## 5.5 Tietokannan käyttö ohjelmasta

SQL-pohjaiset relaatiotietokannat tarjoavat käyttäjille yleensä jonkinlaisen suorakäyttöliittymän, jonka kautta käyttäjä voi antaa SQL–operaatioita. Suorakäyttö soveltuu hyvin

- satunnaisiin kyselyihin,
- erilaisiin ennakoimattomiin nopeaa ratkaisua vaativiin tilanteisiin,
- yksinkertaisiin ylläpito tehtäviin,
- kannanhallintatehtäviin ja
- yksinkertaiseen raportointiin.

SQL on kielenä varsin suppea ja sen toiminnallisuus keskittyy tietokantaoperaatioihin. Se ei sisällä raporttien muotoiluun tarvittavia käskyjä. Se ei sisällä tavanomaisten ohjelmointikielten kontrollirakenteita. Vaikka sen eräänä tavoitteena on ollut helppokäyttöisyys, se on kuitenkin suhteellisen vaikea käyttää muihin kuin aivan yksinkertaisiin kyselyihin. Lisäksi se edellyttää käyttäjältä perehtymistä tietokannan teknisiin rakenteisiin kuten taulu- ja sarakenimiin. Tästä syystä sitä ei voi pitää loppukäyttäjän vaan pikemminkin atk-ammattilaisen työvälineenä.

Markkinoilla on tarjolla SQL:ää helppokäyttöisempiä nimenomaan loppukäyttäjille suunnattuja välineitä kyselyjen laadintaan. Useimmat näistä perustuvat Query-by-Example -malliin. Qyery-by-Example (QBE) kehitettiin myös IBM:n tutkimuslaboratoriossa 70-luvun puolivälissä [Zloof75]. Perusideana kielessä on laatia kysely käyttäen hyväksi graafisia taulupohjia. Näihin kirjataan maskien avulla tulostusvalinnat, valintaehdot ja taulujen väliset kytkennät. Tämä kyselymalli on pohjana mm. Paradox:issa, Delphissä, ja Microsoft Access:issa. Kuvassa 5.8 on pieni esimerkki QBE-tyylisestä kyselystä.. QBE on SQL:ää helppokäyttöisempi satunnaisiin yksinkertaisiin raportointitarpeisiin. Monimutkaiset yhteenvetokyselyt ovat silläkin hankalia eikä QBE-tyyppisten kielten ilmaisuvoima näissä usein edes yllä SQL:n tasolle.

| Kurssi | Koodi | Nimi | Opintoviikot | Luennoija |
|---|---|---|---|---|
| | | Java-ohjelmointi | | _kuka |

| Opettaja | Opetunnus | Nimi | Puhelin | Tyohuone |
|---|---|---|---|---|
| | _kuka | <Print> | | |

Kuva 5.8: QBE- kysely : Java-ohjelmointi kurssin luennoijan nimi.

Pääasiallisesti tietokantaa käytetään kuitenkin sovellusohjelmien kautta. Nämä tarjoavat käyttäjilleen

- paremman käyttöliittymän,
- paremmin muotoillut raportit,
- tarpeiden mukaan kehitetyt palvelut, jotka eivät edellytä käyttäjiltä tietokannan rakenteen yksityiskohtaista tuntemista.

Sovellusohjelmat laaditaan jollakin ohjelmointikielellä. Yleiskäyttöiset ohjelmointikielet kuten Ada, C, C++, Cobol, Pascal, jne. eivät kuitenkaan peruskokoonpanossaan tarjoa mitään välineistöä tietokannan käsittelyyn. Sovellusten toteutusvälinevaihtoehdoiksi muodostuvat tällöin joko erityisen tietokantaohjelmointikielen käyttö tai yleisohjelmointikielen täydentäminen tietokantakäytön mahdollistavilla lisämoduuleilla. Yleisohjelmointikielten laajentamiseen on tarjolla kaksi tekniikkaa

- sulautettu SQL ja
- tietokantaliittymäkirjaston käyttö (database application programming interface, database API).

### *5.5.1 Sulautettu SQL*

Sulautetussa SQL:ssä SQL-lauseita voidaan kirjoittaa ohjelmointikielen lauseiden lomaan, esimerkiksi C-kielen lauseiden joukkoon. Jotta SQL-lauseet kyettäisiin erottamaan varsinaisista ohjelmointikielen lauseista (isäntäkielen lauseista), ne

merkitään kuuluviksi SQL:ään. Standardin mukaan tämä tapahtuu siten, että SQL-osuutta edeltää aina EXEC SQL – alkumerkki. SQL-osuuden loppumerkki on kielikohtainen. Ohjelmointikielen normaali kääntäjä, esimerkiksi C-kääntäjä ei osaa käsitellä ohjelmaan upotettuja SQL-lauseita. Niiden käsittelyyn tarvitaan esikääntäjä. Esikääntäjä prosessoi ohjelman ja muokkaa sen sellaiseen muotoon, että se voidaan edelleen kääntää konekielelle ohjelmointikielen normaalikääntäjällä. Käytännössä tämä tarkoittaa sitä, että kieleen upotetut SQL-lauseet korvataan tietokantaliittymän toteuttavan ohjelmakirjaston funktioiden kutsuilla. Näissä kutsuissa kyselyt ja muut tietokantaoperaatiopyynnöt on muunnettu funktioiden parametreiksi. Liittymäkirjastojen funktiot ovat tietokannanhallintajärjestelmäkohtaisia samoin kuin esikääntäjätkin. Esimerkiksi Oraclen C-esikääntäjä ProC tuottaa tuloksenaan C-kielisen ohjelman, joka käyttää Oracle Call Level Interface –kirjastoa. Tapa, jolla SQL-lauseet upotetaan isäntäkieleen on kuitenkin standardoitu. Joten periaatteessa on mahdollista vaihtaa tietokannanhallintajärjestelmää kääntämällä ohjelma uudelleen uuden järjestelmän esikääntäjällä. Alla on esimerkki Pascal funktiosta, johon on upotettu SQL-lauseita.

```
function keskipalkka(dept:integer):real;
var
    n integer; psumma integer;
    #include SQLCA.INC  /* sql:n käyttämiä tietorakenteita*/
EXEC SQL BEGIN DECLARE SECTION
var  palkka: integer;
     os: integer;
EXEC SQL END DECLARE SECTION
begin
   EXEC SQL DECLARE pal CURSOR FOR
        SELECT salary  from employee where department= :os;
   n:=0; psumma:=0; os:= dept;
   EXEC SQL  open pal;
   EXEC SQL fetch pal into :palkka;
   while sqlcode = 0 do begin
      psumma := psumma + palkka;
      n := n + 1;
      EXEC SQL fetch pal into :palkka;
   end;
   EXEC SQL close pal;
   if n > 0 then
      keskipalkka := psumma/n
   else
      keskipalkka := 0;
end;
```

### *5.5.2 Tietokannan käyttö liittymäkirjaston avulla*

Tietokannanhallintajärjestelmän liittymäkirjasto muodostuu joukosta funktioita. Näitä käyttämällä ohjelmoija voi kohdistaa operaatioita tietokantaan. Liittymäkirjasto on sama kirjasto, jonka funktiden kutsuiksi esikääntäjä kääntää ohjelmaan upotetut SQL-lauseet. Kuten edellä jo todettiin jokaisella tietokannanhallintajärjestelmällä on oma liittymäkirjastonsa.

Järjestelmäkohtaisten kirjastojen lisäksi on olemassa myös korkeamman tason abstraktioon perustuvia liittymäkirjastoja. Tällaisia ovat esimerkiksi ODBC (Microsoft Open Database Connection) ja JDBC (for Java) –kirjastot. Nämä kirjastot tarjoavat tietokannanhallintajärjestelmästä riippumattoman ohjelmointirajapinnan tietokantapalvelujen käyttöön. Tietyn tietokannanhallintajärjestelmän käyttö ODBC tai JDBC rajapinnan kautta edellyttää kuitenkin tietonnanhallintajärjestelmäkohtaisen ajurin (driver) hankintaa. Esimerkiksi Oracle-kannan käyttäminen edellyttää Oracle-ajuria. Ajurin tehtävänä on muuntaa abstraktin kirjastorajapinnan kautta tapahtuvat kutsut tietokannanhallintajärjestelmän käyttämään muotoon.

Esimerkkinä liittymäkirjastoihin perustuvasta tietokantaohjelmoinnista tarkastellaan seuraavassa tietokantasovelluksen toteutusta JDBC-kirjaston avulla. JDBC on Java ohjelmointikieleen liittyvä tietokantaohjelmointirajapinta. Käytöltään se on samankaltainen ODBC-rajapinnan kanssa. Kun ODBC muodostuu joukosta C-tyyppisiä funktioita, muodostuu JDBC Java-luokista ja niiden tarjoamista palveluista. Luokkarakenteen ansiosta rajapinta vaikuttaa ODBC:tä selkeämmältä ja jäsentyneemmältä.

### *5.5.3 JDBC:n perusteet*

JDBC on Java-luokkakirjasto tietokannan käsittelyyn. Hyvä esitys JDBC:stä löytyy lähteestä [Ham97]. Kirjaston keskeiset perusluokat ovat:

- DriverManager,
- Connection,
- Statement ,
- PreparedStatement ja
- ResultSet.

#### DriverManager

Pystyäkseen käyttämään hyväksi tietyn tietokannanhallintajärjestelmän palveluita on ohjelman käytettävä järjestelmäkohtaista ajuria. Ajuri, jos sellainen on olemassa, on saatavissa tietokannanhallintajärjestelmän toimittajalta. Ilman ajuria tietokantaa ei pysty käyttämään. Ajuri otetaan käyttöön DriverManager–luokan avulla Luokkapalvelulla registerDriver ohjelma kytketään käyttämään tiettyä tietokanta-ajuria. Ajurin käyttö tapahtuu JDBC-kirjaston sisäisesti eikä ohjelmoijan tarvitse kytkemisen jälkeen olla sen kanssa missään tekemisissä. Ajurin osaavat yleensä kytkeytä itse kun ne ladataan, joten ohjelmassa riittää ladata ajuri. Myös ODBC:ssä täytyy kiinnittää käytettävä ajuri, mutta tämä tehdään yleensä ohjelman ulkopuolisen hallintatyökalun avulla.

**Esimerkki:**

Otetaan käyttöön Oracle-ajuri

```
Class.forName("oracle.jdbc.driver.OracleDriver”);
```

#### Connection

DriverManager tarjoaa myös palvelun yhteyden luomiseksi tietokantaan. Tietokantayhteydellä (database connection) hallitaan ohjelman ja tietokannan välistä vuorovaikutusta. Yhteyden olemassaolo on edellytys kaikkien tietokantaan

kohdistuvien operaatioiden suoritukselle. Tietokannan suorakäytössä yhteys vastaa tietokantaistuntoa, joka alkaa käyttäjän sisäänkirjoittautumisesta ja päättyy istunnon lopetukseen. Samoin tietokantayhteys alkaa yhteyden perustamisella. JDBC:ssä yhteyden perustaminen tarkoittaa Connection-luokan olion luomista. Yhteys tulisi lopettaa kun sitä ei enää tarvita. Yhteyden lopettaminen vapauttaa siihen kytkettyjä resursseja sekä ohjelman itsensä että tietokantapalvelimen puolelta. JDBC:ssä yhteys suljetaan Connection-olion close-palvelulla. Yhteyden perustaminen vie jonkin verran aikaa (1-2 sekuntia) ja haluttaessa nopeaa palvelua voidaan joskus jättää yhteyksiä 'roikkumaan', jotta niitä voitaisiin nopeasti ottaa uudelleen käyttöön. Ohjelmalla voi olla käytössään useita tietokantayhteyksiä jopa eri tietokantoihin. Samaan tietokantaan tarvitaan yleensä vain yksi yhteys.

**Esimerkki:**

Yhteyden perustaminen Oracle-kantaan.

```
Connection con = DriverManager.getConnection(
 "jdbc:oracle:thin:@dbserver.helsinki.fi:1521:kanta1,
 "info", "expert");
```

Tässä luodaan Oraclen thin-ajuriin perustuva yhteys koneessa *dbserver.helsinki.fi* olevaan porttiin *1521* kytkettyyn tietokantaan, jonka tunnus on *kanta1*.[1] Yhteys perustuu käyttäjätunnukseen *info*, jonka salasana on *expert*.

**Statement**

Tietokantaan kohdistuvat operaatiot suoritetaan JDBC:ssä Statement-olioiden avulla. Statement-olio tarjoaa suoritusympäristön operaatioille. Se tarjoaa mm. palvelut

- *executeQuery* kyselyiden suoritukseen ja
- *executeUpdate* tulostaulua tuottamattomien operaatioiden suoritukseen

[1] dbserver.helsinki.fi nimistä tietokonetta ei ole oikeasti olemassa.

Tietokantaoperaatio suoritetaan jonkin tietokantayhteyden puitteissa. Tämä kytkentä on JDBC:ssä hoidettu siten, että Statement-olio luodaan käyttäen Connection-olion palvelua *createStatement*. Nimestään huolimatta Statement-olio ei sisällä suoritettavaa tietokantaoperaatiota vaan tämä annetaan operaation *executeQuery* tai *executeUpdate* parametrina. Täten jokaista operaatiota varten ei tarvitse välttämättä luoda omaa Statement-oliota. Yhden Statement-olion puitteissa voi kuitenkin olla vain yksi aktiivinen operaatio kerrallaan.

