1. Relaatiomalli

Kaikki relaatiotietokannat - mukaan lukien kirjassa käsitellyt kahdeksan tuotetta - perustuvat IBM:n tutkija E. F. Coddin v.1970 julkaisemaan relaatiomalliin (the relational model). Relaatiomalli määrittelee relaatiotietokantojen teoreettisen pohjan perustuen joukko-oppiin, matematiikkaan ja predikaattilogiikkaan.

Coddin määrittelemä relaatiomalli aiheutti aikanaan vallankumouksen tietokantamaailmassa, sillä siihen perustuvat relaatiotietokantatuotteet ovat syrjäyttäneet aiemmin käytetyt hierarkkiset (esim. DL/1) ja verkkomalliset (esim. IDMS, MDBS) tietokantatyypit.

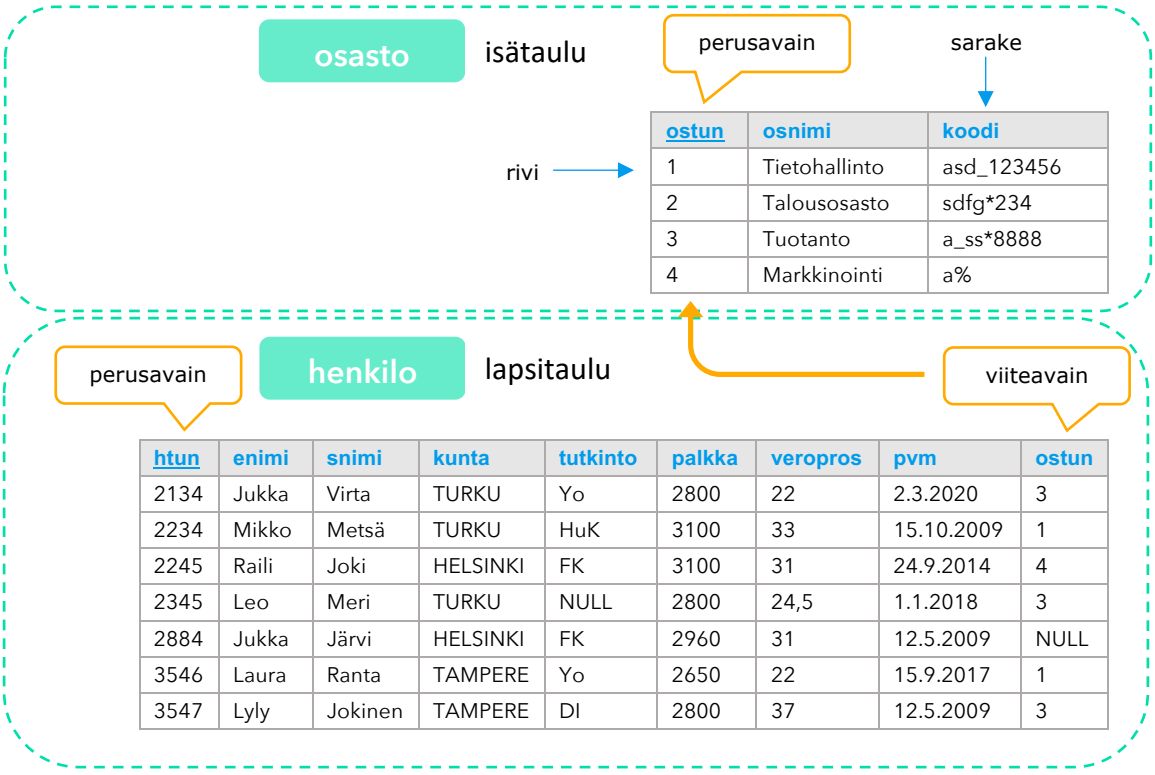
SQL–kieli on standardoitunut kaikkien relaatiotietokantatoimittajien tietokantakieleksi. Valtaosa kaikista uusista tietojärjestelmistä sekä tietovarastoista rakennetaan edelleen relaatiokantatuotteiden avulla. Relaatiomallin ymmärtäminen on jokaisen tiedonhallinnasta kiinnostuneen henkilön peruspilari!

