Tietokannat

Alustava sisältösuunnitelma

- Mikä tietokanta on ? Entä relaatiotietokanta ?
- Relaatiotietokannan suunnittelun periaatteet. Normalisointi.
- MySQL:n käyttö suoraan Linux-palvelimella
- Tietokannan luonti, tietokantamoottorit
- Taulut, tietotyypit, määreet, avaimet
- Taulujen luonti
- Yksinkertaiset kyselyt tietokannasta
- Edistyneemmät kyselyt useasta taulusta
- Apuohjelmia: PHPMyAdmin ja MySQL Workbench
- Näkymät (Views) ja aliohjelmat (Procedures)
- Liipaisin
- Tietokannan käsittely WWW-sivun kautta. PHP.

Vähän johdantoa

Tämä materiaali käyttää MySQL-tietokantaa. Monia muitakin toki on. Eipä niistä nyt sen enempää. MySQL:ää kehittää ruotsalainen yritys MySQL AB. Sun Microsystems osti yrityksen 16. tammikuuta 2008.[2] Ohjelmistoyritys Oracle Corporation osti Sun Microsystemsin huhtikuussa 2009 noin 7,4 miljardilla dollarilla. Kaupan yhteydessä MySQL:n omistus siirtyi Oraclelle. MySQL on saatavissa vapaalla GNU GPL -lisenssillä tai kaupallisella lisenssillä, mikäli asiakas ei halua käyttää GPL lisensoitua ohjelmistoa.

MySQL-tietokannan loi vuonna 1995 suomalainen Michael "Monty" Widenius yhdessä ruotsalaisen David Axmarkin kanssa. MySQL:n ensimmäinen versio julkaistiin 1996. Ubuntu Linuxin mukana tulevat versio on 5.5 (syksyllä 2014).

MySQL-ohjelman kerrotaan saaneen nimensä toisen tekijän My-tyttären mukaan.[3] MySQL:n logo on suomalaisen mainostoimiston tekemä.

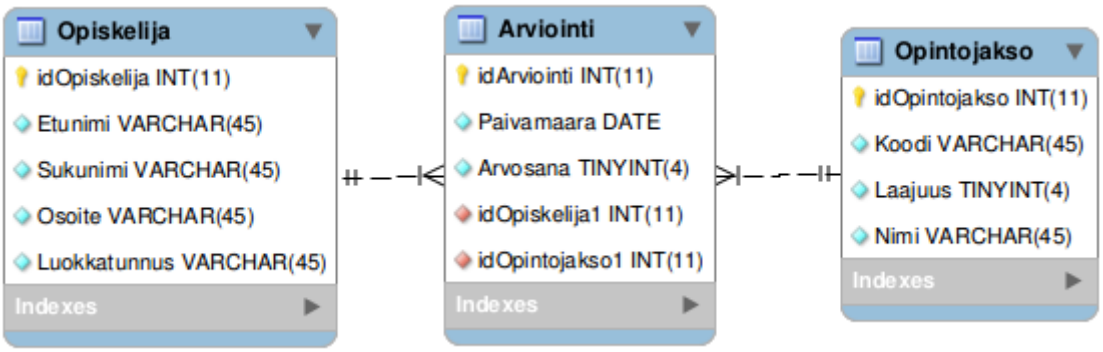
MySQL-tietokanta on hyvin suosittu web-palveluiden tietokantana. MySQL-tietokannan päälle rakennettava ohjelmalogiikka tehdään usein PHP-, Python- tai Perl-ohjelmointikielellä, sivut julkaistaan Apache-webpalvelimella, joka edelleen toimii Linux-käyttöjärjestelmän päällä. Tätä kutsutaan joskus LAMP-alustaksi. Myös muilla ohjelmointikielillä on mahdollista käyttää MySQL-tietokantaa. MySQL sisältää rajapinnan mm. C:lle, C++:lle, C#:lle, Smalltalkille, Javalle, Rubylle ja TCL:lle.

Merkittäviä käyttäjiä: Wikipedia Google Yahoo! NASA Facebook

Mikä on tietokanta ?

Tietokanta on joukko yhteen liittyviä tietoja. Tietokanta saattaa edustaa jotain selvästi rajattua kohdetta reaalimaailmasta. Tällainen kohde voi olla esimerkiksi yrityksen keräämät tiedot asiakkaistaan.

Kuvassa on esimerkkinä opintorekisteri, josta löytyy tiedot opiskelijoista, opintojaksoista ja opintosuorituksista.

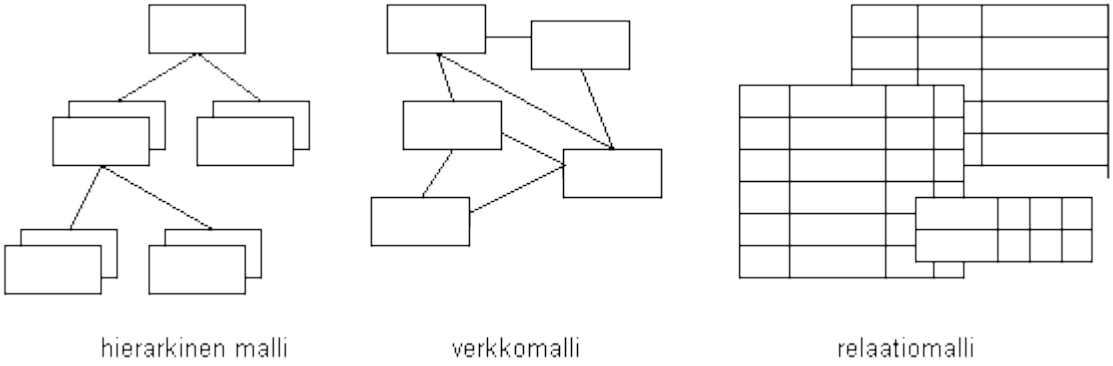


Tietokantojen koot voivat vaihdella suuresti, yhteen tiedostoon tallennetuista taulukoista hyvin suuriin tietokantoihin joissa on useita miljoonia tietueita lukuisista kiintolevyistä koostuvilla levypakoilla. Tietokantaan voidaan tallentaa eri formaateissa olevaa tietoa, esimerkiksi tekstiä, ääntä ja videokuvaa.

