5 SQL tietokantakieli

SQL on standardoitu kieli relaatiotietokantojen käsittelyyn ja määrittelyyn. Sillä voidaan

- määritellä tietokannan käyttäjät ja heidän käyttöoikeutensa
- määritellä tietokannan tietosisältö
- hakea tietoa tietokannasta
  - näytölle tai tiedostoon
  - sovellusohjelman käyttöön
- tehdä päivityksiä tietokantaan (muuttaa dataa)
  - vuorovaikutteisesti
  - sovellusohjelman kautta
- määritellä talletusrakenteita
- kontrolloida tietokannan samanaikaista käyttöä.

SQL-kieli syntyi alunperin IBM:n San Josen tutkimuslaboratoriossa rakennettaessa relaatiotietokannan prototyyppiä nimeltä System R vuosina 1972 -73. SQL on lyhenne termistä Structured English Query Language. Sillä pyrittiin relaatioalgebran ja relaatiokalkyylien matemaattista notaatiota käyttäjäystävällisempään kieleen. Mihinkään huipputuotteeseen käyttäjäystävällisyyden suhteen ei päästy, mutta hyvin monipuoliseen kieleen, jonka perusteiden oppiminen ei ole vaikeaa.

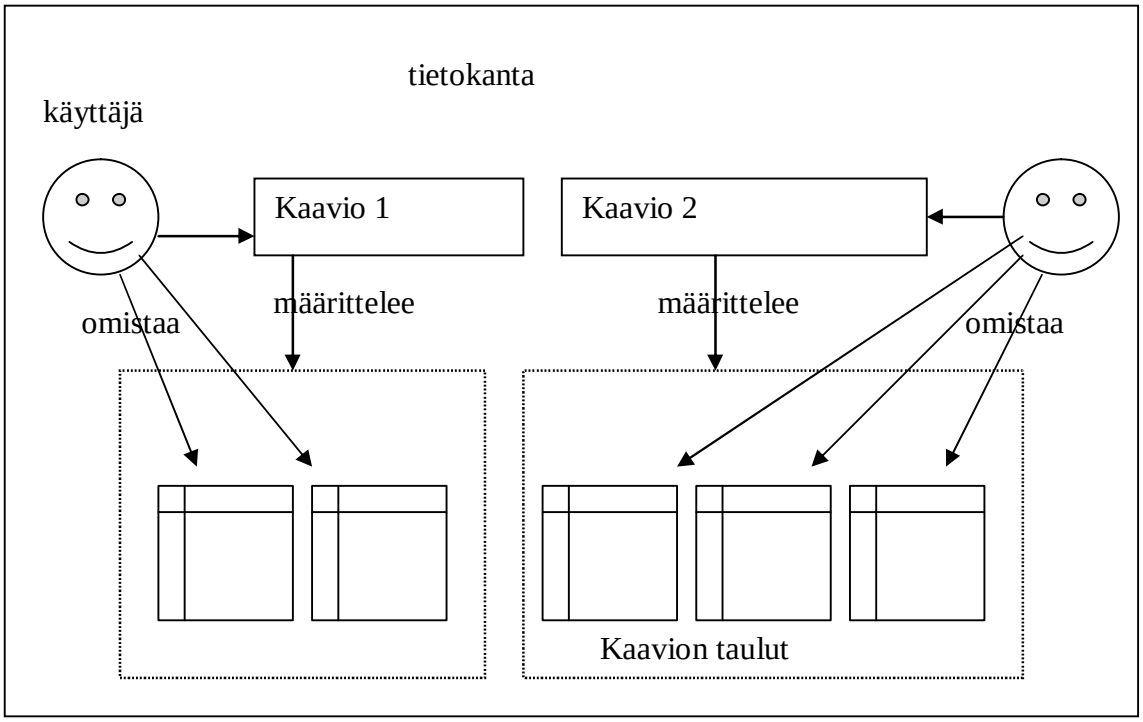
Ensimmäiset kaupalliset relaatiotietokantatuotteet tulivat markkinoille 70-luvun lopulla. IBM:n ensimmäiset kaupalliset relaatiotietokantatuotteet ilmestyivät 80-luvun alussa, varsinainen läpimurtotuote DB2 MVS käyttöjärjestelmälle julkaistiin vuonna 1983. Tätä ennen oli markkinoille tullut ‘käyttöjärjestelmäriippumattomia’ relaatiotietokantatuotteita muilta valmistajilta, esimerkiksi Oracle ja Ingres. Nykyään DB2-järjestelmääkin saa usealle käyttöjärjestelmälle samoin kuin muitakin tietokantatuotteita. Merkittävimmät SQL:ään perustuvat relaatiotietokantatuotteet ovat edellämainitun DB2:n lisäksi markkinajohtaja Oracle, Informix, Sybase ja Microsoftin SQL Server. Näiden lisäksi on tarjolla lukuisia muita SQL-pohjaisia relaatiotietokanta-

järjestelmiä. Vaikka tuotteiden pohjana onkin standardoitu SQL, ne ovat silti ominaisuuksiltaan varsin erilaisia.

SQL:n standardoinnin käynnisti amerikkalainen standardointijärjestö ANSI, joka hyväksyi ensimmäisen SQL-standardin vuonna 1986 (SQL/ANSI–86). Kansainväliseksi ISO-standardiksi ANSI:n standardi hyväksyttiin 1987. ISO:n puitteissa standardiin hyväksyttiin mm. avaimen ja viiteavaimet sisältänyt laajennus vuonna 1989. ANSI ei hyväksynyt laajennusta. Vuonna 1992 sekä ANSI että ISO hyväksyivät uuden työnimellä SQL-2 kulkeneen standardin. Tämä oli huomattavasti alkuperäistä standardia laajempi, sivumääräkin oli kasvanut vuoden -86 noin 60 sivusta noin 1000 sivuun. SQL-2 (tai SQL-92) sisälsi runsaasti uusia piirteitä. Se myös määritteli eri tasoisia SQL-totutuksia perustasosta laajaan toteutukseen. Täydellistä laajan toteutuksen mukaista SQL:ää tuskin saa markkinoilta vieläkään. Standardin kehittäminen on yhä jatkunut. Ohjelmointirajapinta (SQL/CLI – call level interface) -standardi hyväksyttiin –96 samoin kuin tietokantaproseduurien määrittelykielen standardi (SQL/PSM – persistent stored modules). Valmistajat ovat näiden suhteen kuitenkin enimmäkseen pitäytyneet omissa tekniikoissaan. Varsinaista SQL-ytimen uutta standardia on lupailtu jo vuosia, mutta se ei ehtinyt valmiiksi ennen vuosituhannen vaihtumista. Uusi SQL-99 pitää sisällään esimerkiksi automaattisesti käynnistyvät tietokantaproseduurit sekä oliopiirteitä. Näitä piirteitä on toteutettu muutamiin tietokantaohjelmistoihin.

5.1 SQL tietokanta

SQL-tietokanta muodostuu yhden tai useamman kaavion (schema) määrittelemästä joukosta tauluja (table). Kullakin kaaviolla on omistaja (owner), joka omistaa myös kaavion määrittelemät taulut. Omistajana on joku käyttäjä (user) (kuva 5.1).



Kuva 5.1: Kaaviot, taulut ja omistus

SQL:ssä tietojen tallennuksen perusrakenteena on taulu (table), joka jakautuu riveihin (row) ja sarakkeisiin. Taulu on relaatiomallin relaation kaltainen rakenne kuitenkin sillä poikkeuksella, että sen ei välttämättä tarvitse olla matemaattinen joukko, eli siinä voi olla useita keskenään samansisältöisiä rivejä. Käytännössä kaikille perustauluille yleensä määritellään pääavain, jolloin kaikkien rivien täytyy olla erilaisia jo avainsarakkeiden suhteen. Saman rivin esiintyminen useaan kertaan on sensijaan varsin hyödyllinen piirre esimerkiksi kyselyiden tulostauluissa.

Tauluja on kahden tyyppisiä perustauluja (base table) ja johdettuja tauluja (derived table, view). Perustaulujen rivit ovat fyysisesti olemassa. Johdetut taulut määritellään kyselyjen avulla. Niiden rivit muodostetaan tarvittaessa suorittamalla taulun määrittelevä kysely.

5.2 SQL:n kirjoitusasu

SQL-kieli on avainsanojen sekä kaikkien komponenttinimien (taulut, sarakkeet) suhteen kirjoitusasutuntumaton. Nämä voidaan kirjoittaa joko isoilla tai pienillä kirjaimilla tai sekakirjaimin. ‘SELECT’ kelpaa yhtä hyvin kuin ‘Select’ tai ‘select’. Tuntumattomuus ei kuitenkaan välttämättä ulotu tietokannan dataan asti. Merkkijonovakioissa isoja ja pieniä kirjaimia vertaillaan niiden merkkikoodin perusteella ja silloin esimerkiksi iso ‘KALLE’ ei ole sama kuin pikku ‘kalle’ . Tätä käyttäytymistä voidaan joissakin järjestelmissä säädellä asennusparametrein.

5.3 SQL määrittelykielenä

SQL-kieli sisältää yhtenä osanaan tietokannan määrittelykielen (data definition language, DDL). Tämän avulla voidaan määritellä käyttäjiä, kaavioita ja tauluja sekä antaa ja kumota näihin liittyviä oikeuksia. Tietokannan määrittelyyn liittyvät SQL:n *create-* , *alter-* ja *drop*-lauseet. Oikeuksia hallitaan puolestaan *grant-* ja *revoke*-lauseilla.

5.3.1 Käyttäjät

SQL-tietokannalla voi olla useita käyttäjiä. Käyttäjä määritellään lauseella

create user <username> identified by <password>; [1]

Määrittelyssä käyttäjälle annetaan tunnus ja salasana. Lauseessa on usein lisänä erilaisia järjestelmäkohtaisia käyttäjätietoja. Käyttäjillä voi olla erilaisia rooleja (role) Järjestelmät tarjoavat joitain valmiiksi määriteltyjä järjestelmäkohtaisia yleisrooleja kuten esimerkiksi:

- Tietokannanhoitaja (data base administrator, dba)
  Roolin omaavat voivat luoda ja poistaa käyttäjiä sekä hallita tietokantaa kokonaisuutena. Heillä on myös kaikki oikeudet kaikkien käyttäjien tauluihin
- Normaalikäyttäjä

[1] Kulmasuluilla <x> esitetään kielen elementti, jonka tilalle tulee jokin nimi. Kulmasulut eivät tule mukaan lauseeseen. Esimerkiksi ‘create user kalle identified by ellak;’

Roolin haltija voi luoda ja käyttää tauluja ja antaa omiin tauluihinsa liittyviä oikeuksia

- Hyödyntäjä

  Roolin haltija ei saa luoda tauluja, mutta hän voi käyttää muiden luomia tauluja niiden omistajien myöntämien oikeuksien puitteissa.

Käyttäjien määrittely on sallittua vain tietokannanhoitajalle. Tietokannanhoitaja voi myös määritellä uusia rooleja (create role -lauseella) ja kytkeä niitä käyttäjiin lauseella

grant <role> to <user>;

Käyttäjän salasanan voi muuttaa lauseella

alter user <username> identified by <password>;

Tämä on sallittua paitsi tietokannanhoitajalle myös käyttäjälle itselleen. Alter user -lauseella voi muuttaa myös muita käyttäjätietoja, jos järjestelmässä sellaisia on.

Tietokannanhoitaja voi poistaa käyttäjän lauseella

drop user <username> [cascade]; [2]

Jos lauseessa ei ole *cascade*-avainsanaa, estää käyttäjän omistamien taulujen olemassaolo käyttäjän poistamisen. Sen mukanaolo aiheuttaa myös taulujen hävittämisen.

5.3.2 Oikeudet

Käyttäjällä on kaikki oikeudet omistamiinsa tauluihin. Hän voi jakaa oikeuksia muille käyttäjille lauseella:

grant <oikeudet> on <kohde> to {*<username>* | *<role>* | *public* }*;*[3]

Jaettavat oikeudet ilmoitetaan antamalla niiden tietokantaoperaatioiden nimet (alter, select, insert, update, delete), jotka sallitaan käyttäjälle. Oikeuden kohteena voi olla

[2] Hakasulkeissa [x] esitetään valinnainen elementti. Se voi olla lauseessa mukana tai puuttua.

[3] Kaarisulkeiden sisällä esitetään pystyviivoin eroteltuna vaihtoehtoluettelo. Vaihtoehdoista valitaan vain yksi.

kaavio, taulu tai taulun sarakejoukko. Oikeus voidaan myöntää käyttäjälle, kaikille tietyn roolin haltijoille tai kaikille käyttäjille.

Omistaja voi kumota antamansa oikeuden lauseella

revoke <oikeudet>> on <kohde> from {*<username>* | *<role>* | *public* }*;*

5.3.3 Kaaviot ja taulut

Kaavio määritellään lauseella:

create schema <schemaname> authorization <username>;

Lause nimeää kaavion ja määrittelee sen omistajan. Kaikissa tietokannanhallintajärjestelmissä ei tueta kaavion määrittelyä. Oraclessa käyttäjälle muodostuu automaattisesti yksi oletuskaavio käyttäjätunnuksen luonnin yhteydessä.

Kaavio mahdollisine sisältöineen voidaan poistaa lauseella

drop schema <schemaname> [cascade];

Tietokannan taulu määritellään lauseella:

create table [<schemaname>.]<tablename> (

<column definition 1>, ….

