | 3 |
| 2 | NULL | NULL | 2 |
| 7 | HUP | 2 | 1 |
| 2 | PUH | 1 | 1 |

Tauluun esimerkki perustuen saadaan seuraavat yhteenvetofunktioiden arvot:

|  |  |
| --- | --- |
| COUNT(A) | 7 |
| COUNT(DISTINCT A) | 4 |
| COUNT(B) | 5 |
| COUNT(DISTINCT B) | 5 |
| COUNT(C+D) | 4 |
| SUM(C) | 16 |

### Koko aineistoon perustuvat yhteenvedot

Koko aineistoon perustuvissa yhteenvedoissa muodostetaan vain yksi tulosrivi, jonne lasketaan tulosrivin määrittelyssä annetut yhteenvetotiedot (kuva alla).



Seuraavassa esimerkissä selvitetään opiskelijoiden lukumäärä.

Esimerkki Y1. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select count(\*) from opiskelija |

Taulussa opetustehtävä sarake *tunteja*kertoo tehtävän kokonaistuntimäärän. Seuraavassa esimerkissä lasketaan kurssin 'Ohjelmointi 1' harjoituksiin keväällä 2001 varattu kokonaistuntimäärä, sekä harjoitusryhmien lukumäärä sekä harjoitusten pitämisestä maksettava kokonaispalkkio, kun tunnilta maksetaan 183 markkaa (oletaan, että kutakin ryhmää varten on määritelty yksi 'harjoitus'-opetustehtävä).

Esimerkki Y2. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select sum(tunteja) tunteja\_yht,  count(\*) ryhmia, sum(183\*tunteja) kokonaispalkka  from opetustehtava, opintojakso  where lukuvuosi=2001 and  lukukausi='K' and  tyyppi='harjoitus' and  opintojakso.kurssikoodi= opetustehtava.kurssikoodi and  opintojakso.nimi\_suomeksi= 'Ohjelmointi 1' |

Koko aineistoon perustuviin yhteenvetokyselyihin ei saa otettua yhteenvetotietojen lisäksi mitään yksittäisten rivien sarakearvoja. Seuraavalla kyselyllä saadaan selville kurssien suurin opintoviikkomäärä.

Esimerkki Y3. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select max(opintoviikot) suurin  from opintojakso |

Kuitenkaan seuraava kysely ei tuota tulokseksi kurssia, jonka opintoviikkomäärä on suurin, vaan tuloksena on ilmoitus syntaksivirheestä, sillä kyselyssä yritetään hakea sekä yhteenvetotietoa että detaljitietoa.

Esimerkki Y4. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html)HUOM: VIRHEELLINEN

|  |
| --- |
| select nimi\_suomeksi, max(opintoviikot) suurin  from opintojakso |

Ylläolevaan kyselyyn oli aiemmin esillä jo [yksi toimiva ratkaisu](http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely4.html#suurin). Seuraavassa on toinen vaihtoehto, jossa käytetään hyväksi kyselyn FROM-osaan upotettua alikyselyä. Alikysely, jonka tulostaululle annetaan tilapäisnimi 'isoin'  hakee suurimman opintoviikkomäärän yhteenvetokyselyllä. Tässä tulostaulussa on vain yksi rivi. Tulostaulun ja taulun opintojakso välille tehdään liitos.

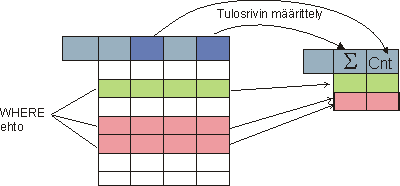
Esimerkki Y5. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select nimi\_suomeksi, suurin  from opintojakso,  (select max(opintoviikot) suurin  from opintojakso) isoin  where opintojakso.opintoviikot= isoin.suurin |

|  |
| --- |
| Tee tehtävät  * SQL-trainer tehtävät y1 - y7 |

### Ryhmiteltyyn aineistoon perustuvat yhteenvedot

Kun yhteenveto tehdään ryhmitellystä aineistosta, lasketaan yhteenvetofunktioiden arvot jokaista ryhmittelyn tuloksena syntynyttä ryhmää kohden. Tulostaulun rivien määrä on sama kuin ryhmien lukumäärä. Alla olevassa kuvassa on kaksi ryhmää, josta kummastakin tuotetaan tulosrivi.



