Tallinna Ülikool, Informaatika Instituut

Veebirakenduste loomine PHP ja MySQLi abil

Jaagup Kippar

Tallinn, 2009

Sisukord

Sissejuhatus

Veeb tänapäevasel kujul asus jõudsalt levima 1990ndate aastate keskel. Suurelt jaolt kasutati seda küll staatiliste tekstide ja piltide mugavaks välja näitamiseks, kuid juba algusest peale olid kasutusel juures lisad, mis võimaldasid lehe sisu vastavalt kasutajale või tema tegevusele eraldi näidata. Levinumate näitena selle kohta kehtiva kuupäeva näitamine või otsing soovitud andmete alusel. Sellisel juhul ei piisa paljast valmistehtud lehe kopeerimisest kasutajale, vaid tuleb lehe sisu tekitada või ühendada mõne programmi võimaluste abil.

Vahendeid selleks on aegade jooksul olnud mitmesuguseid. Siiani kasutusel on võimalus käivitada suvalises meeldivas keeles koostatud programm veebiserveris ning lasta sel suhelda brauserist tulnud päringuga vastava liidese (CGI – Common Gateway Interface) abil. Keskkonnamuutujate kaudu saadakse kätte edastatavad andmed (näiteks otsisõna) ning programmi trükitav väljund suunatakse brauseri poole teele – tehniliselt nõnda lihtne see ongi. Iga programmeerija võis endiselt kirjutada omale sobivas keeles ning kõik toimis.

Veebisisendi ja -väljundiga programmidel on aga mõned eripärad. Olenevalt rakendusest, kuid küllalt sageli tuleb väljastada suures koguses muutumatut HTML-teksti ning sinna vahele vaid üksikud kohad, mis programmiga muuta vaja. Teiseks probleemiks veebirakenduste juures on, et kunagi ei saa usaldada sisendit kasutajalt – üle veebi võib tulla ligi suvaline häkker ning otsisõna asemele panna teele näiteks mõne videofilmi sisu binaarkujul. Sekelduste vältimiseks on seetõttu kasulik lisada sisendile piiranguid ja kontrolle.

Veebi leviku laienedes lisandus ka raamistikke, keeli ja keeletäiendusi, mille abil peaks veebiprogramme olema mugavam kokku panna kui ”tavaliste” programmeerimiskeelte abil. Suure veebileviku osaliseks sai keel nimega PERL, millel võrrelduna näiteks tol ajal muidu väga levinud C-ga olid tunduvalt mugavamad ja mitmekülgsemad vahendid tekstidega ümber käimiseks. Tuntumad eraldi veebi jaoks loodud vahendid ehk ASP (nüüdseks põhjalikult muudetuna ASP.NET), Zope, Java Servletid ja JSPd. Lihtsamate ja ka keskmiselt keerukate veebilehestike juures sai valitsevaks keeleks PHP.

PHP tutvustus

Algselt ühe koolipoisi katsetustest levima hakanud kodulehe koostamise abivahend (Personal Home Page) sai oma lihtsuse ja vabade kasutusõiguste tõttu üllatavalt populaarseks. Eks lihtsusega käivad koos ka mõned ohud, mida on uuemates versioonides püütud lappida. Kuid võlu, et kõik kohe arusaadav ja kasutatav on, paneb paljudki programmeerijad heal meelel PHPst alustama.

Päris algus

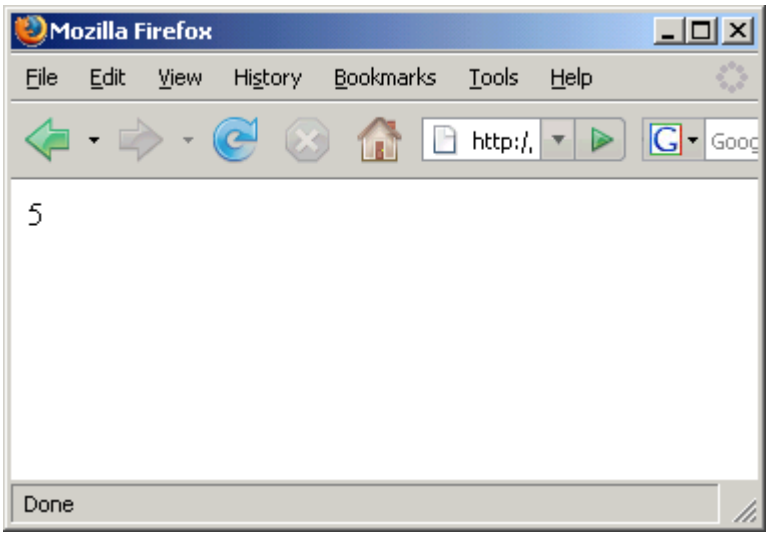
PHP-leht on tavaline teksti (või siis üldjuhul HTML-leht), kus sobivate märkide vahel saab programmikoodi käima lükata – nõnda nagu enamiku muudegi veebiprogrammeerimissüsteemide puhul. Lihtsaim demo näeb välja järgmine:

PHP-võimelise veebiserveri kaudu välja kuvatavasse kataloogi tuleb paigutada järgneva sisuga fail.

```
<?= 3+2 ?>
```

Laiendiks php – näiteks nimega algus.php

Kui nüüd leht veebilehitsejas avada, võiks seal ilutseda üks ilus suur number 5. Kolm ja kaks liideti kokku.



Väike seletus ka juurde: <?= näitab, et nüüd järgneb avaldis, mille väärtus enne kasutajale saatmist kokku arvutatakse. Avaldise lõppu tähistab ?>. Ning vahepealne 3+2 lihtsalt arvutatigi kokku ja trükiti välja. Kui lehel alguses või lõpus oleks veel muudki teksti, trükitaks ka see välja.

Ülesandeid

* Hangi või tee selgeks enesele võimalus PHP-võimelises veebiserveris veebilehtede loomiseks.
* Koosta tervitav leht ja vaata seda veebiserveri kaudu
* Muuda lehe sisu ning uuendusnupu vajutuse järel veendu muutuse kajastumises ka veebilehitsejas.
* Käivita konspektis olnud näide kahe arvu liitmise kohta.

* Muuda arve ja tehet, kontrolli tulemusi.

Muutuja, valik ja kordus

Järgnevalt juba veidi pikem koodilõik. Kui ei piirduta vaid ühe arvu või lause väljastusega, siis tuleb ploki alguseks märkida <?= asemel <?php. Siis võib kuni ?> -ga tähistatud ploki lõpuni rahulikult programmikoodi kirjutada -.see pannakse käima ning tulemus saadetakse veebilehistsejasse.

Programmeerimiskeeltes on levinud võimalus andmeid muutuja ehk märksõna alla meelde jätta. PHPs algavad muutujate nimed dollarimärgiga. See võimaldab neid hiljem vabamalt teksti sisse panna. Lõik

```
$eesnimi="Juku";
echo "Tere, $eesnimi!";
```

trükib aimatavalt välja ”Tere, Juku”.

Valiku jaoks on käsklus if. Tingimus pannakse ümarsulgude sisse. Kui tingimus vastab tõele (praegusel juhul vanus on väiksem kui seitse), siis täidetakse järgnevate looksulgude vahele paigutatud plokk.

```
$vanus=5;
if($vanus<7){
   echo "Oled noor!";
}
```

Korduse ehk silmuse ehk tsükli puhul võidakse looksulgude vahele kirjutatud toimetus ette võtta korduvalt. Järgnev näide teatab viis korda Kuku!

```
for($i=0; $i<5; $i++){
  echo "Kuku!";
}
```

Seletus ka juurde. Muutuja `$i` (PHPs algavad kõik muutujad dollarimärgiga) aitab meeles pidada, mitmenda korra juures ollakse. Käsu for ümarsulgude sees on kolm semikoolonitega eraldatud tsooni. Esimeses neist on algväärtustus, täidetakse üks kord, kohe `for`-ini jõudmise juures. Tüüpiliselt antakse siin loendurile algväärtus, praegusel juhul i-le 0.

