<https://github.com/kasper/tikape-kesa-2014/blob/master/viikko-2/harjoitus-malliratkaisut.md>

<https://github.com/kasper/tikape-kesa-2014/blob/master/viikko-3/harjoitus-malliratkaisut.md>

<https://github.com/kasper/tikape-kesa-2014/blob/master/viikko-4/harjoitus-malliratkaisut.md>

# Harjoitus 2

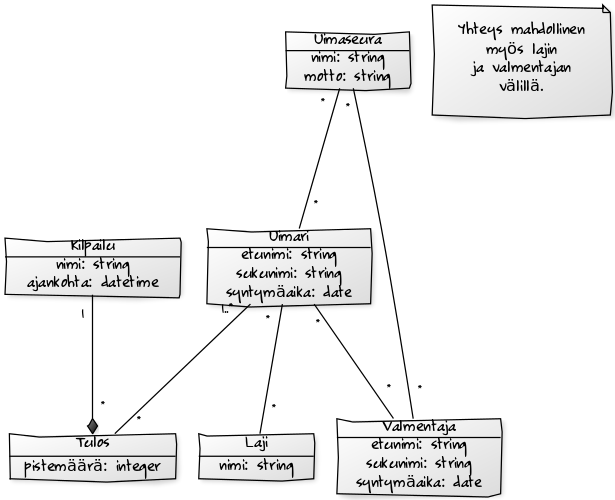
## Relaatiomalli

1. **Tietokohteisiin liittyy attribuutteja, arvoja ja arvojoukkoja. Mitä nämä tarkoittavat? Anna esimerkki.**

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

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

1. **Kartoita** [**viime viikon**](https://github.com/kasper/tikape-kesa-2014/blob/master/viikko-1/harjoitus.md) **tehtävästä 5 tietokohteet, niiden väliset yhteydet ja osallistumisrajoitteet. Mieti tietokohteille attribuutit ja näille sopivia arvoja. Piirrä UML-luokkakaavio tietokohteista. Sisällytä luokkien attribuutit luokkakaavioon.**

[](https://camo.githubusercontent.com/26170ceaaa704a7a22ca2cb0078b03eb3b5ad5eb/687474703a2f2f79756d6c2e6d652f3363376636353031)

1. **Selvitä mitä skeema, taulu, sarake ja rivi tarkoittavat tietokannassa.**

**Skeema:** Relaatiotietokannan taulujen, eheysehtojen ja käyttöoikeuksien määrittely, joka on esitetty tietokannan määrittelykielellä.

**Taulu:** Samaan rivin määrittelyyn liittyvien rivien monijoukko. Taulu on relaation toteutus relaatiotietokannassa. Taulu esitetään usein käyttäjälle siten, että rivit ovat allekkain vaakasuunnassa, sarakkeet pystysuunnassa.

**Sarake:** Taulussa eri riveillä samaan arvoalueen määrittelyyn liittyvien arvojen lista.

**Rivi:** Arvojen lista, jossa esiintyy yksi tietyn arvoalueen arvo kutakin rivin määrittelyssä osoitettua arvoaluetta kohti näiden järjestyksessä.

1. **Mitä tarkoittaa pääavain Primary key (PK) ja viiteavain Foreign key (FK)? Entä yhdistelmäavain?**

**Pääavain:** Monikon (rivin) pääasialliseksi tunnisteeksi valittu avainehdokas.

**Viiteavain:** Attribuutti tai attribuuttiyhdistelmä (sarakenimien yhdistelmä), jonka avulla voidaan osoittaa jokin tietokannan yksittäinen monikko (rivi) käyttäen tämän perusavainta.

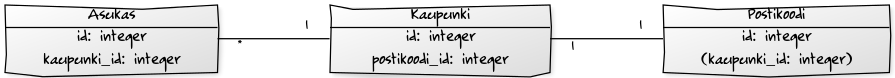
**Yhdistelmäavain:** Kahdesta tai useammasta sarakkeesta muodostuva avain.