Vastausaineisto muodostetaan soveltamalla WHERE-osan ehtoja kyselyn FROM-osan tauluihin. Tämän jälkeen tehdään ryhmittely GROUP BY -määreellä annettujen sarakkeiden arvojen perusteella. Jokainen erilainen vastausaineistosta löytyvä arvoyhdistelmä määrä ryhmän. Tarkastellaan vastausaineistona seuraavaa taulua:

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | D |
| 3 | ABC | NULL | 5 |
| 4 | NULL | NULL | 1 |
| 4 | DEF | 5 | 5 |
| 4 | DEH | 8 | 3 |
| 2 | NULL | NULL | 2 |
| 7 | HUP | 2 | 1 |
| 2 | PUH | 1 | 1 |

Ryhmittelymääre GROUP BY A tuottaa 4 ryhmää sarakkeen A arvojen perusteella

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | D |
| 2 | NULL | NULL | 2 |
| 2 | PUH | 1 | 1 |
| 3 | ABC | NULL | 5 |
| 4 | NULL | NULL | 1 |
| 4 | DEF | 5 | 5 |
| 4 | DEH | 8 | 3 |
| 7 | HUP | 2 | 1 |

Alla olevassa taulukossa on muutamia tähän ryhmittelyyn perustuvia yhteenvetotuloksia

|  |  |
| --- | --- |
| count(C) | min(C) |
| 1 | 1 |
| 0 | NULL |
| 2 | 5 |
| 1 | 2 |

Ryhmittelymääre GROUP BY C tuottaa 5 ryhmää:

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | D |
| 2 | PUH | 1 | 1 |
| 7 | HUP | 2 | 1 |
| 4 | DEF | 5 | 5 |
| 4 | DEH | 8 | 3 |
| 3 | ABC | NULL | 5 |
| 4 | NULL | NULL | 1 |
| 2 | NULL | NULL | 2 |

Tähän ryhmittelyyn perustuen saadaan seuraavia yhteenvetotuloksia

|  |  |  |
| --- | --- | --- |
| MIN(B) | MAX(B) | SUM(D) |
| PUH | PUH | 1 |
| HUP | HUP | 1 |
| DEF | DEF | 5 |
| DEH | DEH | 3 |
| ABC | ABC | 8 |

Ryhmittelyyn perustuvien yhteenvetokyselyiden tulostauluun voidaan ottaa yhteenvetofuntioiden ja vakioiden lisäksi ryhmittelyperustana olevien sarakkeiden arvoja. Nämä arvot ovat samoja kaikissa ryhmän riveissä.

Seuraavassa kyselyssä selvitetään eri kursseille keväällä 2001 ilmoittauneiden opiskelijoiden lukumäärät.

Esimerkki Y6. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select nimi\_suomeksi, opintojakso.kurssikoodi,  count(\*) opiskelijoita  from opintojakso, ilmoittautuminen  where lukuvuosi=2001 and  lukukausi='K' and  perumis\_pvm is null and  opintojakso.kurssikoodi=ilmoittautuminen.kurssikoodi  group by nimi\_suomeksi, opintojakso.kurssikoodi |

Kyselyn GROUP BY -sarakkeina ovat NIMI\_SUOMEKSI ja KURSSIKOODI. KURSSIKOODI yksinään tuottaa saman ryhmittelyn kuin yllä käytetty yhdistelmä. Koska tulostauluun saa yhteenvetotulosten lisäksi vain sellaisten sarakkeiden arvoja, jotka esiintyvät GROUP BY -määreessä, on käytettävä kyselyssä olevaa yhdistelmää, jos kurssin nimi halutaan mukaan. Tämän kyselyn ongelma on se, että kaikki kevään 2001 kurssit eivät välttämättä tule mukaan tulokseen. Tulostaulun ulkopuolelle jäävät kurssit, joihin ei ole ilmoittautunut ketään. Syy tähän on se, että kyseisille opintojaksoille ei löydy paria ilmoittautuminen-taulusta ja näin liitos karsii opintojaksot. Ongelma voidaan hoitaa ulkoliitoksella seuraavasti