Ehkä meidän pitäisi puhua tietokantayhteiskunnasta eikä tietoyhteiskunnasta, sillä tietokannat ovat keskeinen osa elämäämme. Esimerkiksi vakuutukset ja pankkiasiat, kaupassa käynti (bonuskortit), verotus, väestörekisteri, Kela, terveydenhuolto jne.

Webissä on nykyisin runsaasti tietokantoja. Tunnetuin tietokanta netissä lienee Google, mutta onhan niitä muitakin. Netin keskustelupalstat (Vauva-lehti, Suomi24, Kalevan Juttutupa jne...) perustuvat tietokantojen hyödyntämiseen, samoin vaikkapa tällä kurssilla käytetty Optima-ympäristö.

Relaatiotietokantamallin ja käsittelyteorian kehitti Edgar F. Codd vuonna 1970. Codd työskenteli IBM:n tutkimuslaboratoriossa ja hän johti ensimmäisten relaatiotietokantojen testiversioiden kehitystyötä. Vuosina 1973-1976 IBM laboratoriossa kehitettiin relaatiotietokannan prototyyppiä nimeltä System R. IBM:n kaupallinen tietokantatuote DB2 julkaistiin 1982. Tietokannat on käytännöllisintä luokitella tuetun ohjelmointimallin mukaan. Muutamat malleista ovat olleet laajalti käytössä jo jonkin aikaa. **Hierarkkinen malli** toteutettiin ensimmäisenä, sen jälkeen **verkkomalli**, sitten relaatiomalli ohitti ne niin kutsutun **"Flat-File"**-mallin kanssa, joka oli helppo toteuttaa vaatimattomiinkin alustoihin. Hierarkkinen, verkko ja Flat-tiedostomalli eivät perustu vahvaan teoreettiseen pohjaan niin kuin relaatiomalli, vaan ne ovat syntyneet laitteiston ja ohjelmointiteknisten rajoitteiden vaikutuksesta.



Kuva 2: Tietokantamalleja

Tällä kurssilla ei mennä kovin syvälle tietokantojen teoriaan. Se on enemmän tietojenkäsittelytieteilijöiden ja matemaatikkojen asiaa. Muutama tietokannan suunnitteluperiaate riittää varsin pitkälle.

Varsin yleistajuinen kirja on esimerkiksi *Michael J. Hernandez: Database Design for Mere Mortals: A Hands-on Guide to Relational Database Design, Third Edition*.

Relaatiotietokanta

Relaatiotietokanta koostuu useista **taulu**ista (engl. *table*). Tauluissa tiedot esitetään riveissä (**tietue**) ja sarakkeissa (**kenttä**). Tietokannassa tarvittavat tiedot pyritään jakamaan tauluihin siten, että yksi tieto tallennetaan vain yhteen paikkaan. Relaatiotietokantaan tallennetaan myös tieto siitä, miten eri taulut liittyvät toisiinsa. Tässä on esitetty erääseen MySQL-tietokantaan tehdyn taulun (nimeltään "pet") sisältö. Tämä taulu tullaan tekemään harjoituksissa.

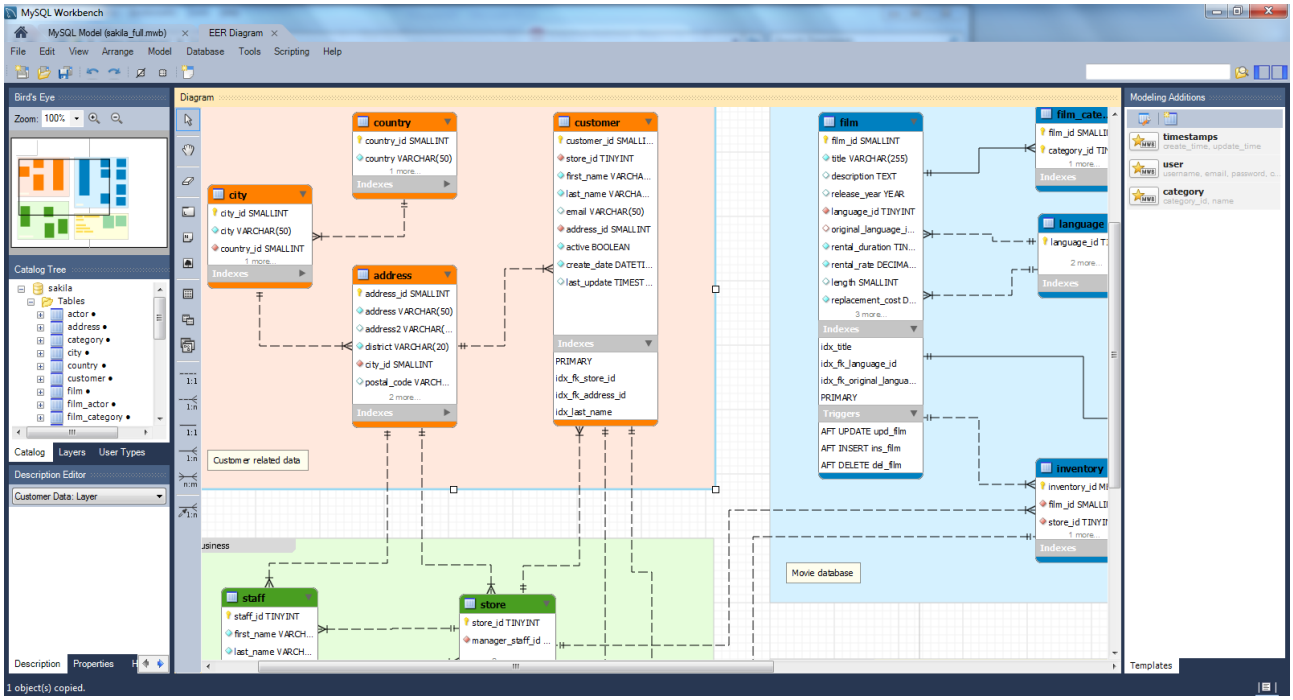
```
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
+----------+--------+---------+------+------------+------------+
```