1. **Miten yhdestä–yhteen (Kaupunki → Postikoodi) ja yhdestä–moneen (Kaupunki → Asukas) relaatio toteutetaan relaatiomallissa? Entä monesta–yhteen (Asukas → Kaupunki)? Anna esimerkki tietokantakaaviona.**

Yhdestä-yhteen -relaatiossa toinen puoli pitää viiteavainta toiseen (kumpi tahansa puoli käy).

Yhdestä-moneen -relaatiossa "moni"-puoli pitää viiteavaimen "yksi"-puoleen, muuten sama kuin yhdestä-yhteen.

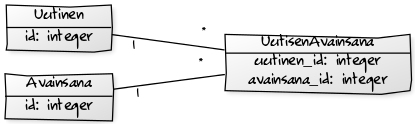
Monesta-yhteen -relaatio on sama kuin yhdestä-moneen.

[](https://camo.githubusercontent.com/9940c0a47882c9893a1317984b2f60587651ecf6/687474703a2f2f79756d6c2e6d652f3934383864623533)

HUOM! Sekä yhdestä-yhteen, että yhdestä-moneen -relaatiot on mahdollista toteuttaa välitaululla, mutta tämä ei pääasiassa ole tarpeellista.

1. **Miten monesta–moneen (Avainsana → Uutinen) relaatio toteutetaan relaatiomallissa? Anna esimerkki tietokantakaaviona.**

Monesta-moneen -relaatio vaatii lisätaulun, joka säilyttää viitteet kummankin osapuolen pääavaimiin.

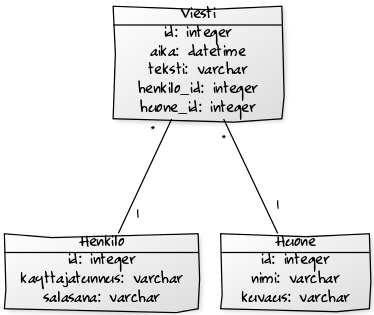
[](https://camo.githubusercontent.com/73a974070a5174348b79dec55844d4ce0a9aade2/687474703a2f2f79756d6c2e6d652f3138363638323763)

1. **Taulujen Opettaa ja Kurssi välille tehdään liitos jossa liitosehtona on sarakkeiden kurssikoodi yhtäsuuruus. Montako riviä on tulostaulussa?**
2. Opiskelija(\_opiskelijanumero\_, nimi, sahkoposti, paaaine) [2000 riviä]
3. Kurssi(\_kurssikoodi\_, nimi, opintopisteita) [200 riviä]
4. Opettaja(\_opettajatunnus\_, nimi, sahkoposti, opetusaine) [100 riviä]
5. Opettaa(\_kurssikoodi → Kurssi\_, \_opettajatunnus → Opettaja\_, \_lukuvuosi\_, \_lukukausi\_, \_tehtava\_, tunnit) [400 riviä]

400 riviä. Kun kyselyn FROM-osassa annetaan useita tauluja, on tuloksena näiden taulujen ristitulo. Liitosehto kuitenkin karsii riviyhdistelmiä. Jokaiseen Opettaa-taulun riviin liittyy nyt oikeat kurssitiedot, mutta mitään ei myöskään rajata pois. Lisäinfoa esim. <http://www.cs.helsinki.fi/u/laine/tkp/sql/kysely3.html>.

1. **Seuraavassa on kuvattuna chat-palvelu. Mitä allevivaukset ja nuolet tarkoittavat tekstipohjaisessa tietokantakuvauksessa? Piirrä chat-palvelu tietokantakaaviona.**
2. Henkilo(\_id\_, kayttajatunnus, salasana)
3. Huone(\_id\_, nimi, kuvaus)
4. Viesti(\_id\_, henkilo\_id → Henkilo, huone\_id → Huone, aika, teksti)

Alleviivatut attribuutit ovat pääavaimia ja nuolet osoittavat viiteavaimen kohteen (kohdetaulun pääavain).