Esimerkki Y7. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select nimi\_suomeksi, kurssi.kurssikoodi,  count(ilm\_aika) opiskelijoita  from opintojakso,  kurssi left outer join ilmoittautuminen i  on kurssi.kurssikoodi= i.kurssikoodi and  kurssi.lukuvuosi= i.lukuvuosi and  kurssi.lukukausi=i.lukukausi and  kurssi.kurssinumero= i.kurssinumero  where kurssi.lukuvuosi=2001 and  kurssi.lukukausi='K' and  perumisaika is null and  opintojakso.kurssikoodi=kurssi.kurssikoodi  group by nimi\_suomeksi, kurssi.kurssikoodi |

Ylläoleva ei toimi Oraclessa, alla vastaava Oracle-versio kokeiltavaksi.

|  |
| --- |
| select nimi\_suomeksi, kurssi.kurssikoodi,  count(ilm\_aika) opiskelijoita  from opintojakso,  kurssi, ilmoittautuminen i  where  kurssi.kurssikoodi= i.kurssikoodi (+) and  kurssi.lukuvuosi= i.lukuvuosi (+) and  kurssi.lukukausi=i.lukukausi (+) and  kurssi.kurssinumero= i.kurssinumero (+) and  kurssi.lukuvuosi=2001 and  kurssi.lukukausi='K' and  perumisaika is null and  opintojakso.kurssikoodi=kurssi.kurssikoodi  group by nimi\_suomeksi, kurssi.kurssikoodi |

Ryhmittely toteutetaan yleensä siten, että vastausaineisto järjestetään ryhmityssarakkeiden perusteella ja näin saadaan samaan ryhmään kuuluvat vastausaineiston rivit peräkkäin, jolloin yhteenvetotiedot on helppo laskea. Jos halutaan varmistaa, että vastaus tulee aina halutussa järjestyksessä, on kyselyyn syytä liittää ORDER BY -määre. Seuraavassa esimerkissä on edelliseen Oracle-esimerkkiin liitetty järjestysmääre, jolla saadaan kurssit osallistujamäärän mukaan laskevassa järjestyksessä, eli suurin kurssi ensin.

Esimerkki Y8. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select nimi\_suomeksi, kurssi.kurssikoodi,  count(ilm\_aika) opiskelijoita  from opintojakso,  kurssi, ilmoittautuminen i  where  kurssi.kurssikoodi= i.kurssikoodi (+) and  kurssi.lukuvuosi= i.lukuvuosi (+) and  kurssi.lukukausi=i.lukukausi (+) and  kurssi.kurssinumero= i.kurssinumero (+) and  kurssi.lukuvuosi=2001 and  kurssi.lukukausi='K' and  perumisaika is null and  opintojakso.kurssikoodi=kurssi.kurssikoodi  group by nimi\_suomeksi, kurssi.kurssikoodi  order by opiskelijoita desc |

### Ryhmien valitseminen

Kyselyyn liitettävällä HAVING -osalla on mahdollista rajoittaa tulokseen otettavia ryhmiä. HAVING-osassa ryhmää koskeva rajoite annetaan yleensä johonkin yhteenvetotietoon perustuvana (muihin kuin yhteenvetotietoihin perustuvat rajoitteet voidaan antaa WHERE-osassa). Seuraavassa täydennetään edellistä esimerkkiä ja valitaan vain sellaiset kurssit joiden osallistujamäärä on yli 5.

Esimerkki Y9. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select nimi\_suomeksi, kurssi.kurssikoodi,  count(ilm\_aika) opiskelijoita  from opintojakso,  kurssi, ilmoittautuminen i  where  kurssi.kurssikoodi= i.kurssikoodi (+) and  kurssi.lukuvuosi= i.lukuvuosi (+) and  kurssi.lukukausi=i.lukukausi (+) and  kurssi.kurssinumero= i.kurssinumero (+) and  kurssi.lukuvuosi=2001 and  kurssi.lukukausi='K' and  perumisaika is null and  opintojakso.kurssikoodi=kurssi.kurssikoodi  group by nimi\_suomeksi, kurssi.kurssikoodi  having count(ilm\_aika)>5  order by opiskelijoita desc |

Seuraavassa kyselyssä halutaan saada selville millä lukukaudella kokonaistyömäärä on ollut suurin, kun harjoitusten työmääräkerroin kokonaistyöaikaa laskettaessa on 3 luentojen 6. Rakennetaan kysely osista. Ensin laaditaan kysely, joka selvittää harjoituksista aiheutuvan kokonaistyöajan lukukausittain