Relaatiomalli voidaan jakaa kolmeen osaan: rakenne, käsittely ja eheyssäännöt. Käyn nämä osat seuraavaksi läpi (tässä ei ole koko relaatiomalli, mutta näillä tiedoilla pääsee jo pitkälle).

1.1 Rakenne

Taulu

Seuraavassa kuvassa on osastoja ja osastoihin liittyviä henkilöitä. Osastoon voi liittyä monta henkilöä ja yksi henkilö kuuluu aina yhteen osastoon.



ostun	osnimi	koodi
1	Tietohallinto	asd_123456
2	Talousosasto	sdfg*234
3	Tuotanto	a_ss*8888
4	Markkinointi	a%

htun	enimi	snimi	kunta	tutkinto	palkka	veropros	pvm	ostun
2134	Jukka	Virta	TURKU	Yo	2800	22	2.3.2020	3
2234	Mikko	Metsä	TURKU	HuK	3100	33	15.10.2009	1
2245	Raili	Joki	HELSINKI	FK	3100	31	24.9.2014	4
2345	Leo	Meri	TURKU	NULL	2800	24,5	1.1.2018	3
2884	Jukka	Järvi	HELSINKI	FK	2960	31	12.5.2009	NULL
3546	Laura	Ranta	TAMPERE	Yo	2650	22	15.9.2017	1
3547	Lyly	Jokinen	TAMPERE	DI	2800	37	12.5.2009	3

Kuva 1.1: Esimerkki kahdesta taulusta: osasto ja henkilo

Tällaiset asiakokonaisuudet, kuten osastot ja henkilöt esitetään relaatiotietokannoissa **tauluina (table)**, kuten yllä taulut osasto ja henkilo. Taulussa on **sarakkeita (column)** kuten yllä sarakkeet ostun, osnimi ja koodi, sekä **rivejä (row)**. Joskus tauluja kutsutaan myös taulukoiksi, sarakkeita kentiksi (field) ja rivejä tietueiksi (record). Sarakkeille määritellään tietotyyppi, kuten numeerinen tai merkkimuotoinen sekä pituus. Kaikki tiedot relaatiokannoissa tallettuvat tauluihin ja vain tauluihin.

Taulut myös perustetaan SQL-kielellä, puhutaan kielen DDL-osasta (Data Definition Language). Lue lisää luvusta Taulujen määrittely ja muuttaminen.

Perus- ja viiteavain

Kussakin taulussa on tunnisteena **perusavain (primary key, PK)**, kuten kuvassa osasto -taulussa ostun ja henkilo -taulussa htun (alleviivattu). Perusavaimen on oltava **yksilöivä** eli uniikki eli sarakkeessa ei saa olla kahdella (tai useammalla) eri rivillä samaa arvoa. Esimerkiksi henkilo -taulussa kullakin rivillä on oma yksikäsitteinen tunnuksensa. Perusavain voi koostua useammastakin sarakkeesta. Perusavaimen määrittelystä kerrotaan kappaleessa Taulujen perustaminen.

Henkilöt liittyvät yrityksiin. Osastolla voi olla monta henkilöä, mutta yksi henkilö kuuluu aina yhteen osastoon. Tätä kutsutaan **isä-lapsi –yhteydeksi** tai yksi-moneen-yhteydeksi. Isällä voi olla monta lasta, mutta lapsella on yksi isä, ks. kuva 1.1. Asian hoitamiseksi on lapsitaulussa eli henkilo -taulussa linkkikenttä eli **viiteavain** (foreign key) ostun, joka viittaa osasto -taulun perusavaimeen ostun. Viittaavaa taulua sanotaan **lapsitauluksi** ja viittauksen kohteena olevaa taulua **isätauluksi**. Viiteavaimia tarvitaan, kun tauluja halutaan yhdistää eli kun tehdään liitoksia. Viiteavaimia kutsutaan joskus myös vierasavaimiksi. Lue lisää liiteavainten määrittelystä kappaleesta Viiteeheys.