Tauluun voidaan tehdä **kyselyjä** (**queries**). Erilaisia kyselykieliä (query languages) on hirveä määrä. Ks. esim. oheinen Wikipedia-artikkeli (http://en.wikipedia.org/wiki/Query_language). Tällä kurssilla käytetään yhtä yleisimmistä eli SQL:ää. Nimitys ”query” on jossain määrin harhaanjohtava. SQL:n avulla voidaan tehdä paljon muutakin kuin kyselyjä.

Esimerkiksi edellinen taulu on tulostettu SQL-lauseella ja se voitaisiin suomentaa ”Valitse kaikki tietokannasta pet”:

```
SELECT * FROM pet;
```

Taulukkojen välille luodaan **yhteys**, jotka nopeuttavat ja tarkentavat tietojen päivittämistä, sillä muutos on tehtävä vain yhteen paikkaan. Relaatiotietokanta on suunniteltava huolellisesti, sillä hyvin suunnitellusta tietokannasta voi helposti tyydyttää monimutkaiset tietotarpeet.



Kuva 3. Esimerkki monimutkaisesta relaatiotietokannasta

Relaatiotietokannassa tiedot esitetään siis tauluina. Yhtä tietokantataulun riviä kutsutaan tietueeksi (engl. **record**). Taulun jokaisella rivillä on yhtä monta tietoa eli kenttää (engl. **field**). Jokaisella rivillä täytyy olla yksikäsitteinen perusavain (engl. **primary key**), joka vastaa jotakin reaalimaailman kohdetta (esim. jokaisella työntekijällä on yksilöllinen tunniste). Kuhunkin kohteeseen liitetään vain siihen välittömästi liittyvät ominaisuudet. Kukin yksittäinen tieto relaatiotietokannassa voidaan hakea ainakin ilmoittamalla taulun nimi, perusavaimen kentän nimi ja avaimen arvo sekä haettavan tiedon kentän nimi. Relaatiotietokannasta tietoa haetaan vain tiedon nimien ja arvojen perusteella, ei siis koskaan tiedon sijainnin tai järjestyksen mukaan.

Tietokannan kentällä on koko joukko ominaisuuksia, jotka täytyy tuntea ja osata määritellä tietokantaa luodessa. Tärkeimmät näistä ominaisuuksista ovat *kentän nimi (field name)* ja *tietotyyppi (datatype)*.

- Kentän nimi (engl. **fieldname**).
- Tietotyyppi (engl. **datatype**), joka voi olla numeerinen, alfanumeerinen (teksti), päivämäärä ja/tai kellonaika, looginen tai bittijono.
- Maksimipituus ja mahdollinen tarkkuus (engl. **maximum length, precision**).

- Onko tieto pakollinen (engl. **required**).
- Tarkistukset syötettävän tiedon oikeellisuudelle tai sallittujen arvojoukkojen joukko (engl. **validity checks**).
- Oletusarvo (engl. **default value**), jota käytetään jätettäessä kenttä tietoja syötettäessä tyhjäksi.
- Syöttömaski (engl. **Input Mask**), jolla kuvataan kenttään kelpuutettavat merkit ja niiden muoto.

Kentän nimenä kannattaa käyttää kentän tietosisältöä kuvaava ja selkeää nimeä. Älä käytä nimessä välilyöntejä, tavuviivaa, erikoismerkkejä äläkä skandinaavisia aakkosia. Niistä aiheutuu todennäköisesti ongelmia.

Mahdollisimman moni kenttä kannattaa määritellä pakolliseksi tai käyttää oletusarvoa. Tällöin vältetään NULL-ongelmia. NULL on erityinen merkintätapa puuttuvalle arvolle. NULL ei tarkoita välilyöntiä tai nollaa. Jos kentälle ei syöttövaiheessa anneta mitään arvoa, niin sen sisällöksi tulee NULL. Se tarkoittaa lähinnä tuntematonta arvoa eli sen arvo voisi periaatteessa olla mikä tahansa tai ei mikään.

Taulujen yhteydet

Tietokanta sisältää yleensä useamman kuin yhden taulun. Yhtä tärkeää kuin tietojen ryhmittely omiin tauluihinsa on se, miten taulut (käsitteet) liitetään toisiinsa. Käsitteiden välillä on looginen yhteys, joka on luotava myös taulujen välille. Tällöin eri tauluissa olevia tietoja voidaan käsitellä samanaikaisesti. Kun tietoja päivitetään yhdessä taulussa, muutokset tallentuvat samalla myös muihin tauluihin. Suuri osa tietokantatyöstä onkin käsitteellistä suunnittelua erilaisia malleja käyttäen (palataan niihin myöhemmin).