[](https://camo.githubusercontent.com/d4a5b092c1f1a19815edcef9ac34154693a1a886/687474703a2f2f79756d6c2e6d652f3039363862373262)

Harjoitus 3

# Harjoitus 3

## SQL

1. **Mitä SQL-standardeja on olemassa ja tukevatko kaikki tietokannat näitä samalla tavalla? Mitä yleisiä tietotyyppejä SQL-tietokannoissa on?**

SQL on ANSI-standardisoitu vuonna 1986 ja ISO-standardisoitu vuonna 1987. Nykyään standardia ylläpitää ISO:on ja IEC:in liittyvä "Joint Technical Committee ISO/IEC JTC 1, Information technology, Subcommittee SC 32, Data management and interchange". SQL-standardia on tarkennettu seitsemän kertaa julkaisun jälkeen. Viimeisin standardi on nimeltään SQL:2011 tai ISO/IEC 9075:2011. SQL-standardi on jaoteltu yhdeksään osaan.

Kaikki tietokannat eivät tue koko standardia. Esimerkiksi lainauksena SQLiten sivuilta:

"SQLite understands most of the standard SQL language. But it does omit some features while at the same time adding a few features of its own."

Pääasiassa suurin osa ydinstandardista on toteutettuna kaikissa tietokannoissa. Suurimmat tietokannat toteuttavat standardin lähes täysin. Lisäksi lähes kaikki tietokannat lisäävät omia ei-standardisoituja ominaisuuksia, kuten apufunktioita. Esimerkkinä MySQL datediff() verrattuna esim. Oraclen date - date.

Yleisiä tietotyyppejä ovat mm. vaihtelevan pituinen merkkijono VARCHAR(n), totuusarvo BOOLEAN, kokonaisluku INTEGER, liukuluku FLOAT ja päivämäärä DATE.

1. **Käyttötapauksia** [**viime viikon**](https://github.com/kasper/tikape-kesa-2014/blob/master/viikko-2/harjoitus.md) **tehtävän 8 chat-palveluun liittyen. Kerro minkälaisella SQL-kyselyllä seuraavat käyttötapaukset luonnistuisivat.**
   * **Palvelun ylläpitäjä haluaa lisätä uuden chat-huoneen. Chat-huoneen nimi on viherpeukalot ja kuvaus on Vihreä kesä.**
   * INSERT INTO Huone (id, nimi, kuvaus) VALUES (1, "viherpeukalot", "Vihreä kesä");
   * **Palvelun ylläpitäjä haluaa lisätä uuden käyttäjän. Uuden käyttäjän käyttäjätunnus on aimo ja salasana porkkana.**
   * INSERT INTO Henkilo (id, kayttajatunnus, salasana) VALUES (1, "aimo", "porkkana");

Huom! Salasanan säilyttäminen tietokannassa salaamattomana on todella huono idea.

* + **Käyttäjä aimo kirjoittaa viestin Moi kaikki! huoneeseen virherpeukalot**
  + INSERT INTO Viesti (id, henkilo\_id, huone\_id, aika, teksti) VALUES (1, 1, 1, "2014-06-01T10:11:12", "Moi kaikki!");
  + **Tuntematon käyttäjä yrittää kirjautua käyttäen käyttäjätunnusta testi ja salasanaa salasana**

Kirjautumisyritys aiheuttaa kantaan kyselyn

SELECT \* FROM Henkilo WHERE kayttajatunnus = "testi" AND salasana = "salasana";

joka palauttaa 0 riviä (oletetaan, että kannassa vain aimo). Ohjelma tulkitsee tämän vääränä tunnus-salasana -yhdistelmänä.

1. **Luo** [**viime viikon**](https://github.com/kasper/tikape-kesa-2014/blob/master/viikko-2/harjoitus.md) **tehtävän 8 tietokantatauluille SQL CREATE-TABLE-lauseet. Mieti sopivat rajoitteet sarakkeille.**
2. CREATE TABLE Henkilo (id INTEGER PRIMARY KEY NOT NULL, kayttajatunnus VARCHAR(255) NOT NULL UNIQUE, salasana VARCHAR(255) NOT NULL);
3. CREATE TABLE Huone (id INTEGER PRIMARY KEY NOT NULL, nimi VARCHAR(255) NOT NULL, kuvaus VARCHAR(255));
4. CREATE TABLE Viesti (id INTEGER PRIMARY KEY NOT NULL, henkilo\_id INTEGER, huone\_id INTEGER, aika DATETIME, teksti VARCHAR(255), FOREIGN KEY (henkilo\_id) REFERENCES Henkilo(id), FOREIGN KEY (huone\_id) REFERENCES Huone(id));

Vaatisi lisäksi Huonetta poistettaessa Viestien poiston aiheuttavan lisäehdon.