Operaatiota suoritettaessa havaitusta virhetilanteesta, esimerkiksi kyselyssä olevasta syntaksivirheestä, ilmoitetaan aiheuttamalla SQLException-tyyppinen poikkeus. Poikkeus tarjoaa metodit, joiden avulla saadaan selville tietokantaohjelmiston generoima virhekoodi ja virheilmoitus. Kaikista tietokannan käsittelyyn liittyvistä virhetilanteista ilmoitetaan samoin. Tietokantaoperaation yhteydessä pitää aina varautua SQLException poikkeukseen.

**ResultSet**

JDBC:ssä kyselyn vastausta käsitellään ResultSet olion palveluilla. Nämä mahdollistavat vastausjoukon rivien käsittelyn rivi kerrallaan. ResultSet tarjoaa myös palveluita metatiedon saamiseksi vastauksesta. Metatietoa ovat esimerkiksi vastauksen sarakkeiden lukumäärä, sarakkeiden nimet ja tietotyypit. ResultSet toteuttaa toimintoja, jotka sulautetussa SQL:ssä liittyvät käsitteeseen kursori (cursor).

Vastausjoukko (ResultSet-olio) syntyy tuloksena Statement-olion palvelun executeQuery suorituksesta, siis kyselyn suorittamisesta. Suoritettava kysely annetaan palvelun parametrina.

**Esimerkki**

```
Statement stmt= con.createStatement();
 ResultSet rs= stmt.executeQuery(
     "select nimi, puhelin from opettaja");
```

Tässä esimerkissä luodaan yhteyden *con* avulla kyselyn suoritusympäristöksi Statement-olio stmt. Sitä käyttäen suoritetaan kysely, joka tulos on käsiteltävissä Resultset olion rs palvelujen avulla.

Vastausta käsitellään rivi kerrallaan. Keskeinen palvelu on Boolen funktio *next()*. Se ottaa aktiiviseksi vastausriviksi vastausjoukossa seuraavana olevan rivin. Jos tällainen rivi on olemassa eli vastausjoukoa ei ole käsitelty loppuun *next()* palauttaa arvon ’*true*’, muuten se palauttaa arvon ’*false*’. Ensimmäisellä kutsukerralla se aktivoi vastausjoukon ensimmäisen rivin. Sulautetun SQL:n fetch-lause toimii samalla tavoin kursorin käsittelyssä. JDBC:n versiossa 1.0 ei ole mitään tapaa siirtyä vastausjoukossa taaksepäin, joten vastaus voidaan ohjelmassa käydä läpi vain kertaalleen alusta loppuun.. Versio 2.0 sisältää palveluita myös taaksepäin, sekä vastausjoukon alkuun ja loppuun siirtymiseksi.

Aktiivisen rivin käsittelyyn ResultSet-oliot tarjoavat tietotyyppikohtaisia get-funktioita sarakkeen arvon saamiseksi ohjelman käyttöön. Vastausriviin sisältyvän merkkijonon sisältö saadaan selville funktiolla getString, kokonaisluvun funktiolla getInt, päiväyksen funktiolla getDate, jne. Näistä funktioista on käytettävissä kaksi muotoa, joista toinen hakee arvon sarakenimen perusteella ja toinen vastaussarakkeen järjestysnumeron perusteella.

**Esimerkki:**

```
String nimi = rs.getString("nimi"); // sarakkeen nimi arvo
String puh = rs.getString(2);       // toisen sarakkeen arvo
```

get-funktiot osaavat jossakin määrin tehdä tietotyyppikonversioita. Esimerkiksi getString-funktiolla voi hakea kaiken tyyppisiä tietoja.

SQL-kyselyt voivat tuottaa tyhjäarvoja sisältäviä vastauksia. get-funktiot käsittelevät näitä eri tavoin. Olioarvoiset funktiot kuten getString palauttavat Javan null-osoittimen Tietotyyppiarvoiset funktiot sensijaan palauttavat, jonkin todellisen arvon. Esimerkiksi getInt palauttaa arvon 0. ResultSet-oliolla on palvelu *wasNull*, jolla voi tarvittaessa tutkia oliko palautettu arvo tyhjäarvo.

Kyselyn vastaus käsitellään vastaussilmukassa, jossa funktiolla next() haetaan vuoroon seuraava vastausrivi kunnes kaikki on käsitelty.

**Esimerkki:** Puhelinluettelo (ei kovin tyylikäs layout)

```
try {
      Statement stmt= con.createStatement();
      ResultSet rs= stmt.executeQuery(
            "select nimi, puhelin from henkilo"+
            " order by nimi");
      while (rs.next()) {
            System.out.println(rs.getString(1)+", "
                  Rs.getString(2));
      }
}
catch (SQLException ex) { do something};
```

Operaatiot, jotka eivät tuota tulosrivejä suoritetaan funktiolla executeUpdate;

**Esimerkki:**

```
stmt.executeUpdate(”update opettaja ”+
      ”set palkka= palkka +100000 ” +
      ”where opetunnus=\’HLAINE\’”);
```

korjaa hieman opettajan palkkaa.

**PreparedStatement**

Tietokannan käsittelyssä tarvitaan usein kyselyitä, joissa rakenteeltaan samanlainen kysely suoritetaan toistuvasti, mutta esimerkiksi hakuehdossa vaihtuu vakioarvo. Tällainen kysely voidaan parametroida niin, että tuo muuttuva vakio annetaan kyselyn parametrina. Parametroidut kyselyt valmistellaan käyttöä varten. Valmistelussa ne käännetään valmiiksi odottamaan suoritusta vaihtuvin parametriarvoin. JDBC tarjoaa parametroitujen kyselyjen käsittelyyn luokan PreparedStatement, joka on luokan Statement aliluokka. Parametroitu SQL-operaatio annetaan parametrina

PreparedStatement-olion luontifunktiolle Connection.prepareStatement. Operaatiossa parametri ilmaistaan kysymysmerkillä (?)

**Esimerkki:**

```
PreparedStatement pst = con.prepareStatement(
   "select nimi, puhelin "+
        "from opettaja"+
        "where nimi like ?" );
```

Tällainen kysely voisi sisältyä vaikkapa puhelinnumeroiden hakujärjestelmään, jossa käyttäjä voi antaa toistuvasti hakukriteerejä, joiden perusteella puhelinnumeroa haetaan. Parametrina tässä on nimeen liittyvä maski.

Ennen parametroidun kyselyn suoritusta on parametreille kiinnitettävä arvot. PreparedStatement tarjoaa tietotyyppikohtaiset set-funktiot parametriarvojen kiinnittämiseksi, esimerkiksi setInt, setString, setDate, jne. Set-funktioilla on kaksi parametria: parametrin järjestysnumero ja arvo joka parametrille annetaan. Esimerkiksi

```
pst.setString(1,"Möttö%");
```

kiinnittää arvon 'Möttö%' parameroidun kyselyn pst ensimmäiselle parametrille, joten kysely suoritettaessa vastaa kyselyä

```
select nimi, puhelin
    from opettaja
    where nimi like ‘Möttö%’
```

Parametroitu operaatio suoritetaan käyttäen palvelua executeQuery tai executeUpdate. Nyt vain näiden yhteydessä ei anneta mitään parametria.

**Esimerkkiohjelma:**

```
// Esimerkkiohjelma – Tietokantojen perusteet
// CLASSPATH polkumäärittelyssä täytyy olla polku hakemistoon
// josta tietokanta-ajuri löytyy
import java.sql.*;
import java.io.*;
public class Esimerkki  {
   public static void main(String args[]) throws Exception {
     String url =
      ”jdbc:oracle:thin:@dbserver.helsinki.fi:1521:kanta1";
     Connection con;
     String query =
        "select KOODI, KURSSI.NIMI AS KNI, OPINTOVIIKOT, "+
        "OPETTAJA.NIMI AS OPNI " +
        "from KURSSI, OPETTAJA " +
        "where KURSSI.LUENNOIJA= OPETTAJA.OPETUNNUS " +
        "order by OPETTAJA.NIMI, KURSSI.NIMI";
     Statement stmt;
     try {
       DriverManager.registerDriver((Driver)Class.forName
        ("oracle.jdbc.driver.OracleDriver").newInstance());
     }
     catch(java.lang.ClassNotFoundException e) {
        // Ilmoitetaan siitä, ettei yhteyttä saada aikaan.
          System.err.print("ClassNotFoundException: ");
          System.err.println(e.getMessage());
     }
     try {
       // Luodaan yhteys kurssin käyttäjätunnuksella
       con = DriverManager.getConnection(
               url, "info", "expert");
       // Luodaan tietokantaoperaation suoritusympäristö
       stmt = con.createStatement();
       // Muuttujaa ope käytetään pitämään kirjaa edellisestä
       // opettajasta. Tällä kontrolloidaan opettajan nimen
       // tulostusta.
       String ope =" ";
       // Tulostetaan otsake
       System.out.println("OPETUSTEHTÄVÄT:");
       System.out.println();
       // Suoritetaan kysely
       ResultSet rs = stmt.executeQuery(query);
       // Käydään läpi vastausrivit
       while (rs.next()) {
         // Eristetään opettajan nimi, huom. alias-nimen
         // käyttö
         String opn = rs.getString("OPNI");
         // Jos opettaja vaihtuu tulostetaan tyhjä rivi ja
         // opettajan nimi, muuten vain kurssin tiedot
         if (ope.equals(opn)) {
            System.out.println("     " +
            rs.getString("KNI")+" ("+rs.getInt("KOODI")+")");
         }
```

```
          else {
            System.out.println();
            ope= opn;
            System.out.println(ope);
            System.out.println("     " +
            rs.getString("KNI")+" ("+rs.getInt("KOODI")+")");
          }
        }  //end-while
        // Suljetaan 'kursori' - vapautetaan vastausjoukon
        // käsittelyyn kiinnitetyt resurssit. Tälle
        // vastausjoukolle ei enää voi tehdä mitään, vaikka
        // sitä ei suljettaisikaan.
        rs.close();
        stmt.close();
        // Suljetaan tietokantayhteys. Yhteys suljetaan vasta
        // siinä vaiheessa kun ohjelmassa ei enää käsitellä
        // kantaa.
        con.close();
      }
      catch(SQLException ex) {
         // Imoitetaan virheestä
         System.err.print("SQLException: ");
         System.err.println(ex.getMessage());
      }
   }
}
```

### *5.5.4 Tietokantaohjelmointikielet*

Tietokantaohjelmointikielissä tietokanta ja sen käsittelyyn tarvittavat rakenteet ovat peruskäsitteenä mukana kielessä. Nämä kielet tarjoavat normaalin ohjelmointikielen kontrollirakenteet ja lisäksi tietokantaliittymän sulautettua SQL:ää miellyttävämmin. Kielten pohjana on jokin yleisohjelmointikieli. Esimerkiksi tietokantaproseduurien laadintaan tarkoitetun standardikielen pohjana on Ada-ohjelmointikieli. Tietokantaohjelmointikieli esiintyy usein osana sovelluskehitintä. (application generator).

Esimerkkinä tietokantaohjelmointikielestä voidaan tarkastella Oraclen PL/SQL kieltä. Se pohjautuu Ada ohjelmointikieleen, mutta on toiminnallisuudeltaan hieman Adaa suppeampi. Kieltä voi käyttää Oraclen SQL-forms ja Developer kehitinten yhteydessä sekä tietokantaproseduurien laadinnassa. Tietokannan käsittelyyn PL/SQL tarjoaa

- sulautetun SQL:n käsitteet sisäänrakennettuina

- SQL:n tietotyyppit ohjelmointikielen tietotyyppeinä
- Oraclen SQL:n funktiot ohjelmointikielen funktioina
- Tietokannan kaavion hyväksikäytön ohjelman tietorakenteiden määrittelyssä
- helpot vastauksen käsittelysilmukat

```
for rivi in kysely loop ... end loop
```

- tietokantavirheet esimääriteltyinä poikkeuksina.

Perustietorakenteiltaan kieli on kuitenkin Adaa köyhempi. Kieli ei myöskään käy yleisohjelmointikieleksi, koska siitä puuttuvat kokonaan esimerkiksi näytön- ja tiedostojen käsittelyyn liittyvät operaatiot.