<column definition n>

[, <constraint 1> …]

);

Esimerkki:

```
CREATE TABLE kurssi (
      koodi numeric(8) NOT NULL ,
      nimi varchar(40) NOT NULL ,
      opintoviikot numeric(5,1) NOT NULL ,
      luennoija varchar(12) NOT NULL,
      PRIMARY KEY (koodi ),
      FOREIGN KEY (luennoija) REFERENCES opettaja
);
```

Jos taulun nimeä edeltää kaavion nimi, liitetään taulu kyseiseen kaavioon. Taululle on annettava kirjaimella alkava nimi, jonka pitää olla yksikäsitteinen joko käyttäjän tau-

lujen tai, jos taulu liitetään kaavioon, niin kyseisen kaavion taulujen joukossa. Nimet SQL:ssä voivat standardin mukaan olla hyvinkin pitkiä (128 merkkiä), mutta järjestelmäkohtaisesti nimen pituus on yleensä paljon lyhyempi. Nimissä käytettävä merkkivalikoima on rajoitettu englanninkielisiin aakkosmerkkeihin, numeroihin ja suppeaan joukkoon erikoismerkkejä.

Sarake

Taulun sarakkeelle annetaan sarakemäärittelyssä vähintään nimi ja tietotyyppi. Lisäksi voidaan antaa sarakkeen sisältöä koskevia rajoitteita ja oletusarvo. Sarakemäärittelyn muoto on:

<column_name> <datatype> [not null]
[default <value>] [<column constraint> …]

Sarakkeiden nimille pätevät samat muotovaatimukset kuin taulujen nimille. Elementti *<datatype>* määrittelee sarakkeeseen sijoittuvien arvojen tyypin. Joidenkin tyyppien kohdalla määritykseen sisältyy arvon pituusmääritys. SQL:n perustietotyypit ovat:

- *Merkkijonot*

  Merkkijonot voivat olla kiinteä- tai vaihtuvapituisia. Ne määritellään seuraavasti:
  character [varying] [(<pituus>)]
  Varying-määreen mukanaolo määrittelee merkkijonon vaihtuvapituiseksi, jolloin arvoille ainakin periaatteessa varataan talletustilaa arvon merkkimäärän mukaisesti. ‘Character varying’ voidaan lyhentää ‘varchar’ ja pelkkä ‘character’ voidaan esittää muodossa ’char’. Jos pituutta ei anneta, sen tulkitaan olevan 1 merkki. Vaihtuvapituisen merkkijonon yhteydessä annettava pituus ilmoittaa merkkijonon enimmäispituuden. Suurin mahdollinen pituus on järjestelmäkohtainen, esimerkiksi 2000 merkkiä. Pitkille merkkijonoille on oma tietotyyppinsä
  CLOB (character large object). Tämän maksimipituus on yleensä useita gigamerkkejä. Merkkijonovakiot esitetään yksinkertaisiin lainausmerkkeihin suljettuina, esimerkiksi ’Kalle’.

- *Binääritieto*

  Binääritietoa ovat esimerkiksi kuva-, video- ja äänitiedostot. Niitäkin voidaan siis tallentaa tietokantaan. SQL.ssä ei oteta kantaa näiden tulkintaan, eikä niihin voi kohdistaa mitään vertailu- tai laskentaoperaatioita. Tietokantaohjelmiston kannalta ne ovat binääristä bittimassaa. Binääritieto määritellään määreellä

  bit [varying] [(<pituus>)]

  Oletuspituus on yksi tavu ja pituuden maksimiarvo on järjestelmäkohtainen. Isoja binääriaineistoja varten on käytettävissä tietotyyppi BLOB (binary large object),.

- *Tarkka numeerinen tieto*

  Tarkalla numeerisella tiedolla tarkoitetaan kokonais- ja desimaalilukuja. Käytössä ovat seuraavat määrittelyt:

  numeric [(<kokonaispituus> [,<tarkkuus>])]
  decimal [(<kokonaispituus> [,<tarkkuus>])] , lyhenne dec
  integer = int (yleensä 4 tavua)
  smallint (yleensä 2 tavua)

  Kokonaispituus ilmaisee numeropaikkojen enimmäismäärän luvussa. Jälleen maksimimäärä vaihtelee järjestelmäkohtaisesti (Oraclessa 44 numeropaikkaa). Tarkkuus ilmoittaa desimaaliosan tarkkuuden. Desimaaliosa lasketaan mukaan kokonaispituuteen. Jos määritys olisi *decimal (5,2),* olisi suurin mahdollinen luku 999.99. Määrityksellä *numeric(5,4)* se olisi 9.9999.

- *Likimääräinen numeerinen tieto*

  Likimääräinen numeerinen tieto tallennetaan tallentamalla eksponentti ja mantissa erikseen. Käytettävissä ovat ohjelmointikielistä tutut tietotyypit *real, float* ja *double precision*. Nämä eivät välttämättä eroa mitenkään toisistaan. *Double precision* on ainakin 8 tavua, *float* ja *real* ovat mahdollisesti lyhempiä.

- *Päiväykset ja aikatieto*

  Tietokantoihin tallennetaan usein päiväyksiä ja muuta aikaan liittyvää tietoa. Esimerkiksi opetukseen ilmoittautumisista voitaisiin kirjata ilmoittautumisaika sekunnin tarkkuudella. Tämän tiedon perusteella määräytyy esimerkiksi jonotusjärjestys. SQL:ssä on aikatietoihin liittyen seuraavat tietotyypit

  Date

  Päiväys, jossa vuosiluku esitetään riittävällä tarkkuudella. Päiväystä ei yleensä tallenneta selväkielisenä merkkijonona, vaan se voidaan viedä kantaan esimerkiksi päivien lukumääränä jostain alkuajankohdasta lähtien. Tietokannan käyttäjälle päiväys näkyy jossakin (usein amerikkalaisessa) päiväyksen esitysmuodossa. Esimerkiksi Oraclessa päiväyksen ulkoisen esitystavan voi valita parametrien avulla. Päiväysvakiot esitetään standardin mukaan muodossa DATE ’vvvv-kk-ppp’, esimerkiksi DATE ’2005-03-31’

  Time

  Kellonaika sekunnin tarkkuudella.

  Timestamp [(<fraction length>)]

  Yhdistetty päiväys ja kellonaika. Fraction length ilmaisee kuinka monella desimaalilla sekunnin osat ovat mukana. Oraclen vanhoissa versioissa (<9) Date tietotyyppi oli oikeastaan timestamp(0).

  Interval

  Aikaero, esim interval 3 day.

  Aikoja voidaan verrata ja niillä voi laskea. Jos esimerkiksi muuttuja *this_day* sisältäisi nykypäiväyksen olisi *this_day + interval 3 day* päiväys kolmen päivän päästä.

Sarakkeeseen liittyviä rajoitteita

Rajoitteiden avulla asetetaan ehtoja sarakkeen arvoille. Järjestelmä ei hyväksy sellaisia lisäys- tai muutosoperaatioita, jotka rikkovat määriteltyjä rajoitteita. Osa rajoitteista voidaan antaa joko sarake- tai taulukohtaisina. Tyypillisesti rajoite, joka koskee vain yhtä taulun saraketta, annetaan sarakekohtaisena. Useampia sarakkeita koskevat rajoitteet annetaan taulukohtaisina.

- *Tyhjäarvojen esto*
  Oletusarvoisesti sarake voi sisältää tyhjäarvon (NULL), olipa sen tietotyyppi mikä tahansa. SQL.ssä tyhjäarvo käyttäytyy vertailuissa siten, että mikä tahansa vertailu, jossa toinen tai molemmat osapuolet ovat tyhjäarvoja tuottaa tuloksen *tuntematon*. Tyhjäarvot voidaan kieltää liittämällä sarakemäärittelyyn määre **not null**. Määre täytyy liittää sarakkeisiin, jotka sisältyvät taulun pääavaimeen.

- *Oletusarvo*
  Sarakkeelle voidaan määritellä oletusarvo, jota käytetään, jos sarakkeen arvoa ei anneta lisäysoperaatiossa. Ellei oletusarvoa määritellä, se on tyhjäarvo NULL.
  Oletusarvo määritellään määreellä:
  default <arvo>

- *Toimiminen avaime*na
  Sarake voidaan määritellä taulun pääavaimeksi liittämällä sarakemäärittelyyn määre **primary key**. Jos taulun pääavain muodostuu useasta sarakkeesta, on avainmäärittely annettava taulukohtaisena.

- *Toimiminen viiteavaimena*
  Sarake voidaan määritellä viiteavaimeksi liittämällä sarakemäärittelyyn määre
  foreign key references <viitattava taulu> [(<avainsarake>)]
  Monisarakkeiset viiteavaimet on määriteltävä taulukohtaisesti. Esimerkiksi Solid:ssa viitattavan taulun avainsarakkeen on oltava mukana määreessä,

Oraclessa sitä ei tarvita. Viiteavainmäärittelyyn voi lisäksi liittää toimintasääntöjä, joita tarkastellaan myöhemmin taulukohtaisten sääntöjen yhteydessä.

- *Yleinen ehto*

  Sarakkeeseen voidaan liittää myös yleisiä ehtoja, jotka tarkistetaan lisäyksen ja muutoksen yhteydessä ja joiden rikkominen estää nämä operaatiot. Ehto esitetään *check* –määreellä, esim.

  Age integer ***check age between 20 and 120***

  Check-määreen ehdot noudattavat kyselyn valintaehtojen rakennetta. Näitä käsitellään myöhemmin.

Taulukohtaiset rajoitteet

Jos avain tai viiteavain muodostuu useasta sarakkeesta, on niihin liittyvä määritys annettava taulukohtaisena. Myös check-määre voidaan antaa taulukohtaisena määreenä, jos se koskee useaa taulun saraketta. Taulukohtaisesti avain määritellään seuraavasti

primary key (<column_name> [,<column_name> …])

Viiteavain määritellään samaan tapaan. Viiteavainmäärittelyyn voidaan liittää myös toimintasääntöjä, jotka määrittelevät miten toimitaan, jos poisto tai muutosoperaatiot rikkovat viite-eheyttä (eli sääntöä, että viittauksen kohteen pitää olla olemassa)

foreign key (<column_name> [, <column_name> …])
references <referred table> [(<columns>)]
[on delete {restrict | cascade | nullify}]
[on update {restrict | cascade | nullify}]

Poistoon ja muutokseen liittyvät säännöt liittyvät tilanteisiin, joissa viittauksen kohteena oleva rivi poistetaan tai viittauksessa käytetty avainarvo muutoksen seurauksena katoaa. Tällöin muutos voidaan estää (restrict - oletus), tai ongelma voidaan korjata poistamalla kaikki kadonnutta avainarvoa käyttäneet rivit (cascade), tai korjata sijoittamalla tyhjäarvo kadonneeseen avainarvoon kohdistuneiden viittausten tilalle (nullify)..

Esimerkki:

```
CREATE TABLE kurssi (
  koodi numeric(8) NOT NULL ,
  nimi varchar(40) NOT NULL ,
  opintoviikot numeric(5,1) NOT
    NULL ,
  luennoija varchar(12) NOT NULL,
  PRIMARY KEY (koodi),
  FOREIGN KEY (luennoija)
    REFERENCES opettaja
);
```

```
CREATE TABLE harjoitusryhma (
  kurssikoodi number(4) NOT NULL,
  ryhmanro number(2) NOT NULL,
  viikonpaiva varchar(12) NOT
    NULL,
  alkamisaika number(2) NOT NULL,
  sali varchar(12) NOT NULL,
  opettaja varchar(12) NOT NULL,
  PRIMARY KEY (kurssikoodi,
    ryhmanro),
  FOREIGN KEY (kurssikoodi)
    REFERENCES kurssi,
  FOREIGN KEY (opettaja)
   REFERENCES opettaja
);
```

Taulussa *Harjoitusryhmä* viiteavain *kurssikoodi* viittaa tauluun *kurssi*. Oletetaan, että taulussa *kurssi* olisi rivi

581287	Info	4	…	Laine

ja taulussa *harjoitusryhmä* siihen viittaavat rivit

581287	1	Ma	…	Niemi
581287	2	Ti	…	Virtanen
581287	3	To	…	Laine

Jos yritetään poistaa taulusta kurssi rivi, jossa koodi=581287, operaatio ei onnistu, myöskään tunnuksen muutos ei onnistu, koska viiteavainmäärittelyyn liittyvät oletusarvoisesti toimintasäännöt ‘*on delete restrict*’ ja ‘*on update restrict*’ .

5.3.4 Taulujen rakenteen muuttaminen

Taulun rakennetta voidaan muuttaa *alter table* -lauseella. Jotkut muutokset ovat mahdollisia vaikka taulussa olisi jo dataa ja se olisi tuotantokäytössä. Tällainen muutos on esimerkiksi sarakkeen lisäys tauluun, esimerkiksi

alter table kurssi add luentotunnit numeric(2);

Uusi sarake tulee järjestyksessä viimeiseksi ja saa olemassaolevien rivien kohdalla arvokseen tyhjäarvon. Useimmat tietokantatoteutukset ovat sellaisia, ettei järjestelmän tarvitse mitenkään muuttaa taulussa jo olevia vanhoja rivejä sarakkeen lisäämisen yhteydessä. Toinen tyypillinen muutos on sarakkeen pituuden kasvattaminen, joka onnistuu myös aina. Esimerkiksi

alter table kurssi modify nimi varchar(60);

Kasvattaa nimen enimmäispituuden 60:een

Pituuden lyhentäminen ja tietotyypin muuttaminen eivät yleensä onnistu, jos taulussa on jo rivejä. Standardi esittelee myös sarakkeen poisto-operaation:

alter table kurssi drop column opintoviikot;

poistaisi sarakkeen opintoviikot.

5.4 SQL –kyselyt

SQL-kyselykieli (query language) on tarkoitettu tietokantaan kohdistuvaan tiedonhakuun. Useimmissa järjestelmissä on tarjolla alkeellinen käyttöliittymä vuorovaikutteiseen tietokannan suorakäyttöön. Tämän liittymän kautta voidaan antaa kyselyjä ja nähdä niiden tuottamat tulokset näytöllä. Tulos voidaan yleensä ohjata myös tiedostoon. Ohjelmointirajapinnan kautta SQL-kyselyjä voidaan käyttää myös ohjelmissa.