1. **Tee SQLite-tietokanta, johon suoritat edelliset tehtävät. Lisää tietokantaan lisää käyttäjiä, kanavia ja viestejä. Lisää tietokantaan myös joku uusi vapaavalintainen taulu ja siihen tietoja. Pääset tehtävässä alkuun katsomalla lyhyen** [**SQLite-oppaan**](https://github.com/kasper/tikape-kesa-2014/blob/master/sqlite.md)**. Ota komennot ja tulokset talteen.**

Tämän voi tehdä esimerkkiratkaisujen mukaisesti omatoimisesti.

1. [**Tik**](http://tik.herokuapp.com) **on tehtävälistasovellus. Löydät sovelluksen SQLite-tietokannan** [**tästä**](https://github.com/kasper/tikape-kesa-2014/blob/master/tehtavat/tik.sqlite?raw=true)**. Suorita seuraavat käyttötapaukset ja ota ne talteen.**
2. sqlite> .schema
3. CREATE TABLE "schema\_migrations" ("version" varchar(255) NOT NULL);
4. CREATE TABLE "taggings" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "tag\_id" integer, "task\_id" integer, "created\_at" datetime, "updated\_at" datetime);
5. CREATE TABLE "tags" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "name" varchar(255), "created\_at" datetime, "updated\_at" datetime);
6. CREATE TABLE "tasks" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "description" varchar(255), "completed" boolean, "owner\_id" integer, "created\_at" datetime, "updated\_at" datetime, "creator\_id" integer);
7. CREATE TABLE "teams" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "name" varchar(255), "created\_at" datetime, "updated\_at" datetime);
8. CREATE TABLE "users" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "username" varchar(255), "created\_at" datetime, "updated\_at" datetime, "team\_id" integer, "password\_digest" varchar(255), "admin" boolean);
9. CREATE UNIQUE INDEX "unique\_schema\_migrations" ON "schema\_migrations" ("version");
   * **Hae kaikki käyttäjän kasper tehtävät**
   * sqlite> SELECT description, users.id AS user\_id, creator\_id FROM tasks, users WHERE tasks.creator\_id = users.id AND users.username = "kasper";
   * description user\_id creator\_id
   * ----------------- ---------- ----------
   * Create a new task 1 1
   * Code Tik 1 1
   * **Lisää käyttäjälle jack uusi tehtävä, jonka omistaja on kasper**
   * sqlite> SELECT id, username FROM users;
   * id username
   * ---------- ----------
   * 1 kasper
   * 2 jack
   * sqlite> INSERT INTO tasks (creator\_id, owner\_id, description) VALUES (2, 1, "uusi tehtävä");
   * sqlite> SELECT description, creator\_id, owner\_id, id FROM tasks;
   * description creator\_id owner\_id id
   * ----------------- ---------- ---------- ----------
   * Create a new task 1 1 1
   * Code Tik 1 1 2
   * Create another ta 2 2 3
   * uusi tehtävä 2 1 4
   * **Liitä tehtävään avainsanoiksi critical ja enhancement**
   * sqlite> SELECT name, id FROM tags;
   * name id
   * ---------- ----------
   * critical 1
   * enhancemen 2
   * feature 3
   * sqlite> SELECT description, creator\_id, owner\_id, id FROM tasks;
   * description creator\_id owner\_id id
   * ----------------- ---------- ---------- ----------
   * Create a new task 1 1 1
   * Code Tik 1 1 2
   * Create another ta 2 2 3
   * uusi tehtävä 2 1 4
   * sqlite> INSERT INTO taggings (tag\_id, task\_id) VALUES (1, 4);
   * sqlite> INSERT INTO taggings (tag\_id, task\_id) VALUES (2, 4);
   * sqlite> SELECT tags.name, description FROM tags, taggings, tasks WHERE tags.id = taggings.tag\_id AND tasks.id = taggings.task\_id AND tasks.id = 4;
   * name description
   * ---------- --------------
   * critical uusi tehtävä
   * enhancemen uusi tehtävä
   * **Lisää uusi User Experience-tiimi ja liitä sekä kasper että jack sen jäseniksi**
   * sqlite> INSERT INTO teams (name) VALUES ("User Experience");
   * sqlite> SELECT name, id FROM teams;
   * name id
   * ---------- ----------
   * Software 1
   * User Exper 2
   * sqlite> SELECT users.id as user\_id, teams.id as team\_id, username, name FROM users, teams WHERE teams.id = users.team\_id;
   * user\_id team\_id username name
   * ---------- ---------- ---------- ---------------
   * 1 1 kasper Software
   * 2 1 jack Software
   * sqlite> UPDATE users SET team\_id = 2 WHERE username = "jack" OR username = "kasper";
   * sqlite> SELECT users.id as user\_id, teams.id as team\_id, username, name FROM users, teams WHERE teams.id = users.team\_id;
   * user\_id team\_id username name
   * ---------- ---------- ---------- ---------------
   * 1 2 kasper User Experience
   * 2 2 jack User Experience
   * **Poista lopuksi Software-tiimi**
   * sqlite> SELECT name, id FROM teams;
   * name id
   * ---------- ----------
   * Software 1
   * User Exper 2
   * sqlite> DELETE FROM teams WHERE id = 1;
   * sqlite> SELECT name, id FROM teams;
   * name id
   * --------------- ----------
   * User Experience 2
10. **Mitä tarkoittaa tietokantatransaktio? Milloin transaktiota kannattaa käyttää ja mitä hyötyä siitä on?**