# 6 WWW-tietokantasovellukset

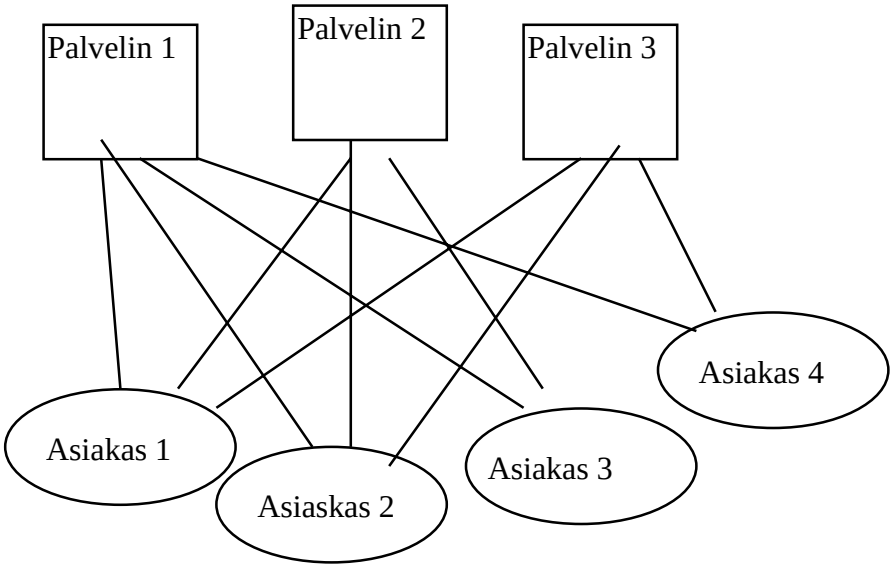
WWW (world wide web) on muutamassa vuodessa noussut ylivoimaisesti suosituimmaksi alustaksi erilaisten tiedotus- ja tiedonjakelujärjestelmien toteutuksessa. Myös vuorovaikutteisten WWW-alustalla toimivien sovellusten määrä on jo nykyään suuri ja voimakkaassa kasvussa.

Tyypillinen WWW-sovellus on laajalle asiakaskunnalle tarkoitettu tiedotus- tai tiedonjakeluärjestelmä. Tällaisessa järjestelmässä informaatiota välitetään hyvin erilaisille asiakkaille, jopa sellaisille, joista tiedon tarjoajalla ei ole mitään ennakkotietoa. Järjestelmien käyttäjämääristäkään ei välttämättä ole tarkkaa tietoa. Järjestelmille tyypillistä on tiedon usein toistuva hakeminen. Tieto itsessään saattaa olla aika muuttumatonta, joten päivitystarve voi olla vähäinen. Joissain tilanteissa esimerkiksi johdon informaatiojärjestelmien kohdalla käyttäjäkunta on selkeästi rajattu. Käyttäjien tiedontarve sen sijaan ei ole täsmällisesti määriteltävissä, vaan tietoa pitää olla saatavissa useassa muodossa. Tietotarpeet eivät tällöin useinkaan ole paikkaan tai aikaan sidottua, vaan tiedon pitää olla saatavilla kun sitä satutaan tarvitsemaan.

Vuorovaikutteisten WWW-sovellusten perustyyppi on suurelle yleisölle tarkoitettu tapahtumankäsittelyjärjestelmä, esimerkiksi jonkin pienehkön palvelun itsepalvelu-

periaatteella tarjoava järjestelmä. Tällainen palvelu voisi olla lipun varaaminen johonkin tilaisuuteen, ilmoittautuminen kurssille, tavaran tai palvelun tilaaminen, laskun maksaminen, yms. Tällaisten järjestelmien kasvupotentiaali on suuri.

WWW on hajautettu tietoverkossa (Internet) toimiva hypertekstijärjestelmä. Järjestelmiä käytetään asiakassovelluksina toimivien WWW-selainten (WWW-browser) avulla. Tietoa ja palveluja on haettavissa suuresta joukosta WWW-palvelimia (WWW-server) (kuva 6.1). Sisäisen verkon (Intranet) tapauksessa asiakas on sidottu tiettyihin palvelimiin. Ulkoisen verkon tapauksessa asiakkaalle tarjolla oleva palvelinvalikoima on rajoittamaton.




Kuva 6.1: WWW-asiakkaat ja palvelimet

WWW-selain on asiakasohjelma, joka toimii käyttäjän työasemassa. Selaimia on saatavissa usealta eri toimittajilta. Tunnetuimmat selaimet ovat:

- Netscape ja
- MS Internet Explorer. (IE)

Selaimilla esitetään käyttäjälle palvelimilta noudettuja, standardimuotoisia (HTML-kieli) dokumentteja (tekstiä, kuvia, yms.). Selain kykenee myös välittämään tietoja,

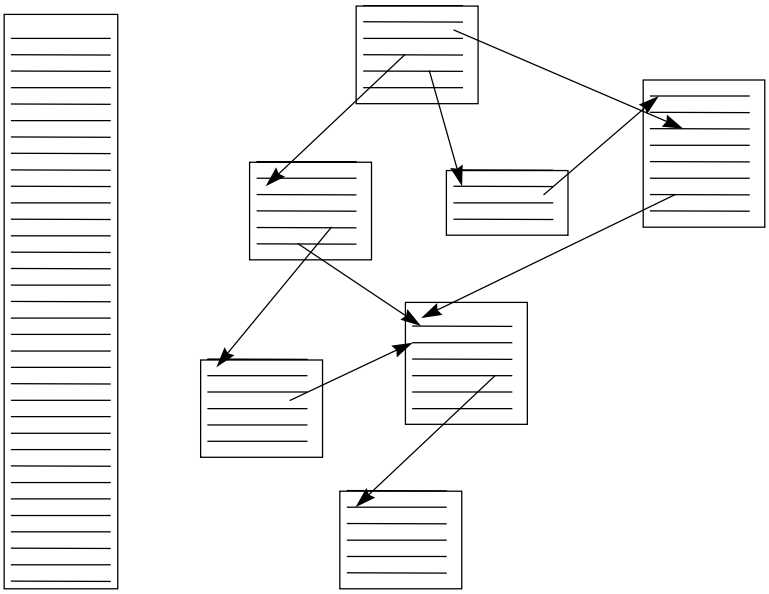
dokumenttien noutopyyntöjä sekä ohjelmien käynnistyspyyntöjä palvelimelle. Nykyiset selaimet kykenevät myös suorittamaan asiakaskoneessa palvelimelta haettuja ohjelmia.

Selaimen ja palvelimen välinen yhteistyö perustuu näytettävissä dokumenteissa oleviin linkkeihin ja lomakkeisiin. Käyttäjä osoittaa jotakin linkkiä, jonka seurauksena asiakasohjelma pyytä osoitetun olion palvelmelta. Jos osoitettu olio oli dokumentti, palvelin välittää sen asiakaskoneeseen selaimelle, joka näyttää dokumentin. Jos osoitettu olio oli palvelimeen asennettu ohjelma, palvelin suorittaa sen ja välittää tuloksen selaimelle näytettäväksi. Käyttäjä voi myös täyttää selaimen avulla lomakkeen (vaikkapa laatia kyselyn) ja lähettää sen palvelimelle. Palvelin käsittelee lomakkeen ja tuottaa vastausdokumentin, jonka selain esittää.

Palvelimet voivat sijaita missä tahansa (Internetissä). Palvelimen tehtävänä on tuottaa ja käsitellä tietomateriaalia. Käyttöliittymän hallinta on kokonaan asiakassovelluksen, siis selaimen vastuulla. Alunperin lähtökohtana WWW-tekniikassa on ollut kuvata HTML-kielellä dokumentin rakenne ja jättää kokonaan selaimen päätettäväksi käyttöliittymä ja dokumentin ulkoasu. Kehitys on kuitenkin johtamassa siihen, että dokumentit yhä enemmän sisältävät dokumentin ulkoasuun liittyvää informaatiota.

## 6.1 Hyperteksti

Perustana WWW:ssä on hypertekst (hypertext). Hyperteksti muodostuu vapaassa järjestyksessä luettavaksi tarkoitetuista elementeistä (solmuista). Elementit muodostuvat tekstistä ja kuvista. Elementtejä kytkevät toisiinsa hyperlinkit. Nämä ovat osoittimia elementteihin tai tiettyihin kohtiin elementeissä ja mahdollistavat selailevan liikkumisen (navigation) elementistä toiseen. Hyperteksti mahdollistaa assosiatiivisen asioiden yhdistelyn. Asiasta siirrytään suoraan tutkimaan jotain siihen liittyvää asiaa. Vastakohtana hypertekstille on normaali lineaarinen teksti, joka etenee alusta loppuun (kuva 6.2).



Kuva 6.2: Lineaarinen teksti vs. Hyperteksti.

Hypertekstin solmut voivat sisältää, mm:

- tekstiä
- taulukoita
- kuvia
- animaatiota
- videoita
- ääntä
- aktiivisia ohjelmakomponentteja

Solmut saattavat olla hyvinkin eri kokoisia, rivin kokoisesta tekstistä tai pienestä parin sadan tavun kuvasta useiden megatavujen kokoisiin dokumentteihin, kuviin tai videoihin.

Hyperlinkit yhdistävät solmuja. Ne voivat olla viittaavia osoittaen dokumenttia täydentävään materiaaliin. Korvaava linkki osoittaa elementin tilalle tulevaan elementtiin. Huomautus-tyyppiset linkit selventävät jotain dokumentin käsitettä. Ehdolliset linkit voivat tarjota vaihtoehtoja tai olla käytettävissä tiettyjen ehtojen voimassaollessa. Toimintalinkit mahdollistavat toimintojen käynnistyksen.

Hypertekstin käyttö on periaatteessa hyvin vapaata. Käyttäjällä on päätösvalta. Hän päättää, miten etenee hyperavaruudessa. Hyperteksti on oikeastaan yleistys valikkojärjestelmästä. Nyt valikot eivät ole erillisinä osina sovellusta, kuten monissa perinteisissä sovellusmalleissa, vaan valikot on upotettu tekstiin tai kuviin. Samoin kuin valikkojenkin kohdalla, eteneminen on helppoa. Se edetäänkö tavoitteeseen voikin olla eri asia. Aina ei tiedetä, mitä linkin takaa löytyy. Linkki saattaa viedä käyttäjän harhapolulle, jolta pitää palata takaisin. Voi olla, että siinä vaiheessa kun tarve paluuseen todetaan, paluu ei enää olekaan mahdollista. Käyttäjä on eksynyt (lost in hyperspace). Pelkästään linkkien avulla tapahtuvan samoilun (browsing) rinnalle tarvitaankin erilaisia aputoimintoja, esimerkiksi:

- haku sisällön perusteella
- paluu kuljettua polkua taaksepäin
- siirtyminen tunnettuun solmuun
- sisällysluettelot
- kartat.

Hypertekstille on useita luontevia sovelluskohteita:

- dokumentit,
- käsikirjat,
- sanakirjat,
- esittelymateriaalit,
- opetusmateriaalit,
- arkistomateriaali,
- ohjelmien opasteet,
- jne.

Hypertekstijärjestelmä koostuu kahdenlaisista osista

- käyttäjän selailuvälineestä, jota käytetään luettaessa hypertekstiä
  esim. Windows Help, Netscape, ...
- tekijäjärjestelmästä (hypertext editor), jolla tuotetaan hypertekstiin sisältyvä informaatio ja linkit.

esim. HyperCard, tekstinkäsittelyjärjestelmien laajennokset, HTML-editorit, Help-editorit

Tekijäjärjestelmillä saadaan yleensä aikaan staattista hypertekstiä. Tulevaisuudessa hypertekstit tulevat kuitenkin muuttumaan dynaamisemmiksi. Hypertekstiesitys informaatiolle generoidaan tarvittaessa perustiedot sisältävästä tietokannasta. Tällöin tarvitaan kehittimiä, joilla on helposti tuotettavissa hypertekstiä tuottavia ohjelmia.

Hypertekstin perusideasta on erilaisia toteutuksia. Useissa hypertekstijärjestelmissä on oma esitysmuotonsa hypertekstille. Esimerkkejä tällaisista ovat HyperCard ja Windowsin avustustiedosto (Help) -rakenteet. WWW:ssä käytetään standardoitua HTML-rakennetta (hyper text markup language) [W3C]. Rakenteen viimeisin standardiluonnos on versionumeroltaan 4.0 (ainakin tätä kirjoitettaessa). HTLM-standardista on muutamassa vuodessa (4) saatu aikaan useita standardiversioita. Kehitys on siis todella nopeaa verrattuna esimerkiksi SQL:ään, jossa versioiden väli on ollut luokkaa 4-5 vuotta, jopa enemmänkin.

## 6.2 HTML-kieli

HTML (HyperText Markup Language) perustuu ns. SGML-standardiin (Standard Generalized Markup Language). SGML on tarkoitettu dokumenttien rakenteen kuvaamiseen. SGML-standardi määrittelee puitteet erilaisten dokumentin rakenneosien merkitsemiselle. HTML-kielessä on kiinnitetty joukko dokumentin rakenneosia ja niiden merkitsemistapoja.