Keskmises tsoonis kontrollitakse tingimuse kaudu, et kas on põhjust `for`-ile järgnevate looksulgude vahel olevaid käske täitma hakata. Kui tingimus on tõene siis jah, muidu mitte. Võib ka juhtuda, et tingimus on juba esimesel korral väär - sellisel juhul ei täideta tsükli keha ühtegi korda.

Viimasesse ehk kolmandasse tsooni paigutatakse tsüklis edasiliikumise tegevus(ed) - siin juhul suurendatakse `$i` väärtust, et järgmisel korral oleks juba suurem arv loenduriga võrrelda. Lugemisega alustatakse tüüpiliselt nullist - siis hiljem massivide puhul kergem toimetada.

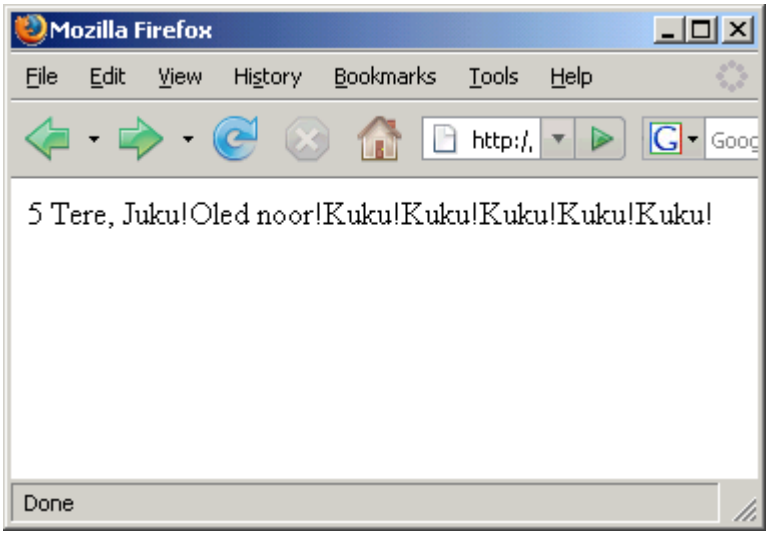
```
<?php
  $eesnimi="Juku";
  echo "Tere, $eesnimi!";

  $vanus=5;
  if($vanus<7){
     echo "Oled noor!";
```

```
  }

  for($i=0; $i<5; $i++){
    echo "Kuku!";
  }
?>
```

Koodilõigu loodud väljund järgmine:



Ülesandeid

* Käivita nähtud näide, muuda andmeid, jälgi tulemusi
* Lisa tervitatavate inimeste eesnimesid, igaüks oma muutujas
* Lisa tingimus üle saja-aastaste jaoks teatega "oled väga vana".
* Lisa for-tsükli sisse iga Kuku järjekorranumber.

PHP HTMLi sees

Nagu näha, võib PHP ka lihtsat teksti väljastada. Kui aga tahta kokku panna viisaka kujundusega veebilehte kus ka kasutaja midagi sisestada saab, siis tuleb HTMLi reeglitega arvestama hakata. Viisakasti tasub `<!DOCTYPE` abil ära määrata HTMLi versioon ning edasi juba kõik elemendid omadel kohtadel: väline element <html>, tema sees peamiste suurte plokkidena <head> ja <body>. Esimesse neist tulevad enamikus metaandmed ehk siis andmed dokumendi kohta, <body> sisse nähtav tekst ise.

Lihtsaim asjalik lõik PHPd HTML koodi sees võiks välja näha ehk järgmine:

```
 <?php
  echo "Kell on: ".date("H:i:s");
  ?>
```

Nagu aimata võite, võib selle tulemusena näha veebiserveri kella aega veebilehel kliendi masinas. Tekst "Kell on " tuleb välja jutumärkide vahelt. Järgnev punkt on operaator tekstide liitmiseks (sidurdamiseks). Ning käsklus date võimaldab kuupäeva ja kellaaja väljastada ettemääratud kujul. Siin näites kasutatud H tähendab tunde 24 tunni süsteemis, i minuteid (sest m ehk month on kuude jaoks) ning s sekundeid. Koolonid trükitakse nende vahele niisama välja. Leht tervikuna siis:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
 <head>
     <title>PHP katsetused</title>
     <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
 </head>
 <body>
   <h1>PHP katsetused</h1>
   <?php
      echo "Kell on: ".date("H:i:s");
   ?>
 </body>
</html>
```

Sisestusega veebileht

Lehele sisestuselementide lisamiseks on sinna kõigepealt vaja panna kujunduse mõttes nähtamatu plokk nimega form. Parameeter action näitab, millisele lehele saadetakse elementidesse kirjutatud andmed. Praeguses näites on fail ise nimega "teine.php" ning action saadab andmed ka faili "teine.php" ehk siis failile iseenesele. Kui andmed kirjutatakse väljadesse ja vajutatakse submit-nuppu, siis pannakse nad praegusel juhul kaasa avatava faili aadressiribale -

```
   <form action="teine.php">
      Eesnimi: <input type="text" name="eesnimi" />
      Vanus: <input type="text" name="vanus" />
      <input type="submit" value="Sisesta" />
   </form>
```

Järgmisel avamisel võib näha aadressiribal failinime näiteks kujul

```
teine.php?eesnimi=Juku&vanus=7
```

Järgmine samm on vormi kaudu aadressiribale jõudnud andmed kätte saada ning nendega midagi ette võtta. Andmete püüdmiseks sobib massiiv nimega $_REQUEST (teadjamatele: sinna jõuavad kokku andmed massiividest $_GET ja $_POST). Sisestatud tegelase lihtsaks tervitamiseks sobib rida

```
echo "Tere, $_REQUEST[eesnimi]";
```

Selle juures on aga probleemiks, et tervitada püütakse ka juhul, kui nime andmeid tegelikult ei ole. Sel juhul ilmuks ekraanile paljas tere koos komaga.

Kui me pole kindlad, kas andmed saabuvad või mitte, siis on viisakas nende olemasolu enne kasutamist kontrollida. Kusjuures on kaks täiesti erinevat andmete puudumise juhtu. Ühel puhul lihtsalt leht avati niisama, sisestades aadress aadressiribale. Teine puudumise puhk on juhul, kui lehele küll sisenetakse vormi kaudu andmeid sisestades ja submit-nupule vajutades, kuid tekstivälja väärtus jäeti tühjaks. Aadressiriba tekib siis kujul näiteks

```
teine.php?eesnimi=&vanus=7
```

Elemendi olemasolu kontrollib PHPs funktsioon nimega isSet, väljastades tõeväärtuse true/false. Näiteks

```
if(isSet($_REQUEST["eesnimi"])){...}
```

Sisestatud teksti olemasoluks on aga hea moodus kindlaks teha selle teksti pikkus käsuga strlen. Kui tekst juhtub tühi olema, siis lihtsalt on selle pikkus 0. Nõnda saabki kokku koodilõigu, mis esimesel avamisel ei ütle midagi. Nimelahtri tühjaksjätmisel aga teatab nime puudumisest.