Yhteys kahden taulun välille muodostetaan **viiteavaimen** avulla. Ns. ensisijaisen taulukon **perusavain eli pääavain** kopioidaan toiseen taulukkoon viiteavaimeksi.

Käsitteiden välillä on kolmenlaisia riippuvuuksia: yhden suhde yhteen, yhden suhde moneen ja monen suhde moneen [1].

Yhteystyypit:

- **Yksi-yhteen:** Kukin ensimmäisen taulun tietue on liitetty vain yhteen toisen taulun tietueeseen ja päinvastoin. Harvoin käytetty yhteys, koska yleensä tällä tavoin liitetyt tiedot ovat samassa taulussa. Esim. mies-vaimo.

1 Hieman yksinkertaistaen.

- **Yksi-moneen:** kukin ensimmäisen taulun tietue on liitetty yhteen tai useampaan toisen taulun tietueeseen. Tavallisimmin käytetty yhteys. Esim. äiti-lapsi.

- **Monta-moneen:** Kukin ensimmäisen taulun tietue on liitetty useaan toisen taulun tietueeseen, ja kukin toisen taulun tietue on liitetty useaan ensimmäisen taulun tietueeseen. Esim. opettaja-oppilas.

Jotta taulujen tiedot pysyisivät yhtenäisinä, voidaan yhteyksiä määriteltäessä määrätä taulujen välille **viite-eheys**. Viite-eheys varmistaa, että taulujen sisältämät tiedot pysyvät ajan tasalla siinä tilanteessa, että jokin tietue poistetaan.

Tietokannan suunnittelu

Hyvä tavoite tietokannan suunnittelussa on, että tietoa on helppoa lisätä, hakea ja muuttaa.

1. **Tietokannan taulut ja niiden kentät ovat pysyviä**

Tietokanta on suunniteltava valmiiksi ennen kuin sitä voi käyttää. Olemassa olevan tietokannan sisällön muuttaminen on vaikeaa ja monesti jopa mahdotonta.

2. **Yksi tietokannan kenttä saa sisältää vain yhden tiedon**

Tietokannan kenttä saa sisältää yksittäisen tiedon, kuten henkilön nimen, tuotteen hinnan, viestin sisällön tai päivämäärän. Sen sijaan kentässä ei saa olla listaa henkilöistä, tuotteista, viesteistä, päivämääristä tai muista erillisistä asioista.

Esimerkiksi jos tietokanta sisältää verkkokaupan asiakkaiden ostoskorien sisällöt, ei ole järkevää tallentaa asiakkaan kaikkia ostoksia yhdelle riville, joka sisältää merkkijonokentän muotoa "lanttu,purjo,selleri". Tällaisen kentän käsittely SQL-kyselyillä olisi vaivalloista. Järkevä ratkaisu on antaa jokaiselle asiakas-tuote-parille oma rivi, jolloin yhdessä kentässä on vain yksi tuote.

3. **Sama tieto saa esiintyä vain yhdessä paikassa tietokannassa.**

Tietokannassa ei saa olla toistuvaa ja päällekkäistä tietoa. Jos samaa tietoa tarvitaan monessa paikassa, tieto on tallennettu vain yhteen paikkaan ja muualla on viittaus siihen.

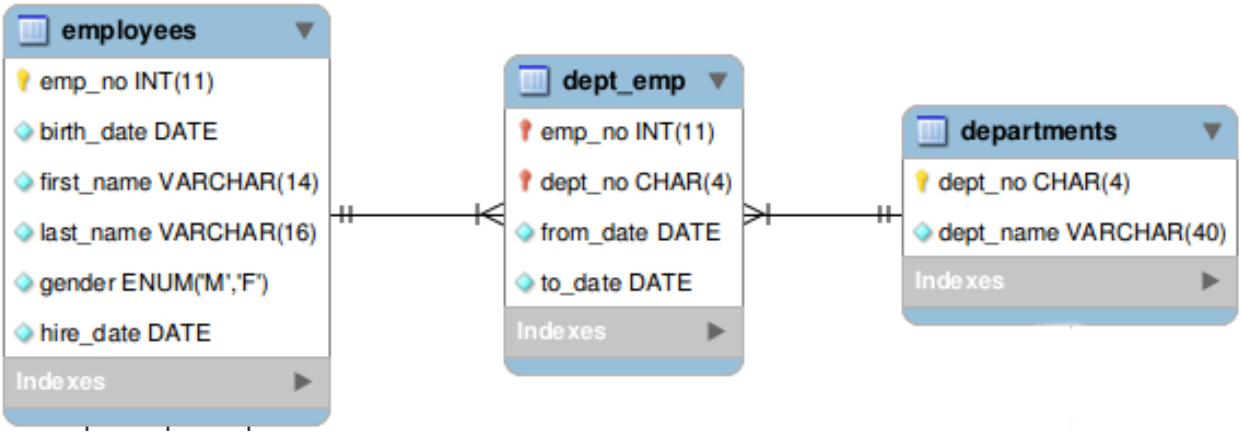
Tästä on huono esimerkki edellä esitetty pet-taulu. Siinähän omistaja-tieto esiintyy useaan kertaan.

4. **Jokaisessa taulussa on oltava kenttä, joka yksilöi jokaisen tietueen**

Esimerkiksi, jos taulussa on ihmisten nimiä, eivät pelkät etunimet ja sukunimet riitä varmaan yksilölliseen tunnistukseen. Useilla ihmisillä voi olla sama nimi. Matti Virtasia on Suomessa monta. Sen sijaan henkilötunnus on jokaiselle Suomen kansalaiselle varmasti yksilöllinen.