HTML-kielinen dokumentti on tavallinen tekstitiedosto. Se sisältää sisällöllistä tekstiä sekä merkittyjä elementtejä. Merkitty elementti voisi olla vaikka otsake:

```
<title>WWW-esimerkki</title>
```

Kieli perustuu rakenneosien loogiseen merkitsemiseen. WWW-selain muotoilee lopullisesti esitettävän dokumentin ja päättää sen ulkoasun. Rakenneosa merkitään

siten, että sen eteen sijoitetaan alkumerkki ja jälkeen loppumerkki. Nämä ovat muotoa:

<TUNNUS>  = osan TUNNUS alkumerkki,

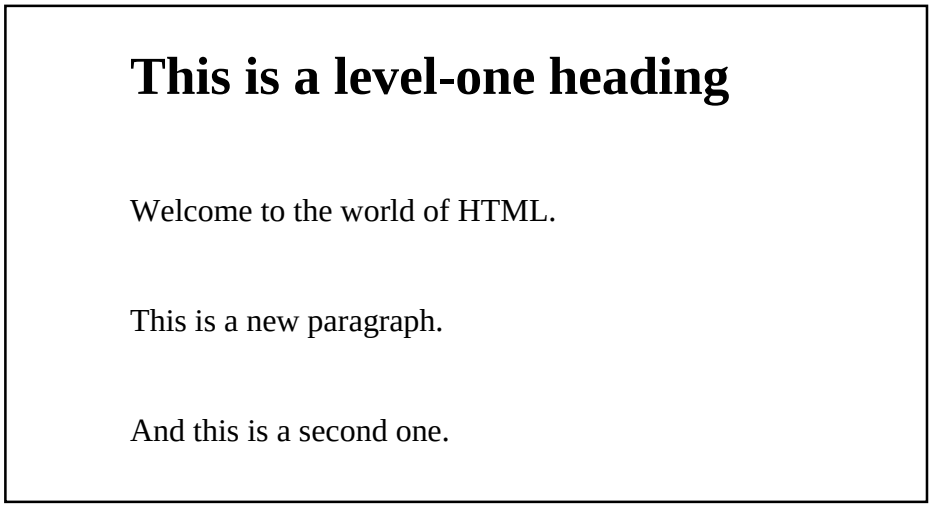
</TUNNUS>  = osan TUNNUS loppumerkki.

Alku- ja loppumerkeissä ei ole väliä kirjoitetaanko ne isoilla vai pienillä kirjaimilla, esim.  <TUNNUS>, <tunnus> tai <TuNNuS>.

Seuraavassa esimerkin yksinkertaisessa HTML-dokumentissa on otsake, näkyvä pääotsake sekä kolme tekstikappaletta.

```
<TITLE>The simplest HTML example </TITLE>
<H1>This is a level-one heading </H1>
<P>Welcome to the world of HTML.</P>
<P>This is a new paragraph.</P>
<P>And this is a second one.</P>
```

WWW-selain voisi näyttää dokumentin vaikkapa seuraavasti (kuva 6.3):




Kuva 6.3: WWW-sivun elementtejä.

HTML-kielinen dokumentti jakautuu otsakeosaan (head) ja runko-osaan (body). Otsakeosassa voidaan esittää erilaista dokumenttiin liittyvää kuvailutietoa, kuten

- sisältöä kuvaava otsake,
- dokumentin laatija,
- dokumentin kieli,
- käytetty tekijäjärjestelmä,
- noudatettava standardi, yms.

Osassa annettavat tiedot eivät näy dokumentissa, mutta hakukoneet ja selaimet saattavat käyttää niitä hyväkseen.

Runko-osa sisältää dokumentin varsinaisen näkyvän sisällön. Alla on esimerkki suositeltavasta dokumentin rakenteesta. Esimerkki alkaa dokumenttityypin määrittelyllä. Tässä tapauksessa dokumentti on määritelty HTLM-standardin 4.0 mukaiseksi ja suomenkieliseksi. Esimerkin tekstiosan lopussa on tieto laatimisajankohdasta ja laatijan sähköpostiosoite. Sen otsakkeena on ’Kuvaava otsake’ ja tekijänä Harri Laine, joka on tavoitettavissa annetusta sähköpostiosoitteeesta.

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.0//FI”>
<HTML>
    <HEAD>
     <TITLE>Kuvaava otsake</TITLE>
     <LINK REV=”made”
      HREF=”mailto:harri.laine@cs.helsinki.fi”>
   </HEAD>
   <BODY>
     Tässä olisi sitten se varsinainen sisältö.
     <HR>
     Created : 1997
     <A href=”mailto:Harri.Laine@cs.helsinki.fi”>
      Harri.Laine@cs.helsinki.fi</A>
   </BODY>
</HTML>
```

HTML-koodin saa muotoilla tiedostossa haluamallaan tavalla. Tyhjien rivien ja sisennysten käyttö saattaa selventää koodia. Rivinvaihdot eivät normaalisti ole merkityksellisiä. Selaimet eivät niitä huomioi. Selaimissa useat perättäisetkin välilyönnit, sarkainmerkit ja rivinvaihdot tulkitaan vain yhdeksi välilyönniksi.

### *6.2.1 HTML-elementtejä*

- Otsikot

  HTML:ssä on tarjolla 6 näkyvää otsikkotasoa (<H1> .. <H6>). Yleensä dokumentin alussa on ylimmän tason (taso 1) otsikko.

- Linkit

  Dokumenttiin upotettavat hyperetekstilinkit tekevät HTML:stä hypertekstikielen. Linkit voivat osoittaa

  - muihin dokumentteihin tai niitä tuottaviin ohjelmiin tai
  - saman dokumentin muihin osiin.

  Selain esittää linkiksi määritellyn sanan tai kohdan **korostettuna**
  esimerkiksi eri värisenä tai alleviivattuna. Linkin syntaksi on:
  **<A HREF=”URL”>dokumentissa näkyvä teksti</A>**

  URL (Universal Resource Locator) on tietoresurssin universaalinen osoite. Täydellisenä se sisältää koneen verkko-osoitteen ja identifioi täten resurssin maailmanlaajuisesti. Osoite voidaan esittää myös suhteellisena parhaillaan käsiteltävän dokumentin osoitteen suhteen.
  URL:n yleinen muoto on

  **protokolla://kone.alue[:portti]/polku/tiedosto**

  Protokolla on esimerkiksi:

  - http: HTML-muotoiset dokumentit
  - file: paikallinen tiedosto
  - ftp: FTP-palvelimessa oleva tiedosto
  - gopher, news ja telnet.

  Esimerkiksi

  ”http://www.cs.helsinki.fi/u/laine/info/syksy97/index.html”

  viittaa tiedostoon

  ~laine/public_html/info/syksy97/index.html

  Helsingin ylipiston tietojenkäsittelytieteen laitoksen tiedostopalvelimella. Tarkastellaan yllämainittua dokumenttia. Siihen sisältyvä suhteellinen linkki

  <A HREF=”h1s.txt”>Harjoitus 1</A>

  viittaa tiedostoon

~laine/public_html/info/syksy97/h1s.txt

samassa ympäristössä.

Paitsi kokonaisiin tiedostoihin HTML:ssä voi viitata myös tiettyyn kohtaan tiedostoa. Tällöin käytetään paikkamerkkejä (anchors).

**Esimerkki**

dokumentissa B

```
juuri tässä on <A NAME=”tarkeapaikka”>
tärkeä paikka</A>, johon haluan viitata
muualta
```

dokumentissa A

```
Tässä viittaan
<A HREF= ”documenttiB.HTML#tarkeapaikka”>
dokumentin B kohtaan ’tärkeä paikka’</A>.
```

Dokumentin sisältä voi nimettyyn kohtaan viitata suhteellisesti:

```
<A HREF = “#tarkeapaikka”>sisäinen viittaus</A>
```

- Listat –

Numeroimaton lista

```
<UL>
<LI>Musti
<LI>Lassie
</UL>
```

näyttää tältä:

- Musti
- Lassie

Numeroitu lista

```
<OL>
<LI>Suomen pystykorva
<LI>Suomen ajokoira
<LI>Tanskan doggi
</OL>
```

näyttää tältä (numeroiden esitysmuoto ja nymeroinnin aloituskohta määritel-tävissä):

1. Suomen pystykorva
2. Suomen ajokoira
3. Tanskan doggi

- Listat - Määrittelylista

```
<DL>
<DT>Dobermanni
 <DD>Tukevahampainen nakertaja
<DT>Schäfer
<DD>Lojaali louskuttaja
</DL>
```

näyttää tältä:

Dobermanni

Tukevahampainen nakertaja

Schäfer

Lojaali louskuttaja

- Valmiiksi muotoiltu teksti

Esimerkiksi ohjelmateksti tai taulukot

```
<PRE>
     Tammi     Helmi     Maalis
     3800      4532      3440
      940      1422      1503
</PRE>
```

näyttää tältä :

```
    Tammi     Helmi     Maalis
    3800      4532      3440
     940      1422      1503
```

- Taulukot

  HLML:ssä voi määritellä ‘oikeitakin’ taulukoita:

```
<table border=1 align=center width=80%>
<tr>
     <th width=20%>koodi</th>
     <th width=60%>nimi</th>
    <th width 20%>ov</th>
</tr>
<tr>
     <td>1134</td>
     <td>Informaatiojärjestelmät</td>
     <td>4</td>
</tr>
<tr>
     <td>1135</td>
     <td>Java ohjelmointi</td>
     <td>3</td>
</tr>
<tr>
     <td>1136</td>
     <td>Oliotietokannat</td>
     <td>4</td>
</tr>
<tr>
     <td>1137</td>
     <td>Tietorakenteet</td>
     <td>5</td>
</tr>
</table>
```

  Taulukko näyttää seuraavalta:

| **koodi** | **nimi** | **ov** |
|---|---|---|
| 1134 | Informaatiojärjestelmät | 4 |
| 1135 | Java ohjelmointi | 3 |
| 1136 | Oliotietokannat | 4 |
| 1137 | Tietorakenteet | 5 |

- Muotoillut tekstit

  Muotoilumääreillä määritellään tekstille looginen tyyli. Tämä ei tarkoita tarkkaa ulkoasun määritystä. Eri selaimet voivat näyttää loogisen tyylin erilaisena. Loogisia tyylejä ovat mm:

  - <EM>korostettu</EM> j
  - <CITE>lainaus</CITE>
  - <CODE>ohjelmakoodi</CITE>
  - <STRONG>vahvennettu</STRONG>

  Vaihtoehtona loogiselle tyylille on käyttää fyysisiä tyylejä. Näillä halutaan nimenomaan tietty ulkoasu. Fyysisiä tyylejä ovat esimerkiksi:

  - <B> lihavoitu</B>
  - <I> kursiivi </I>
  - <TT>konekirjoitusjälki</TT>

  Fyysisiin tyyleihin kuuluu myös mahdollisuus suoraan määritellä käytettävä fonttikoko ja väri sekä jopa käytettävä kirjasinleikkaus.

```
<FONT size=“+2” COLOR=“red”>
     ISO PUNAINEN
</FONT>
```

  Fyysisten tyylien kohdalla voi tietenkin käydä niin, etteivät kaikki selaimet kykene kaikissa ympäristöissä niitä toteuttamaan (kuten tämäkään paperi ei osaa näyttää edellä määriteltyä punaista tekstiä).

- Kuvat

  Eräänä HTML:n suureen suosioon vaikuttaneista tekijöistä on ollut mahdollisuus välittää dokumenteissa paitsi tekstiä myös kuvia. Kuvilla saadaan toisaalta aikaan näyttäviä dokumentteja, toisaalta pystytään välittämään informaatiota, jota ei tekstimuodossa ole mahdollista välittää. HTML:n kuvaelementti on <IMG>. Kuvaelementtiin liittyvinä parametrina annetaan kuvan URL (parametri SRC) ja sijoittelu- sekä kokotietoja. Esimerkiksi.