SQL-kysely esitetään select-lauseena. Select –lause määrittelee tulostaulun. Lauseen yleisrakenne on seuraava:

select <tulostietomäärittely>
from <kohdetaulut>
[where <valintaehdot>]
[group by <ryhmitystekijät>]
[having <ryhmärajoitteet>]
[order by <järjestysperusta>]

Tulostietomäärittelyssä määritellään tulostaulun sarakkeet; niiden sisältö ja nimet. Kohdetaulut on luettelo tauluista, joista tietoa haetaan. Valintaehdot määrittelevät perusteet, joilla tulostaulun muodostamiseen käytettävät rivit valitaan ja kytketään toisiinsa. Ryhmitystekijöiden avulla voidaan määritellä tulostaulun muodostus ryhmäperustaiseksi ja ryhmärajoitteella valita, mitä ryhmiä otetaan mukaan. Järjestysperustan määrittelyllä saadaan tulostaulun rivit järjestettyä sisällön perusteella.

Esimerkiksi kyselyssä

```
select merkki, reknro
     from auto
     where vmalli=1996 and
          vari ='punainen' and merkki like 'Fo%'
     order by merkki, reknro
```

haetaan vuoden 1996 mallia olevien punaisten merkiltään 'Fo'-alkuisten autojen merkki ja rekisterinumero auton merkin ja saman merkin sisällä rekisterinumeron mukaan järjestettynä.

5.4.1 Tulostietomäärittely

Kysely tuottaa nimettömän, vain hetkellisesti olemassa olevan, tulostaulun. Tulostietomäärittelyssä kuvataan, mitä sarakkeita tulostaulussa on, ja miten näiden sarakkeiden arvot muodostetaan. Määrittely koostuu jonosta pilkulla erotettuja alkiomäärittelyjä

<alkio1>, <alkio2>, …., <alkio N> .

Kukin alkiomäärittely määrittelee tulostaulun sarakkeen. Alkiomäärittelyssä määritellään sarakkeen sisältö ja annetaan mahdollisesti tulostaulun sarakkeelle nimi muodossa

<lauseke> [[AS] <nimi>]

Nimen määrittely ei ole välttämätöntä. Jos nimeä ei anneta, nimeää tietokantaohjelmisto tulossarakkeet omalla tekniikallaan. Esimerkiksi Oracle antaa nimeksi koko lausekkeen. Yleisesti, jos lausekkeena on sarakenimi, se tulee myös tulossarakkeen nimeksi. Muunlaisten lausekkeiden tuottamien nimien muoto vaihtelee järjestelmäkohtaisesti..

Yksinkertaisimmillaan alkiomäärittelyyn sisältyvä lauseke on kohdetaulun sarakkeen nimi tai vakioarvo. Näiden lisäksi kyseeseen tulevat aritmeettiset lausekkeet, merkkijono- tai päiväysoperaatioita sisältävät lausekkeet sekä SQL-funktioita sisältävät lausekkeet. ja kaikkien edellä esiteltyjen yhdistelmät.

SQL:ssä merkkivakiot suljetaan yksinkertaisiin lainausmerkkeihin. Merkkijonon kirjoitusasulla on merkitystä. Numeeriset vakiot esitetään ilman lainausmerkkejä. Päiväykset esitetään yksinkertaisissa lainausmerkeissä. Päiväyksen esitystapa on järjestelmäkohtaista ja siihen voi olla mahdollista vaikuttaa järjestelmän asetuksilla. Tyhjäarvoon viitataan merkinnällä *null*.
Esimerkkejä

30	numeerinen vakio
’kirja’	merkkijonovakio
date ’1999-02-24’	päiväys (standardin mukainen muoto)
null	tyhjäarvo

Aritmeettisten lausekkeiden muodostuksessa voi käyttää yhteen- (+), vähennys- (-), kerto- (*) ja jakolaskua (/). Lausekkeissa voi käyttää sulkeita ohjelmointikielten tapaan. Esimerkiksi

```
0.25*Paino/Pituus AS Painoindeksi
0.4*(Harjoituspisteet - 20) + 0.6*Koepisteet  AS Kokonaispisteet
```

Merkkijono-operaationa SQL:ssä on merkkijonojen yhteenliittäminen (katenointi). Yhteenliittämisoperaattori on ’||’. Esimerkiksi

```
Sukunimi || ’ ’ || Etunimi AS Nimi
```

muodostaa sarakkeista Sukunimi ja Etunimi yhden Nimi-sarakkeen laittamalla aluksi sukunimen sitten väliin erottimeksi välilyöntimerkin ja lopuksi etunimen.

SQL:ssä on tarjolla useita funktioita. Yhteenvetofunktiot MIN, MAX, AVG, COUNT, SUM ovat tarjolla kaikissa SQL-toteutuksissa. Yksittäisarvoihin perustuvien ns. skalaarifunktioiden kohdalla sensijaan on melkoisesti eroja eri tietokannanhallinta-järjestelmien välillä. Matemaattiset funktiot ovat tarjolla melko samanlaisina useimmissa järjestelmissä. Esimerkkejä matemaattisista funktioista:

abs(luku)	luvun itseisarvo
floor(luku)	suurin kokonaisluku, joka on pienempi tai yhtäsuuri kuin luku
mod(luku1,luku2)	jakojäännös kun luku1 jaetaan luvulla2
round(luku, tarkkuus)	pyöristys tarkkuuden osoittamaan desimaalitarkkuuteen
truncate(luku,tarkkuus)	katkaisu tarkkuuden osoittamaan desimaalitarkkuuteen

Merkkijonofunktioiden kohdalla funktioiden nimissä on jo huomattavasti enemmän kirjoa eri toteutusten välillä sekä poikkeamia standardista. Esimerkkejä

concat(mjono1, mjono2)	liittää merkkijonot peräkkäin
length(mjono)	merkkijonon pituus merkkeinä

Osamerkkijonon eristys ja merkkijonon sisältymisen tutkiminenkin löytyvät useista järjestelmistä, mutta funktioiden syntaksi vaihtelee.

Standardissa osamerkkijono eristetään funktiolla

Substring(merkkijono from alkukohta for pituus).

Oraclessa saman funktion esitystapa on

substr(merkkijono, alkukohta, pituus) ja

Solidissa

substring(merkkijono, alkukohta, pituus)

Merkkijonon alkukohdan toisen merkkijonon sisällä saa standardin mukaisesti selville funktiolla

position (alijono in isojono).

Oraclessa saman arvon saa selville funktiolla

instr(isojono, alijono)

Päiväysten käsittelyyn on järjestelmissä tarjolla runsaasti funktioita, mutta jälleen standardista huolimatta esitys eri toteutuksissa on varsin erilainen, esimerkkinä voidaan tarkastella Oraclella ja Solidilla tehtyjä kyselyjä, jotka kirjaavat tämän päivän päiväyksen. Onerow olkoon taulu, jossa on yksi rivi.

Oracle (oletetaan että päivien ja kuukausien nimet saadaan suomeksi)

```
select
     'Tänään on '||
     lower( rtrim(to_char(sysdate,'DAY')) ||
     to_char(sysdate,' DD.')) || ' '||
     rtrim(to_char(sysdate,'MONTH')) || 'ta' ||
from onerow;
```

Solidi: (tulos englanniksi)

```
select
     'To-day is '|| dayName(curDate()) ||' '||
     convert_Char(DayOfMonth(curDate())) ||'th of '||
     MonthName(curDate())
from onerow;
```

Kumpikaan esitystapa ei ole standardin mukainen, joskin Solid:in muoto on hieman lähempänä. Standardista puuttuvat C-tyyppiset turhat kaarisulut '()' parametrittomien funktionimien perästä.

Edelleen järjestelmät tarjoavat funktioita esimerkiksi tietotyyppien väliseen konversioon (convert_char ja to_char edellisessä esimerkissä), tyhjäarvon korvaamiseen jollain todellisella arvolla (ifnull(korvattava, korvaava)), käyttäjän tietokantakohtaisen käyttäjätunnuksen selville saantiin (user) ja käyttäjän järjestelmäkohtaisen käyttäjätunnuksen selvitykseen (system).

Koska funktioiden nimissä esiintyy vaihtelua eri toteutusten välillä, tulisi funktioiden käyttöä välttää mikäli pyritään siirrettävään sovellukseen.

Edellä on käsitelty sitä, miten tulostaulun sarakkeet määritellään yksi kerrallaan. Käytössä on myös merkintätapa, jolla tulokseen voidaan ottaa mukaan kaikki from-osassa lueteltujen kohdetaulujen sarakkeet (*) tai vain jonkin kohdetaulun sarakkeet (taulu.*). Esimerkiksi kysely

```
select * from auto;
```

tuottaa tulostaulun, joka sisältää kaikki taulun auto rivit kaikkine sarakkeineen.

5.4.2 Viittaukset tauluihin ja sarakkeisiin

Täydellinen viittaus tauluun pitää sisällään tietokannan nimen, kaavion nimen ja taulunimen. Täydellinen sarakeviittaus puolestaan sisältää yllä mainittujen lisäksi vielä sarakenimen:

[[[tietokanta.]kaavio.]taulu.]]sarakenimi

Täydellistä viittausta on käytettävä, jos käytetään jossain toisessa tietokannassa sijaitsevia tauluja. Tietokantanimi voi puuttua, jos käytetään eri kaaviossa olevia (tai toisen käyttäjän omistamia) tauluja. Sarakeviittaukseen pitää sisällyttää taulunimi, jos sarakenimi ei ole yksikäsitteinen kyselyn kohdetaulujen suhteen, eli useassa kohdetaulussa on samanniminen sarake.

Esimerkiksi viitaus

```
demo.laine.auto.merkki
```

viittaa tietokannan *demo* kaavion *laine* taulun *auto* sarakkeeseen *merkki*.

5.4.3 Toistuvat tulosrivit

Tulostauluun tuotetaan normaalisti yksi rivi jokaista valintaehdot täyttävää kohdetaulujen ristitulon riviyhdistelmää kohti. Oletetaan, että taulussa *auto* on tiedot 100 punaisesta vuoden 1996 vuosimallin Fordista. Kysely

```
select merkki
from auto
where vmalli=1996 and
     vari ='punainen' and merkki like 'Fo%'
     order by merkki;
```

tuottaa tällöin tulostauluun rivin ’Ford’ 100 kertaa. Kysely toimii siis toisin kuin relaatioalgebran projektio, joka karsii toistuvat arvot. Projektion kaltainen toiminta on saatavissa aikaan myös SQL:llä. Tällöin tulostietomäärittelyn alkuun on lisättävä avainsana ***distinct***.. Kysely

```
select distinct merkki
from auto
where vmalli=1996 and
     vari =‘punainen’ and merkki like ‘Fo%’
     order by merkki;
```

tuottaa tulokseen vain yhden arvon ’Ford’ sisältävän rivin.

Yleensä raportteihin ei haluta samanlaisia, samaa asiaa ilmaisevia rivejä moneen kertaan. Mikäli tällaisia rivejä voi kyselyn valintaehtojen tuloksena syntyä, ne on syytä karsia lisäämällä tulostietomäärittelyyn ’distict’. Jos esimerkiksi haluamme tietää, mitä automerkkejä on olemassa, mitä hyötyä siitä olisi, että Ford tulee listalle sata kertaa? Sensijaan tilanteessa, jossa samanlaiset rivit eivät ilmaise samaa asiaa, niitä ei ole syytä karsia. Jos vaikkapa haluamme listan kurssin opiskelijoista, ja kurssilla on kaksi samannimistä opiskelijaa, ei nimen esiintyminen kahdesti listassa ole tarpeetonta toistoa vaan varsin hyödyllinen informaatio. Se, että SQL:ssä ei ole pitäydytty pelkästään projektioon johtuu käytännön vaatimuksista. Ajatellaanpa vaikka, että haluamme laskea yrityksen työntekijöiden keskipalkan ja sitä varten haemme tietokannasta työntekijöiden palkat SQL-kyselyllä. Kysely

```
select palkka from tyontekija;
```

voisi tuottaa tuloksen {9000,9000,9000,9000,9000,9000,15000,21000} ja vastaava projektiokysely

```
select distinct palkka from tyontekija;
```

tuottaisi tuloksen {9000,15000,21000}. Edellisestä laskettuna keskiarvo on 11250 ja jälkimmäisestä saadaan keskiarvoksi joko 15000 (jos lasketaan työntekijämääräkin noiden palkkojen määrän perusteella) tai 5625 (jos oikea työntekijämäärä on saatu selville jotain toista tapaa käyttäen).

5.4.4 Kyselyn kohdetaulut

Kyselyn from-osassa annetaan tuloksen muodostamiseen tarvittavat taulut. Kohdetaululuetteloon on välttämättä kuuluttava kaikki sellaiset taulut, joiden sarakkeisiin viitataan tulostietomäärittelyssä. Rivien valintaan voi alikyselyiden[4] kautta osallistua myös sellaisia tauluja, joiden sarakkeet eivät esiinny tulostietoluettelossa. Loogisesti tulos lasketaan muodostamalla kohdetaululuettelon taulujen välinen ristitulo ja valitsemalla siitä valintaehdot täyttävät riviyhdistelmät. Jokaisesta jäljelle jääneestä yhdistelmästä muodostetaan rivi tulostauluun (ellei distinct karsi toistuvia rivejä).

Kohdetaululuettelo voi sisältää

- tauluja,
- alikyselyn tulostauluja
- liitostauluja.

Näitä käsitellään tarkemmin seuraavissa aliluvuissa.