Tietokantatransaktio on tietokantaa käsittelevä prosessin osa, jonka vaikutusten halutaan muodostavan yhden jakamattoman (atomisen) kokonaisuuden.

Esimerkiksi tilisiirtotapahtuma, kolmiosainen kysely:

* + muokkaa siirtäjän tilitietoja
  + muokkaa saajan tilitietoja
  + lisää tilitapahtumiin tieto.

Esimerkissä jakamattomuus merkitsee sitä, että kaikki 3 tietokantaoperaatiota suoritetaan eikä vain osaa niistä.

Käsittelyssä voi sattua häiriöitä missä vaiheessa tahansa. Nämä voivat johtua ulkoisista syistä tai olla tietokannan itse aiheuttamia. Ilman transaktiota voisi siis käydä siten, että tililtä lähtee rahat, mutta saaja ei ikinä saa rahojaan – tällöin operaatio ei olisi jakamaton.

Transaktiota kannattaa käyttää kun joukko muutoksia pitää tehdä kokonaisuudessaan ollakseen validi tapahtuma. Toisin sanoen: jos on mahdollisuus, että tietokannan tila on väärä kun vain osa kyselystä toteutuu, käytä transaktiota. Transaktio mahdollistaa mm. sen perumisen kokonaisuudessaan sen epäonnistuessa. Kyselyt suoritetaan myös kokonaisuutena, joten muutoksia ei voi tapahtua transaktion "keskellä".

1. **Mitä tarkoittaa SQL-injektio ja miten siihen voi varautua? Kokeile suorittaa SQL-injektioita** [**Väestörekisteriin**](http://t-khirviko.users.cs.helsinki.fi/vaestorekisteri/)**. Arto Vihavaisen lompakko tippui Unicafen kassalle ja tiedämme hänen SOTU:n olevan 010190-111A.**

SQL-injektiossa ideana on saada ohjelma suorittamaan SQL-kyselyitä, joita ohjelmaa ei ole tarkoitettu tekemään. Esimerkiksi ohjelmakoodissa kysely voitaisiin muodostaa näin:

statement = "SELECT \* FROM users WHERE name ='" + userName + "';"

ja käyttäjä voisi määritellä userName:ksi

' or '1'='1

syntyisi kysely

SELECT \* FROM users WHERE name = '' OR '1'='1';

Tällöin jos kyselyä käytettäisiin esimerkiksi autentikointiin tai tietojen listaukseen, saisi SQL-injektion tekijä oikeudet tai listauksen kaikista käyttäjistä.