```
<IMG SRC=”kuvat/kaavio1.gif” ALIGN=LEFT>
```

liittää dokumenttiin sen alkuperähakemiston alihakemistosta *kuvat* kuvan ”*kaavio1.gif*”. Kuva sijoitetaan näytön vasempaan laitaan ja teksti tulee sen oikealle puolelle.

Kuville on olemassa useita esitysmuotoja. Selaimien yleisesti tunnistamat kuvamuodot ovat nykyään GIF ja JPEG. Molemmat ovat pakattuja bitti-karttamuotoja.

GIF-muodossa kuva pakataan häviöttömästi. Tekniikka soveltuu hyvin kuviin, joissa on vähän värejä (esim. kaavioihin). GIF-kuvissa voi olla korkeintaan 256 väriä/sävyä. GIF-tekniikalla voidaan välittää myös kuvasarjoja ns. GIF-animaatioita. Näitä käytetään esimerkiksi mainoksissa. GIF-tekniikka mahdol-listaa myös läpinäkyvän taustan käytön. Eräs GIF:n ongelma on, että se ei ole julkinen standardi.

JPEG tekniikassa kuvan pakkaaja voi säätää kuvan pakkausastetta. Kuvan laatu saattaa kuitenkin kärsiä pakkaamisessa, jos halutaan kovin tiivis esitys. Esimerkiksi valokuvien kohdalla JPEG tuottaa pienempiä kuvatiedostoja kuin GIF. JPEG on julkinen standardi, joten sen käyttöön ei liity lisenssimaksuja. Värisävyjä JPEG kuvissa voi olla $2^{24}$ .

PNG (Portable Network Graphics) on uusin erityisesti WWW:n kautta välitet-tävien kuvien esitysmuodoksi suunniteltu muoto. Se pakkaa kuvan häviöttö-mästi ja tiiviimmin kuin GIF. Värisävyjä on käytössä yhtä paljon kuin JPEG:ssä. Taustan läpinäkyvyys on tarjolla ja standardi on julkinen. Merkittävimpien selainten uusimmat versiot osaavat jo näyttää PNG-muo-toisia kuvia.

3-ulotteisten kuvien välittämiseen on määritelty VRML-standardi. Näiden katselu voi edellyttää selaimeen liitettyjä lisämoduuleja.

Voisi ajatella, että erilaiset kaaviot muodostaisivat merkittävän osan doku-mentteihin sisältyvistä kuvista. Kaaviot saa kaikkein tiiviimmin esitettyä jollain vektorigrafiikkapohjaisella esitysmuodolla. Yleiskäyttöistä esitys-

muotoa tällaiseen käyttöön ei kuitenkaan ole tarjolla. XML-standardi tuonee ratkaisun kaaviografiikan välittämiseen.

- Ääni, video, animaatiot

  HTML-sivujen kautta voidaan välittää myös ääntä, videopätkiä sekä jopa jatkuvaa kuvaa ja/tai ääntä. Näiden hyväksikäyttö ei kuitenkaan yleensä ole selaimen perustoiminto, vaan niitä varten selaimeen on kytkettävä kyseiseen ääni tai videoesitysmuotoon liittyvä täydennysosa. Liikkuvan kuvan esitysmuotoja ovat mm. AVI ja MPEG.

### *6.2.2 WWW-pohjaiset tietokantasovellukset*

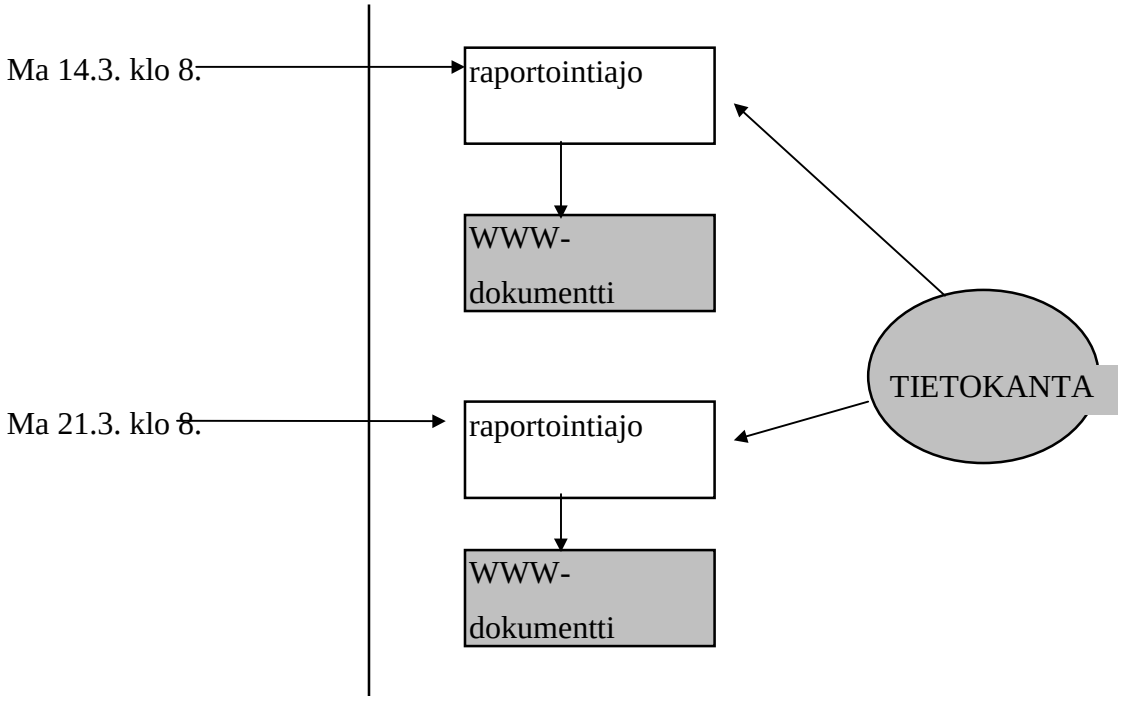
HTML-kieliset dokumentit muodostavat periaatteessa staattisen aineiston. Dokumenttien ylläpito on hankalaa, etenkin jos sama tieto esiintyy useassa dokumentissa. Dokumenttikokoelman pitäisi olla hyvin dokumentoitu ja hallittu, jotta ylläpito toimisi. Käytäntö on osoittamassa, että staattisiin dokumentteihin perustuva tiedotus pysyy huonosti ajantasalla, linkit katkeilevat, tietoa ei löydy tai, jos löytyy se on vanhentunutta. Tiedon löytäminen hypertekstistä on aina vaikeaa, jollei lukija sisäistä dokumentin laatijan käyttämää informaation ‘jäsennysmallia’. Erilaisista hakukoneista (esim. AltaVista) saattaa olla apua tiedon löytymisessä. Tämäkin tosin saattaa edellyttää, että tekijät osaavat kuvailla dokumenttinsa oikein hakukoneita varten.

Dokumenttikokoelman ylläpitoa voidaan helpottaa perustamalla kokoelma staattisten dokumenttien asemasta generoitaviin dokumentteihin. Tällaisessa ratkaisussa dokumentit tuotetaan generointiohjelman avulla tietokantaan sisältyvän aineiston perusteella. Tuottaminen voi olla:

- ajoittaista

  Käytössä on erillinen ohjelma, joka suorittaa tietokantakyselyt ja muokkaa saamansa vastaukset HTML-kieliseksi dokumentiksi, joka asetetaan tarjolle ja

uudistetaan kun dokumentin perustana olevissa tiedoissa on tapahtunut muutoksia. Uudistaminen voisi tapahtua säännöllisin aikavälein (kuva 6.4). Tietokannan muutos voisi myös automaattisesti laukaista uudistuksen. Esimerkiksi tuoteluettelo voitaisiin uudistaa kerran viikossa (tuotteet tietokannassa). Kurssitarjontaluettelo voitaisiin uudistaa aina kun on perustettu uusia kursseja (kurssit tietokannassa). Uudistamismalli sopii tilanteisiin, joissa muutoksia tietokantaan tehdään suhteellisen harvoin. Lähestymistavan käyttöön tarvitaan tietokannanhallintajärjestelmä ja mahdollisesti raportointiväline, joka helpottaa vastauksen saattamista HTML-muotoon (aika monet raporttikehittimet osaavat jo tämän).

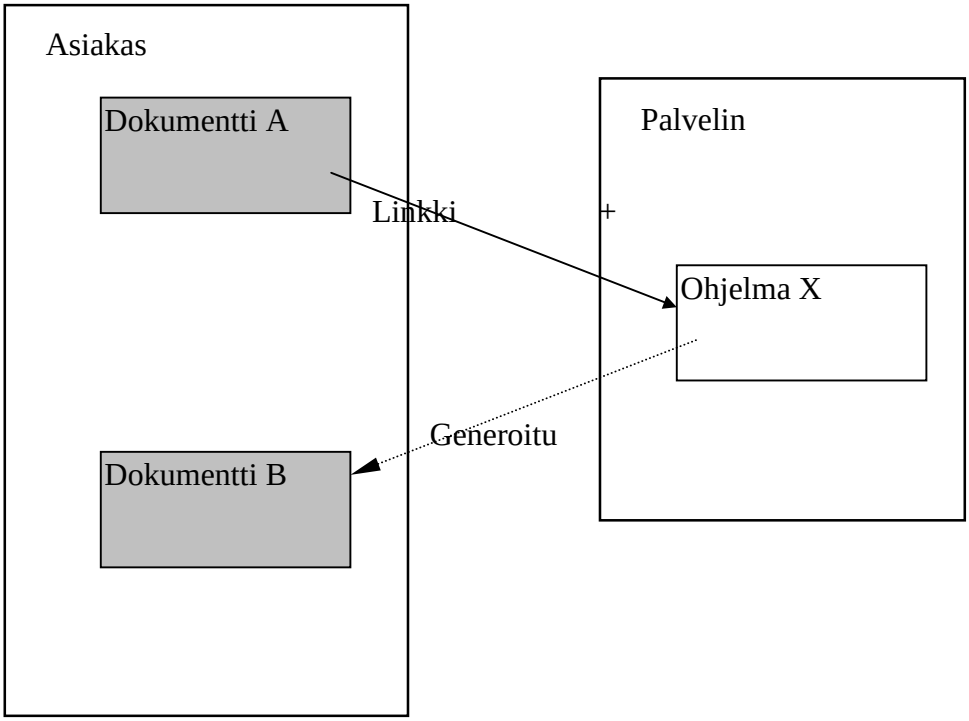



Kuva 6.4: Määräajoin tuotettavat WWW-dokumentit

- pyydettäessä tapahtuvaa

  Dokumentti tuotetaan uudelleen aina kun sitä halutaan. Tässä tapauksessa dokumentin laatii WWW-palvelimessa oleva ohjelma, joka käynnistetään aina kun dokumenttia halutaan. Dokumenttipyyntö on tällöin siis ohjelman käynnistyspyyntö. Ohjelmien kaynnistys ja niiden tuottamien tulosten välitys

WWW-asiakkaalle hoidetaan WWW:ssä perinteisesti CGI (common gateway interface) tekniikan avulla (kuva 6.5). Vaihtoehdoksi CGI:lle on noussut uudempi Java palvelinkomponentteihin (servlet) perustuva tekniikka, joka mahdollistaa CGI:tä nopeamman palvelun.




Kuva 6.5: Dokumenttien generointi dokumenttia pyydettäessä.

#### *6.2.2.1 CGI*

CGI:n kautta käynnistettävälle ohjelmalle voi välittää parametreja liittämällä ne URL:iin.

Esmerkiksi:

```
”teoshaku?isbn=0-123456-67-1&kkoodi=FIN”
```

välittää ohjelmalle teoshaku parametrina kysymysmerkin jälkeisen merkki-jonon

Parametreja voi välittää ohjelmalle myös antamalla ne HTML-lomakkeella.

Palvelimessa toimiva ohjelma voi tehdä mitä tahansa, esimerkiksi

- kirjata tilauksen
- lähettää sähköpostin
- tuottaa uuden WWW-dokumentin
- laskuttaa asiakkaan luottokorttia.

Ohjelma voi käyttää tietokantaa tuotettavan uuden sivun laatimiseen. Ohjelma voi olla laadittu millä ohjelmointikielellä tahansa C, Perl, C++, Java. Relaatiotietokantaan kohdistuvan SQL-kyselyn voi hoitaa upottamalla sen ohjelmaan tai käyttämällä ohjelmointirajapintakirjastoa .

#### *6.2.2.2 HTML-lomakkeet*

HTML-kielessä on tarjolla lomake-elementti, jonka avulla voidaan välittää tietoa palvelimessa käynnistettävälle ohjelmalle:

```
<FORM miten ja minne>
     lomakkeen kenttien määrittely
</FORM>
```

’Miten ja minne’ yllä määrittelee parametrien välitystavan sekä käynnistettävän ohjelman. Lomake-elementin alku voisi olla:

```
     <FORM method=”get”
     ACTION=“HTTP://WWW.helsinki.fi/cgi-bin/prog1”>
```

Tällöin lomakkeen lähetyksen yhteydessä käynnistyy palvelimella *WWW.helsinki.fi* hakemistossa *cgi-bin* oleva *prog1* –niminen ohjelma. Parametrit tälle ohjelmalle välitettäisiin tarkoitukseen varatun ympäristömuuttujan kautta (*method=get*). Parametrit voidaan välittää myös standardisyöttöjonon kautta (*method=post*). Standardisyöttöjonoon perustuvaa välitystapaa on käytettävä, jos väitettävän parametritiedon määrä on suuri. Lomakkeen lähetyksen yhteydessä parametrien arvot talletetaan ympäristömuuttujaan tai syöttöjonoon samassa muodossa, jossa ne esitettäisiin URL:iin liitettyinä.

Lomakkeen erottaa muun tyyppisestä HTML_dokumentista se, että lomake voi sisältää kenttiä. Vähimmillään lomakkeella on oltava lähetysnappi, jota painamalla kenttien arvot lähetetään parametreina kutsuttavalle ohjelmalle.[2]
Lomakkeen kentät esitetään input-, textarea tai select-elementteinä.