```
if(isSet($_REQUEST["eesnimi"])){
   if(strlen($_REQUEST["eesnimi"])>0){
     echo "Tere, $_REQUEST[eesnimi]!";
   } else {
     echo "Nimi kirjutamata!";
   }
}
```

Eks arvude puhul saab olemasolu samamoodi kontrollida. Esialgu on ka vanus arvuti jaoks lihtsalt tekst ehk sümbolid. Seepärast ka sisestuslahter sai kirja

```
Vanus: <input type="text" name="vanus" />
```

Kui tahta kontrollida, et sinna lahtrisse saaks kirjutada vaid numbreid, siis tuleks selleks eraldi Javaskripti lõik kirjutada või mõne raamistiku abil sisse panna.

Kui serveris aga tahta saabunud arvuga viisakalt ümber käima hakata, siis on viisakas saabunud tekst mälus arvulisele kujule ümber muundada. Täisarvude puhul aitab seda teha funktsioon nimega intval. Edasi võib veebist tulnud arvuga käituda juba sarnaselt nagu iga muu arvuga. Siin siis tervitame nime sisestanut nõnda palju kordi, palju tal aastaid on.

```
if(strlen($_REQUEST["vanus"])>0){
  $v=intval($_REQUEST["vanus"]);
  for($i=0; $i<$v; $i++){
    echo "&Otilde;nne! ";
  }
}
```

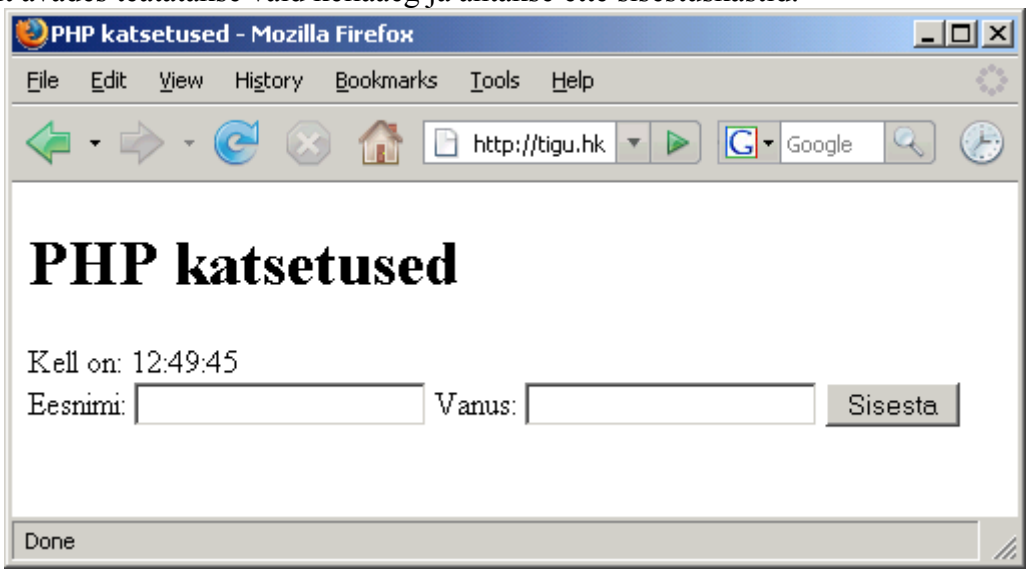
Veebisisestuse üle ei saa aga kunagi kindel olla: kui juhtub, et mõni katsetaja kirjutab oma vanuseks miljoni, siis see koodijupp tervitab teda rõõmsalt miljon korda, kulutades vastavalt serveri aega. Selline tegevus pole küll serveri andmetele ohtlik, kuid kui häkkerid tahaksid hakata masinat kõvasti koormama, siis paarikümnest kohast pidevalt end miljoni kaupa tervitada laskmine on hea moodus serveri kiusamiseks. Kõike selliseid võimalusi ei jõua ega saagi kinni panna. Aga mõnigikord peab mõtlema, et kas, mida ja kui palju on põhjust turvata, ehk siis kiusajatele takistusi teha.

Nime ja vanuse järgi tervitav ja õnnitlev leht siis järgmine.

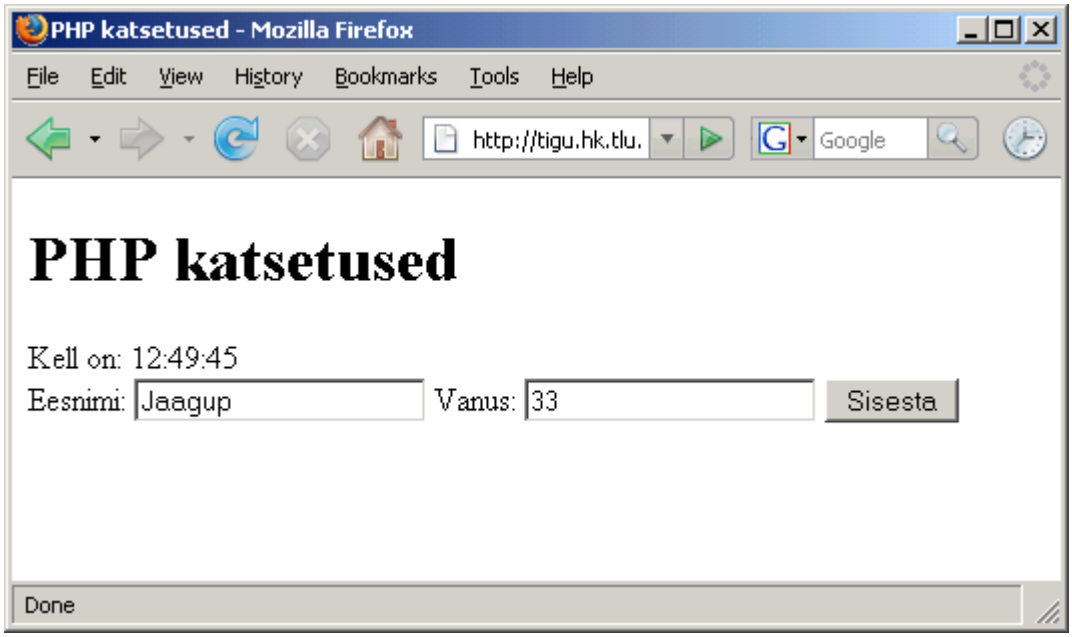
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
 <head>
     <title>PHP katsetused</title>
     <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
 </head>
 <body>
   <h1>PHP katsetused</h1>
   <?php
      if(isSet($_REQUEST["eesnimi"])){
         if(strlen($_REQUEST["eesnimi"])>0){
           echo "Tere, $_REQUEST[eesnimi]!";
         } else {
           echo "Nimi kirjutamata!";
         }
      }
      echo "Kell on: ".date("H:i:s");
   ?>
   <br />
```

```
    <?php
      if(strlen($_REQUEST["vanus"])>0){
        $v=intval($_REQUEST["vanus"]);
        for($i=0; $i<$v; $i++){
          echo "&Otilde;nne! ";
        }
      }
    ?>
    <form action="teine.php">
       Eesnimi: <input type="text" name="eesnimi" />
       Vanus: <input type="text" name="vanus" />
       <input type="submit" value="Sisesta" />
    </form>
  </body>
</html>
```

Kõigepealt avades teatatakse vaid kellaaeg ja antakse ette sisestuskastid.