SQL-injektioita voidaan estää esimerkiksi parametrisoimalla kyselyt. Tällöin kyselyt ovat muotoa

INSERT INTO PRODUCT (name, price) VALUES (?, ?)

ja puuttuvat arvot (?) annetaan erikseen parametreina kyselylle. Kyselyn parametrit automaattisesti suodatetaan ("escaping" tai vastaava).

Lisää tietoa esim. <http://en.wikipedia.org/wiki/Sql_injection> tai <http://www.unixwiz.net/techtips/sql-injection.html>

Väestörekisterikyselyssä tarvitsee tehdä oletuksia kyselyn muodosta. Voimme esimerkiksi olettaa, että kysely olisi muotoa

SELECT \* FROM taulu WHERE sotu = ?

tai koska Väestörekisteri on huonosti suunniteltu voi ? olla muuttujana koodissa ja saamme määritellä sen itse. Eli oletetaan koodiksi jotain tällaista:

String kysely = "SELECT \* FROM taulu WHERE sotu = '" + sotu + "'";

Nyt haluamme taulun koko sisällön eli laitamme lomakkeeseen kyselyksi sopivan pätkän, joka antaa hyväksyvän totuusarvon jokaisen rivin kohdalla taulussa. Eli

' OR 'x' = 'x

Huomioi '-merkkien kohdat. Nyt kysely on muotoa

SELECT \* FROM taulu WHERE sotu = '' OR 'x' = 'x'

eli "kaikki henkilöt, joiden sotu on tyhjä TAI kirjain x on kirjain x" ja testaamalla lomakkeeseen saadaan ulos suuri määrä henkilötietoja.

Harjoitus 4

# Harjoitus 4

## Optimointi

1. **Selvitä mitä tarkoittavat funktionaalinen riippuvuus, Boyce-Codd analyysi ja -normaalimuoto. Onko alla oleva taulu normaalimuodossa? Mitä funktionaalisia riippuvuuksia siitä löytyy? Mieti ehdotelma miten ongelman voisi ratkaista.**

| **id** | **nimi** | **syntymaaika** | **osasto\_id** | **osasto\_nimi** | **osasto\_sijainti** |
| --- | --- | --- | --- | --- | --- |
| 1 | Arvelo | 1.3.80 | 3 | myynti | Helsinki |
| 2 | Ahtisaari | 4.5.81 | 3 | myynti | Helsinki |
| 3 | Ahtovuo | 8.6.87 | 4 | hallinto | Espoo |
| 4 | Äimä | 2.4.95 | 4 | hallinto | Espoo |
| 5 | Alaja | 1.2.85 | 6 | tuotanto | Espoo |

1. Funktionaalisella riippuvuudella tarkoitetaan sitä, että sarakkeen arvon avulla pystytään selvittämään yksikäsitteinen arvo sarakkeesta funtionaalisesti riippuvalle sarakkeelle. Esimerkiksi henkilötunnuksen perusteella saadaan selville yksikäsitteinen nimi, syntymäaika, osoite, yms.
2. **Boyce-Codd -normaalimuoto määritellään seuraavasti:**
3. Relaatiokaavio on Boyce-Codd -normaalimuodossa, jos ja vain jos relaatiokaavioon ei liity yhtään sellaista funktionaalista riippuvuutta, jossa määrääjä ei sisältäisi relaation avainta.
4. Boyce-Codd normaalimuodossa oleva tietokantakaavio voidaan määrätä minimaalisen riippuvuusjoukon perusteella seuraavasti:
   1. Ryhmittele riippuvuudet ryhmiin yhteisen määrääjän perusteella
   2. Muodosta kutakin ryhmää kohden relaatiokaavio, johon otat kaikki ryhmän riippuvuuksissa esiintyvät attribuutit
   3. Jos relaatiokaavion avain ei sisälly muodostettuihin relaatioihin, muodosta siihen kuuluvista attribuuteista oma relaatio
   4. Karsi mahdolliset saman asian toisteiset esitykset
   5. Anna muodostuneille relaatiokaavioille nimet. Jos niille löytyy nimet ja luonnollinen merkitys jakoa voi pitää onnistuneena.

Kun tietokannan relaatiot ovat kaikki Boyce-Codd normaalimuodossa, ei tietokannassa ole lainkaan toisteista tietoa.