Kuva 1.2: Lapsilla on isän perusavain (jonka arvo tässä kuvassa on 1) viiteavaimena

Tyhjä-arvo NULL

Kuvasta 1.1 huomaamme, että henkilön 2345 tutkinnon kohdalla on NULL. Jos tietokantaan syötettäessä sarakkeella ei ole arvoa, tulee tuohon kohtaan tauluun erityinen merkintä, ns. NULL-arvo eli tyhjäarvo. NULL ei tarkoita välilyöntiä eikä nollaa, vaan tuntematonta arvoa. Tuntemattoman arvon mukana olon vuoksi relaatiokannoissa on ns. kolmiarvoinen predikaattilogiikka. Kysyttäessä onko tietyn henkilön tutkinto FK, on vastaus joko tosi, epätosi tai ehkä. Ehkä on siis vastaus, jos kantaan on tallentunut NULL-arvo. On siis kolme arvoa tavanomaisen kahden arvon sijasta.

Mihin NULL-arvoja sitten tarvitaan? Katsotaan esimerkkinä taulua, jossa on päivämääriä ja lämpötiloja ja muitakin mittaustuloksia. Joka aamu tauluun talletetaan mittausarvoja, ks. taulun kuva alla. Neljäntenä päivänä lämpömittari on rikki ja nyt asteiksi talletetaan NULL eli tuntematon lämpöila. SQL:n keskiarvofunktio AVG laskee keskiarvon ohittamalla NULL-arvot, kuten pitääkin. Jos NULL-arvoa ei olisi käytössä, olisi hankalaa päättää, mitä tuolloin talletettaisiin lämpötilaksi kun mittari on rikki. Monissa muissa tilanteissa NULL-arvot aiheuttavat vaikeuksiakin, palaan niihin myöhemmin.

pvm	asteet	muu_mittaus
1.12.2020	3	35
2.12.2020	0	27
3.12.2020	-1	27
4.12.2020	NULL	29
5.12.2020	1	31
...	...	...

SQL-kielessä on omat operaationsa NULL-arvon haulle, ks. kappale NULL-arvot. Taulua perustettaessa voi erikseen määritellä, onko NULL-arvo sallittu vai ei.

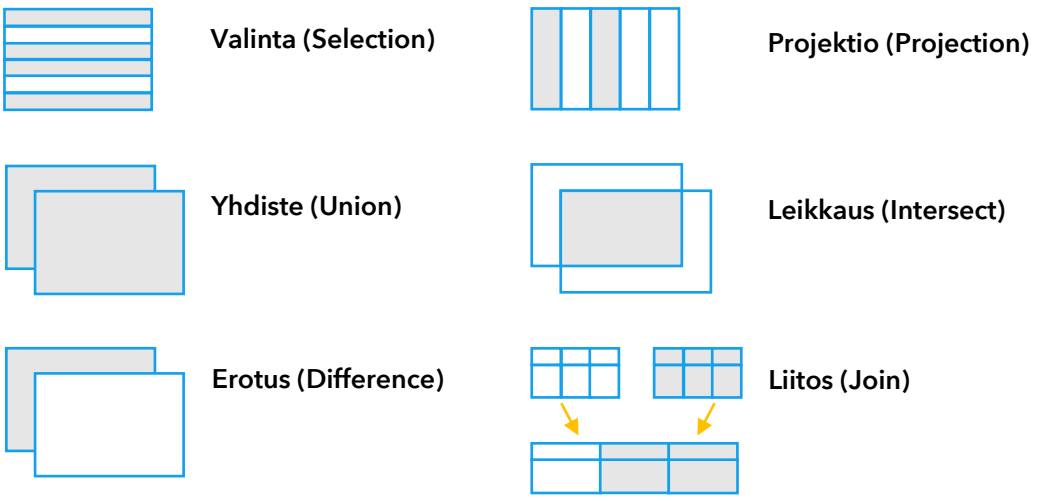
1.2 Käsittely

Joukko-oppi taustalla

Coddin nerokas oivallus relaatiomallin yhteydessä oli, että tietoja käsitellään **joukko-opillisesti**. Taulu muodostuu joukosta rivejä; tähän joukkoon voi sitten kohdistaa joukko-operaatioita, kuten "hae kaikki turkulaiset henkilot (valinta), niistä etunimi ja sukunimi (projektio)". Katso esimerkkejä tauluille tehtävissä olevista joukko-operaatiosta seuraavasta kuvasta.