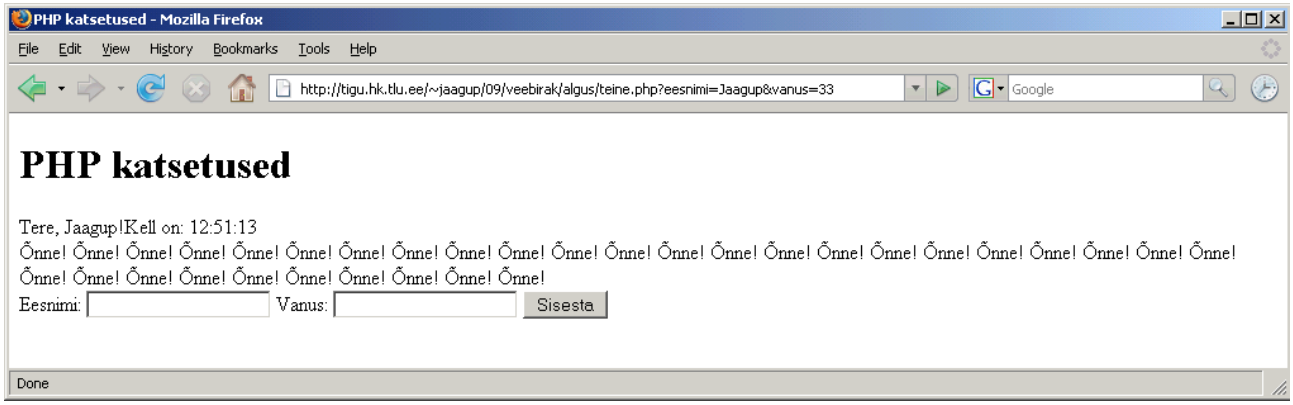
Sinna kannatab andmed sisse kirjutada.



Edasi jõuavad need andmed aadressiriba kaudu faili juurde kuhu form-i action suunab. Ning selle lehe ülesandeks on juba saabunud andmetele vastav sisu kuvada.

Ülesandeid

* Lase kasutajal sisestada kaks arvu. Programm väljastab nende korrutise
* Trüki kasutaja määratud ridade ja veergude arvuga korrutustabel



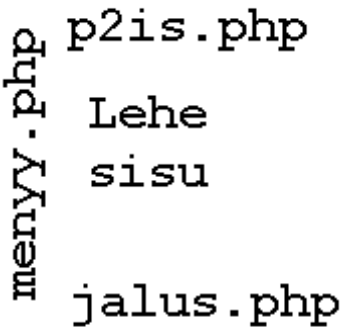
Lehe koostamine alamosadest

Aastate eest oli tähtsaks näitajaks serveri või teenusepakkuja juures, et kas too toetab SSI-d ehk Server Side Include't. Nüüd saab sama tulemuse PHP käskudega mugavasti kätte.

Paljude sarnase kujundusega lehtede juures on paratamatult hulk korduvat koodi - soovitakse samasse kohta paigutada logo, menüü ja ehk muudki. Üheks võimaluseks on algul blanketileht teha, seal tähtsamad kujunduseasjad ära määrata ning seejärel blanketist sobiv hulk koopiaid teha. Kui lehestik nõnda ühekordselt kokku pannakse, siis selline lähenemine täiesti sobib. Samuti aitavad mõned keskkonnad (nt. FrontPage) selliseid sarnaseid lehti genereerida.

Kui aga tehakse lehti lihtsa HTML-redaktoriga, samuti kui tahetakse lehe koodi ise kontrollida ning sinna hiljem programmilõikegi sisse panna, sellisel juhul on sisseloetavad alamosad igati head abilised.

Näitena koostame bussiliinide andmeid avaldava lehestiku, kus muutuvad ainult liini andmed. Lehe päis, jalus ja menüü jäävad ikka samadesse kohtadesse. Alamosad loetakse sisse järgneval joonisel näha olevatest failidest:



Päiseosa saab failist p2is.php. Sinna tuleb kogu HTMLi algusots: DOCTYPE, head ja title ning lehe keha algus.

p2is.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
 <head>
      <title>Bussiliinid</title>
      <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
      <style type="text/css">
        @import "kujundus.css";
      </style>
 </head>
 <body>
 <?php require("menyy.php"); ?>
```

Samuti loetakse sinna juurde sisse menüü failist menyy.php.

menyy.php

```
<div id="menyy">
  <h2>Men&uuml;&uuml;</h2>
  <ul>
     <li><a href="726.php">726</a></li>
     <li><a href="727.php">727</a></li>
  </ul>
</div>
```

Menüü on hea eraldi kohas hoida. Siis on sinna mugav lisada uusi ning eemaldada vanu lehekülgi. Samuti võib hiljem vajadusel paigutada menüükihi lehe koodis mõnda muusse, hetkel sobivamasse kohta.

Ka kujundus on paigutatud eraldi faili. Ehkki siinse sisselugemise teel saaks kujunduseosa suhteliselt mugavalt panna ka päisefaili, on vähegi pikema kujunduslõigu puhul see mõttekas panna omaette faili. Päisefaili kaudu lugedes arvatakse see kujunduslõik iga kord eraldi faili osaks olema ning sikutatakse kogupikkuses serverist kohale. Kui aga viidatakse eraldi css-failile, siis piisab selle ühe korra kohale tõmbamisest, ülejäänud osa ajast saab puhvris olevaid andmeid kasutada.

Kujundusefailis määratakse ära lehel asuvate plokkide asukohad. Käsklus float paneb kihi lehe peal "ujuma". Kuna nii menüükihil kui ka sisukihil on omadus float:left, siis nad ujuvad üksteisel sabas. Menüü on nõnda lai, nagu teksti laius teda lükkab, sisukihil on praegu määratud laiuseks 70% akna suurusest, et ta kasvaks ja kahaneks koos aknaga. Lõputeate kihi clear:left ütleb, et tuleb taas uuelt realt oma paiknemist alustada, nii jõuab ta ilusti eelmiste elementide alla.

kujundus.css

```
body{
  background-color: #ffeb90;
}

#menyy{
 float: left;
 padding-right: 30px;
}

#sisu{
 width: 70%;
 float: left;
}

#loputeade{
 clear: left;
}
```

Jalus praeguse seisuga lihtne fail, kus lõputeade ning HTML-lehe ots.

jalus.php

```
    <div id="loputeade">Lehe koostas Jaagup</div>
  </body>
</html>
```

Sisuleht

Kui abitükid olemas, siis saab asuda sisulehti koostama. Nemad loevad enesele require-käsuga sisse ette päise ja taha jaluse. Päis haarab enese külge veel kaasa menüü. Sisule ka omaette väike kiht paigutamiseks ümber ning võibki rahumeeli vajalikud andmed sinna sisse kirjutada.

```
<?php require("p2is.php"); ?>
  <div id="sisu">
    <h2> 726 HAAPSALU-TAEBLA-PALIVERE-RISTI-LAITSE-TALLINN</h2>