5.4.5 Yhteen tauluun kohdistuvat valinnat

Jos kyselyn from-osassa on vain yksi taulu tulevat tulokseen mukaan kaikki rivit, jotka täyttävät where-osassa annettavat valintaehdot.[5] Ellei kyselyyn sisälly valintaehtoja eli se on muotoa

```
select  jotain from taulu;
```

tulevat mukaan kaikki taulun rivit. Rivien järjestys on määrittelemätön, mikä tarkoittaa sitä, että käyttäjä ei voi laskea mitään sen varaan, että hän saisi kyselyn jokaisella suorituskerralla rivit samassa järjestyksessä.

Where-osassa annettavat ehdot rajoittavat vastaukseen mukaan tulevia rivejä relaatioalgebran valinta-operaation kaltaisesti. Valintalauseke koostuu loogisilla operaattoreilla AND, OR ja NOT yhteen kytketyistä alkeisehdoista. Alkeisehdossa vertailun osapuolena voi olla tulostietoluettelon yhteydessä esitellyn kaltainen

[4] Alikyselyitä käsitellään myöhemmin.

[5] Yhteenvetofunktioiden käyttö muuttaa tilanteen. Tätä käsitellään myöhemmin.

lauseke. Osapuolena voi olla myös alikysely. Alikyselyjä käsitellään omassa luvussaan myöhemmin.

Vertailuoperaattoreina ovat ohjelmointikielistä tutut operaattorit pienempi kuin (<), suurempi kuin (>), pienempi tai yhtäsuuri kuin (<=), suurempi tai yhtäsuuri kuin (>=) ja erisuuri kuin (<> ja !=) . Nämä soveltuvat käytettäväksi muiden tietotyyppien paitsi binääritiedon vertailuun. Erittäin suurten merkkijonojen vertailuun voi liittyä rajoituksia. Numeerisia tietoja vertaillaan niiden numeerisen arvon perusteella, merkkijonoja merkkien merkkikoodin perusteella ja päiväyksiä sekä aikoja ajallisen edeltävyyden perusteella. Siispä merkkijonoina ’22.9.1999’ >’10.10.1999’, mutta päiväyksinä ’22.9.1999’ < ’10.10.1999’.

Tyhjäarvo (null) käyttäytyy kaikissa vertailussa siten, että vertailun tulos on **aina tuntematon**, jos jompikumpi tai molemmat osapuolet ovat tyhjäarvoja.

SQL:ssä on lisäksi käytössä vertailuoperaattorit ’**in**’ ja ’**not in**’, joilla verrataan alkion kuulumista joukkoon. Esimerkiksi

’a’ in (’a’, ’b’, ’c’) on tosi ja
6 in (1,2,3,4) on epätosi.

Arvovälivertailussa tutkitaan kuuluuko arvo annetulle välille. Esimerkiksi

```
arvo between 3 and 6
```

on sama kuin

```
arvo>=3 and arvo<=6
```

ja

```
arvo not between 3 and 6
```

on sama kuin

```
not (arvo>=3 and arvo<=6)
```
eli
```
arvo<3 or arvo>6.
```

Merkkijonoja voi verrata myös maskeihin. Maskissa voidaan käyttää jokerimerkkejä ’%’ (prosentti), joka korvaa minkä tahansa merkkijonon (vrt. unix:n ja dos:in *) ja ’_’ (alaviiva), joka korvaa minkä tahansa merkin. Maskivertailussa operaattorina on LIKE tai NOT LIKE, esimerkiksi

```
reknro LIKE 'OG%'
```

OG-alkuiset rekisterinumerot

```
reknro NOT LIKE '_ _ _ _23'
```

muut kuin kuusimerkkiset numeroihin 23 päättyvät rekisterinumerot

Arvon tyhjyyttä voidaan testata operaattoreilla '**is null**' ja '**is not null**'. Esimerkiksi

```
sarake is null
```

on tosi, jos sarakkeessa on tyhjäarvo ja epätosi, jos siinä on jokin todellinen arvo

Esimerkki:

Tarkastellaan taulua

```
CREATE TABLE opiskelija (
    onumero numeric(5) NOT NULL ,
    nimi varchar(40) NOT NULL ,
    paa_aine varchar(12) NOT NULL,
    kaupunki varchar(30),
    aloitusvuosi numeric(4),
PRIMARY KEY (onumero ) ) ;
```

Espoossa asuvat matematiikan opiskelijat saa selville kyselyllä:

```
select *
from opiskelija
where paa_aine='MAT' and kaupunki='Espoo';
```

Vertailuissa olevien merkkijonojen on oltava kirjoitusasultaan samanlaisia kuin tietokannassa olevien arvojen. Matematiikka pääaineena on tallennettu sarakkeeseen paa-aine arvona 'MAT', joten ehto paa_aine='mat' tuottaisi tyhjän tulostaulun.

Vuosina 1994-1997 opintonsa aloittaneet sai selville kyselyillä

```
select * from opiskelija
where aloitusvuosi >=1994 and aloitusvuosi<=1997;
```

tai

```
select * from opiskelija
where aloitusvuosi between 1994 and 1997;
```

tai

```
select * from opiskelija
```

```
where aloitusvuosi in (1994,1995,1996,1997);
```

Tapoja on siis useita.

Oletetaan, että esimerkkitaulussa opiskelijan nimi on tallennettu NIMI-sarakkeeseen muodossa sukunimi+väli+etunimi. Kaikki opiskelijat, joiden etunimi alkaa kirjaimella ’L’ saa tällöin selville kyselyllä

```
select * from opiskelija
where nimi like '% L%';
```

5.4.6 Tuloksen järjestäminen

Käyttäjän kannalta järjestetty tulos on usein hyödyllisempi kuin järjestämätön, Esimerkiksi järjestetystä raportista on huomattavasti helpompi etsiä tietoa kuin järjestämättömästä. SQL:ssä kyselyn tulostaulu voidaan järjestää sisältönsä perusteella. Järjestysmääre annetaan kyselyn lopussa muodossa:

```
order by lauseke [ asc[ending] | desc[ending] ][, …]
```

Lausekkeen arvo toimii järjestyksen määrääjänä. Järjestyksen määräjiä voi olla useampia: ensisijainen, toissijainen, jne. Suuntamääreet ascending ja descending ilmoittavat mihin järjestykseen kyseisen lausekkeen perusteella tulosrivit on järjestettävä:

- ascending: kasvavaan (nousevaan) järjestykseen
- descending: vähenevään (laskevaan) järjestykseen.

Ascending on oletusmääre, jota sovelletaan, ellei suuntamäärettä ole annettu.

Tulosrivit lajitellaan ensisijaisesti ensimmäisenä luettelossa annettavan lausekkeen perusteella. Toisena luettelossa olevaa lauseketta käytetään määräämän järjestys niiden rivien joukossa, joissa ensimmäisen lausekkeen arvo on sama, kolmatta niiden joukossa, jossa toisenkin arvo on sama, jne. (kuva 5.2)

Esimerkki:

A	B	C	D
2	4	8	7
1	4	6	7
3	1	5	2
2	3	5	1
1	1	4	2
3	2	4	6
1	5	5	2

a)

A	B	C	D
1	4	6	7
1	1	4	2
1	5	5	2
2	4	8	7
2	3	5	1
3	1	5	2
3	2	4	6

b)

A	B	C	D
1	**1**	4	2
1	**4**	6	7
1	**5**	5	2
2	**3**	5	1
2	**4**	8	7
3	**1**	5	2
3	**2**	4	6

c)

A	B	C	D
3	**1**	5	2
3	**2**	4	6
2	**3**	5	1
2	**4**	8	7
1	**1**	4	2
1	**4**	6	7
1	**5**	5	2

d)

a) järjestämätön
b) order by A ascending
c) order by A ascending, B ascending
d) order by A descending, B ascending

Kuva 5.2: Tuloksen järjestäminen

Merkkitiedon järjestäminen voi perustua merkkien merkkikoodijärjestykseen. Toisaalta esimerkiksi Oraclessa on mahdollista määritellä aakkoston kansallisen järjestyksen huomioiva järjestäminen, jossa suomenkielessä skandimerkitkin sijoittuvat oikeille paikoilleen ja V=W. Tällaisessa järjestämisessä isot ja pienet kirjaimet ovat samanarvoisia.

Esimerkki: Opiskelijat pääaineittain kauimmin opiskelleista uudempiin.

```
select paa_aine, aloitusvuosi, nimi
from opiskelija
order by paa_aine, aloitusvuosi;
```

SQL:n kannalta järjestyksen määräämiseen käytetyt sarakkeet voivat sijaita tulostaulussa missä tahansa. Tuloksen käytettävyyden kannalta ne on kuitenkin syytä sijoittaa vasempaan laitaan (alkuun).

5.4.7 Liitokset

Jos kyselyn from-osan kohdetaululuettelossa on useita tauluja, muodostetaan näiden ristitulo, ellei kyselyn where-osassa ole ehtoa, joka kytkisi rivit yhteen. Mikäli tällainen yhteen kytkevä ehto on mukana, kyseessä onkin taulujen liitos.

Esimerkki:

Tarkastellaan tauluja kurssi ja opettaja

```
CREATE TABLE kurssi (
     koodi  numeric(8) NOT NULL ,
     nimi  varchar(40) NOT NULL ,
     opintoviikot  numeric(5,1) NOT NULL ,
     luennoija  varchar(12) NOT NULL,
     PRIMARY KEY (koodi ),
     FOREIGN KEY (luennoija) REFERENCES opettaja);

CREATE TABLE opettaja (
     opetunnus varchar(12) NOT NULL ,
     nimi varchar(40) NOT NULL ,
     puhelin varchar(12) ,
     tyohuone varchar(12),
PRIMARY KEY (opetunnus) );
```

Kysely:

```
select kurssi.nimi, opettaja.nimi
from opettaja, kurssi
where kurssi.luennoija= opettaja.opetunnus
order by kurssi.nimi;
```

antaa tiedon kurssien luennoijista liittämällä kuhunkin kurssi-riviin viiteavaimen (luennoija) perusteella opettajan, johon tuo viiteavain osoittaa.

Edellisen esimerkin kaltainen viiteavaimen ja avaimen yhtäsuuruuten perustuva liitos on tyypillisin tapa liittää tauluja yhteen. Muunkinlaisia liitoksia voi toki esiintyä. Esimerkiksi seuraavassa kyselyssä yritetään nimien yhtäsuuruuden perusteella selvittää, ketkä opiskelijat toimivat myös opettajina.

```
select opiskelija.nimi
from opiskelija, opettaja
where opiskelija.nimi=opettaja.nimi
order by opiskelija.nimi;
```

Liitosoperaation vastauksen kannalta taulujen järjestyksellä from-osassa ei ole mitään merkitystä, kuten ei myöskään sillä, missä järjestyksessä liitosehdot ovat kyselyn ehto-osassa. Näillä kummallakin tekijällä saattaa olla vaikutusta kyselyn tehokkuuteen. Vaikutus on kuitenkin järjestelmäkohtainen ja se mikä yhdessä järjestelmässä saattaa johtaa tehokkaaseen toteutukseen toimiikin toisessa päinvastoin.

5.4.7.1 Tilapäinen uudelleennimeäminen (correlation name)

Taulut voidaan from-osassa nimetä uudelleen. Uusi nimi esitellään muodossa:

```
taulu [AS] uusinimi
```

Annettu uusi nimi on voimassa vain kyselyn sisällä ja sitä on käytettävä aina kun kyselyssä viitataan kyseiseen tauluun, esimerkiksi tarkenteissa. Uudelleennimeämistä on pakko käyttää, jos sama taulu esiintyy from-osassa useaan kertaan. Muissa tilanteissa sitä voidaan käyttää esimerkiksi korvaamaan pitkät taulunimet lyhenteillä.

Esimerkki: Kurssiparit, joilla on sama luennoija.

```
select A.nimi, B.nimi
from kurssi A, kurssi B
where A.luennoija=B.luennoija and A.koodi<B.koodi
order by A.nimi, B.nimi
```

Kurssi-taulu esiintyy kyselyssä kahdesti. Ensimmäinen esiintymä on nimetty uudelleen A:ksi ja toinen B:ksi. Parinmuodostus tapahtuu ehdolla A.luennoija=B.luennoija. Ehto A.Koodi<B.koodi on mukana karsimassa pois parit, joissa kurssi esiintyy itsensä parina (sama koodi) ja myös varmistamassa sen, että tietty pari tulee vain kertaalleen (jokaisessa parissa ensimmäisen osapuolen koodi pienempi kuin toisen) tulokseen.

5.4.7.2 Alikyselyn tulos kyselyn kohteena

Vuoden 92 SQL-standardissa sallittiin kyselyn from-osassa myös alikyselyjen tulostaulut sekä liitostulokset (joined table). Alikyselyn tulostaulu otetaan kyselyn kohdetauluksi muodossa:

```
(alikysely) [[as] alias [( sarakeluettelo)] ]
```

Tässä alikysely on normaali SQL-kysely, kuitenkin ilman järjestysmääritystä. Alias antaa sille kyselynsisäisen nimen ja sarakeluettelo uudelleennimeää haluttaessa taulun sarakkeet. Alikyselystä kohdetaululuettelossa on hyötyä yhdistettäessä rivikohtaista yksityiskohtaista tietoa ja yhteenvetotietoa. Tätä käsitellään yhteenvetotietojen yhteydessä. Muissa tilanteissa from-osaan sijoitetut alikyselyt pikemminkin hankaloittavat kyselyt ymmärtämistä. Rakenne ei myöskään ole käytettävissä kaikissa SQL-toteutuksissa.