Joukko-operaatiolla voi käsitellä koko taulun tai useampiakin tauluja. Joukko-opillisuus koskee myös päivityksiä. Yhdellä päivityskäskyllä voi esim. keskeltä Java-ohjelmaa päivittää ison joukon taulun rivejä, ilman silmukoita. Tämä tekee tietokantakäsittelystä hyvin tuottavaa.

Joukko-oppi toteutetaan käytännössä SQL –kielellä, joka siis nojautuu täysin joukko-oppiin. Kaikki kuvassa mainitut operaatiot voi tehdä SQL:n erilaisilla SELECT-käskyn muodoilla.



Kuva 1.3: Tärkeimmät joukko-operaatiot

7. Liitokset

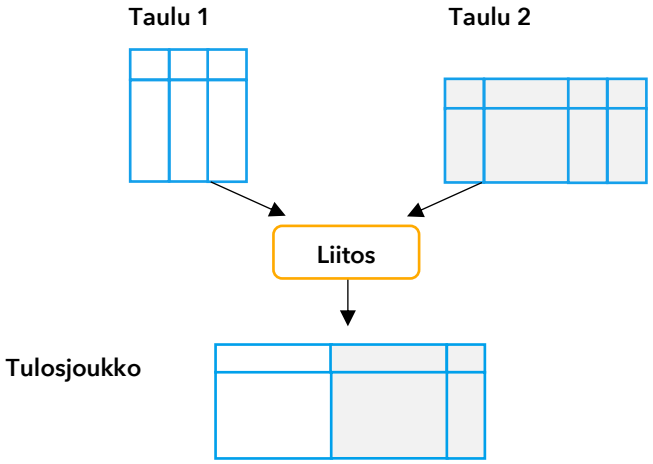
Tähän asti olemme hakeneet tietoja vain yhdestä taulusta kerrallaan. Kunnolla suunnitelluissa käytännön tietokannoissa tiedot ovat kuitenkin jakautuneet useisiin tauluihin (ns. normalisoitu) ja tietoja on yhdisteltävä eri tauluista halutun lopputuloksen saamiseksi. Tämä tehdään liitosten (join) avulla. Liitokset ovat erittäin keskeinen osa SQL-kyselyjä, niinpä käsittelen niitä sangen laajasti.

Liitoksen voi tehdä kahdella erilaisella syntaksilla. Uudempi ja suositeltava on ns. Join -syntaksi. Kutsun tässä vanhempaa tapaa perinteiseksi syntaksiksi. Esittelen ensin Join -syntaksi ja sitten perinteisen syntaksin ja lopuksi vertaan niitä.

7.1 Join -syntaksi

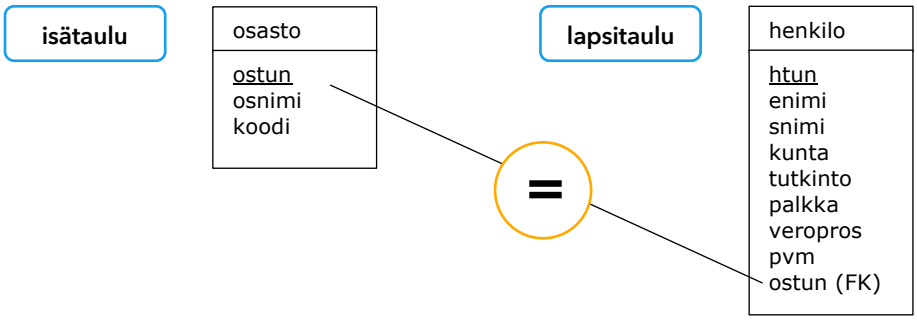
Kahden taulun liitos

Seuraava kuva esittää liitoksen periaatetta.



Kuva 7.1: Liitos yhdistelee useamman taulun sarakkeita samaan tulosjoukkoon.

Hae kunnassa Turku työskentelevien henkilöiden nimet ja osastot; hae ostun, osaston nimi, henkilön nimi ja kunta. Lajittele osaston nimellä.