<pre>
12:00              HAAPSALU
12:05        UUEMÕISA (LÄÄNEMAA)
12:10        RANNAKÜLA
12:15        TAEBLA
12:17        PRIGULDI
12:19        VÕNTKÜLA
12:23        PALIVERE EIK
12:25        PALIVERE
12:30        JAAKNA
12:35        RISTI (LÄÄNEMAA)
12:38        REHEMÄE
12:45        ELLAMAA
12:50        TURBA (HARJUMAA)
```

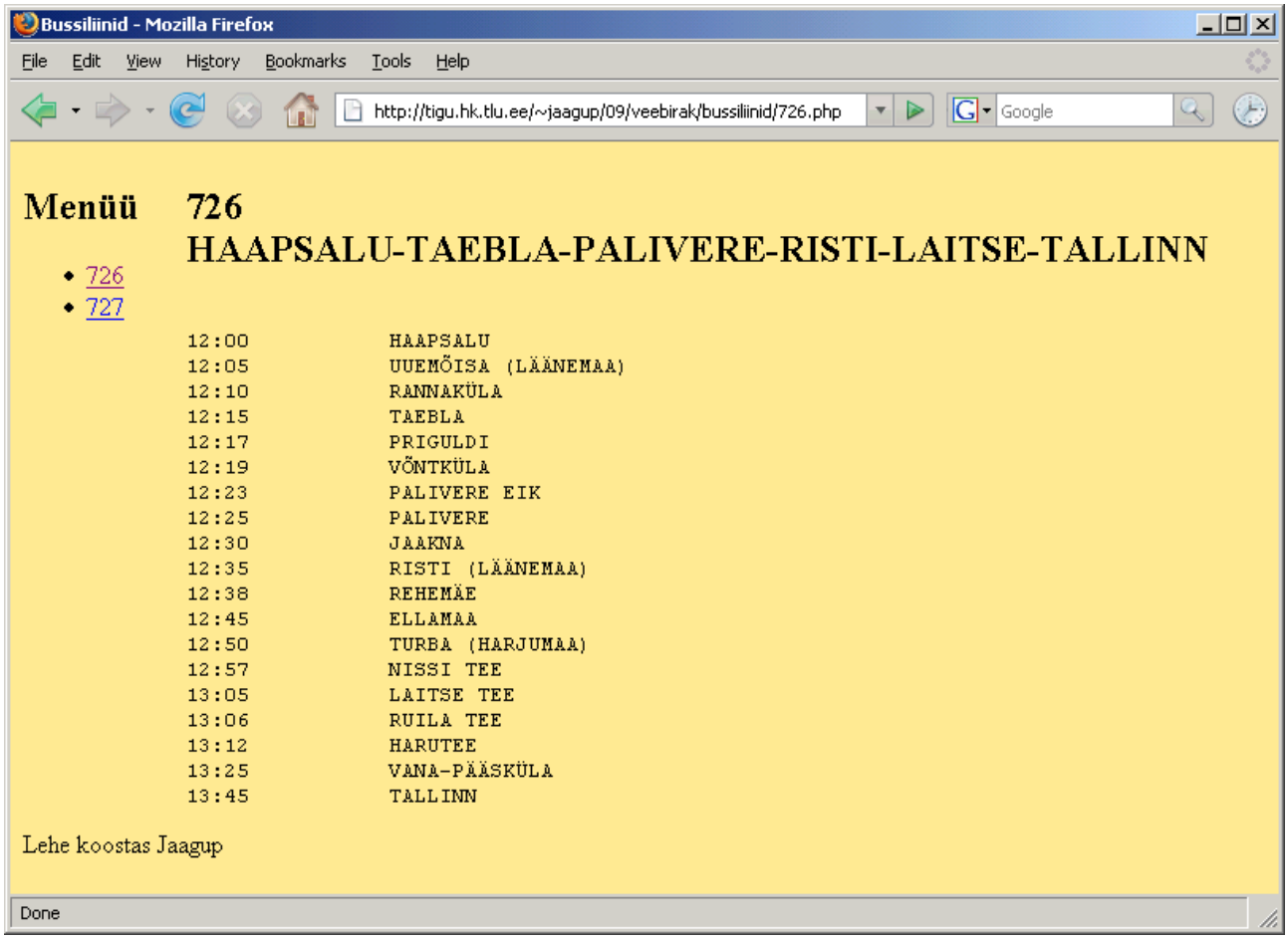
```
12:57        NISSI TEE
13:05        LAITSE TEE
13:06        RUILA TEE
13:12        HARUTEE
13:25        VANA-PÄÄSKÜLA
13:45        TALLINN
</pre>
  </div>
<?php require("jalus.php"); ?>
```

Kaks kõrvutist lehte erinevad vaid sisu poolest - siin siis teise bussiliini ajad ja peatused. Päis ning jalus võetakse külge samasugustena. Käsklus require erineb mõnel pool levinumast include-st selle poolest, et require annab otsitava faili puudumisel veateate, include jätab selle lihtsalt näitamata. Tegemise juures vigade vältimiseks on esimene variant kindlam.

```
<?php require("p2is.php"); ?>
  <div id="sisu">
    <h2>Liin 727, HAAPSALU-TAEBLA-PALIVERE-RISTI-TALLINN</h2>

<pre>
13:00              HAAPSALU
13:05        UUEMÕISA (LÄÄNEMAA)
13:10        RANNAKÜLA
13:15        TAEBLA
13:17        PRIGULDI
13:19        VÕNTKÜLA
13:23        PALIVERE EIK
13:25        PALIVERE
13:30        JAAKNA
13:35        RISTI (LÄÄNEMAA)
13:38        REHEMÄE
13:45        ELLAMAA
13:50        TURBA (HARJUMAA)
13:57        NISSI TEE
14:12        HARUTEE
14:25        VANA-PÄÄSKÜLA
14:45        TALLINN
</pre>
  </div>
<?php require("jalus.php"); ?>
```

Veebilehitsejas tuleb avada sisuleht. Viimane haarab omale ette ja taha vajalikud tükid külge ning võibki rahumeeli sõiduplaani vaadata.



Ülesandeid

* Tee näited läbi
* Lisa kolmanda bussiliini andmed

* Koosta sarnasel moel väike lehestik suvise matka korralduse tarbeks: osalejad, varustus, marsruut.

Andmefailid eraldi kataloogis

Kuni kümnekonna harva muudetava faili puhul on eelmine lähenemine mugav küllalt. Andmete lisamist ja eemaldamist saab aga ka mugavamaks teha. Järgnevas näites korraldati nõnda, et liini andmete näitamiseks piisab vaid sobiva nimega faili paigutamisest selle jaoks ette nähtud kataloogi. Ülejäänuga saab PHP-leht juba ise hakkama.

Lehe sisu näitamiseks tuli juurde fail liinivaataja.php. Temale antakse aadressiriba kaudu ette näidatava liini number. Liinivaataja haarab ette päise, taha jaluse ning liiniandmete kataloogist keskele sisse etteantud numbriga algava tekstifaili sisu. Eeldatakse, et failinimed on kujul 726.txt, 727.txt jne. Samuti eeldatakse, et faili nimi enne laiendit on number - sissemurdmise vastaseks kaitseks töödeldakse saabunud parameetrit `$_REQUEST["liininr"]` käsuga intval, mis muudab iga talle antud teksti arvuks. Arvud jäävad ikka pärast .txt-ga liitmist samasuguseks tekstiks. Kui aga

parameetriks antakse midagi muud, siis annab intval sellele tulemuseks 0. Nõnda pole karta, et keegi võiks failinime sisse paigutada kataloogide vahel liikumise või muid andmete õngitsemiseks tarvilikke käske.

Käsklus file_get_contents tagastab parameetriga ette antud faili sisu. @-märk käsu eel annab teada, et tekkivaid veateateid ei kuvataks ekraanile - need jällegi asjad, mis kipuvad häkkeritele masina ülesehituse kohta teavet andma ning sellest tasub turvalisuse huvides hoiduda.

liinivaataja.php

```
<?php require("p2is.php"); ?>
<div id="sisu">
 <pre><?php
     if(isSet($_REQUEST["liininr"])){
        $sisu=@file_get_contents("liiniandmed/".
                      intval($_REQUEST["liininr"]).".txt");
        if($sisu){
          echo $sisu;
        } else {
          echo "Andmed puuduvad";
        }
     } else {
       echo "Liininumber puudub";
     }
   ?>
 </pre>
</div>
<?php require("jalus.php"); ?>
```

Päis endiselt samasugune

p2is.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
 <head>
      <title>Bussiliinid</title>
      <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
      <style type="text/css">
        @import "kujundus.css";
      </style>
 </head>
 <body>
 <?php require("menyy.php"); ?>
```