Tällainen yksilöllinen tunniste on nimeltään **pääavain** (primary key, lyhenne PK). Usein, jos taulussa ei ole hetun kaltaista itsestäänselvää tunnistetta, on pääavain automaattisesti kasvava kokonaisluku.

5. **Taulujen välillä ei saa olla monesta moneen-yhteyksiä**



Kuva 4. Monesta moneen yhteys.

Taulujen välillä ei saa olla monesta moneen-tyyppisiä yhteyksiä. Yhteyksien on oltava tyyppiä yhden suhde moneen. Esimerkki monesta moneen yhteydestä: ”Osastolla on monta työntekijää. Sama työntekijä voi työskennellä monella osastolla”.

Tällaiset yhteydet puretaan ”välitaulun” kautta yhdestä moneen-yhteyksiksi. Kuvassa `dept_emp` on välitaulu, johon kootaan työntekijä-osasto-avainparit. Sekä työntekijöillä että osastoilla on yhden suhden moneen yhteys tähän välitauluun. Ilmoittautumistauluun on koottu viittaukset työntekijä- ja osastotaulujen pääavaimiin (Primary key, kentät `emp_no` ja `dept_no`) ns. *viiteavaimilla* (Foreign Key, FK). Viiteavaimista käytetään myös suoraa käännöstä *vierasavain*.

Ohjelmiston vaatimusmäärittely

Tietokantaa suunniteltaessa pitää jäsentää, mitä tietoja kantaan ollaan tallentamassa. Tätä tarkoitusta varten tehdään yleensä vaatimusmäärittely, jonka pohjana toimivat käyttäjien haastattelut (http://fi.wikipedia.org/wiki/Ohjelmiston_vaatimusm%C3%A4%C3%A4rittely)

Ohjelmiston vaatimusmäärittely on dokumentti, jossa kuvataan ohjelmistoprojektin tavoitteita ja vaatimuksia. Siinä määritellään miten lopullisen ohjelmiston tulisi toimia ja millä keinoilla nämä toiminnallisuudet saavutetaan. Toiminnallisuuksia kuvataan usein **käyttötapausten** avulla. Käyttötapaukset kuvaavat käyttäjän ja ohjelmiston välistä vuorovaikutusta. Vaatimusmäärittely jaotellaan pääpiirteissään kahteen osaan eli **toiminnallisiin** ja **ei-toiminnallisiin** vaatimuksiin. Ei-toiminnallisia vaatimuksia ovat laadulliset vaatimukset ja resurssivaatimukset. Eri osat voivat olla myös omia erillisiä dokumenttejaan. Vaatimusmäärittely tehdään yhdessä asiakkaan kanssa.

- Toiminnallisissa vaatimuksissa määritellään ohjelmiston toiminnallisuudet eli ne asiat mitä lopullisen ohjelmiston tulee käytännössä sisältää ja miten sen tulee toimia. Toiminnot kuvataan yleensä syötteinä ja tulosteina. Toiminnallisissa vaatimuksissa ei niinkään

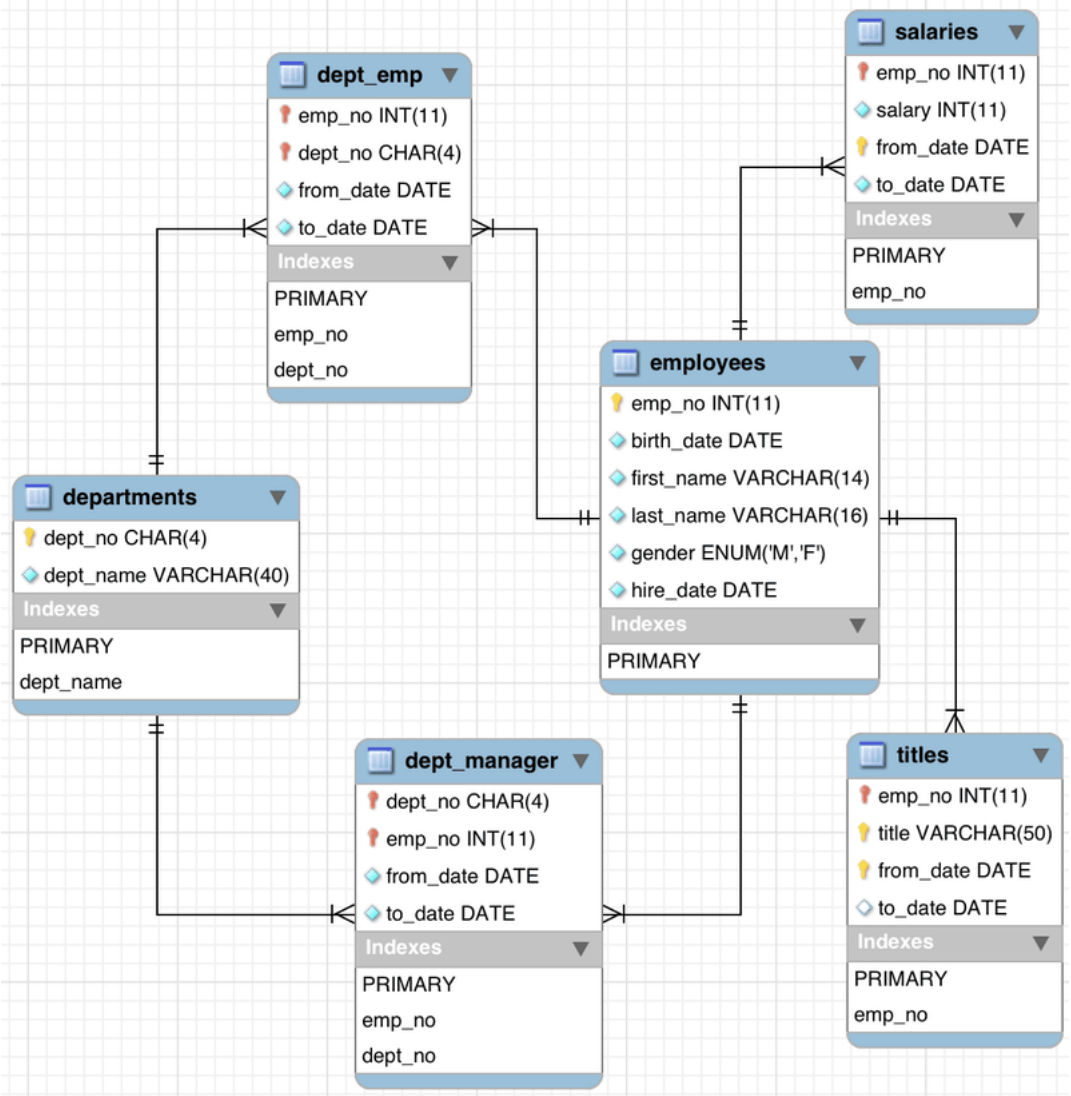
keskitytä sisäisiin toimintoihin vaan siihen, miten järjestelmä reagoi ulkoa tuleviin tapahtumiin.