Liitostulokset antavat aiemmin kuvatun tavan rinnalle toisen tavan liitoksen esittämiseen. Liitostuloksena voidaan määritellä myös ulkoliitos. Liitostulos määritellään seuraavasti:

```
taulu1 [<join type>] JOIN taulu2 [ ON <liitosehdot>]
<join type> =
     inner | left outer | right outer | full outer
```

’Inner join’ tai pelkästään ’join’ on normaaliliitos. Aiemmin esillä ollut kysely kurssin opettajista voidaan sitä käyttäen esittää muodossa:

```
select  kurssi.nimi, opettaja.nimi
from opettaja join kurssi on luennoija=opetunnus
order by kurssi.nimi
```

Tässä liitosehto tulee selkeämmin esiin kuin where-osaan upotettuna. Normaali-liitoksessa tulokseen tulevat mukaan vain ne lähtötaulujen rivit, joille löytyy liitos-ehdon täyttävä pari. Esimerkin tapauksessa vastaus ei sisällä mitään tietoa kursseista, joilla ei ole luennoijaa eikä opettajista, jotka eivät luennoi. Tällaisen informaation mukaan saaminen edellyttää ulkoliitoksen käyttöä.

Kyselyllä:

```
select kurssi.nimi, opettaja.nimi
from kurssi left outer join opettaja on
    luennoija=opetunnus
order by kurssi.nimi
```

saadaan tulokseen kaikki kurssit, myös sellaiset, joilla ei ole luennoijaa. Näiden kohdalla opettajan nimenä on tyhjäarvo null. Vastaavasti kyselyllä:

```
select opettaja.nimi, kurssi.nimi
from kurssi right outer join opettaja on
    luennoija=opetunnus
order by  opettaja.nimi
```

saadaan mukaan kaikki opettajat, myös ne, jotka eivät luennoi. Luennoimattomien opettajien kohdalla kurssin nimenä on tyhjäarvo.

Tyypillinen virhe liitoksissa on jonkin liitosehdon puuttuminen. Näin voi käydä erityisen helposti silloin kun tauluja joudutaan liittämään vertailemalla monisarak-keisia avaimia ja viiteavaimia. Jos kohdetauluja on n kappaletta, tarvitaan niiden liittämiseksi n-1 liitosehtoa. Jos jotkin näistä liitoksista perustuvat monisarakkeisiin avaimiin, kasvaa alkeisehtojen määrä suureksi. Yhdenkin ehdon puuttuminen johtaa virheelliseen tulokseen ja odotettua suurempiin tauluihin. Yleensä kyselyt rakentuvat siten, että niissä on yksi kokoava taulu, johon liitetään muista tauluista muodostuvia polkuja. Kokoavasta taulusta itsestään ei välttämättä suoraan tule mitään tietoa lopputulokseen.

Esimerkki:

Tarkastellaan edellä esiteltyjen taulujen lisäksi tauluja harjoitusryhmä ja ilmoittautuminen:

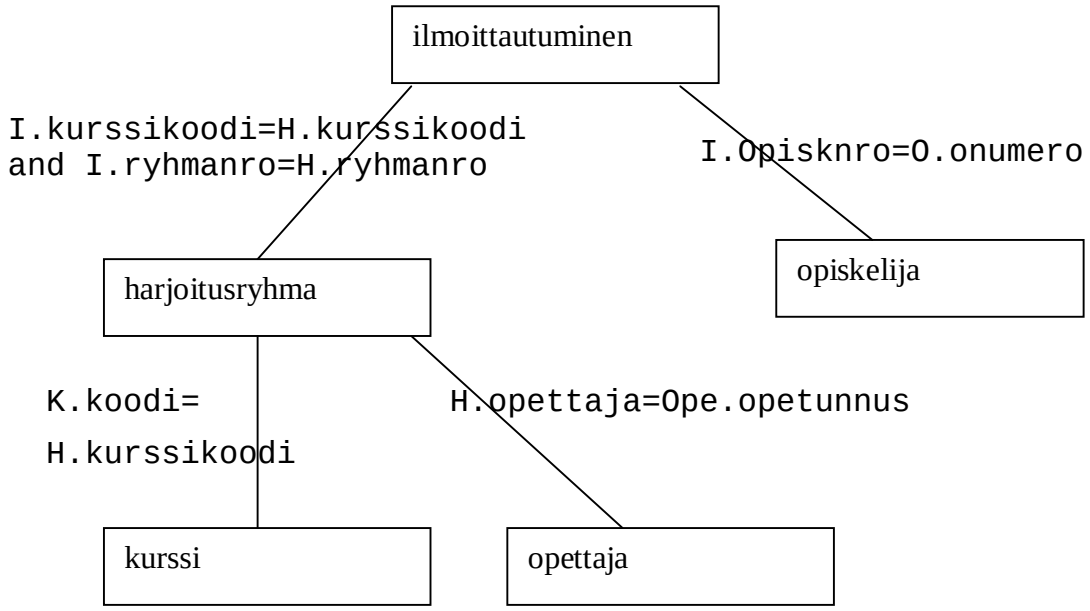
```
CREATE TABLE harjoitusryhma (
        kurssikoodi  numeric(4) NOT NULL ,
        ryhmanro  numeric(2) NOT NULL ,
        viikonpaiva  varchar(12) NOT NULL ,
        alkamisaika  numeric(2) NOT NULL ,
        sali varchar(12) NOT NULL,
        opettaja varchar(12) NOT NULL,
        PRIMARY KEY (kurssikoodi, ryhmanro) ,
        FOREIGN KEY (kurssikoodi) REFERENCES kurssi,
        FOREIGN KEY (opettaja) REFERENCES opettaja );

CREATE TABLE ilmoittautuminen (
        kurssikoodi numeric(8) not null,
        ryhmanro numeric(2) not null,
        opisknro  numeric(5) NOT NULL ,
        ilm_aika  date NOT NULL ,
        PRIMARY KEY (opisknro, kurssikoodi) ,
        FOREIGN KEY (kurssikoodi, ryhmanro)  REFERENCES
              harjoitusryhma ,
        FOREIGN KEY (opisknro) REFERENCES opiskelija );
```

Halutaan saada raportti kurssin ’Java ohjelmointi’ harjoitusryhmistä.

```
select H.ryhmanro rno, Ope.nimi ope,
       H.viikonpaiva, H.alkamisaika,
       O.Nimi opiskelija
from Harjoitusryhma H, opettaja Ope, opiskelija O,
     ilmoittautuminen I, kurssi K
where
     K.nimi='Java ohjelmointi' and
     K.koodi= H.kurssikoodi and
     I.kurssikoodi=H.kurssikoodi and
     I.ryhmanro=H.ryhmanro and
     H.opettaja=Ope.opetunnus and
     I.Opisknro=O.onumero
order by H.ryhmanro, O.nimi;
```

Kyselyssä *ilmoittautuminen* on kokoava taulu, joka yhdistää muut taulut. Yhtään taulun *ilmoittautuminen* saraketta ei kuitenkaan tule kyselyn tulokseen. Kyselyn rakennetta kuvaa alla oleva puu (kuva 5.3).



Kuva 5.3: Liitokset kytkentäpuuna

5.4.8 Alikyselyt

Alikyselyllä tarkoitetaan kyselyyn upotettua toista kyselyä. Edellä todettiin, että upotettua kyselyä voidaan käyttää kyselyn from-osassa, mutta sitä voidaan käyttää myös where-osassa valintaehdon operandina. Kuten varsinainen kysely, niin alikyselykin tuottaa tuloksenaan taulun (=joukon rivejä). Alikyselyiden käyttöön valintaehdoissa on omia predikaatteja ja lisätarkenteita, jotka määrittelevät, miten ehdon operandia sovelletaan alikyselyn tulokseen

Aiemmin käsitellyistä vertailuoperaatioista operaatiolla IN tarkasteltiin arvon kuulumista joukoon. IN-vertailun joukko-osapuoli voidaan korvata alikyselyllä. Tällöin ehto on tosi, jos verrattava arvo sisältyy alikyselyn tulokseen.

Esimerkki: Luennoivat opettajat.

```
select nimi from opettaja
where opetunnus in
   (select luennoija from kurssi)
order by nimi;
```

Alikyselyllä haetaan kaikkien luennoijien tunnukset. Pääkyselyssä vaaditaan, että opettajan tunnus kuuluu tähän joukkoon. Samaa alikyselyä voi käyttää apuna myös selvitettäessä, ketkä opettajat eivät luennoi:

```
select nimi from opettaja
where opetunnus not in
   (select luennoija from kurssi)
order by nimi;
```

IN-vertailua voidaan käyttää myös verrattaessa arvojonoa monisarakkeisen alikyselyn tulokseen. Esimerkiksi harjoitusryhmät, joille ei ole ilmoittautunut ketään, saadaan kyselyllä:

```
select * from harjoitusryhmä
where (kurssikoodi, ryhmanro) not in
   (select kurssikoodi,ryhmanro
     from ilmoittautuminen);
```

Muut esillä olleet vertailuoperaatiot perustuvat yksittäisten arvojen vertailuun. Näidenkin osapuolena voi käyttää alikyselyä, mikäli on varmaa, että kysely tuottaa vain yhden tulosrivin. Vertailuun liitettävillä lisämääreillä SOME (tai ANY) ja ALL voidaan yksittäisten arvojen vertailuoperaatioita käyttää myös monirivisten alikyselyn tulostaulujen yhteydessä. Lisämääreellä SOME varustettu ehto on tosi, jos jokin joukon alkioista toteuttaa ehdon. Lisämääreellä ALL varustettu ehto on tosi, jos kaikki joukon alkiot toteuttavat ehdon.

Esimerkki: Luennoivat opettajat

```
select nimi from opettaja
where opetunnus =
      some (select luennoija from kurssi)
order by nimi;
```

Esimerkki: Mistä kurssista saa eniten opintoviikkoja?

```
select nimi from kurssi
where opintoviikot >=
      ALL (select opintoviikot from kurssi)
order by nimi;
```

Alikyselyllä haetaan kaikkien kurssien opintoviikkomäärät. Olkoon ne vaikka välillä 1-8. Ulompi kysely testaa, että kurssin opintoviikkomäärän täytyy olla suurempi tai yhtäsuuri kuin kaikki nämä, eli suurempia tai yhtä suuria kuin suurin näistä eli 8. Vain ne kurssit valitaan, joiden opintoviikkomäärä on 8.

Yksittäisen sarakkeen tyhjyyttä testataan **is null** tai **is not null** -predikaateilla. Myös alikyselyn tulosjoukon tyhjyyttä voidaan testata. Tähän tarkoitukseen SQL tarjoaa **exists** ja **not exists** –predikaatit. **Exists** on tosi, jos alikyselyn tulostaulussa on ainakin yksi rivi. **Not exists** on tosi, jos alikyselyn tulostaulu on tyhjä .

Aiemmin on tarkasteltu vain sellaisia alikyselyjä, jotka ovat olleet riippumattomia siitä kyselystä, mihin ne sisältyvät. Tällaisen alikyselyn voisi suorittaa myös erillisenä kyselynä. Se ei sisällä viittauksia ulomman kyselyn tauluihin. Exist ja not exists – predikaattien käyttäminen ei ole mielekästä riippumattomien alikyselyjen yhteydessä, sillä predikaatilla olisi sama arvo, joko tosi tai epätosi, jokaisen ulomman kyselyn rivin kohdalla.

Kytketyssä alikyselyssä viitataan ulomman kyselyn taulujen sarakkeisiin. Näin alikysely tavallaan suoritetaan uudelleen jokaisen ulommassa kyselyssä käsiteltävän rivin yhteydessä käyttäen ehdoissa niitä arvoja, mitä vuorossa olevalla rivillä on.

Esimerkki: Luennoivat opettajat:

```
select nimi from opettaja
where exists
    (select luennoija from kurssi
     where luennoija= opettaja.opetunnus)
order by nimi;
```

Tässä alikysely on ehdon ’luennoija=opettaja.opetunnus’ avulla kytketty ulommassa kyselyssä käsiteltävään opettaja-tauluun. Se suoritetaan, ainakin loogisesti, uudelleen aina kun siirrytään tarkastelemaan seuraavaa opettaja taulun riviä. Tällaisessa kyselyssä ei ole merkitystä sillä millainen sen tuottama tulosrivi on. Tässä tulokseksi tulee opettajan tunnus. Tulos voisi olla yhtä hyvin * tai vaikka vakio.

Sama kysely voidaan SQL:ssä ilmaista usealla eri tavalla: Tarkastellaan muutamia tapoja esittää kysely ’Ketkä luennoijat pitävät myös harjoituksia luennoimillaan kursseilla ja millä kursseilla’

a)
```
Select O.nimi, K.nimi from opettaja O, kurssi K
where O.opetunnus=K.luennoija and
      O.opetunnus in
        (select opettaja from harjoitusryhma where
        kurssikoodi=K.Koodi);
```

b)
```
Select O.nimi, K.nimi from opettaja O, kurssi K
where O.opetunnus=K.luennoija and
exists (select 'A' from harjoitusryhma
    where opettaja=o.opetunnus and
    kurssikoodi=K.Koodi);
```

c)
```
Select O.nimi, K.nimi from opettaja O, kurssi K
where O.opetunnus=K.luennoija and
(O.opetunnus,K.koodi) in
(select opettaja,kurssikoodi from harjoitusryhma );
```

Alikyselyitä voi myös ketjuttaa siten, että alikysely pitää sisällään toisen alikyselyn, jne. Esimerkkinä olkoon kysely 'Java –alkuisten kurssien harjoituksia pitävien opettajien nimistä:

```
select nimi from opettaja
where opetunnus in
     (select opettaja from harjoitusryhma
      where kurssikoodi in
           (select koodi from kurssi
           where nimi like 'Java%'));
```

5.4.9 Joukko-opin operaatiot

SQL-kyselyjen tulostauluja voidaan yhdistää joukko-opin yhdiste (union), leikkaus (intersect) ja erotus (except, Oraclessa minus) operaatioilla. Yhdiste operaatio karsii toistuvat rivit. Aiemmin esillä ollut kysely opiskelijoista, jotka toimivat myös opettajina voitaisiin esittää leikkausta käyttäen esimerkiksi seuraavasti:

```
select nimi from opettaja
intersect
select nimi from opiskelija
order by nimi;
```

Kaikki Java-alkuisia kursseja opettavat saadaan selville kyselyllä:

```
select nimi from opettaja
where opetunnus in
     (select opettaja from harjoitusryhma
      where kurssikoodi in
           (select koodi from kurssi
           where nimi like ‘Java%’)
      union
       (select luennoija from kurssi
        where nimi like ‘Java%’  ));
```

Joukko-opin operaatiot edellyttävät, että osapuolet ovat samarakenteisia. Operaation ensimmäinen osapuoli määrä tulostaulun sarakkeiden nimet.