Menüü skanneerib läbi kõik liiniandmete kaustas leiduvad failinimed. PHP 5st alates kättesaadav funktsioon `scandir` tagastab etteantud kataloogis leiduvatest failinimedest koosneva massiivi. foreach-tsükliga võetakse sealsed nimed ükshaaval ette, käsklus explode tükeldab failinime punkti koha pealt massiiviks nii, et ühe punkti korral tekib kaheelemendiline massiiv. Kohal 0 on nimi ise ning kohal 1 on laiend. Kontrollitakse üle, et kui faililaiend ikka on .txt, siis lisatakse menüüloetellu vastava faili nimi - pannes see ka ühtlasi viitega kaasa saadetavaks parameetriks.

menyy.php

```
<div id="menyy">
  <h2>Men&uuml;&uuml;</h2>
  <ul>
   <?php
    $failinimed=scandir("liiniandmed");
    foreach($failinimed as $failinimi){
      $m=explode(".", $failinimi);
      if($m[1]=="txt"){
        echo "<li><a href='liinivaataja.php?liininr=$m[0]'>$m[0]</a></li>";
      }
    }
   ?>
  </ul>
</div>
```

Jalus endiselt lihtne.

jalus.php

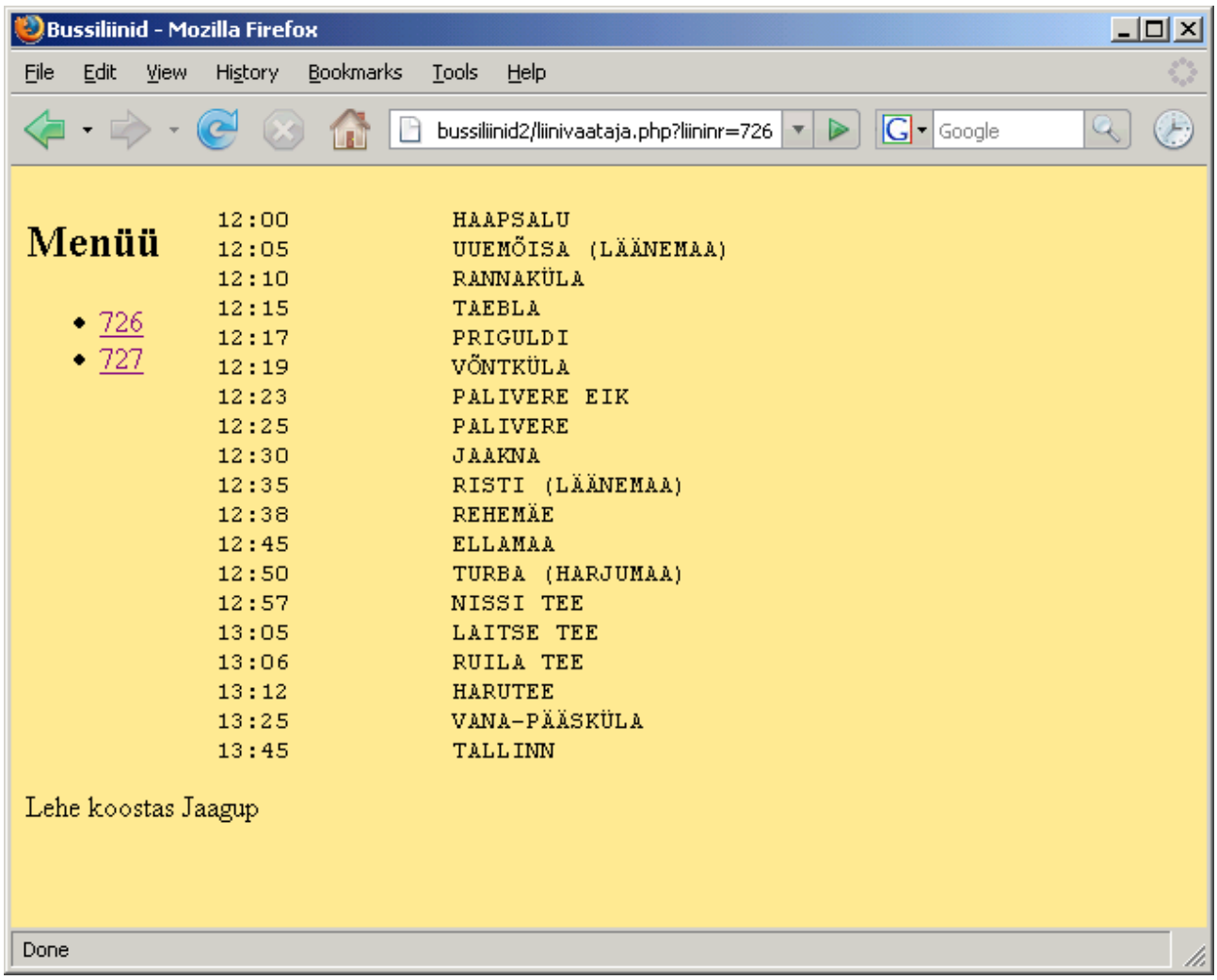
```
    <div id="loputeade">Lehe koostas Jaagup</div>
  </body>
</html>
```

Nüüd piisab liiniandmete faili juures vaid andmetest endist, päist ja muid käsklusi pole lisada enam vaja, need tulevad liinivaataja.php juurest.

liiniandmed/726.txt

```
12:00              HAAPSALU
12:05        UUEMÕISA (LÄÄNEMAA)
12:10        RANNAKÜLA
12:15        TAEBLA
12:17        PRIGULDI
12:19        VÕNTKÜLA
12:23        PALIVERE EIK
12:25        PALIVERE
12:30        JAAKNA
12:35        RISTI (LÄÄNEMAA)
12:38        REHEMÄE
12:45        ELLAMAA
12:50        TURBA (HARJUMAA)
12:57        NISSI TEE
13:05        LAITSE TEE
13:06        RUILA TEE
13:12        HARUTEE
13:25        VANA-PÄÄSKÜLA
13:45        TALLINN
```

Ja jällegi võib ilusti omale sobivad sõiduajad valida :-)



Ülesandeid

* Tee näited läbi
* Koosta sarnaselt laulusõnade lehestik: iga laul omaette lehel. Lihtsamal juhul on laulul pealkirjaks number. Keerukamal juhul on failinimi vabam, aga tasub kontrollida, et erisümboleid sisse ei jääks.
* Koosta eelmise näite abil pildigalerii. Kataloogis olevaid pilte saab veebi kaudu vaadata.
* Jaga galerii alateemadeks, ehk siis alamkataloogideks. Koosta navigeerimisvahendid lehtede vahel liikumiseks.

Andmebaas veebilehestiku juures

Nagu eelnenud näidetest näha, pole veebi koostamisel andmebaas sugugi hädavajalik lisandus. Küll aga võimaldab veebilehe andmebaasitabeliga ühendamine mitmete toimetustega mugavamal ja/või ohutumalt hakkama saada. Hulk võimalusi, mis muidu tuleks ise kodeerida, on juba andmebaasiprogrammidesse sisse ehitatud. Tüüpilised kohad, kus andmebaaside eelised välja tulevad on järgmised:

* samas veebilehestikus kasutatavad mitmesuguse struktuuriga seotud andmed
* otsing parameetrite järgi
* andmete muutmine veebi kaudu - eriti mitme administraatori korral

Eelnevalt nähtud suhteliselt staatilised lehed seega, kus kasutatakse lihtsalt ühiseid päiseid ning sarnase struktuuriga pikematekstilisi andmeid - selliste lehtede puhul pole andmebaas sugugi hädavajalik.

Kui aga soovida veebi kaudu muudetavat/täiendatavat lehestikku teha, siis on andmebaasi abi kindlasti omal kohal.

Enne näite juurde minemist märgime siia edaspidi kasutatavad mõisted.