Tehtävän tauluun liittyen voidaan löytää riippuvuudet:

id → nimi

id → syntymaaika

id → osasto\_id

osasto\_id → osasto\_nimi

osasto\_id → osasto\_sijainti

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

id → nimi

id → syntymaaika

id → osasto\_id

ja

osasto\_id → osasto\_nimi

osasto\_id → osasto\_sijainti

Näiden perusteella määräytyvät kaaviot:

kaavio\_1(id, nimi, syntymaaika, osasto\_id) ja

kaavio\_2(osasto\_id, osasto\_nimi, osasto\_sijainti).

Kun kaavioille annetaan uudet nimet, päädytään tauluihin:

Työntekijä(\_id\_, nimi, syntymaaika, osasto\_id → Osasto) ja

Osasto(\_id\_, osasto\_nimi, osasto\_sijainti).

1. **Relaatiotietokantojen suunnittelussa lähtöideana on tiedon toiston välttäminen. Tähän on muutamia syitä: toistuva tieto vie "turhaan" tilaa ja toistuvan tiedon ylläpitäminen on vaikeaa sillä jokaista kopiota on ylläpidettävä. Seuraavassa on esitetty Pankkitapahtuma-niminen relaatio. Jaa relaatio useampaan osaan siten, että suurin osa toisteisesta tiedosta vältetään.**
2. Pankkitapahtuma(\_id\_, pankkitilin\_numero, asiakkaan\_etunimi, asiakkaan\_sukunimi,
3. asiakkaan\_henkilötunnus, otto\_vai\_pano, summa, tapahtumapäivä,
4. pankkivirkailija\_etunimi, pankkivirkailija\_sukunimi, tapahtuman\_lisakuvaus)

Löydetään riippuvuudet:

|  |  |
| --- | --- |
| id → otto\_vai\_pano | Pankkitapahtumalla on tyyppi |
| id → summa | Pankkitapahtumaan liittyy tietty määrä rahaa |
| id → tapahtumapäivä | Pankkitapahtuma tapahtuu tiettyyn aikaan |
| id → tapahtuman\_lisakuvaus | Pankkitapahtumaan voi liittyä lisätietoja |
| id → asiakas\_id | Pankkitapahtuma liittyy tietyn asiakkaan tietoihin ja tiliin. Tämä sarake on uusi ja voisi olla tässä tapauksessa myös asiakkaan\_henkilötunnus. |
| asiakas\_id → asiakkaan\_henkilötunnus | Asiakkaalla on henkilötunnus |
| asiakas\_id → pankkitilin\_numero | Asiakkaalla on pankkitili, tässä esimerkkiratkaisussa sallitaan asiakkaalle vain yksi tili, jos tilejä olisi enemmän per asiakas, pitäisi ne erottaa vielä omaan tauluunsa |
| asiakas\_id → asiakkaan\_etunimi | Asiakkaalla on etunimi |
| asiakas\_id → asiakkaan\_sukunimi | Asiakkaalla on sukunimi |
| id → pankkivirkailija\_id | Pankkitapahtuman käsittelee virkailija. Tämä sarake on uusi ja tässä tapauksessa lähes pakollinen lisäys, tunnistavaksi tekijäksi tuskin kannattaa laittaa etunimi-sukunimi -yhdistelmää. |
| pankkivirkailija\_id → pankkivirkailija\_etunimi | Pankkivirkailijalla on etunimi |
| pankkivirkailija\_id → pankkivirkailija\_sukunimi | Pankkivirkailijalla on sukunimi |

Saadaan kaaviot:

kaavio\_1(id, otto\_vai\_pano, summa, tapahtumapäivä, tapahtuman\_lisakuvaus, asiakas\_id, pankkivirkailija\_id)

kaavio\_2(asiakas\_id, asiakkaan\_henkilötunnus, pankkitilin\_numero, asiakkaan\_etunimi, asiakkaan\_sukunimi)

kaavio\_3(pankkivirkailija\_id, pankkivirkailija\_etunimi, pankkivirkailija\_sukunimi)

ja näistä nimeällä taulut:

Pankkitapahtuma(\_id\_, otto\_vai\_pano, summa, tapahtumapäivä, tapahtuman\_lisakuvaus, asiakas\_id → Asiakas, pankkivirkailija\_id → Pankkivirkailija)

Asiakas(\_id\_, asiakkaan\_henkilötunnus, pankkitilin\_numero, asiakkaan\_etunimi, asiakkaan\_sukunimi)

Pankkivirkailija(\_id\_, pankkivirkailija\_etunimi, pankkivirkailija\_sukunimi)

1. **Selvitä mitä tarkoittaa ORM (Object-relational mapping). Mitä ORM-toteutukset (esim. Rails ActiveRecord tai Spring ORM) yleensä tarjoavat ja mikä niiden hyöty on?**

ORM muuntaa dataa tietokantojen ja olio-ohjelmointikielten välillä. ORM toimii siis rajapintana dataan ja sen avulla pystytään käsittelemään tietoa suoraan ohjelmointikielen keinoin ilman kyselyjen kirjoittamista.

Esimerkiksi: sen sijaan, että kirjoitettaisiin

String kysely = "SELECT ... FROM kayttajat WHERE id = 10";

TkKomento komento = new TkKomento(kysely);

Tulos tulos = komento.execute();

String nimi = res.get(0)["NIMI"];

voidaan kirjoittaa

Kayttaja k = kayttajaRepository.find(10);

String nimi = k.getNimi();

Jokainen kirjasto tarjoaa oman toteutuksen tähän, joten syntaksieroja on paljon.

ORM-toteutukset tarjoavat mm.

\* Tietokantamallien muuntamisen olio-ohjelmointiparadigmaan (sarakkeista olioiden kentiksi jne.)

\* Olioiden validoinnin ennen tallennusta

\* Välimuistit

\* Helpot rajapinnat tiedon hakemiseen, tallentamiseen, päivittämiseen ja poistamiseen

\* Skeeman ylläpito (migraatiot, esim. tietokantataulujen sarakkeiden muutokset)

\* Yhteensopivuuden eri tietokantajärjestelmien kanssa (SQLite, PostgreSQL, Oracle, MongoDB)

1. **Relaatiotietokannat ovat vain eräs tietokantatyyppi. 2000-luvulla nousi suosioon NoSQL-käsite. Mitä NoSQL:llä tarkoitetaan ja mitä NoSQL-tietokantoja on olemassa?**

NoSQL-tietokannat on tehty säilyttämään ei-relationaalista dataa. Tällaista dataa on esim. avain-arvo -parit, graafit ja dokumentit.

NoSQL-tietokantoja ovat mm.

\* Hadoop

\* MongoDB

\* Redis

\* Neo4j

NoSQL-tietokannat tarjoavat optimoituja malleja tiedon tallentamiseen tietyissä käyttötarkoituksissa (mm. suuret määrät dataa, tehokas rinnakkaisuus ja spesifit tietorakenteet kuten verkot).

1. **NoSQL-tietokannat ovat menettäneet hieman nostettaan. Mistä tämä voisi johtua?**

Suurin osa datasta sisältää relaatioita. NoSQL-tietokantoja sopivat huonosti yhteyksiä sisältävän datan käsittelyyn (monta kyselyä vs. relaatiotietokannan ~yksi) ja relaatiot on hallittava itse. Käyttötarkoitus ja tietokantatyyppi on siis sovitettava yhteen — jokaisella on hyvät ja huonot puolet.

1. **MongoDB on yksi dokumenttipohjainen NoSQL-tietokanta. Mitkä ovat sen edut? Tutustu MongoDB:n käyttörajapintaan osoitteessa** [**http://try.mongodb.org**](http://try.mongodb.org)**.**
   1. Kyselykieli on hyvin tarkka, verrattavissa SQL:ään yhden taulun tasolla (<http://docs.mongodb.org/manual/core/read-operations-introduction/>)
   2. Skeematon (ei välttämättä etu)
   3. Rakennettu skaalautuvaksi (esim. monelle palvelimelle)
   4. MongoDB:n dokumentit ovat hyvin verrattavissa esim. JSON:iin ja Rubyn Hasheihin ja mm. näissä kielissä dokumenttien käsittely on helppoa