5.4.10 Yhteenvetofunktiot

Tähän asti on käsitelty kyselyjä, joissa tuotetaan yksi tulosrivi jokaista valintaehdot täyttävää kohdetaulujen riviyhdistelmää kohti. Tällaiset kyselyt tuottavat tuloksenaan detaljitietoa. SQL:llä on mahdollista tuottaa myös yhteenvetotietoa (aggregate data). Yhteenvetotietojen muodostukseen ovat käytettävissä funktiot

- AVG keskiarvo
- COUNT lukumäärä
- MAX suurin arvo
- MIN pienin arvo ja
- SUM summa.

Joissakin järjestelmissä on näiden lisäksi myös muita tilastollisia suureita, esimerkiksi keskihajonnan, laskevia funktioita.

Yhteenvetotietoja tuotettaessa tuloksen muodostamisperiaate muuttuu. Yhteenveto voidaan tuottaa joko koko vastausaineistosta tai ryhmiin jaotellusta vastausaineistosta. Vastausaineistolla tarkoitetaan tässä valintaehdot täyttäviä riviyhdistelmiä.

Tuotettaessa yhteenveto koko vastausaineistosta saadaan kyselyn tulokseksi aina yksi rivi. Jos vastausaineisto on tyhjä, tuottavat muut funktiot paitsi COUNT tuloksenaan tyhjäarvon (NULL). COUNT tuottaa tyhjästä vastausaineistosta arvon 0.

Esimerkkejä:

Ilmoittautumisten lukumäärä:

```
select count(*) from ilmoittautuminen;
```

Kurssien pienin ja suurin opintoviikkomäärä

```
select min(opintoviikot) as pienin,
      max(opintoviikot) as suurin
from kurssi;
```

Helsinkiläisten opiskelijoiden keskimääräinen aloitusvuosi

```
select avg(aloitusvuosi) from opiskelija
where kaupunki='Helsinki';
```

Funktioille annetaan argumentiksi lauseke, jonka perusteella funktion arvo lasketaan. Funktiolle COUNT voi antaa argumentiksi myös tähden (*) osoittamaan, että tulokseksi halutaan rivien lukumäärä. Argumenttiin voidaan liittää määre DISTINCT ilmaisemaan sitä, että funktion arvo tulisi laskea sarakkeen erilaisten arvojen perusteella. Funktion arvoa laskettaessa tyhjäarvot jätetään huomiotta, eli ne eivät vaikuta tulokseen. Ainoa poikkeus tästä on COUNT(*), joka laskee mukaan kaikki rivit, vaikka jokin rivi sisältäisi pelkkiä tyhjäarvoja. Sen sijaan sarakekohtaisesti COUNT-funktiokin ilmoittaa vain todellisten arvojen lukumäärä.

Esimerkki:

Taulu R(A,B,C) :

A	**B**	**C**
1	1	2
1	2	2
3	null	2
3	null	2
5	2	2

select count(*) from R = > 5

select count(A) from R => 5

select count(B) from R => 3

select count(distinct B) from R => 2

select count(*) from R where B is null => 2

select (distinct C) from R => 1

DISTINCT –määreen käyttö toistuvien arvojen ottamiseksi vain kertaalleen mukaan laskentaan on yleensä järkevää vain COUNT-funktion yhteydessä. Keskiarvon tai summan laskeminen erillisistä arvoista tuottaa harvoin mitään hyödyllistä tulosta.

Esimerkki: Monellako eri paikkakunnalla opiskelijat asuvat?

```
select 'Opiskelijoiden asuinpaikkakuntia: ',
     count(distinct kaupunki)
     from opiskelija;
```

Yhteenvetofunktioita voi käyttää lausekkeissa, ja niiden argumentteina voi käyttää lausekkeita. Toista yhteenvetofunktiota ei kuitenkaan voi käyttää toisen argumenttina eikä lausekkeisiin tai ylipäätään tulokseen saa mukaan samasta vastusaineistosta sekä rivikohtaista tietoa että yhteenvetotietoa.

Esimerkkejä:

Suurimman ja pienimmän opintoviikkomäärän erotus

```
select MAX(opintoviikot)-MIN(opintoviikot) erotus
from kurssi;
```

Kertokoon opiskelijanumeron parillisuus opiskelijan olevan naispuolinen. Naispuolisten opiskelijoiden osuus prosentteina saadaan tällöin selville kyselyllä:

```
select
  100*(count(onumero)-sum(mod(onumero,2)))/count(onumero)
  from opiskelija;
```

Tässä mod(onumero,2) on miespuolisilla opiskelijoilla 1, joten sum(mod(onumero,2)) ilmoittaa miespuolisten opiskelijoiden lukumäärän. Erotuksella saadaan naispuolisten lukumäärä ja jakolaskulla heidän osuutensa.

Kysely

```
select nimi, max(opintoviikot)
from kurssi;
```

on syntaktisesti virheellinen. Siinä halutaan samasta aineistosta yhteenvetotietoa ja rivikohtaista tietoa. Miltä riviltä tuo rivikohtainen tieto otettaisiin? Jos kyselyllä

haluttaan saada selville, minkä kurssin opintoviikkomäärä on suurin, se pitäisi esittää esimerkiksi seuraavasti

```
select nimi, opintoviikot
from kurssi
where opintoviikot =
      (select max(opintoviikot) from kurssi);
```

tai

```
select nimi, maksi
from kurssi K,  (select max(opintoviikot) maksi from kurssi) M
where K.opintoviikot=M.maksi;
```

Kummassakin näistä vaihtoehdoista maksimi lasketaan loogisesti eri rivijoukosta kuin mistä rivikohtainen tieto *nimi* haetaan.

5.4.10.1 Ryhmiin jaottelu

Ryhmiin jaottelu saadaan aikaan lisäämällä kyselyyn ryhmittelymääre

```
group by <ryhmitysperusteet>.
```

Ryhmitysperusteina annetaan sarakeluettelo ja ryhmät muodostetaan siten, että ryhmään kuuluvat kaikki sellaiset vastausaineiston rivit, joilla on yhtenevät arvot kussakin sarakeluettelon sarakkeessa (kuvat 5.4 ja 5.5)

X= taulu T group by A (Muodostuu 3 ryhmää.)

A	**B**	**C**	**D**	**group**
1	4	6	7	A=1
1	1	4	2	A=1
1	1	5	2	A=1
2	4	8	7	A=2
2	3	5	1	A=2
3	1	5	2	A=3
3	2	4	6	A=3
3	2	1	5	A=3

Kuva 5.4: Ryhmittely yhden sarakkeen perusteella

Y= taulu T group by A,B (Muodostuu 6 ryhmää)

A	B	C	D	group
1	1	4	2	A=1, B=1
1	1	5	2	A=1, B=1
1	4	6	7	A=1, B=4
2	3	5	1	A=2, B=3
2	4	8	7	A=2, B=4
3	1	5	2	A=3, B=1
3	2	4	6	A=3, B=2
3	2	1	5	A=3, B=2

Kuva 5.5: Ryhmittely kahden sarakkeen perusteella.

Ryhmitetystä aineistosta yhteenvetofunktioiden arvot lasketaan ryhmäkohtaisesti, ja tulokseen tulevien rivien lukumäärä on sama kuin ryhmien lukumäärä. Yllä olevasta kuvan 5.4 ryhmittelystä X tulee siis 3 tulosriviä, ja kuvan 5.5 ryhmittelystä Y tulee 6 tulosriviä (kuva 5.6).

Esimerkkejä:

```
select  A, sum (B) , count(C) from T group by A;
```

A	sum(B)	count(C)
1	6	3
2	7	2
3	5	3

```
select A,B, count(C), sum(D) from T group by A,B;
```

A	B	count(C)	sum(D)
1	1	2	4
1	4	1	7
2	3	1	1
2	4	1	7
3	1	1	2
3	2	2	11

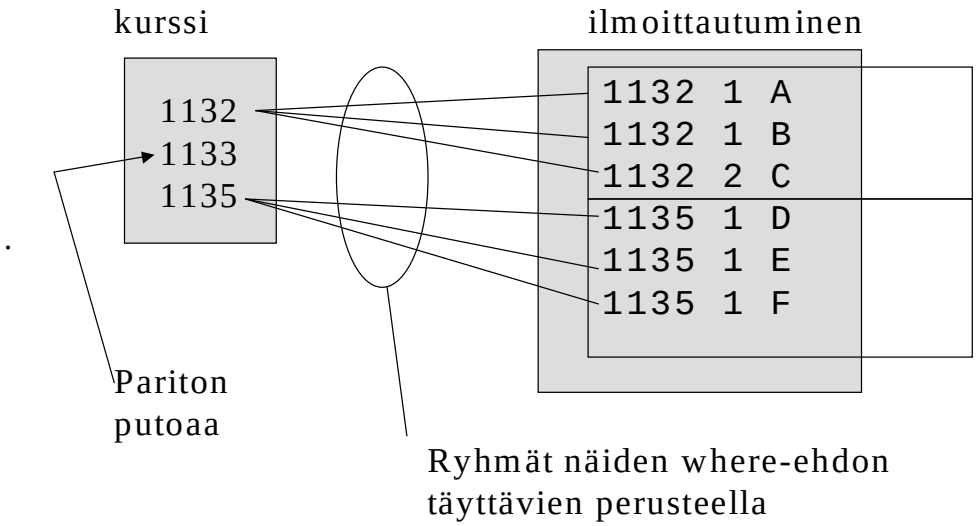
Kuva 5.6: Esimerkkejä ryhmittelyn vaikutuksesta kyselyn tulokseen

Esimerkki: Kurssien harjoitusryhmiin ilmoittautuneiden opiskelijoiden lukumäärä

Kysely

```
select nimi, ryhmanro, count(*)
   from kurssi, ilmoittautuminen
   where ilmoittautuminen.kurssikoodi=kurssi.koodi
   group by nimi, ryhmanro;
```

tuottaa oikean tuloksen, jos jokaiseen ryhmään on ilmoittautunut vähintään yksi opiskelija. Tyhjät ryhmät jäävät kuitenkin pois vastauksesta, sillä ne karsiutuvat liitosoperaation johdosta vastausaineistosta jo ennen ryhmitystä (kuva 5.7)



Kuva 5.7: where- ehto karsii rivit ennen ryhmitystä.

Tyhjät ryhmätkin tuottava kysely olisi esimerkiksi:

```
select K.nimi, I.ryhmanro, count(*)
from kurssi K, ilmoittautuminen I
where I.kurssikoodi=K.koodi
group by K.nimi, I.ryhmanro
union
(select C.nimi, H.ryhmanro, 0
from kurssi C, harjoitusryhma H,
where C.koodi=H.kurssikoodi and
      (C.koodi,H.ryhmanro) not in
        (select kurssikoodi, ryhmanro
        from ilmoittautuminen));
```

Myös ulkoliitosta voisi käyttää:

```
select nimi, h.ryhmanro, count(distinct opisknro)
from kurssi,
     harjoitusryhma h left outer join ilmoittautuminen i
     on h.kurssikoodi= i.kurssikoodi and
        h.ryhmanro=I.ryhmanro
where kurssi.koodi= h.kurssikoodi
group by nimi, h.ryhmanro
```

Huomaa, että edellä ei voi käyttää funktiota COUNT(*), sillä tyhjistäkin ryhmistä tulee nyt yksi rivi vastausaineistoon.

Ryhmittelyyn perustuvissa kyselyissä voidaan tulokseen ottaa mukaan vain group by –osassa lueteltuja ryhmittelyssä käytettyjä sarakkeita sekä yhteenvetofunktioihin perustuvia lausekkeita. Jos esimerkiksi kursseista haluttaan tietoina koodi, nimi, luennoijan nimi ja ryhmien lukumäärä, täytyy kysely esittää muodossa

```
select koodi, kurssi.nimi, opettaja.nimi, count(*) ryhmia
from kurssi, opettaja, harjoitusryhma
where kurssi.koodi=harjoitusryhma.kurssikoodi and
    kurssi.luennoija=opettaja.opetunnus
group by koodi, kurssi.nimi, opettaja.nimi;
```

Ryhmät muodostuisivat täysin samoiksi vaikka ryhmittelyyn käytettäisiin vain kurssi-taulun saraketta koodi. Kurssin nimi ja opettajan nimi on kuitenkin liitettävä group by –osaan, jotta ne saataisiin mukaan tulokseen. Niiden jättäminen pois group by –osasta johtaisi yllä olevan kyselyn kohdalla käännösaikaiseen virheilmoitukseen.

Ryhmitetystä aineistosta tuotetaan vastaukseen yksi rivi jokaista ryhmää kohden. Ryhmien mukaan ottamista voidaan kuitenkin säädellä liittämällä kyselyyn ryhmiä koskeva valintaehto. Tämä esitetään määreellä

```
having <valintaehdot>
```

Valintaehdot ovat rakenteeltaan samanlaisia kuin kyselyn where-osassa annettavat valintaehdot. Yleensä ehdot kuitenkin perustuvat yhteenvetofunktioiden arvoihin.

Esimerkki: Ryhmät, joihin on ilmoittautunut yli 20 opiskelijaa

```
select nimi, ryhmanro, count(*)
from kurssi, ilmoittautuminen
where ilmoittautuminen.kurssikoodi=kurssi.koodi
group by nimi, ryhmanro
having count(*) >20;
```

Käytännössä ryhmien muodostaminen tarkoittaa vastausaineiston järjestämistä ryhmityssarakkeiden perusteella. Vastauksen varsinaisen järjestyksen määrittelee kuitenkin order by –määre.