* Andmebaasiprogramm, andmeohjur - masinasse installeeritud programm või draiver andmebaasidega ümber käimiseks. Nt. MySQL, Oracle, PostgreSQL, Access.

* Andmebaas - andmeohjuri poolt hallatav tabelite komplekt. Nt. konkreetse veebipoe jaoks tarvilikud andmetabelid. Ühes masinas võib olla (enamasti ongi) mitu andmebaasi. Igas andmebaasis saab olla palju andmetabeleid. Ühe andmebaasi piires on eri andmetabelitest andmeid mugavam/kiirem siduda kui eri andmebaaside vahel.

* Andmetabel - ridade ja veergudega andmete hoidmise koht. Igal veerul ehk tulbal on nimi ja tüüp - nt. tekst, kuupäev, täisarv, reaalarv.

* SQL - tõenäoliselt levinuim keel andmebaasile käskude andmiseks. Selle abil saavad andmebaasiga suhelda nii inimesed kui andmebaasivälised rakendusprogrammid (nt. PHP). Inimeste jaoks tehakse vahel ka graafilisi haldusvahendeid, kuid teised programmid suhtlevad üldjuhul andmebaasiga SQLi kaudu.

Siinses õppematerjalis kasutame andmebaasina MySQLi. Ta sobib PHPga hästi kokku, kuna on suunatud enamvähem sarnasele sihtgrupile: lihtsamate veebirakenduste lihtne loomine tasuta tarkvara abil. Eestis vähemasti 2009nda aastani tõenäoliselt levinuim moodus veebirakenduste loomiseks. (Üli)suurtes süsteemides nagu nt. pangalehed jääb vahenditesse sisseehitatud kontrolle ja stuktueeritust väheks. Aga enamike veebirakenduste nagu foorumid, uudistelehed, registreerumisvormid jaoks on võimalusi piisavalt. Arendamine ja ülespanek nõuab riistvaraliselt vähe ressursse, lehtede ülespanekuks kasutatavaid teenusepakkujaid (nt zone.ee, elkdata.ee) on piisavalt. Täiesti korraliku PHP ja MySQLi veebimajutuse leiab 2009nda aasta seisuga alates 100st kroonist kuus.

Ühe andmetabeliga seotud veebilehestik

MySQLis ja ka teistes relatsioonilistes ehk tabelipõhistes andmebaasides hoitakse ja väljastatakse pea kõiki andmeid tabelitena. Ning viimastel aastakümnetel üle ¾ andmebaasidest ongi relatsioonilised.

Andmebaasis toimetamiseks peab kõigepealt sellesse sisenema. Oma arvutisse nt. XAMPP abil veebiloomiskomplekti installides on MySQL kohe kättesaadav. Avalikumas serveris tuleb sisenemiseks enne administraatorilt kasutajakonto paluda. Ning mõnes teenusepakkuja keskkonnas saab ligi vaid graafilise kasutajaliidese (nt. PHP MyAdmin) kaudu.

Käsurealt sisenemine näeb välja ligukaudu järgmine. Võti -ujaagup näitab, et kasutajaks (user) on jaagup. Järgmine jaagup tähistab andmebaasi nime. Ning -p teatab, et parool küsitakse eraldi nõnda, et selle tähti ekraanile ei kuvata. Kui kõik õnneks läheb, siis ilmub lõpuks ette käsuviip mysql >

```
jaagup@tigu:~$ mysql -ujaagup jaagup -p
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 47574
Server version: 5.0.51a-24+lenny2-log (Debian)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Tabeli loomiseks käsklus CREATE TABLE. Käsu nimele järgneb tabeli nimi (praegusel juhul "lehed"). Ning siis sulgudes komadega eraldatuna tulpade nimed ning nende parameetrid. Iga tabeli esimeseks tulbaks on üldjuhul id - identifikaator, mille abil hiljem ridu eristada ja neile viidata. Parameetrid võivad lihtsamate rakenduste puhul enamasti samaks jääda. Selgitused:

INT - täisarv
NOT NULL - väärtus ei tohi puududa
AUTO_INCREMENT - server arvutab lisamisel ise juurde sobiva seni veel kasutamata väärtuse
PRIMARY KEY - selle tulba väärtust kasutatakse edaspidi tabeli vastavale reale viitamisel (näiteks muutmise või kustutamise juures).

Tulp pealkiri siin näites tüübiga VARCHAR(50) ehk siis tekst pikkusega kuni 50 tähte. Sisu tüübiks TEXT, mis tähendab, et pikkust ei piirata.

Kokku siis lause järgmine, mis tasub valmis kirjutada ning MySQLi käsuviibale kopeerida:

```
CREATE TABLE lehed(
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  pealkiri VARCHAR(50),
  sisu TEXT
);
```

Kui vastuseks tuli Query OK, siis järelikult ettevõtmine õnnestus. Muidu tuleb veateateid uurida ja uuesti proovida.

Andmete lisamiseks on loodud käsklus INSERT INTO. Järgneb tabeli nimi, siis sulgudes tulpade nimed kuhu lisatavad andmed tulevad. Edasi sõna VALUES ning sulgude sisse komadega eraldatult igale tulbale vastav väärtus. Tekstilised andmed paigutatakse ülakomade vahele.

```
INSERT INTO lehed (pealkiri, sisu) VALUES ('Ilmateade', 'Kuiv ilm');
```

Tahtes rohkem andmeid lisada, tuleb INSERT lauset lihtsalt mitu korda käivitada, igal korral eraldi andmed sisse pannes.

```
mysql> INSERT INTO lehed (pealkiri, sisu) VALUES ('Korvpall', 'Treening reedel kell 18');
Query OK, 1 row affected (0.00 sec)
```

Kui mõned sees, siis on hea vaadata ja kontrollida, et mis sinna täpsemalt sai. Andmete küsimiseks on SQLis loodud käsklus SELECT. Tärn tähendab, et kuvatakse kõikide olemasolevate tulpade andmed. Sõnale FROM järgneb tabeli nimi ning käsu lõppu käivitamiseks semikoolon. Lehtede tabeli sisu tuleb siis välja järgnevalt.

```
mysql> SELECT * FROM lehed;
```

```
+----+-----------+-------------------------+
| id | pealkiri  | sisu                    |
+----+-----------+-------------------------+
|  1 | Ilmateade | Kuiv ilm                |
|  2 | Korvpall  | Treening reedel kell 18 |
+----+-----------+-------------------------+
2 rows in set (0.00 sec)
```

Nagu näha, id-väärtused on automaatselt ise pandud, kuna vastaval tulbal on juures omadus AUTO_INCREMENT.

Tahtes andmeid veel juurde panna, tuleb taas käivitada INSERT-lause sobivate andmetega. Teksti sisestamisele vastab kõige korrasoleku puhul MySQL taas "Query OK", lisades sinna vahel ka mõjutatud ridade arvu ja kulunud aja - tähtis pigem suuremate andmestike korral.

```
mysql> INSERT INTO lehed (pealkiri, sisu) VALUES ('Matemaatika', 'Homme tunnikontroll');
Query OK, 1 row affected (0.00 sec)
```

SQLi selecti abil saab andmeid kergesti sobivas järjekorras ja kujul välja küsida. Kui lause lõppu lisatakse ORDER BY koos vastava tulba nimega, siis tulevad andmed välja selle tulba järgi tähestiku järjekorda panduna (kui vastav tulp oli tekstitulp).