- Laadullisissa vaatimuksissa määritellään se miten ohjelmisto tekee ne asiat mitä sen vaaditaan tekevän. Yleisiä ohjelmiston laadun määritelmiä ovat esimerkiksi oikeellisuus, käytettävyys, suorituskyky, luotettavuus/turvallisuus, siirrettävyys, uudelleenkäytettävyys ja ylläpidettävyys.
- Resurssivaatimuksissa määritellään kuinka paljon asiakas on valmis sijoittamaan rahaa projektiin. Usein joudutaan tekemään kompromisseja kulujen ja projektin valmistumiseen kuluvan ajan välillä.

Tietokannan mallintaminen: ER-diagrammi

Vaatimusmäärittelyn pohjalta tietokannan suunnittelija muodostaan tarvittavat taulut, taulujen sisältämät tiedot ja taulujen väliset yhteydet. Tämän materiaalin kirjoittajan mielestä tietokannan suunnittelu on melkeinpä vaikein asia mihin tietokantojen yhteydessä törmää. Siksi ei mennä vielä tässä vaiheessa kurssia tähän aihealueeseen. Opetellaan ensin tarvittavat perustiedot ja työkalut.

Seuraavassa kuvassa on esitetty MySQL:n esimerkkitietokannan Employees rakenne. Se on esimerkki hieman monimutkaisemmasta tietokannasta. Kuvaaja on piirretty MySQL Workbench-ohjelmalla.



Kuva 5. MySQL:n Employees-tietokannan rakenne ER-mallia ja Crows Foot-notaatiota käyttäen.

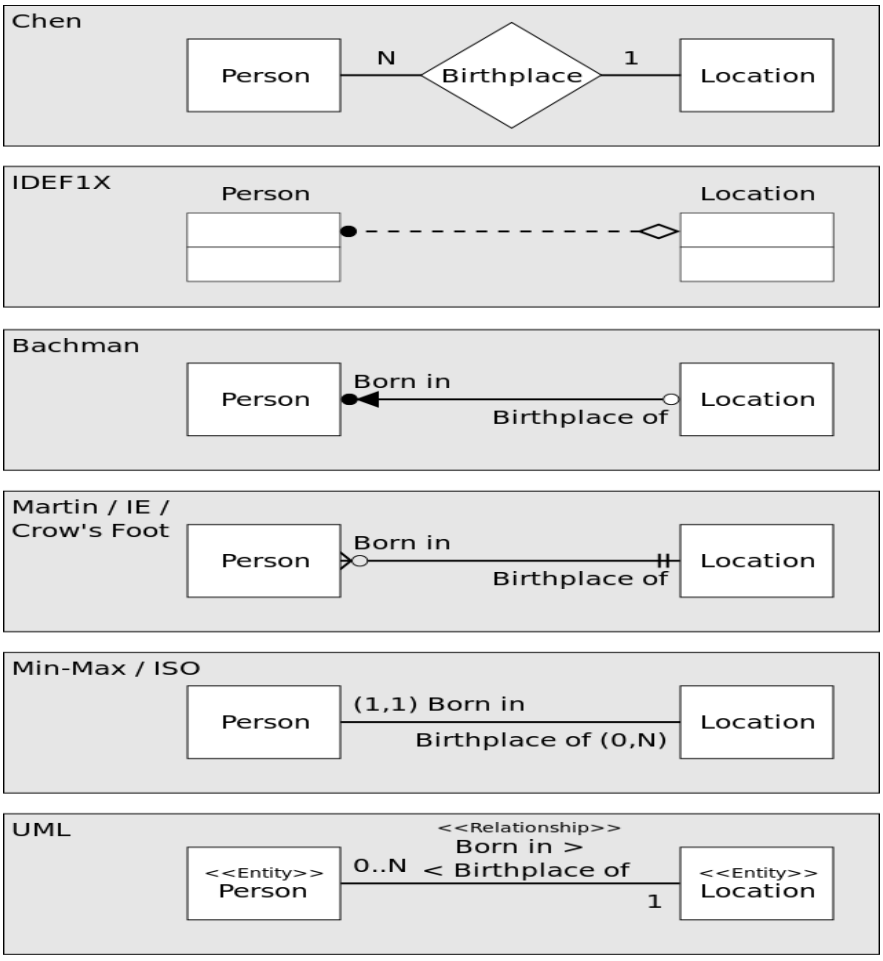
Edellisestä kuvasta voidaan päätellä muutamia asioita.