Esimerkki Y10. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select lukuvuosi, lukukausi, sum(3\*tunteja) ht  from opetustehtava  where  tyyppi='harjoitus'  group by lukuvuosi, lukukausi |

Vastaavalla tavalla saadaan luentotunneista aiheutuva kokonaistyömäärä. Jotta pystyttäisiin laskemaan lukukausittainen kokonaistyöaika, pitää harjoitusten kokonaistyömäärä ja luentojen kokonaistyömäärä saada samalle riville Seuraavassa tämä saadaan aikaan FROM- osaan upotetuilla alikyselyillä, joista toinen tuottaa harjoitusten ja toinen luentojen kokonaistyömäärät.

Esimerkki Y11. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select luennot.lukuvuosi, luennot.lukukausi, ht+lt  from (select lukuvuosi, lukukausi, sum(3\*tunteja) ht  from opetustehtava  where tyyppi='harjoitus'  group by lukuvuosi, lukukausi) harkat,  (select lukuvuosi, lukukausi, sum(6\*tunteja) lt  from opetustehtava  where tyyppi='luento'  group by lukuvuosi, lukukausi) luennot  where harkat.lukuvuosi=luennot.lukuvuosi and  harkat.lukukausi=luennot.lukukausi |

Lopputuloksena haluttiin vain se lukukasui, jolla kokonaistyömäärä on suurin. Tässä ei voida käyttää max-funktiota seuraavsasti:

Esimerkki Y12. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html)) HUOM: VIRHEELLINEN

|  |
| --- |
| select luennot.lukuvuosi, luennot.lukukausi, max(ht+lt)  from (select lukuvuosi, lukukausi, sum(3\*tunteja) ht  from opetustehtava  where tyyppi='harjoitus'  group by lukuvuosi, lukukausi) harkat,  (select lukuvuosi, lukukausi, sum(6\*tunteja) lt  from opetustehtava  where tyyppi='luento'  group by lukuvuosi, lukukausi) luennot  where harkat.lukuvuosi=luennot.lukuvuosi and  harkat.lukukausi=luennot.lukukausi |

Miltä riviltä lukuvuosi ja lukukausi otettaisiin?

Sen sijaan kyselyyn voidaan liittää lisäehto, joka valitsee lukukauden

Esimerkki Y13. (ks. [esimerkkikannan kaavio](http://www.cs.helsinki.fi/u/laine/tkp/esimkanta/esimkanta.html))

|  |
| --- |
| select luennot.lukuvuosi, luennot.lukukausi, ht+lt  from (select lukuvuosi, lukukausi, sum(3\*tunteja) ht  from opetustehtava  where tyyppi='harjoitus'  group by lukuvuosi, lukukausi) harkat,  (select lukuvuosi, lukukausi, sum(6\*tunteja) lt  from opetustehtava  where tyyppi='luento'  group by lukuvuosi, lukukausi) luennot  where harkat.lukuvuosi=luennot.lukuvuosi and  harkat.lukukausi=luennot.lukukausi and  ht+lt >=  all (select ht+lt  from (select lukuvuosi, lukukausi, sum(3\*tunteja) ht  from opetustehtava  where tyyppi='harjoitus'  group by lukuvuosi, lukukausi) harkat,  (select lukuvuosi, lukukausi, sum(6\*tunteja) lt  from opetustehtava  where tyyppi='luento'  group by lukuvuosi, lukukausi) luennot  where harkat.lukuvuosi=luennot.lukuvuosi and  harkat.lukukausi=luennot.lukukausi  ) |

[Harri Laine](mailto:harri.laine@helsinki.fi):*Tietokantojen perusteet, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 2005*

# Keittiökalustekanta, SQL harjoitteluun

Taulun nimeä näpäyttämällä saat esiin uuteen ikkunaan taulun sisällön.

/\*

Taulu ptype esittelee yrityksen tuotevalikoimaan kuuluvat tuotetyypit

\*/

create table [ptype](http://www.cs.helsinki.fi/u/laine/tkpv/keittio/table_ptype.html) (

ptypeid varchar(8) not null primary key, // tuotetyypin tunnus

productname varchar(40) // tuotetyypin nimi

);

/\*

Taulussa Model esitellään mallisarjat.