```
<INPUT TYPE=”tyyppi” NAME=”nimi” VALUE=”arvo”>
```

Kenttätyyppejä ovat (attribuutti type):

- text – merkkijonoarvoinen yhdelle riville sijoittuva tekstikenttä
- radio – poissulkeva valintanappi (name kertoo ryhmän, johon nappi kuuluu)
- checkbox – valintanappi
- submit – lähetysnappi
- image – kuvalla varustettu lähetysnappi
- reset – tyhjennysnappi
- password – kuten text, mutta syötettävä teksti ei näy
- hidden – kokonaan piilossa oleva kenttä (näkyy toki katsottaessa dokumenttia source-muodossa).

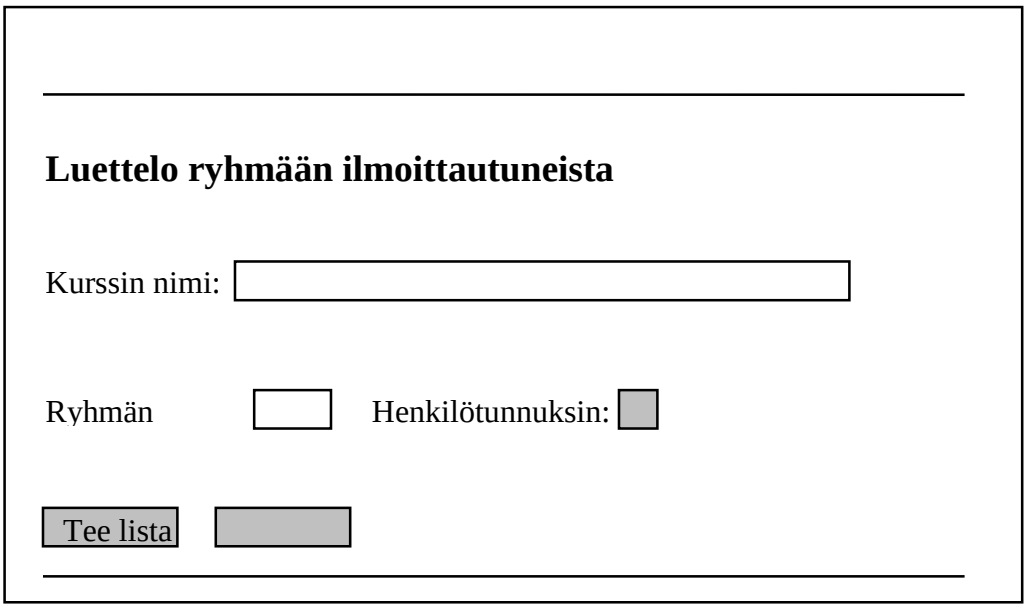
Attribuutit *NAME* ja *VALUE* ovat tarpeen useimpien kenttätyyppien kohdalla. *NAME* ilmoittaa nimen, joka kentän arvoon liitetään kun arvo välitetään parametrina. *VALUE* ilmoittaa kentän alkuarvon, tai kenttään liittyvän parametriarvon, jos sitä ei voi käyttöliittymän kautta muuttaa. Lähetys- ja tyhjennysnappien kohdalla *VALUE*-määre ilmoittaa napissa näkyvän tekstin. Joihinkin kenttätyyppeihin voidaan liittää myös muita määreitä, esimerkiksi *SIZE* ilmoittamaan kentän pituuden merkkeinä, *MAXLENGTH* ilmoittamaan kenttään kirjoitettavan arvon maksimipituuden. Valintanappeihin voidaan liittää *CHECKED*-määre ilmoittamaan, että nappi on ’alhaalla’.

[2] Javascriptillä saa lomakkeen lähetettyä myös ilman lähetysnappia.

Esimerkki:

```
<HR>
<H2> Luettelo ryhmään ilmoittautuneista</H2>
<FORM METHOD=“Get”
      ACTION= ”http://WWW.helsinki.fi/cgi-bin/ophaku”>
Kurssin nimi:
<INPUT TYPE=“TEXT” NAME=“kurssi” SIZE= 40 MAXLENGTH=40> <P>
Ryhmän numero:
<INPUT TYPE=“text” NAME=“ryhma”
      VALUE=“” SIZE=4 MAXLENGTH=4>
Henkilötunnuksilla:
<INPUT TYPE=“CHECKBOX” NAME=“cb1” VALUE=“tunnuksin” >
<P>
< INPUT TYPE =“SUBMIT” VALUE=“Tee lista”>
< INPUT TYPE =“RESET” VALUE=“Tyhjennä“>
</FORM>
<HR>
```

tuottaa lomakkeen:




Kun ’Tee lista’ –nappia painetaan lähtee käynnistyspyyntö ohjelmalle ophaku. Oletetaan, että kurssin nimi kentässä on arvo ’Java ohjelmointi’ ja ryhmänumero kentässä arvo 1 ja henkilötunnukset on valittu. Tällöin käynnistyspyyntöön liitettävä parametrimerkkijono olisi:

```
kurssi=Java+ohjelmointi&ryhma=1&cb1=tunnuksin.
```

Käynnistettävän ohjelman pitää ensityökseen purkaa esiin tarvitsemansa parametriarvot tästä merkkijonosta.

Lomakkeelle voi sijoittaa myös useammasta rivistä muodostuvia tekstialueita (textarea) sekä valikkoja (select)

```
<Select name="valikko" size=14>
<option selected>Ensimmäinen
<option>Toinen
….
<option>Kymmenes
</select>
```

Valikko voi olla yksivalintainen (tavallisesti) tai monivalintainen (määre MULTIPLE). Monivalintaisen valikon asemesta saattaa olla parempi käyttää valinta-nappiryhmää.

Uusimmassa HTML-standardissa on jo välineistö kenttien sijoittelun tarkkaan määrittelyyn. Jos tällaisia tietoja ei käytetä, selain sijoittelee kentän omien sijoittelu-periaatteidensa mukaisesti. Toivotun sijoittelun aikaansaaminen voi olla hankalaa. Eräs tapa saada aikaan siisti sijoittelu, jossa esimerkiksi kenttien vasen reuna on tasattu, on sijoittaa lomake-elementit taulukkoon.

Toteutettaesa tietokantasovellus HTML-lomakkeita käyttäen, erääksi pääongelmaksi muodostuvat erilaiset tarkistukset, joita syötettävälle datalle pitää tehdä. Kaikkien selainten yhteydessä toimii ainoastaan sellainen tekniikka, jossa kenttien arvot tarkastetaan vasta lomakkeen käsittelevässä palvelinohjelmassa. Jos jossakin kentässä on virhe, lomake palautetaan korjattavaksi. Tämä on toimiva tekniikka, mutta hyvän interaktiivisen käyttöliittymän oletetaan reagoivan välittömästi virheeseen, eikä vasta koko lomakkeen täytön jälkeen.

Tarkistukset voidaan määritellä skriptikielillä. HTML-standardiversiossa 4.0 on määritelty kätevät ripustuspisteet (onchange, onblur) tarkastusten määrittelyyn. Skriptikielet ovat ohjelmointikieliä, joilla laaditut lähdekieliset ohjelmamääritykset voidaan välittää osana HTML-kielistä dokumenttia. Selaimen pitäisi osata tulkita skriptejä ja suorittaa ne. Ongelma kuitenkin on se, että selaimet osaavat kovin eri tavoin käsitellä eri skriptikieliä. Netscape hallitsee hyvin JavaScriptin ja Internet Explorer VBScriptin (uusin versio toki JavaScriptinkin). Jotkin selaimet eivät hallitse

mitään skriptikieliä. Näinollen skriptikielen valinta sitoo sovellukset tiettyyn selaimeen ja mahdollisesti sitä kautta tiettyyn toimintaympäristöön.

### *6.2.3 Esimerkkejä tietokantasovelluksen laatimisesta*

Seuraavissa esimerkeissä tarkastellaan WWW-alustalle toteutettujen tietokantasovelluksen toteutusta Oracle Web Server –palvelimen avulla. Oracle Web Server mahdollistaa tietokantaproseduurien suoran hyväksikäytön WWW-dokumenttien kautta käynnistettävinä ohjelmina. Web Server hoitaa mm. automaattisesti hhtp-ohjelmakutsussa annettavien parametrien sitomisen tietokantaproseduurin parametreihin. Esimerkissä käytetään tauluista opettaja, kurssi, harjoitusryhmä ja ilmoittautuminen koostuvaa tietokantaa.. Seuraavassa käsitellään lähinnä opiskelija-taulua.

Oracle-tietokantaan liittyvät tietokantaproseduurit laaditaan PL/SQL-ohjelmointikielellä.[3] Kieli on melko lähellä SQL:n tietokantaproseduuri –standardia. Se on Ada-kielen rakenteisiin perustuva ohjelmointikieli, jossa tietokannan käsittely on integroitu kieleen tiiviimmin kuin upotettua SQL:ää käytettäessä. Seuraavassa esitellään lyhyt tiivistelmä PL/SQL-kielen piirteistä. Tarkastelun kohteena on proseduurin määrittely. Lisää tietoa löytyy englanninkielisestä manuaalimateriaalista.

PL/SQL –proseduurimäärittely jakautuu viiteen osaan:

- proseduurin nimi
- parametrien esittely
- paikallisten tietorakenteiden esittely
- toimintaosa ja
- poikkeukset.

```
create or replace procedure proseduurin_nimi
```

[3] PL/SQL ei sisälly Tietokantojen perusteet kurssin vaatimuksiin. Kieli on esitelty tässä koska pääosa Tietokantasovellusten harjoitustyö kurssin töistä toteutetaan sen avulla.

```
        ( parametrit ) is
        paikalliset tietorakenteet
begin
        toiminta_osa
execption
        poikkeukset
end;
```

Näistä osista parametrit, paikalliset tietorakenteet ja poikkeusosa voivat puuttua.

**Parametrit**

Yleisesti PL/SQL-proseduurin parametrien voivat olla syöte-, tulos- tai sekä syöte- että tulosparametreja. Parametrin esittelyssä määritellään kunkin parametrin osalta käyttötapa (in / out / in out), nimi ja tietotyyppi. Tietotyyppeinä tulevat kyseeseen SQL:n tietotyypit sekä PL/SQL:n omat tietotyypit.

```
create or replace procedure testi
     (in teksti1 varchar2, in luku1 integer) is ….
```

**Paikalliset tietorakenteet**

Paikallisina tietorakenteina voidaan määritellä yksinkertaisia muuttujia, tietueita, taulukoita ja kursoreita. Yksinkertaisen muuttujan määrittelyssä annetaan muuttujan nimi ja tietotyyppi. Tietotyyppejä ovat SQL:n tietotyypit integer, number, char, varchar ja date. Ne määritellä kuten SQL:ssä. Lisäksi on käytettävissä Boolean tietotyyppi.

Tietotyyppi voidaan määritellä myös toisen muuttujan tai taulun sarakkeen tietotyypin avulla. Lausekkeella

```
s1kopio virtanen.tauluA.sarake1%TYPE
```

määritellään muuttuja *s1kopio* saman tyyppiseksi kuin käyttäjän Virtanen omistaman taulun *taulua* sarake *sarake1*.

Tietuerakenne voidaan määritellä tauluun tai kyselyyn perustuen.

```
rivi1 virtanen.TauluA%ROWTYPE
cursor c1 is
     select * from virtanen.TauluB;
```

```
rivi2 c1%ROWTYPE;
```

Kyselyt määritellään kursoreiden avulla, kuten sulautetussa SQL:ssä. PL/SQL tarjoa kursoreiden läpikäyntiin kursorisilmukan, joka yksinkertaistaa käsittelyä sulautettuun SQL:ään verrattuna.

**Toimintaosa**

PL/SQL –kielen toimintaosassa ovat käytettävissä tavanomaiset ohjelmointikielen operaatiot ja rakenteet:

- sijoitusoperaatio ( `luku := 19;` )
- lausekkeet, ( `luku * 2 + toinen_luku` )
- perusaritmetiikka (`luku :=luku * 2 + toinen_luku` )
- merkkijonojen liittäminen (`Nimi:= etunimi || ’ ’ || sukunimi`)
- proseduureja ja funktioita, mm. kaikki Oraclen SQL:n funktiot ohjelmointikielen funktioina

```
etukirjain := substr(Sukunimi,1,1);
viikonpaiva:= to_char(sysdate,’DAY’);
```

- ehtorakenne if-lausekkeina

```
if ehto then
   toiminto;
end if;

if ehto then
   toiminto1;
else
   toiminto2;
   toiminto3;
end if;

if ehto1 then
   toiminto1;
elsif ehto2 then
   toiminto2;
   toiminto3;
else
   toiminto4;
end if;
```

- Silmukat, mm
  - while-silmukka – suoritetaan, niin kauan kun ehto on tosi

```
while ehto loop
   toiminta;
end loop;
```

  - for-silmukka – suoritetaan kaikilla arvovälin arvoilla

```
for muuttuja in (alaraja .. yläraja )loop
   toiminta;
end loop;
```

  - kursorisilmukka – suoritetaan jokaiselle kyselyn tulosriville

```
for rivimuuttuja in kursori loop
toiminta;
end loop;
```

Kursorisilmukassa suoritetaan aluksi kysely. Sitten suoritetaan määritelty toiminta jokaiselle riville ja lopuksi suljetaan kursori. Rivimuuttuja osoittaa kyselyn tulosriviin eikä sitä määritellä erikseen. Rivin sarakkeisiin viitataan muodossa :**rivimuuttuja.sarakenimi**.