- Työntekijällä (`empolyee`) voi olla kerrallaan vain yksi nimike (`title`). Sama nimike voi olla usealla työntekijällä. Taulujen välinen yhteys on siis yhden suhde moneen.
- Yksi työntekijä voi työskennellä monella eri osastolla (`department`). Osastoilla on monia työntekijöitä. Yhteys on siten tyyppiä monen suhde moneen. Se ei ole sallittu, joten työntekijät ja osastot yhdistetään välitaulun `dept_emp` kautta.
- Taulujen yhdistäminen tapahtuu avainten avulla. Esimerkiksi `employees`-taulun pääavain `emp_no` löytyy myös taulusta `dept_emp`. Samoin `departments`-taulun pääavain löytyy taulusta `dept_emp`.

Tietokantojen yhteydessä käytetään tavallisimmin ns. ER-mallia (Entity-Relationship-Diagram, ERD) [2]. Erilaisia tapoja mallintaa asioita ja niiden välisiä yhteyksiä on paljon. Siihen tarkoitukseen on kokonainen oma mallinnuskieli UML (Unified Modeling Language). ER-malli on yksi osa UML-kieltä.

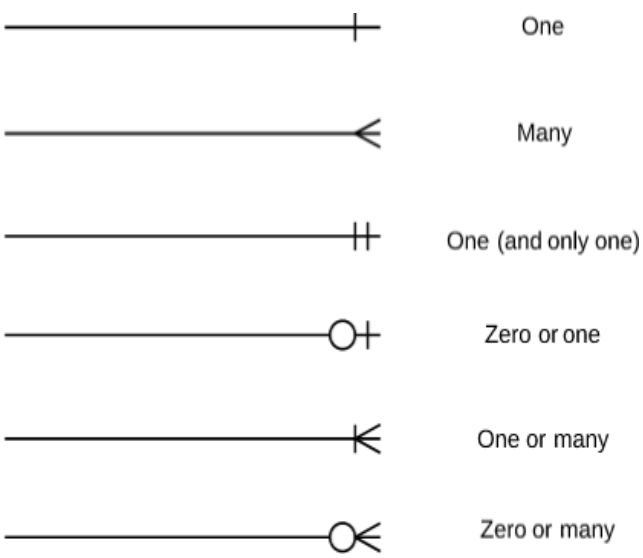
- Entity = entiteetti, jokin asia jota mallinnetaan. Käytännössä tietokannan yksi taulu
- Relationship = entiteettien (taulujen) välinen yhteys

ER-mallin sisälläkin on vielä useita eri merkintätapoja eli notaatioita (kuva).



2Huom: MySQL Workbench-ohjelma käyttää ER-mallin laajennettua versiota ”EER”-malli (Extended ER): *”The EER model includes all of the concepts introduced by the ER model. Additionally it includes the concepts of a subclass and superclass (Is-a), along with the concepts of specialization and generalization. Furthermore, it introduces the concept of a union type or category, which is used to represent a collection of objects that is the union of objects of different entity types.”*

Tällä kurssilla käytetään ”harakanvarvasnotaatiota” Crows Foot Notation, jossa taulujen väliset yhteydet merkitään seuraavasti:



MySQL-kielen tietotyypit

Tietokannan tauluja muodostettaessa on jokaisella kentällä oltava tietotyyppi. Esimerkiksi nimet ovat tekstityyppiä, kouluarvosana on kokonaisluku väliltä 4-10 jne. Tietotyyppi voi olla esimerkiksi aakkosnumeerinen, numeerinen, päivämäärä tai looginen. Alla on taulukoitu MySQL:n tietotyypit.

TEXT TYPES

```
CHAR( )         A fixed section from 0 to 255 characters long.
VARCHAR( )      A variable section from 0 to 255 characters long.
TINYTEXT        A string with a maximum length of 255 characters.
TEXT            A string with a maximum length of 65535 characters.
BLOB            A string with a maximum length of 65535 characters.
MEDIUMTEXT      A string with a maximum length of 16777215 characters.
MEDIUMBLOB      A string with a maximum length of 16777215 characters.
LONGTEXT        A string with a maximum length of 4294967295 characters.
LONGBLOB        A string with a maximum length of 4294967295 characters.
```

Suluissa voidaan antaa suurin merkkimäärä, esim.

```
VARCHAR(20)
```

CHAR and VARCHAR eniten käytettyjä tietotyyppejä. CHAR on kiinteän mittainen merkkijono ja sitä käytetään, kun datan pituus ei muutu paljon (esim. sosiaaliturvatunnus, auton rekisterinumero). VARCHAR on muuttuvanmittainen merkkijono, jota käytetään kun merkkijonon pituutta ei tiedetä etukäteen tai se voi muuttua.

CHAR-tyyppistä tietoa voi olla nopeampi hakea tietokannasta kuin VARCHAR-tyyppistä. Jos samassa taulussa on käytössä molempia tyyppeja, MySQL muuttaa automaattisesti CHAR:n VARCHAR:ksi.

BLOB tulee sanoista Binary Large Object. Sekä TEXT että BLOB ovat vaihtuvanmittaisia tyyppejä, joihin voidaan tallentaa suuria tietomääriä. Niiden käsittely on hidasta.