Samaa tuotetta voidaan valmistaa useassa eri sarjassa.

\*/

create table [model](http://www.cs.helsinki.fi/u/laine/tkpv/keittio/table_model.html) (

modelid varchar(12) not null primary key, // mallisarjan tunnus

modelname varchar(30) ); // mallisarjan nimi

/\*

Taulussa unit määritellään tarjolla olevat perustuotteet.

Näistä voidaan koota isompia kokonaisuuksia.

Samaa tuotetta voi olla saatavissa useina eri malli- ja

värivaihtoehtoina. Sarake producttypeid yhdistää tuotteen ja sen

tyyppiin. Sarake unitid yksilöi tuotteen. Tällä hetkellä

tuotetunnuksesta selviää tuotteen tyyppi ja erilaista mittatietoa,

mutta näin ei välttämättä ole jatkossa. Tuotetiedoissa kerrotaan

lähinnä tuotteiden mittoja. Mittoja voi puuttua.

\*/

create table [unit](http://www.cs.helsinki.fi/u/laine/tkpv/keittio/table_unit.html) (

unitid varchar(30) not null primary key, // tuotteen tunnus

producttypeid varchar(12), // tuotteen tuotetyypin tunnus

unitname varchar(40), // tuotteen nimi

height decimal(6,2), // mittoja, korkeus cm

width decimal(6,2), // leveys cm

depth decimal(6,2), // syvyys cm

foreign key (producttypeid) references ptype

);

/\*

Tuoteen valmistukseen käytetty materiaali riippuu tuotetyypista ja

mallisarjasta. Taulussa modelmaterial voidaan tuotetyyppi, malli -yhdistelmälle

määritellä yksi materiaali.

\*/

create table [modelmaterial](http://www.cs.helsinki.fi/u/laine/tkpv/keittio/table_modelmaterial.html) (

producttypeid varchar(12) not null, // tuoteen tyyppitunnus

modelid varchar(12) not null, // mallisarjan tunnus

material varchar(24) not null, // valmistusmateriaali

primary key (producttypeid, modelid),

foreign key (modelid) references model,

foreign key (producttypeid) references ptype

);

/\*

Tuotteiden värivaihtoehdot on määritelty malliriippuvina. Kaikkia mallin

tuotteita saa kaikissa mallikohtaisesti kiinnitetyissä väreissä.

\*/

create table [coloring](http://www.cs.helsinki.fi/u/laine/tkpv/keittio/table_coloring.html) (

modelid varchar(12) not null, // mallisarjan tunnus

color varchar(24) not null, // värin nimi englanniksi

primary key (modelid,color),

foreign key (modelid) references model

);

/\*

Taulussa unitprice määritellään perustuotteiden hinnat yleisimmin kysytyille

tuotteille. Muitakin voi saada tilauksesta. Hinnat riippuvat tuotteesta ja

mallisarjasta.

/\*

create table [unitprice](http://www.cs.helsinki.fi/u/laine/tkpv/keittio/table_unitprice.html) (

unitid varchar(12) not null, // tuotteen tunnus

modelid varchar(12) not null, // mallisarjan tunnus

price integer not null, // hinta euroina (huom kokonaisluku)

primary key (unitid, modelid),

foreign key (unitid) references unit,

foreign key (modelid) references model

);

/\*

Perustuotteita voidaan yhdistää yhdistelmakalusteeksi. Taulussa composition

määritellään joukko valmiita yhdistelmiä. Yhdistelmiin kuuluu lähinnä runkoja,

hyllyjä ja vetolaatikoita. Niihin ei kuulu jalkoja eikä kahvoja.

Kullakin yhdistelmällä on tunnus ja nimi.

\*/

create table [composition](http://www.cs.helsinki.fi/u/laine/tkpv/keittio/table_composition.html) (

compositionid varchar(12) not null primary key, // yhdistelmän tunnus

compositionname varchar(80) // nimi

);

/\*

Taulussa partof määritellään miten yhdistelmät koostuvat perusosista.

Samaa perusosaa voi olla yhdistelmässä useita kappaleita.

Sarake partcount ilmoittaa kuinka monta. Yhtä yhdistelmää

kohden taulussa on useita rivejä.