```
cursor kkursori is
     select jotakin from jostakin;
…
for rivi in kkursori loop
     a:= rivi.jotain;
       …
end loop;
```

Kyselyn tulosrivejä voi käydä läpi myös while- silmukassa. Tällöin kursori on ensin avattava Open-lauseella ja rivillä olevat arvot täytyy hakea muuttujiin *Fetch*-lauseella kuten sulautetussa SQL:ssä.

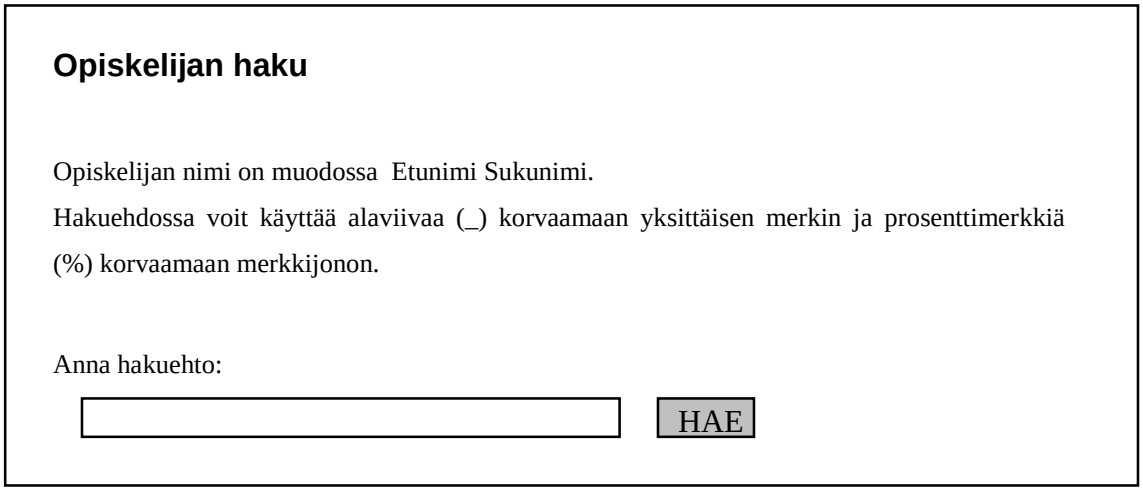
*Esimerkki 1: Opiskelijoiden haku*

Sovelluksen ensimmäinen manuaalisesti laadittu HTML-sivu (ophaku.HTML) sisältää lomakkeen, jonka avulla käyttäjä voi hakea opiskelijan tietoja opiskelijan nimen perusteella. Lomakkeen käsittelee tietokantaproseduuri *opiskelija_data*. Proseduuri tuottaa HTML-kielisen tulosdokumentin, jossa näkyvät hakuehdon täyttävien opiskelijoiden tiedot.

Dokumentti: ophaku.HTML

```
<HTML>
<HEAD>
<TITLE>Opiskelijan haku –lomake INFO k97</Title>
</HEAD>
<BODY BGCOLOR=yellow>
<H1>Opiskelijan haku</H1>
<blockquote>
Opiskelijan nimi on muodossa Etunimi Sukunimi.
<br>
Hakuehdossa voit käyttää alaviivaa (_) korvaamaan yksittäisen
merkin ja prosenttimerkkiä (%) korvaamaan merkkijonon.<P>
<Form method=”get” action=
”http://kontti.helsinki.fi:8889/ttst/owa/info1.opiskelija_data”
>
Anna hakuehto:
<table cellpadding=10><tr>
<td><input type=”text” name=”onimi” size=40 maxlength=40> </td>
<td><input type=”submit” value=”HAE”> </td>
</table></form>
<hr>
<Font size=’-1’>
Haku kohdistuu Informaatiojärjestelmät-kurssin
esimerkkitietokantaan
<i>Harri Laine</i>
</body>
</HTML>
```

Dokumentissa lomake-elementit on sijoitettu taulukkoon. Käynnistettävä proseduuri on käyttäjän info1 omistama. Lomake näyttää seuraavalta:



Proseduuri opiskelija_data:

```
create or replace procedure opiskelija_data
   (onimi in varchar2) is
cursor op is
      select onumero, nimi, paa_aine, kaupunki,
        aloitusvuosi
      from info1.opiskelija
      where nimi like onimi
      order by nimi;
t varchar2(80);
nobody boolean;
begin
   nobody:=true;
   htp.HTMLOpen;
   htp.headOpen;
   htp.title('Kyselyesimerkki');
   htp.headClose;
   htp.bodyOpen(NULL,'BGCOLOR="WHITE"');
   t:='<h2>Opiskelijat, hakuehdolla NIMI LIKE
          '''||onimi||'''</h2>';
   htp.p(t);
   for o in op loop
      nobody := false;
      htp.bold(o.nimi);
      htp.blockquoteOpen;
      htp.p('tunnus='||to_char(o.onumero)||',
            kaupunki= '||o.kaupunki);
      htp.p(', pääaine= '||o.paa_aine);
      htp.blockquoteClose;
      htp.para;
   end loop;
   if nobody = true then
      htp.p('Ketään ei löytynyt!');
   end if;
   htp.bodyClose;
   htp.HTMLClose;
end;
/
```

Tässä esimerkissä on käytetty pakkaukseen *htp* sisältyviä proseduureja tulostamaan tietoja generoitavaan HTML-dokumenttiin. Pakkauksessa on tarjolla proseduurit kaikille HTML:n elementeille. Haluttaessa voitaisiin kaikki materiaali tulostaa htp.p –proseduurin avulla, jolloin olisi itse huolehdittava siitä, että tulostettava merkkijono sisältää kaikki tarvittavat HTML-koodit. Yllä näin on tehty vain otsikon <H2> kohdalla.

Kun tietokantaproseduuri suoritetaan hakuehdolla *'Ville%'* saadaan tuloksena seuraava dokumentti:

## Opiskelijat, hakuehdolla NIMI LIKE ‘Ville%’

**Ville Lahti**

tunnus=3, kaupunki= Helsinki , pääaine= TKT

**Ville Lumme**

tunnus=11, kaupunki= Helsinki , pääaine= MAT

**Ville Salo**

tunnus=9, kaupunki= Espoo , pääaine= TKT

**Ville Tuomi**

tunnus=7, kaupunki= Helsinki , pääaine= TKT

**Ville Varjo**

tunnus=8, kaupunki= Espoo , pääaine= MAT

**Ville Vieras**

tunnus=10, kaupunki= Espoo , pääaine= FYS

Dokumentin määrittelevä HTML-koodi on ohjelmassa tuotettua:

```
<HTML>
<HEAD>
<TITLE>Kyselyesimerkki</TITLE>
</HEAD>
<BODY BGCOLOR=”WHITE”>
<h2>Opiskelijat, hakuehdolla NIMI LIKE ‘Ville%’</h2>
<B>Ville Lahti</B>
<BLOCKQUOTE>
tunnus=3, kaupunki= Helsinki
, pääaine= TKT
</BLOCKQUOTE>
<P>
<B>Ville Lumme</B>
<BLOCKQUOTE>
tunnus=11, kaupunki= Helsinki
, pääaine= MAT
</BLOCKQUOTE>
<P>
<B>Ville Salo</B>
<BLOCKQUOTE>
tunnus=9, kaupunki= Espoo
, pääaine= TKT
```

```
</BLOCKQUOTE>
<P>
<B>Ville Tuomi</B>
<BLOCKQUOTE>
tunnus=7, kaupunki= Helsinki
, pääaine= TKT
</BLOCKQUOTE>
<P>
<B>Ville Varjo</B>
<BLOCKQUOTE>
tunnus=8, kaupunki= Espoo
, pääaine= MAT
</BLOCKQUOTE>
<P>
<B>Ville Vieras</B>
<BLOCKQUOTE>
tunnus=10, kaupunki= Espoo
, pääaine= FYS
</BLOCKQUOTE>
<P>
</BODY>
</HTML>
```

Esimerkki 2: Opiskelijatietojen muuttaminen.

Edellisessä esimerkissä generoitiin tulosdokumenttina raportti, joka näytti opiskelijatietoja. Tulosdokumenttiin voidaan sisällyttää myös linkkejä ja lomakkeita. Esimerkiksi virhetilanteessa palutetaan yleensä samanlainen lomake jolta tietoa syötettiin virheilmoituksella täydennettynä. Tässä esimerkissä lähdetään liikkeelle edellisen esimerkin hakusivusta, mutta käynnistetäänkin tietokantaproseduuri, joka tuottaa tuloksenaan saman hakulomakkeen täydennettynä löydettyjen opiskelijoiden yksityiskohtaisten tietojen haun käynnistävällä valikolla. Ensimmäisen hakulomakkeen käsittelevän proseduurin (opiskelija_sel) koodi on tällöin:

```
create or replace procedure opiskelija_sel (onimi in
varchar2) is
  cursor op is
    select onumero, nimi, paa_aine, kaupunki,
aloitusvuosi
      from info1.opiskelija
      where nimi like onimi
      order by nimi;
  t varchar2(80);
  o op%rowtype;
  jatkuu boolean;
begin
```

```
   htp.HTMLOpen;
   htp.headOpen;
   htp.title('Opiskelijan haku – esimerkkilomake INFO
k97');
   htp.headClose;
   htp.bodyOpen(NULL,'BGCOLOR=silver');
   htp.header(1,'Opiskelijan haku – esimerkkilomake');
   htp.blockquoteOpen;
   htp.p('Opiskelijan nimi on muodossa Etunimi
Sukunimi.');
   htp.p('Hakuehdossa voit käyttää alaviivaa (_)
korvaamaan yksittäisen merkin ja');
   htp.p('prosenttimerkkiä (%) korvaamaan merkkijonon.');
   htp.para;
   htp.formOpen('info1.opiskelija_sel','GET');
   htp.p('Anna hakuehto: ');
   htp.formText('onimi',40,40,'');
   htp.formsubmit('HAE');
   htp.fontOpen(NULL,NULL,-1);
   htp.italic('<br>Haku kohdistuu
esimerkkitietokantaan');
   htp.fontClose;
   htp.formClose;
   htp.hr;
   jatkuu:=True;
   open op;
   fetch op into o;
   if op%notfound then
      htp.bold('Ketään ei löytynyt!');
   else
      htp.header(2,'Seuraavat opiskelijat täyttivät
hakuehdon:');
      htp.tableOpen(1,'center',NULL,NULL,'width 50%');
      htp.tableRowOpen;
      htp.tableHeader('Nimi');
      while jatkuu loop
         htp.tableRowOpen;
         htp.tableData(htf.anchor(
          'info1.opiskelija_data?onimi='||
              replace(o.nimi,' ','+'),o.nimi));
         fetch op into o;
         if op%notfound then
            jatkuu:= false;
         end if;
      end loop;
      htp.tableClose;
   end if;
   htp.bodyClose;
   htp.HTMLClose;

end;
/
show errors;
exit;
```

Tässä proseduurissa tulostetaan aluksi samanlainen kyselylomake, jolta lähdettiin liikkeelle. Lomakkeen käsittelijänä on proseduuri *opiskelija_sel*. Lomakkeen alaosaan tulostetaan taulukko, jossa näkyvät haussa löytyneiden opiskelijoiden nimet tai ilmoitus siitä, että haulla ei löytynyt opiskelijoita. Jokaiseen opiskelijanimeen liitetään linkki, joka osoittaa esimerkissä 1 esitettyyn tietokantaproseduuriin (opiskelija_data), parametriksi proseduurille annetaan opiskelijan nimi. Kun hakuehtona on annettu *'Tele%'*, näyttää tulosdokumentin HTML-koodi seuraavalta:

```
</HEAD>
<BODY BGCOLOR=silver>
<H1>Opiskelijan haku – esimerkkilomake</H1>
<BLOCKQUOTE>
Opiskelijan nimi on muodosssa Etunimi Sukunimi.
Hakuehdossa voit käyttää alaviivaa (_) korvaamaan
yksittäisen merkin ja
prosenttimerkkiä (%) korvaamaan merkkijonon.
<P>
<FORM ACTION=”info1.opiskelija_sel” METHOD=”GET”>
Anna hakuehto:
<INPUT TYPE=”text” NAME=”onimi” SIZE=”40” MAXLENGTH=”40”>
<INPUT TYPE=”submit” NAME=”HAE” VALUE=”Submit”>
<FONT SIZE=”-1”>
<I><br>Haku kohdistuu esimerkkitietokantaan</I>
</FONT>
</FORM>
<HR>
<H2>Seuraavat opiskelijat täyttivät hakuehdon:</H2>
<TABLE  1 ALIGN=”center” width 50%>
<TR>
<TH>Nimi</TH>
<TR>
<TD><A   HREF=”info1.opiskelija_data?onimi=Tele+Fax”>Tele
Fax</A></TD>
<TR>
<TD><A HREF=”info1.opiskelija_data?onimi=Tele+Kopio”>Tele
Kopio</A></TD>
<TR>
<TD><A HREF=”info1.opiskelija_data?onimi=Tele+Visio”>Tele
Visio</A></TD>
</TABLE>
</BODY>
</HTML>
```

Samalla periaatteella voidaan laatia ylläpitolomakkeita ja lisäyslomakkeita. Näitä käsittelevissä proseduureissa suoritetaan pelkkien kyselyiden asemesta

ylläpidon tapauksessa update-lauseet ja lisäyksen tapauksessa insert-lauseet. Lisäyksiä, muutoksia ja poistoja tehtäessä on syytä pitää mielessä, että tietokantayhteys katkeaa aina kun proseduuri on suoritettu. Tapahtumaa ei siis voi vahvistaa vasta seuraavalla sivulla. Jos halutaan mahdollistaa tapahtuman peruutus on muutetut tai postetut tiedot erikseen vietävä johonkin aputauluun ja palautettava arvot sieltä.

Tässä esitetty tapa WWW-alustalla toimivien tietokantasovellusten laatimiseksi on yksi monista vaihtoehdoista. Tarjolla on esimerkiksi välineitä, joissa on oma määrittelykielensä tietokantayhteyttä varten. Samoin on välineitä, jotka mahdollistavat SQL-kyselyjen kirjoittamisen suoraan HTML_koodin sekaan (esim. PHP) Eräänä vaihtoehtona on laatia tietokantasovellus Java-sovelmana (applet) tai ActiveX-komponenttina. Dokumentti sisältää tällöin vain komponentin käynnistyskoodin. Komponentti hoitaa itse kaikki tietokantayhteyteen liittyvät asiat. Näin saadaan aikaan juuri sellainen käyttöliittymä kuin halutaan.

# 7 Relaatiotietokannan suunnittelusta

Tietokantasuunnittelun pääperiaatteena on tiedon toiston välttäminen. Tiedon toistumiseen liittyy monenlaisia ongelmia

- toistuva tieto vie tilaa
- ylläpito muodostuu hankalaksi
- ylläpito-operaatioilla voi olla odottamattomia sivuvaikutuksia.

Tietokannan suunnittelun vaiheita ovat

- tietosisällön kartoitus,
- loogisten rakenteiden suunnittelu ja
- teknisten rakenteiden suunnittelu.

Tietosisällön kartoituksessa selvitetään, mitä tietoa tietokantaan halutaan tallentaa Samassa yhteydessä selvitetään myös erilaisia sääntöjä ja lainalaisuuksia, jotka ovat voimassa tietojen kohdealueella ja joiden pitäisi heijastua myös tietokantaan. Kartoi-

tuksen tulos esitetään käsitetason tietomallina. Käsitetason tietomalleja ja kartoituksen kulkua käsitellään kurssilla ’Johdatus sovellussuunnitteluun’. On olemassa ohjeistoja ja jopa ohjelmia, jotka tuottavat relaatiotietokannan rakenteen käsitetason mallin pohjalta. Tälläinen ohjeisto sisältyy kurssin ’Tietokantasovellusten harjoitustyö’ –materiaaliin.

Tällä kurssilla kuitenkin ohitamme kartoitusvaiheen ja oletamme, että olemme saaneet jotenkin selvitettyä sen, mitä tietoa kantaan tulisi tallentaa. Ongelmaksi jää, miten tiedot kootaan relaatioiksi. Perussääntönä on koota yhteenkuuluvat tiedot samaan relaatioon. Toisin sanoen kaikkien saman relaatiokaavioon sijoitettavien attribuuttien tulisi kuvata samaa ilmiötä tai kohdetta. Esimerkiksi kaikki opiskelija-relaation attribuutit esittävät jonkin opiskelijasta kiinnostavan ominaisuuden ja kuuluvat tästä syystä yhteen.

Relaatiotietokantoihin liittyy mittava suunnitteluteoria, jota käsitellään tarkemmin kurssilla Tietokantojen mallintaminen. Suunnitteluteoriassa yhteenkuuluvuus määritellään tietojen välisten riippuvuuksien avulla. Keskeinen peruskäsite suunnitteluteoriassa on funktionaalinen riippuvuus (functional dependency). Attribuutti B on funktionaalisesti riippuva attribuutista A (A määrää funktionaalisesti B:n), jos ja vain jos kaikissa relaatiokaavion R ilmentymissä kuvaus A:n arvojoukolta B:n arvojoukolle on funktionaalinen. Kuvaus f:v(A)->v(B) on funktionaalinen, jos jokainen A:n arvo kuvautuu yhdelle B:n arvolle eli, jos riveillä r ja s attribuutilla A on sama arvo (r.A=s.A), niin näillä riveillä täytyy myös B-attribuuteilla olla keskenään sama arvo (r.B=s.B). Funktionaalinen riippuvuus tarkoittaa sitä, että attribuutin B arvo on yksikäsitteisesti selvitettävissä kun tiedetään attribuutin A arvo. Selvittäminen voisi tapahtua kyselyllä

```
select distinct B from R where A=a;
```

Jos B on funktionaalisesti riippuva A:sta tuottaa ylläoleva kysely joko yhden tai ei yhtään tulosriviä, mutta ei koskaan enempää. Tämän ehdon on oltava voimassa aina, ei pelkästään sattumalta hetkellisesti.

Funktionaalista riippuvuutta, jossa A määrää B:n merkitään A→B. Attribuuttia A kutsutaan määrääjäksi. Yksittäisen attribuutin A tilalla voi olla myös attribuuttiyhdistelmä. Tavoitteena on kuitenkin löytää yhdistelmät, joissa on minimaalinen

määrä attribuutteja. sillä, jos A→B, voidaan määrääjään lisätä mikä tahansa attribuutti x ja pätee Ax→B.

Relaatiokaaviossa attribuutin ei tarvitse olla funktionaalisesti riippuva yhdestäkään muusta attribuutista. Se voi olla riippuva jostain yksittäisestä attribuutista tai useamman attribuutin yhdistelmästä. Kaikki relaatiokaaviot attribuutit ovat funktionaalisesti riippuvia relaation avaimesta eli, jos suoritamme ylläolevan kyselyn siten, että A on relaation avain, saamme aina vastaukseksi joko ei yhtään riviä tai vain yhden rivin olipa attribuutti B mikä tahansa. Avaimen kohdalla kyselyssä ei tarvita edes distinct määrettä.

Tarkastellaan relaatiokaaviota

Kurssilainen(Kurssikoodi, Hetu, OpiskelijaNimi, KurssiNimi, TehtavaLkm).

Oletetaan että relaatio sisältää tietoja useista opiskelijoista ja useista kursseista.

Attribuutit Kurssikoodi ja Hetu eivät ole funktionaalisesti riippuvia mistään muusta attribuutista. OpiskelijaNimi on funktionaalisesti riippuva attribuutista Hetu, sillä kyselyn

```
select distinct OpiskelijaNimi from Kurssilainen where Hetu= a;
```

voidaan edellyttää tuottavan korkeintaan yhden tulosrivin kaikilla a:n arvoilla. Samoin on voimassa Kurssikoodi→KurssiNimi. Mikään yksittäinen attribuutti ei määrää funktionaalisesti attribuuttia TehtavaLkm. Sillä

```
select distinct TehtavaLkm from Kurssilainen where Hetu=a;
```

tuottaa tuloksenaan opiskelijan a tekemien tehtävien määrät kaikilta kursseilta, jonne opiskelija on osallistunut, eikä tehtävämäärä välttämättä ole sama. Vastaavasti

```
select distinct TehtavaLkm from Kurssilainen where Kurssikoodi=b;
```

tuottaa kaikkien kurssille b osallistuneiden opiskelijoiden tekemien tehtävien määrät.. Sensijaan Hetu ja Kurssikoodi määräävät yhdessä attribuutin TehtavaLkm, koska voimme edellyttää, että kysely

```
select distinct TehtavaLkm from Kurssilainen
where Kurssikoodi=b and Hetu=a;
```

tuottaa korkeintaan yhden tulosrivin olivatpa a ja b mitä tahansa arvoja.

Yhteenkuuluvuus voidaan määritellä funktionaalisten riippuvuuksien avulla. Seuraavassa tarkastelemme vain yhtä yhteenkuuluvuussääntöä, ns. Boyce-Codd normaalimuodon sääntöä. Tämän säännön mukaan

- relaatiokaavion R attribuutit kuuluvat yhteen, jos ja vain jos relaatiokaavioon R ei liity yhtään sellaista funktionaalista riippuvuutta, jossa määrääjä ei sisältäisi relaation avainta.

Yllä olevassa esimerkissä Hetu ja Kurssikoodi yhdessä muodostavat avaimen. Relaatiokaavion attribuutit eivät ole tämän säännön mukaisesti yhteenkuuluvia, koska esimerkiksi riippuvuus Hetu→OpiskelijaNimi on yhteenkuuluvuussäännön vastainen.

Jos attribuutit eivät kuulu yhteen ne on uudelleenjärjesteltävä sellaisiksi relaatiokaavioiksi, jossa yhteenkuuluvuussäännöt ovat voimassa. Esimerkin tapauksessa päädytään relaatiokaavioihin

Opiskelija(Hetu, OpiskelijaNimi)
Kurssi(Kurssikoodi, KurssiNimi)
Osallistuminen(Hetu, Kurssikoodi, TehdytTehtavat).

Alkuperäisessä Kurssilainen kaaviossa esiintyi toistoa. esimerkiksi kurssin nimi olisi pitänyt tallentaa jokaisen kurssin opiskelijan yhteydessä samoin opiskelijanimi jokaisen kurssiosanoton yhteydessä. Uudelleensijoittelun mukaisissa kaavioissa toistoa ei esiinny.

Eräs tapa muodostaa relaatoita on toimia siten, että tietosisältökartoituksen perusteella muodostetaan alustavat relaatiokaaviot. Tämän jälkeen määrätään relaatiokaavioiden avaimet ja niissä voimassa olevat funktionaaliset riippuvuudet. Jos osoittautuu, että jossain kaaviossa on yhteenkuuluvuussäännön rikkovia riippuvuuksia järjestetään tämän kaavion attribuutit uudelleen siten, että yhteisen minimaalisen määrääjän sisältämät attribuutit kootaan omaksi relaatiokaavioksi.

Esimerkki:

Tilauslomaketta analysoitaessa löydettiin seuraavat attribuutit,:
lomakenumero, tilaajan tunnus, tilaajan nimi, tilaajan osoite, tilaajan puhelinnumero, toimitusosoite, rivinumero, tavaran koodi, tavaran nimi, tilattu määrä, ja tilauspäivä.

Tarkastellaan relaatiokaaviota U johon nämä kaikki kuuluisivat. Tähän kaavioon liittyen voimme tunnistaa riippuvuudet:

- lomakenumero → tilaajan tunnus (lomakkeella voidaan ilmoittaa vain yksi tilaaja)
- tilaajan tunnus → tilaajan nimi (tunnus identifioi tilaajan, joten sen perusteella saamme selville kaikki tilaajaan liittyvät tiedot)
- tilaajan tunnus → tilaajan osoite,
- tilaajan tunnus → tilaajan puhelinnumero,
- lomakenumero → toimitusosoite (lomakkeella voidaan tilata tavaroita vain yhteen paikkaan ja sama tilaaja voi tilata eri paikkoihin)
- tavaran koodi → tavaran nimi (tavaran koodi on tavaran tunniste, jonka kayttaa päästään kaikkiin tavaran tietoihin)
- lomakenumero, rivinumero → tavaran koodi (lomakkeen rivillä voi ilmoittaa yhden tilattavan tavaran koodin)
- lomakenumero, rivinumero → tilattu määrä ((lomakkeen rivillä voi ilmoittaa yhden tilattavan tavaran määrän)
- lomakenumero → tilauspäivä

Ylläoleva riippuvuusjoukko on minimaalinen. Näiden riippuvuuksien lisäksi relaatiokaaviossa on voimassa useita muitakin näistä johdettavissa olevia riippuvuuksia. Jatkotoimia varten minimaalisen riippuvuusjoukon löytäminen on oleellista (tästä kerrotaan lisää Tietokantojen mallinnus kurssilla). Kaavion U avain on yhdistelmä lomakenumero, rivinumero. Kaavion liittyy täten useita riippuvuuksia, jotka rikkovat yhteenkuuluvuussääntöä. Attribuutit on siis uudelleensijoiteltava. Kun teemme sijoittelun yhteisen määrääjän perusteella päädymme kaavioihin

X(<u>tilaajan tunnus</u>, tilaajan nimi, tilaajan osoite, tilaajan puhelinumero)
Y(<u>tavaran koodi</u>, tavaran nimi)
Z(<u>lomakenumero</u>,tilauspäivä,tilaajan_tunnus,toimitusosoite)
T(<u>lomakenumero, rivinumero</u>, tavaran koodi, tilattu määrä)

nämä täyttävät yhteenkuuluvuussäännön. Yhtenä testinä sille olemmeko päätyneet toimivaan relaatiotietokantakaavioon voidaan pitää sitä onko syntyneillä relaatio-

kaavioilla jokin luonnollinen merkitys eli onko niille löydettävissä kuvaava nimi. Tässä tapauksessa näin on:

X= tilaaja,

Y= tavara,

Z= tilaus,

T= tilausrivi

Olemme siis löytäneet toistottoman toimivan tietokantakaavion.

**Kirjallisuusluettelo:**


| | |
|---|---|
| [BRJ98] | Booch G., Rumbaugh J., Jacobson I.: The unified modeling language user guide, Addison-Wesley, 1998 |
| [Chen 76] | Chen, P., P.: The entity-relationship model: toward a unified view of data, ACM Trans. On Database Systems, 1:1, pp 9-36, 1976 |
| [Codd70] | Codd E., F,: A relational model of data for large shared data banks, Comm. Of the ACM, 13:6, pp 377-387, 1970 |
| [Ham97] | Hamilton G., Cattell R, Fisher M.: JDBC Database Access with Java –A tutorial and Annodated Reference, Addison-Wesley, 1997 |
| | |
| [EN99] | Elmasri R., Navathe S. B.: Fundamentals of Database Systems, Addison-Wesley,1999 |
| W3C | HTML Markup language, Home Page, http://www.w3.org/MarkUp/ |
| [Zloof75] | Zloof: Query by Example, National Computinc Conference, AFIPS,44, 1975 |