NUMBER TYPES	
TINYINT( )	-128 to 127 normal 0 to 255 UNSIGNED.
SMALLINT( )	-32768 to 32767 normal 0 to 65535 UNSIGNED.
MEDIUMINT( )	-8388608 to 8388607 normal 0 to 16777215 UNSIGNED.
INT( )	-2147483648 to 2147483647 normal 0 to 4294967295 UNSIGNED.
BIGINT( )	-9223372036854775808 to 9223372036854775807 normal 0 to 18446744073709551615 UNSIGNED.
FLOAT	A small number with a floating decimal point.
DOUBLE(,)	A large number with a floating decimal point.
DECIMAL(,)	A DOUBLE stored as a string , allowing for a fixed decimal point.

Kokonaislukutyypit voivat olla myös UNSIGNED (etumerkitön).

DATE TYPES	
DATE	YYYY-MM-DD.
DATETIME	YYYY-MM-DD HH:MM:SS.
TIMESTAMP	YYYYMMDDHHMMSS.
TIME	HH:MM:SS.

MISC TYPES	
ENUM ( )	Short for ENUMERATION which means that each column may have one of a specified possible values.
SET	Similar to ENUM except each column may have more than one of the specified possible values.

ENUM on enumeroitu tietotyyppi. Se siis sisältää listan ennalta määrättyjä arvoja (esimerkiksi vaikkapa viikonpäivien nimet). Taulun sarake voi sisältää vain yhden ENUM-tyyppisen arvon.

```
ENUM('y','n')
```

Listan pituus voi olla maksimissaan 65535 arvoa.

SET on samantyyppinen tietorakenne kuin lista, mutta se voi sisältää maksimissaan 64 arvoa ja siihen voi tallentaa enemmän kuin yhden arvon per sarakake.

Varatut sanat

Seuraavat sanat on tarkoitettu itse SQL-kielen käyttöön. Niitä ei voi käyttää omien tietokantojen tai taulujen nimissä.

ADD	ALL	ALTER
ANALYZE	AND	AS
ASC	ASENSITIVE	AUTO_INCREMENT
BDB	BEFORE	BERKELEYDB
BETWEEN	BIGINT	BINARY
BLOB	BOTH	BY
CALL	CASCADE	CASE
CHANGE	CHAR	CHARACTER
CHECK	COLLATE	COLUMN
COLUMNS	CONDITION	CONNECTION
CONSTRAINT	CONTINUE	CREATE
CROSS	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURSOR	DATABASE
DATABASES	DAY_HOUR	DAY_MICROSECOND
DAY_MINUTE	DAY_SECOND	DEC
DECIMAL	DECLARE	DEFAULT
DELAYED	DELETE	DESC
DESCRIBE	DETERMINISTIC	DISTINCT
DISTINCTROW	DIV	DOUBLE
DROP	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	FALSE
FETCH	FIELDS	FLOAT
FOR	FORCE	FOREIGN
FOUND	FRAC_SECOND	FROM
FULLTEXT	GRANT	GROUP
HAVING	HIGH_PRIORITY	HOUR_MICROSECOND
HOUR_MINUTE	HOUR_SECOND	IF
IGNORE	IN	INDEX
INFILE	INNER	INNODB
INOUT	INSENSITIVE	INSERT
INT	INTEGER	INTERVAL
INTO	IO_THREAD	IS
ITERATE	JOIN	KEY
KEYS	KILL	LEADING
LEAVE	LEFT	LIKE
LIMIT	LINES	LOAD
LOCALTIME	LOCALTIMESTAMP	LOCK
LONG	LONGBLOB	LONGTEXT
LOOP	LOW_PRIORITY	MASTER_SERVER_ID
MATCH	MEDIUMBLOB	MEDIUMINT
MEDIUMTEXT	MIDDLEINT	MINUTE_MICROSECOND

```
MINUTE_SECOND           MOD                     NATURAL
NOT                     NO_WRITE_TO_BINLOG      NULL
NUMERIC                 ON                      OPTIMIZE
OPTION                  OPTIONALLY              OR
ORDER                   OUT                     OUTER
OUTFILE                 PRECISION               PRIMARY
PRIVILEGES              PROCEDURE               PURGE
READ                    REAL                    REFERENCES
REGEXP                  RENAME                  REPEAT
REPLACE                 REQUIRE                 RESTRICT
RETURN                  REVOKE                  RIGHT
RLIKE                   SECOND_MICROSECOND      SELECT
SENSITIVE               SEPARATOR               SET
SHOW                    SMALLINT                SOME
SONAME                  SPATIAL                 SPECIFIC
SQL                     SQLEXCEPTION            SQLSTATE
SQLWARNING              SQL_BIG_RESULT          SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT        SQL_TSI_DAY             SQL_TSI_FRAC_SECOND
SQL_TSI_HOUR            SQL_TSI_MINUTE          SQL_TSI_MONTH
SQL_TSI_QUARTER         SQL_TSI_SECOND          SQL_TSI_WEEK
SQL_TSI_YEAR            SSL                     STARTING
STRAIGHT_JOIN           STRIPED                 TABLE
TABLES                  TERMINATED              THEN
TIMESTAMPADD            TIMESTAMPDIFF           TINYBLOB
TINYINT                 TINYTEXT                TO
TRAILING                TRUE                    UNDO
UNION                   UNIQUE                  UNLOCK
UNSIGNED                UPDATE                  USAGE
USE                     USER_RESOURCES          USING
UTC_DATE                UTC_TIME                UTC_TIMESTAMP
VALUES                  VARBINARY               VARCHAR
VARCHARACTER            VARYING                 WHEN
WHERE                   WHILE                   WITH
WRITE                   XOR                     YEAR_MONTH
```