Esimerkki: Harjoitusryhmät, ilmoittautujalukumäärän mukaan laskevassa järjestyksessä.

```
select nimi, ryhmanro, count(*)
from kurssi, ilmoittautuminen
where ilmoittautuminen.kurssikoodi=kurssi.koodi
group by nimi, ryhmanro
order by count(*) desc;
```

Seuraavassa esimerkissä tarkastellaan kohtuullisen monimutkaista ryhmittelyyn perustuvaa kyselyä. Tällaisten kyselyjen yhteydessä kannattaisi jo harkita myöhemmin esiteltävien näkymien käyttöä helpottamaan kyselyn muodostusta.

Esimerkki:

Millä kurssilla on suurin keskimääräinen ryhmäkoko.

Ensimmäinen yritys:

```
select nimi, H.ryhmanro, max(avg(count(*)))
from kurssi, harjoitusryhma H, ilmoittautuminen I
where koodi=H.kurssikoodi and
      H.kurssikoodi=I.kurssikoodi and
      H.ryhmanro=I.ryhmanro
group by nimi, H.ryhmanro;
```

Yritys johtaa virheilmoitukseen, sillä yhteenvetofunktio ei voi olla toisen argumenttina. Lähdetään etsimään oikeaa kyselyä sen osasten kautta. Seuraavassa ei huomioida tyhjiä ryhmiä. Ryhmään ilmoittautuneiden lukumäärät ei-tyhjien ryhmien osalta saadaan kyselyllä:

```
select kurssikoodi,ryhmanro, count(*) lkm
from ilmoittautuminen
group by kurssikoodi, ryhmanro
```

Oletetaan, että tämän suoritus tuottaa tuloksen

KURSSIKOODI	RYHMANRO	LKM
1134	1	6
1134	2	4
1134	3	3
1135	1	6
1135	2	5
1135	3	2
1136	1	6
1137	1	5

Kysely voidaan ottaa alikyselyksi kyselyyn, joka laskee kullekin kurssille keskimääräisen ryhmäkoon:

```
select koodi, nimi, avg(lkm)
   from kurssi,
       (select kurssikoodi,ryhmanro, count(*) lkm
        from ilmoittautuminen
        group by kurssikoodi, ryhmanro)  A
        where koodi= A.kurssikoodi
    group by koodi, nimi;
```

Suoritettuna samassa kannassa kuin edellinen kysely tämä tuottaisi tuloksen

KOODI	NIMI	AVG(LKM)
1134	Informaatiojärjestelmät	4,33333333
1135	Java ohjelmointi	4,33333333
1136	Oliotietokannat	6
1137	Tietorakenteet	5

Maksimi saadaan selville having-ehdolla, jolloin koko kysely olisi:

```
select koodi, nimi, avg(lkm)
   from kurssi,
        (select kurssikoodi,ryhmanro,count(*) lkm
         from ilmoittautuminen
         group by kurssikoodi, ryhmanro)  A
        where koodi= A.kurssikoodi
   group by koodi, nimi
   having avg(lkm)>= ALL
       (select avg(lkm)
        from kurssi,
           (select kurssikoodi,ryhmanro,count(*) lkm
              from ilmoittautuminen
              group by kurssikoodi, ryhmanro)  A
        where koodi= A.kurssikoodi
        group by koodi)
```

Having–ehdon alikyselyssä on tulokseen otettu vain yheenvetofunktion arvo ilman ryhmittelysaraketta. Alikyselyn tulos tässä tapauksessa on

AVG(LKM)
4,33333333
4,33333333
6
5

Käytettäessä yhteenvetofunktioita yhdeksi merkittäväksi ongelmaksi muodostuu tuloksen oikeellisuuden todentaminen. Tulokseksi saadaan vain yksi rivi tai joukko ryhmäkohtaisia rivejä, jotka sisältävät yhteenvedon tuloksen. Miten voimme tietää onko tuo tulos oikein? Onko se laskettu oikean rivijoukon yli? Ovatko kyselyyn liittyvät ehdot oikein? Tyypillinen valintaehdon puuttuminen johtaa siihen, että jokin luku tulee laskentaan mukaan moneen kertaan. Tällöin summa moninkertaistuu ja keskiarvo lasketaan painottuneesta joukosta. Tilanne on ongelmallinen etenkin, jos tietokannasta yritetään saada selville jotain uutta informaatiota tunnusluvulla, jonka oikeasta suuruusluokastakaan ei ole varmaa etukäteistietoa. Tunnuslukuun uskotaan ja sitä käytetään päätöksenteossa. Puuttuvan tai virheellisen valintaehdon perusteella voidaan tuottaa merkittävästi virheellisen tunnusluku. Tällä voi sitten olla päätöksenteossa suurikin merkitys. Tunnuslukuja tuottavien kyselyiden testaukseen tulisikin kiinnittää erityishuomiota. Testaamisessa pitää tuottaa apu- ja välituloksia, joilla varmistutaan kokonaistuloksen oikeellisuudesta. Suuren tietokannan kohdalla tällaiset apu- ja välitulokset voivat olla suuria, mutta niiden kohdalla virhe voi ilman aputuloksia jäädä kokonaan huomaamatta. Esimerkiksi vastausjoukon koko voi joissain tilanteissa olla riittävä aputulos. Joissain toisissa se ei riitä vaan lisäksi tarvitaan muita kyselyjä selvittämään millainen vastausjoukon tulisi olla.

5.4.11 Näkymät

Näkymällä (view) tarkoitetaan kyselyn avulla määriteltyä johdettua taulua (derived table). Tällaista taulua voidaan käyttää kyselyissä kuten perustaulujakin. Myös näkymiin kohdistuvat ylläpito-operaatiot voivat olla rajoitetusti mahdollisia. Näkymä määritellään *create view* –lauseella, joka on muotoa:

create view <*taulunimi*> [(<*sarakke1*>, …)] as <*kysely*>;

Taulunimi nimeää näkymään liitetyn kyselyn tulostaulun. Määrittelyyn liittyvässä sarakeluettelossa sarakkeille voidaan antaa nimet. Jos sarakeluettelo puuttuu, käytetään sarakkeista kyselyn tuottamia nimiä. Kysely määrittelee taulun sisällön.

Esimerkki: Tyhjät harjoitusryhmät

```
Create view tyhja_ryhma (kurssikoodi, nimi, ryhmanro) as
     select kurssikoodi, nimi, ryhmanro
     from kurssi, harjoitusryhma
     where koodi= kurssikoodi and
          (kurssikoodi, ryhmanro) not in
            (select kurssikoodi, ryhmanro
             from ilmoittautuminen)
```

Näkymätaulun rivejä ei tallenneta minnekään.[6] Kyselyn kohdistuessa näkymätauluun kyselyn käsittelijä limittää kyselyn ja näkymämäärittelyssä olevan kyselyn ja suorittaa näin aikaansaadun yhdistetyn kyselyn. Esimerkiksi kysely

```
select * from tyhja_ryhma;
```

aiheuttaisi näkymän määrittelevän kyselyn suorituksen

Näkymien käyttöön on kolme perussyytä:

- tietoriippumattomuuden aikaansaaminen,
- täsmäsuojauksen mahdollistaminen ja
- kyselyjen helpottaminen.

Tietoriippumattomuudella tarkoitetaan sitä, että ohjelmaa ei tarvitse muuttaa, jos tietokantaa muutetaan. Hyvin laaditut kyselyt (ei ’select *’ –muotoisia) takaavat jo jonkin verran tietoriippumattomuutta. Jos ohjelma käsittelee kantaa näkymien kautta, voidaan tietoriippumattomuutta yhä lisätä. Tietokantaan voidaan tehdä laajojakin muutoksia, esimerkiksi pilkkoa taulu useaksi tauluksi ja silti tarjota käyttäjälle samanlainen näkymä. Muutokset edellyttävät vain näkymän määrittelevän kyselyn muuttamista tuottamaan samanlainen taulu muuttuneen kannan pohjalta.

Oletetaan, että tauluun *ilmoittautuminen* liittyisi näkymä ilmo, joka olisi määritelty seuraavasti:

---

[6] Joissakin järjestelmissä on tarjolla näkymän kaltaisesti määriteltävä vedos (snapshot), joka muodostetaan kyselyllä mutta jonka rivit tallennetaan kantaan. Toisin kuin näkymä vedos ei muutu perustaulujen muuttuessa.

```
create view ilmo as
   select kurssikoodi, ryhmanro, opisknro, ilm_aika
   from ilmoittautuminen;
```

Tämä on rakenteeltaan ja sisällöltään identtinen taulun *ilmoittautuminen* kanssa. Kun tauluun *ilmoittautuminen* on kertynyt ilmoittautumisia pitkältä ajanjaksolta runsaasti, taulun käyttö alkaa hidastua. Tällöin voidaan päätyä ratkaisuun, jossa passiiviset vanhat ilmoittautumiset siirrettään historiatauluun *vanhat_ilmoittautumiset*. Erilaiset tilastot edellyttävät kuitenkin sekä nykyisiä että vanhoja ilmoittautumisia. Jos tilastointi olisi perustettu näkymään *ilmo*, toimisivat tilastot edelleen kun näkymän määrittely muutetaan muotoon

```
create view ilmo as
   select kurssikoodi, ryhmanro, opisknro, ilm_aika
   from ilmoittautuminen
union
(select kurssikoodi, ryhmanro, opisknro, ilm_aika
   from vanhat_ilmoittautumiset);
```

Näkymien käyttö suojauksessa perustuu siihen, että käyttäjille annetaan käyttöoikeuksia näkymiin. Näin voidaan rajata käyttöoikeus vain joihinkin taulun riveihin tai vain yhteenvetotietoihin. Esimerkiksi kuka tahansa voisi saada tiedon kurssien opiskelijamääristä, mutta ei tietoa kurssien yksittäisistä opiskelijoista. Roolille ’javaopettaja’ voitaisiin antaa kaikki oikeudet näkymään

```
create view javailmoittautuminen as
   select kurssikoodi, ryhmanro, opisknro, ilm_aika
   from ilmoittautuminen
   where kurssikoodi in
       (select koodi from kurssi
        where nimi=’Java ohjelmointi’)[7]
```

[7] Jos näkymää käytetään usin saattaisi olla parempi määritellä se kurssikoodin eikä kurssin nimeen perustuvana.

Seuraava näkymämäärittely perustuu oletukseen, että *opettaja*-taulussa käytetään opettajatunnuksena (opetunnus) opettajan tietokantakäyttäjätunnusta. Funktio *user* antaa kyselyn suorittajan tietokantakäyttäjätunnuksen.

```
create view oma_kurssi as
select * from kurssi where luennoija=user;
```

Näkymän käyttöoikeus voitaisiin tehdä julkiseksi. Luennoijille taulussa näkyvät hänen luennoimiensa kurssien rivit. Muille käyttäjille taulu on tyhjä. Vaikka käyttöoikeus on julkinen eivät muut kuin kurssin luennoija pääse tämän näkymän kautta kurssitietoihin.

Näkymää voidaan käyttää myös kyselyjen yksinkertaistamiseen. Monimutkainen kysely upotetaan näkymämäärittelyyn ja käyttäjä muodostaa omat kyselynsä yksinkertaisina näkymätauluihin perustuvina kyselyinä.

Edellä määriteltyä näkymää tyhjä_ryhma voisi käyttää näkymän ryhmakoko määrittelyssä:

```
create view ryhmäkoko (koodi, nimi, ryhma, lkm) as
    select nimi, ryhmanro, count(*)
    from kurssi, ilmoittautuminen
    where ilmoittautuminen.kurssikoodi=kurssi.koodi
    group by nimi, ryhmanro
    union
    (select kurssikoodi, nimi, ryhmanro, 0
     from tyhja_ryhma);
```

Tätä käyttämällä sen kurssin selvittäminen, jonka keskimääräinen ryhmäkoko on suurin sujuisi seuraavasti:

```
select koodi, nimi, avg(lkm)
from ryhmakoko
group by koodi, nimi
having avg(lkm)>= ALL
    (select avg(lkm) from ryhmakoko
    group by koodi)
```

5.4.12 Tietokannan ylläpito

SQL sisältää myös tiedon muokkausvälineet tietokannan ylläpitoa varten. Välineistö muodostuu lauseista:

- insert rivien lisäys,
- update rivien muutos ja
- delete rivien poisto.

5.4.12.1 Lisäykset

Insert-lauseella on kaksi muotoa. Toisella lisätään yksittäisiä vakioarvoista koostuvia rivejä, toisella usean rivin joukko. Yksittäisen rivin lisäävän insert-lauseen rakenne on

```
insert into <taulu> [(<sarakenimet>)] values (<arvot>)
```

Sarakenimet on luettelo sarakenimiä. Se on valinnainen ja sitä tarvitaan tilanteissa, joissa annetaan arvot vain osalle lisättävän rivin sarakkeista. Arvot –osassa annetaan lisättävän rivin sarakkeiden arvot pilkulla eroteltuina. Ellei lauseeseen sisälly sarakenimiosaa, on sarakkeiden arvot annettava siinä järjestyksessä, missä sarakkeet esiintyvät taulussa. Jos lauseeseen sisältyy sarakenimiosa, se määrää annettavien arvojen järjestyksen.

Esimerkki:

Tarkastellaan taulua kurssi, joka on määritelty seuraavasti:

```
CREATE TABLE kurssi (
    koodi  numeric(8) NOT NULL ,
    nimi  varchar(40) NOT NULL ,
    opintoviikot  numeric(5,1),
    luennoija  varchar(12),
    PRIMARY KEY (koodi ),
    FOREIGN KEY (luennoija) REFERENCES opettaja);
```

Lause

```
insert into kurssi
values (1234,'Tietokantojen perusteet',2,'HLAINE');
```

lisää rivin tauluun kurssi. Rivin kaikille sarakkeille on annettu arvot.