\*/

create table [partof](http://www.cs.helsinki.fi/u/laine/tkpv/keittio/table_partof.html) (

compositionid varchar(12) not null, // yhdistelmän tunnus

partid varchar(12) not null, // yhdistelmään kuuluvan osan tunnus

partcount integer, // osien lukumäärä yhdistelmässä

primary key (compositionid,partid),

foreign key (compositionid) references composition,

foreign key (partid) references unit

);

/\*

Asiakkaat voivat laatia keittiösuunnitelmia suunnittelutyökalulla.

Suunnittelutyökalu kirjaa suunnitelman tauluun PLAN ja suunnitelman sisällön

tauluun PLANELEMENT. Suunnitelma voidaan haluttaessa muuntaa tilaukseksi.

Taulun PLAN rivi sisältää sunnitelman tunnustietojen ja erilaisten aikatietojen

lisäksi myös asiakkaan henkilötiedot. Näitä ei ole sijoitettu omaksi taulukseen

koska on oletettu, ettei samalla asiakkaalla ole monia suunnitelmia.

Näin voi kuitenkin olla, jolloin asiakastiedot joudutaan toistamaan.

Tilauksen erottaa pelkästä suunnitelmasta se, että tilauspäivämäärä on

kirjattu sarakkeeseen dateordered. Toimitetun tilauksen kohdalla on

myös toimituspäivä kirjattuna.

\*/

create table [plan](http://www.cs.helsinki.fi/u/laine/tkpv/keittio/table_plan.html) (

plannumber integer not null primary key, // suunnitelman numero

customername varchar(30) not null, // asiakkaan nimi

customerphone varchar(20), // asiakkaan puhelinnumero

datemade date not null, // milloin suunnitelma on tehty

dateordered date, // milloin suunnitelma on muutettu

// tilaukseksi

howtodeliver varchar(12), // miten pitäisi toimitta:

// home:kotiin, store: liikkeeseen

whentodeliver date, // milloin toimitus halutaan

whendelivered date, // milloin toimitettiin

customeraddress varchar(64) not null, // katuosoite

customercity varchar(64) not null // kaupunki

);

/\*

Taulussa PLANELEMENT esitetään suunnitelman sisältö: mitä osia

suunnitelmaan kuuluuu. Suunnitelman osaksi voidaan ottaa joko

yhdistelmäkalusteita tai erillisiä perustuotteita.

Jos suunniteltu osa on yhdistelmäkaluste, annetaan sen tunnus sarakkeessa

compositionid. Jos suunniteltu osa on erillinen perustuote, annetaan

sen tunnus sarakkeessa unitid. Yhdellä PLANELEMENT rivillä voidaan

määritellä vain yhdenlainen osa. Täten toinen sarakkeista compositionid

tai unitid on aina tyhjä.

Yhdistelmäkalusteet muodostuvat rungosta ja ovista tai etupaneeleista.

Niiden kohdalla annetaan rungon mallisarja ja väri (sarakkeet bodymodel

ja bodycolor) sekä erikseen ovien mallisarja ja väri (sarakkeet fronmodel ja

frontcolor). Erillisten perustuotteiden malli ja väri annetaan sarakkeissa

bodymodel ja bodycolor.

Sarakkeessa amount kerrotaan montako kertaa osa toistuu suunnitelmassa.

\*/

create table [planelement](http://www.cs.helsinki.fi/u/laine/tkpv/keittio/table_planelement.html) (

plannumber integer not null, // suunnitelman tunnus

itemnumber integer not null, // osan numero suunnitelmman sisällä

compositionid varchar(12), // yhdistelmäkalusteen tunnus, jos

// suunniteltu osa on yhdistelmäkaluste,

// muuten tyhjä

unitid varchar(12), // perustuotteen tunnus, jos suunniteltu

// osa on perustuote, muuten tyhjä

bodymodel varchar(12), // yhdistelmän rungon malli tai

// perustuotteen malli

bodycolor varchar(24), // yhdistelmän rungon väri tai

// perustuotteen väri

frontmodel varchar(12), // yhdistelmän ovien malli

frontcolor varchar(24), // yhdistelmän ovien väri

amount integer, // kuinka monta kappaletta

primary key (plannumber, itemnumber),

foreign key (compositionid) references composition,

foreign key (unitid) references unit,

foreign key (bodymodel,bodycolor) references coloring,

foreign key (frontmodel,frontcolor) references coloring

) ;