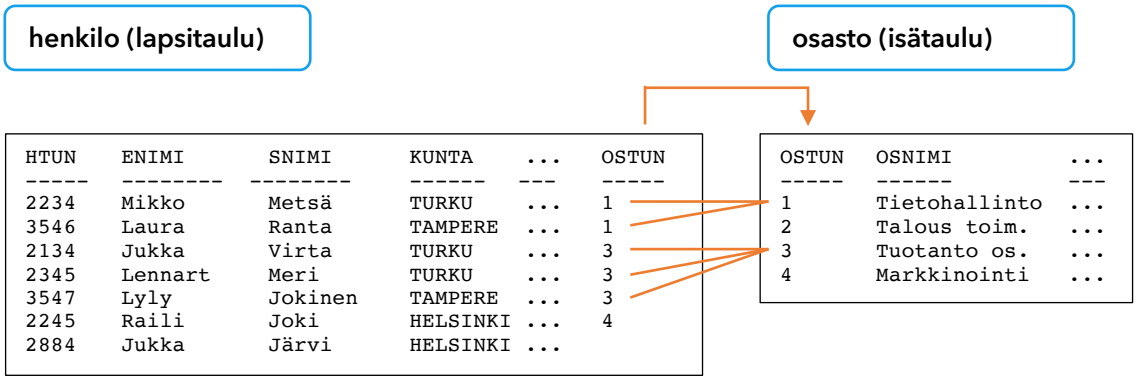
Kuva 7.2: Taulu henkilo on lapsitaulu, jonka viiteavain (FK) ostun viittaa isätaulun osasto perusavaimeen ostun. Tämä yhteys kerrotaan liitoskyselyn liitosehdossa

```
SELECT osasto.ostun, osnimi, snimi, enimi, kunta
FROM osasto
     JOIN henkilo
       ON (osasto.ostun = henkilo.ostun)  -- Liitosehto JOIN-lauseen ON-osiossa
WHERE kunta = 'TURKU'                     -- sulut vain selvyyden vuoksi
ORDER BY osnimi
```

```
   OSTUN  OSNIMI          SNIMI      ENIMI      KUNTA
-------- --------------- ---------- ---------- -------
       1 Tietohallinto   Metsä      Mikko      TURKU
       3 Tuotanto        Virta      Jukka      TURKU
       3 Tuotanto        Meri       Leo        TURKU
```

Vaadittavien tietojen hakemiseksi tarvittiin kahta taulua, nimttäin tauluja osasto ja henkilo. Ne kuvataan yllä olevalla tavalla FROM -lauseessa. Nyt voidaan SELECT-lauseessa viitata kaikkiin näiden taulujen sara¬kenimiin. Ei ole väliä kumman taulun mainitset ensin FROM-lauseessa.

Kun liitoksessa liitämme rivejä kahdesta taulusta toisiinsa, on tärkeää tietää, mitkä rivit yhdistetään minkäkin rivin kanssa. Tämä esitetään *liitosehdolla* FROM -lauseen ON -osiossa kertomalla, että yhdistävässä sarak¬keessa pitää molemmissa tauluissa olla sama arvo. Jos molemmissa tauluissa on sama arvo, yhdistyvät rivit yhdeksi riviksi tulosjou¬kossa. Lähes aina yhdistämme isä-taulun perusavaimen (yllä osasto.ostun) lapsi-taulun viiteavaimeen (yllä henkilo.ostun), ks. seuraava kuva. Vasemmalla on henkilo-taulu, jossa näkyvät turkulaiset ja oikealla osasto –taulu. Viivat kertovat, miten nyt saman osaston tunnuksen omaavat rivit yhdistyvät.



Kuva 7.3: Kuva näyttää konkreettisesti, mitkä rivit yhdistyvät toisiinsa.

Huomaa yllä olevassa SQL-esimerkissä sarakenimi osasto.ostun, joka tarkoittaa: osasto -taulusta ostun. Sanomme, että olemme *kvalifioineet* ostun-sarakkeen laittamalla taulun nimen sarakkeen eteen. Kvalifiointi on *pakollista* silloin, *kun saman niminen sarake esiintyy useassa FROM -lauseen taulussa*. Esimerkissä ostun on sekä osasto- että henkilo -tauluissa. Ilman kvalifiointia järjestelmä ei tietäisi, kummasta taulusta ostun-saraketta haetaan, ja protestoisi virheilmoituksella. Sarakkeita onimi, snimi, enimi ja kunta ei siis ole pakko kvalifioida, sillä ne esiintyvät vain yhdessä kyselyn tauluista.