Jos kurssia lisättäessä ei vielä ole tiedossa kurssin luennoijaa, voidaan lisäys tehdä lauseella

```
insert into kurssi
values (1234,'Tietokantojen perusteet',2,NULL);
```

tai käyttämällä sarakeluetteloa

```
insert into kurssi (koodi, nimi, opintoviikot)
values (1234,'Tietokantojen perusteet',2);
```

Niille sarakkeille, joille ei anneta arvoa lisäyslauseessa, tulee arvoksi oletusarvo. Ellei tätä ole sarakkeelle määritelty create table –lauseen default-määreellä, käytetään oletusarvona tyhjäarvoa. Edellisen esimerkin kaksi viimeistä insert-lausetta lisäävät siis samanlaisen rivin. Jos oletusarvoa ei ole määritelty ja puuttuvaan sarakkeeseen on liitetty taulumäärittelyssä määre NOT NULL , lisäys epäonnistuu. Samoin käy, jos lisäys rikkoo muita eheysehtoja, esimerkiksi avaimen yksikäsitteisyyttä tai viite-eheyttä.

Lisäyslauseen toisen vaihtoehdon avulla voidaan tauluun lisätä kyselyn tulosrivit. Lauseen muoto on seuraava:

```
insert into <taulu> [(<sarakenimet>)] <kysely>.
```

Tällä lauseella voidaan samoin kuin yksittäistenkin rivien tapauksessa lisätä sekä täydellisiä rivejä että rivejä, joista puuttuu joitain sarakkeita. Kyselyn tulosrivit lisäävä operaatio on hyödyllinen esimerkiksi kopioitaessa taulu, siirrettäessä tietoja aktiivisesta taulusta historiatauluihin, perustettaessa testiaineistoja ja suoritettaessa tietokannan uudelleenorganisointia.

Esimerkki:

Kurssin lisääminen kun ei tiedetä luennoijan tunnusta, mutta tiedetään nimi.

```
insert into kurssi
      select 1234, 'Tietokantojen perusteet', 2, opetunnus
      from opettaja where nimi ='Laine Harri';
```

Tässä kyselyssä vain opetunnus tulee taulusta opettaja. Muut vastausrivin tiedot ovat vakioita.

Esimerkki:

Kurssin 'Ohjelmoinnin perusteet' opiskelijoiden siirto kurssille 'Java ohjelmointi'.

```
Insert into ilmoittautuminen
   select java.kurssikoodi, ryhmanro, opisknro, sysdate[8]
   from kurssi java, kurssi ohpe, ilmoittautuminen
   where java.nimi='Java-ohjelmointi' and
         ohpe.nimi='Ohjelmoinnin perusteet' and
         ohpe.koodi=ilmoittautuminen.kurssikoodi;
```

5.4.12.2 Rivien muutokset

Rivien sisältöä muutetaan update-lauseella. Update lauseen rakenne on

```
update <taulu> set  <sijoitus>[, <sijoitus> …]
[where <valintaehdot>]
```

missä sijoitus on muotoa

```
<sarake>=<lauseke>
```

Lauseessa oleva valintaehto määrittelee muutoksen kohteena olevat rivi. Muutos tehdään kaikkiin niihin riveihin, joiden kohdalla valintaehto on tosi. Jos ehto puuttuu tehdään muutos kaikkiin taulun riveihin. Muutosta ei hyväksytä, jos se aiheuttaa eheysehdon rikkoutumisen.

[8] sysdate on Oraclen mukainen nykyhetken antava funktio.

Esimerkki:

Kurssin ’Java-ohjelmointi’ opintoviikkomäärä kasvatetaan yhdellä.

```
update kurssi
set opintoviikot= opintoviikot+1
where nimi=’Java-ohjelmointi’;
```

Esimerkki:

Kaikki 4 opintoviikon kurssit jaetaan kahdeksi 2 opintoviikon kurssiksi (osa 1 ja osa 2). Kurssikoodiin ja nimeen lisätään osan tunnus.

Tehdään aluksi uudet puolikaskurssit

```
insert into kurssi
  select kurssikoodi*10+1, nimi||’, osa 1’,2, luennoija
  from kurssi where opintoviikot=4;
```

ja muutetaan sitten vanhat kurssit toisiksi puolikkaiksi

```
update kurssi
  set kurssikoodi=kurssikoodi*10+2,
      nimi= nimi||’, osa 2’
      opintoviikot=2
where opintoviikot=4;
```

5.4.12.3 Rivien poisto

Poistolauseella

```
delete from <taulu> [where <valintaehto>]
```

poistetaan taulusta kaikki valintaehdon täyttävät rivit. Jos valintaehto puuttuu poistetaan kaikki taulun rivit. Poistot, jotka rikkovat eheysehtoja epäonnistuvat.

Esimerkki

Poistetaan harjoitusryhmät, joihin ei ole ilmoittautunut ketään.

```
delete from harjoitusryhma
where (kurssikoodi, ryhmanro) not in
      (select kurssikoodi, ryhmanro
       from ilmoittautuminen);
```

5.4.12.4 Ylläpito-operaatioiden yhteisvaikutus

Jos ylläpito-operaatioita suoritetaan sarjassa useita, on pidettävä huolta siitä, että ne suoritetaan oikeassa järjestyksessä. On kiinnitettävä huomiota siihen, etteivät sarjassa suoritettavien operaatioiden kohteen määrittelevät valintaehdot aiheuta odottamattomia sivuvaikutuksia esimerkiksi sen takia, että sarjassa aiemmin ollut operaatio muuttaa sarjassa myöhemmin olevan operaation kohdejoukkoa joksikin muuksi kuin on ajateltu..

Esimerkki:

Tarkastellaan taulua

```
bonustili(asiakasnimi, ... , saldo, ...)
```

Olkoon siellä rivit

asiakasnimi	**…**	**saldo**	**…**
Roope Ankka		3100	
Aku Ankka		2000	
Ines Ankka		2700	
Hannu Hanhi		2900	

Tilille on tarkoitus antaa bonus siten, että saldoa korotetaan 10% jos saldo on 3000 tai alle ja 15% jos saldo on yli 3000. Jos bonuksen maksu hoidetaan seuraavasti

```
update bonustili set saldo=1.1*saldo
where saldo<=3000;
update bonustili set saldo=1.15*saldo
where saldo>3000;
```

olisi tulos

asiakasnimi	**…**	**saldo**	**…**
Roope Ankka		3565	
Aku Ankka		2200	
Ines Ankka		2970	
Hannu Hanhi		**3668**	

ja Hannu Hanhella on taas käynyt tuuri. Ongelma suoritusjärjestyksessä on se, että ensimmäinen päivitys muuttaa rivejä siten, että toisen päivityksen kohdejoukko muuttuu. Suoritettaessa muutokset oikeassa järjestyksessä

```
update bonustili set saldo=1.15*saldo
where saldo>3000;
update bonustili set saldo=1.1*saldo
where saldo<=3000;
```

tulos on

asiakasnimi	**…**	**saldo**	**…**
Roope Ankka		3565	
Aku Ankka		2200	
Ines Ankka		2970	
Hannu Hanhi		**3190**	

5.4.12.5 Rivien siirto ja kopiointi

Joissakin tietokannanhallintajärjestelmissä on tarjolla mahdollisuus tallentaa kyselyn tulostaulu suoraan oikeaksi tauluksi create table tai create snapshot -komennoilla Esimerkiksi Oraclessa taulu voitaisiin luoda seuraavasti:

```
create table ilmotilanne_09_20 as
select * from ilmoittautuminen;
```

Jos tämä suoritettaisiin 20.9., se vastaisi nimeään ja sisältäisi kaikki luontihetkellään taulussa ilmoittautuminen olleet ilmoittautumiset. Ellei tietokannanhallintajärjestelmä

tarjoa tällaisia mahdollisuuksia, on kopioinnit suoritettava insert-lauseiden avulla. Rivien siirtäminen taulusta toiseen edellyttää sekä insert- että delete-lauseiden käyttöä.

Esimerkki:

Vanhojen ilmoittautumisten siirto historiatauluun.

```
insert into ilmohistoria
   select * from ilmoittautumiset
   where ilm_aika<‘1.1.1999’;
delete from ilmoittautumiset
   where ilm_aika<‘1.1.1999’;
```

5.4.12.6 Näkymiin perustuva ylläpito

Joissakin järjestelmissä on mahdollista kohdistaa ylläpito-operaatio myös näkymään. Kaikki näkymät eivät kuitenkaan tule kyseeseen ylläpidon kohteina. Ylläpito on mahdollista vain, jos jokaiselle näkymän riville löytyy yksikäsitteinen vastinrivi näkymän perustaulusta ja muutos näkymäriviin on yksikäsitteisesti muunnettavaksi muutokseksi sen vastinriviin. Yhteenvetotietoja tarjoavat näkymät eivät ole tällaisia. Eivät myöskään näkymät, joiden sarakkeet on määritelty lausekkeiden avulla. Yleensä edellytetään, että kaikki ylläpidettävän näkymän rivit kuuluvat samaan tauluun ja taulun pääavain on mukana näkymässä. Lisäksi vaaditaan, että näkymän määrittely sisältää pelkästään valintarajoituksia. Aiemmin esillä olleista näkymistä *oma_kurssi* täyttää nämä vaatimukset samoin ilman yhdiste-operaatiota määritelty *ilmo*.

5.4.12.7 Tietokantatapahtumat

Usein joudutaan jonkin tietokantamuutoksen aikaansaamiseksi suorittamaan useita tietokantaoperaatioita. Esimerkiksi suoritettaessa tilisiirto otetaan rahaa yhdeltä tililtä ja siirretään ne toiselle. On oleellista, että kumpikin operaatio suoritetaan.

Esimerkki:

Tarkastellaan taulua: `tili(tilinumero,…,saldo,…)`

```
update tili set saldo=saldo-500
    where tilinumero=123456;
update tili set saldo=saldo+500
    where tilinumero=654321;
```

siirtää 500 markkaa tililtä 123456 tilille 654321. On oleellista, että ei tehdä vain tililtäottoa tai tilillepanoa, vaan molemmat.. Samoin, mikään ulkopuolinen operaatio, vaikka pankin vastuiden laskentaohjelma, ei pääse näkemään tilien saldoja siirto-operaation ollessa kesken.

Tietokantatransaktiolla (tapahtumalla) tarkoitetaan yhtenä jakamattomana kokonaisuutena pidettävää tietokantaoperaatioiden joukkoa. Edellä mainittu tilisiirto on tällainen. Tietokannanhallintajärjestelmän tulisi taata, että

- transaktio suoritetaan joko kokonaan tai sitä ei suoriteta lainkaan,
- ulkopuoliset näkevät vain kokonaan suoritettujen transaktioiden lopputulokset.

Tietokantatransaktion tietokantaan tekemät muutokset ovat peruttavissa siihen asti kunnes järjestelmä sitoutuu transaktioon. Käyttäjä voi luottaa siihen, että tietokantamuutokset, joita on tehty sellaisen transaktion kuluessa, johon järjestelmä on sitoutunut, eivät peruunnu. Käyttäjän kannalta järjestelmän sitoutuminen transaktioon tarkoittaa sen onnistunutta loppuunvientiä.

SQL:ssä transaktion päättyminen ilmaistaan commit-komennolla, jonka muoto on

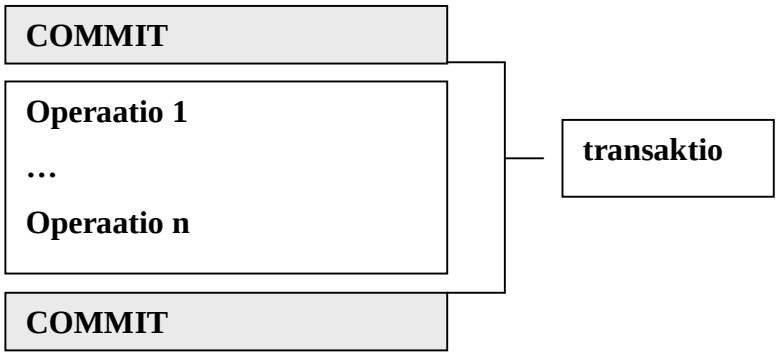
```
commit [work].
```

Commit-komento ilmoittaa, että transaktio päättyy ja esittää järjestelmälle pyynnön sitoutua viemään transaktio päätökseen. Samalla se on ilmoitus siitä, että mahdolliset jatkossa tulevat operaatiot kuuluvat eri transaktioon. Transaktioon kuuluvat siis kahden commit-komennon välissä annetut operaatiot (kuva 5.8).

Käyttäjä voi lopettaa transaktion myös antamalla perumispyynnön

```
rollback [work]
```

Perumispyyntö aikaansaa kaikkien edellisen commit:in jälkeen tehtyjen muutosten perumisen.



Kuva 5.8: Transaktio SQL:ssä

Esimerkki:

Yritetään poistaa tyhjät harjoitusryhmät. Oletetaan, että ilmoittautumisten viiteavaimeen liittyy on delete cascade -määre

```
commit;
select count(*) from harjoitusryhma;
>> 150 <<
select count(*) from ilmoittautuminen;
>> 3500 <<
delete from harjoitusryhma
   where ryhmanro is not null;              ßVIRHE
select count(*) from ilmoittautuminen;
>>0<<                                       (:-?)
select count(*) from harjoitusryhma;
>>0<<                                       =(:-o) OHO!
rollback;
select count(*) from ilmoittautuminen;
>>3500<<                                    J
```

Tietokantatransaktiot pitäisi tehdä lyhytkestoisiksi ja vähän resursseja varaaviksi. Esimerkiksi varautuminen peruutukseen vie resursseja koska muutosta edeltänyttä tilaa on säilytettävä commit-operaation asti.