**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.**

Funktionaalinen riippuvuus: sarakkeen arvolla saadaan selville yksikäsitteinen arvo funktionaalisesti riippuvalle sarakkeelle.

**Boyce-Codd**

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

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

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

Postinumero->Kaupunki ja Kaupunki, Katuosoite -> Postinumero

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

**Normaalimuodossa**:

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.

**Riippuvuudet**:

| **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 |

id -> kolme arvoa (nimi, syntymäaika, osastoid)

osastoid -> kaksi arvoa (osnimi ja ossijainti)

Taulun avain on id -> kaikki rivissä tämän sarakkeen puolesta. Esiintyy toistoa, jako kahteen tauluun: Työntekijä (nimi, syntymäaika ja osid) ja osasto (osid, osnimi ja ossijainti).

2. 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.

Pankkitapahtuma(\_id\_, pankkitilin\_numero, asiakkaan\_etunimi, asiakkaan\_sukunimi,

asiakkaan\_henkilötunnus, otto\_vai\_pano, summa, tapahtumapäivä,

pankkivirkailija\_etunimi, pankkivirkailija\_sukunimi, tapahtuman\_lisakuvaus)

Yksi taulu (kuten edellinen -> jako osiin).

pankkitapahtuma -> id otto/pano, summa, tapahtumapäivä, lisäkuvaus

asiakas -> id, pankkitilin numero, etunimi, sukunimi, henkilötunnus,

pankkivirkailija -> id, pvetunimi, pvsukunimi

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)

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

<http://en.wikipedia.org/wiki/Object-relational_mapping>

**Object-relational mapping** (**ORM**, **O/RM**, and **O/R mapping**) in computer science is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to create their own ORM tools.

In object-oriented programming, data management tasks act on object-oriented (OO) objects that are almost always non-scalar values. For example, consider an address book entry that represents a single person along with zero or more phone numbers and zero or more addresses. This could be modeled in an object-oriented implementation by a "Person object" with attributes/fields to hold each data item that the entry comprises: the person's name, a list of phone numbers, and a list of addresses. The list of phone numbers would itself contain "PhoneNumber objects" and so on. The address book entry is treated as a single object by the programming language (it can be referenced by a single variable containing a pointer to the object, for instance). Various methods can be associated with the object, such as a method to return the preferred phone number, the home address, and so on.

However, many popular database products such as structured query language database management systems (SQL DBMS) can only store and manipulate scalar values such as integers and strings organized within tables. The programmer must either convert the object values into groups of simpler values for storage in the database (and convert them back upon retrieval), or only use simple scalar values within the program. Object-relational mapping is used to implement the first approach.

The heart of the problem is translating the logical representation of the objects into an atomized form that is capable of being stored in the database, while preserving the properties of the objects and their relationships so that they can be reloaded as objects when needed. If this storage and retrieval functionality is implemented, the objects are said to be persistent.

ORM ohjelmointitekniikka muuntamaan dataa olio-ohjelmointikielten ja tietokantojen välillä.

ORM ohjelmoidaan niin että mäppää tietokannoista oliokerroksen ohjelmointikielen päälle.

* yksinkertaisuus (välttää sql-lausekkeet), yhteensopivuus (ei tarvitse ohjelmointikielen spessuja juttuja)
* skeeman hallinta
* Tietokantojen muuttaminen, tallentaminen, päivittäminen, hakeminen helpompi

Esim:

The following is a simple example. The below is [C#](http://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29) code to deliver a query, itself written in [SQL](http://en.wikipedia.org/wiki/SQL), to a database engine.

String sql = "SELECT ... FROM persons WHERE id = 10";

DbCommand cmd = new DbCommand(connection, sql);

Result res = cmd.Execute();

String name = res[0]["FIRST\_NAME"];

In contrast, the following makes use of an ORM API, allowing the writing of code which naturally makes use of the features of the language.

Person p = repository.GetPerson(10);

String name = p.getFirstName();

Compared to traditional techniques of exchange between an object-oriented language and a relational database, ORM often reduces the amount of code that needs to be written.[2]

Disadvantages of ORM tools generally stem from the high level of abstraction obscuring what is actually happening in the implementation code. Also, heavy reliance on ORM software has been cited as a major factor in producing poorly designed databases.[3]

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

NoSQL databases

Another approach is to use an object-oriented database management system (OODBMS) or document-oriented databases such as native XML databases that provide more flexibility in data modeling. OODBMSs are databases designed specifically for working with object-oriented values. Using an OODBMS eliminates the need for converting data to and from its SQL form, as the data is stored in its original object representation and relationships are directly represented, rather than requiring join tables/operations.

Document oriented databases also prevent the user from having to "shred" objects into table rows. Many of these systems also support the XQuery query language to retrieve datasets.

Object-oriented databases tend to be used in complex, niche applications. One of the arguments against using an OODBMS is that switching from an SQL DBMS to a purely object-oriented DBMS means that you may lose the capability to create application independent queries for retrieving ad hoc combinations of data without restriction to access path.[citation needed] For this reason, many programmers find themselves more at home with an object-SQL mapping system, even though most object-oriented databases are able to process SQL queries to a limited extent. Other OODBMS (such as RavenDB[4]) provide replication to SQL databases, as a means of addressing the need for ad hoc queries, while preserving well-known query patterns.

-> dokumenttipohjainen, skeemaltaan muunneltavissa? Not only sql.

skaalautuvuus = mahdollisuus jakaa dataa eri ”koneisiin” / tietokantoihin. Relaatiotietokannoissa eri tietokantojen pitäisi pystyä keskustelemaan keskenään. NoSQL tietokannoissa ei yleensä relaatioita eli jakaminen eri tietokantoihin mahdollista. Relaatiotietokantojen skaalautuvat myös mutta vaikeahko ja monimutkainen prosessi.

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

Paljon datasta sisältää relaatioita -> relaatiotietokannalla vähemmällä vaivalla kysely ja hallinta(?). Riippuu mikä käyttötarkoitus. MongoDB vaikuttaa toimivan. SQL-standardoitu, vanhempi -> enemmän tukea.

6. MongoDB on yksi dokumenttipohjainen NoSQL-tietokanta. Mitkä ovat sen edut verrattuna relaatiotietokantoihin?

<https://www.mongodb.org/>

Agility, scalability, performance -> muuntautuvuus/ketteryys, skaalautuvuus ja suorituskyky – omilta sivuilta

Saa tallennettua yhteen Book-nimiseen dokumenttiin datan (relaatiotietokanta pilkkoisi tauluihin). Eli helpompi kopioida näitä (isompia) kokonaisuuksia.