```
mysql> SELECT * FROM lehed ORDER BY sisu;
+----+-------------+-------------------------+
| id | pealkiri    | sisu                    |
+----+-------------+-------------------------+
|  3 | Matemaatika | Homme tunnikontroll     |
|  1 | Ilmateade   | Kuiv ilm                |
|  2 | Korvpall    | Treening reedel kell 18 |
+----+-------------+-------------------------+
3 rows in set (0.00 sec)
```

Saab küsida ka ainult ühe rea väärtusi:

```
mysql> SELECT pealkiri, sisu FROM lehed WHERE id=3;
+-------------+---------------------+
| pealkiri    | sisu                |
+-------------+---------------------+
| Matemaatika | Homme tunnikontroll |
+-------------+---------------------+
```

Kui leitakse, et rida pole enam vajalik, siis selle kustutamiseks sobib käsklus DELETE, kus soovitavalt id järgi määratakse ära, milline rida kustutada.

```
mysql> DELETE FROM lehed WHERE id=3;
Query OK, 1 row affected (0.00 sec)
```

Uue SELECT-päringuga saab kontrollida, mis siis sinna tegelikult alles jäi. Kui nüüd juhtutaks INSERT-lausega taas andmeid lisama, siis sellele reale enam id väärtust 3 välja ei antaks - välistamaks näiteks olukorda, kus vanale teatele pandud kommentaarid satuksid uue külge. Primaarvõtmetulba id väärtuseks tuleks uue rea lisamisel vähemasti 4.

```
mysql> SELECT * FROM lehed;
+----+-----------+-------------------------+
| id | pealkiri  | sisu                    |
+----+-----------+-------------------------+
|  1 | Ilmateade | Kuiv ilm                |
|  2 | Korvpall  | Treening reedel kell 18 |
+----+-----------+-------------------------+
```

MySQList saab välja käsuga `quit`

Ülesandeid

* Tee näited läbi
* Välju MySQList, sisene uuesti.
* Lisa teadete tabelisse veel mõned read.
* Väljasta teated sorteerituna pealkirja järgi

* Loo tabel "veised" tulpadega id, nimi, mass ja vanus
* Lisa mõned andmed
* Väljasta andmed sordituna vanuse järgi
* Väljasta ühe veise andmed id järgi.
* Kustuta see veis.
* Väljasta veised, kelle mass on 200-500kg (kahe tingimuse vahel AND).

Andmetabeli sisu kuvamine PHP abil.

MySQLi ja PHP ühendamiseks on aegade jooksul kasutatud mitmesuguseid vahendeid. Levinuimad on tõenäoliselt käsud mysql_connect, mysql_select_db ja mysql_query, millede abil on ka suurem osa kasutatavaid rakendusi kirjutatud. Selle komplekti puuduseks on aga sisendandmete suhteliselt vaba sattumine SQLi lausesse, mis teeb rakenduse kergemini haavatavaks. Aastaid on abiks kasutusel olnud eraldi loodud teek nimega PEAR (http://pear.php.net/). PHP 5. versiooniga aga tuli kaasa andmebaasipäringute poolest sarnaseid võimalusi pakkuv pakett MySQL Improved, mille abil ka siinse õppematerjali näited tehakse.

Järgnev PHP-leht kuvab andmetabelis olnud teated ekraanile. MySQL Improved Extensioni (http://ee.php.net/mysqli) võimaluste kasutamiseks tuleb kõigepealt vastav objekt luua.

```
$yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");
```

Edasi tehakse valmis käsklus.

```
$kask=$yhendus->prepare("SELECT id, pealkiri, sisu FROM lehed");
```

Siis määratakse, millistesse muutujatesse satuvad tulpadest tulevad andmed ning käivitatakse käsklus andmebaasiserveris. Muutujatesse pannakse andmed samas järjekorras, kui need tulevad välja SELECT-lausest.

```
$kask->bind_result($id, $pealkiri, $sisu);
$kask->execute();
```

Kui andmed käes, siis võib neid ühe rea kaupa ette võtma hakata. Iga `fetch()` täidab `bind_result`-käsklusega määratud muutujad uue rea andmetega. Tsükkel while jätkab senikaua kuni päringu vastuseks veel ridu on, ehk kuni `$kask->fetch` tagastab tõese väärtuse kursori edasiliikumise õnnestumise kohta.

Veebilehel näitamiseks pannakse pealkirjale ja sisule ümber käsklus htmlspecialchars - see asendab HTML-koodis tekstiosas lubamatud sümbolid (nt. < ja >) vastavate asenduskombinatsioonidega (`<` ja `>` nende märkide puhul).

```
while($kask->fetch()){
  echo "<h2>".htmlspecialchars($pealkiri)."</h2>";
```

```
        echo "<div>".htmlspecialchars($sisu)."</div>";
      }
```

Lõpuks on viisakas ühendus kinni panna. Ilma vastava käsuta küll ühendus suletakse ka mingi aja pärast, kuid jääb siiski mõneks ajaks rippuma ning tiheda kasutusega serveris võib tekkida olukord, kus vabade andmebaasiühenduste limiit saab otsa ja mõnda aega pole võimalik baasipäringuid kasutada. Kui aga ühendus selgesõnaliselt kinni panna, siis on vastav kanal vaba ning selle võib avada uue päringu tarbeks.

```
  $yhendus->close();
```

Tabeli andmeid veebilehel näitav kood tervikuna:

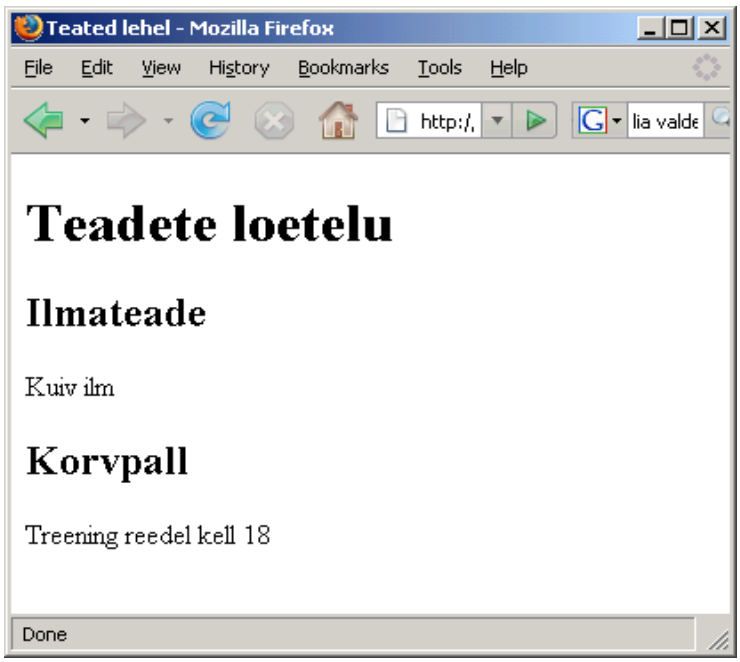
```
<?php
  $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");
  $kask=$yhendus->prepare("SELECT id, pealkiri, sisu FROM lehed");
  $kask->bind_result($id, $pealkiri, $sisu);
  $kask->execute();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
     <title>Teated lehel</title>
  </head>
  <body>
     <h1>Teadete loetelu</h1>
     <?php
       while($kask->fetch()){
         echo "<h2>".htmlspecialchars($pealkiri)."</h2>";
         echo "<div>".htmlspecialchars($sisu)."</div>";
       }
     ?>
  </body>
</html>
<?php
  $yhendus->close();
?>
```

Käivitamise tulemusena valmis HTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
     <title>Teated lehel</title>
  </head>
  <body>
     <h1>Teadete loetelu</h1>
     <h2>Ilmateade</h2><div>Kuiv ilm</div>
<h2>Korvpall</h2><div>Treening reedel kell 18</div>  </body>

</html>
```

Ning selle põhjal järgmise väljanägemisega veebileht:



Ülesandeid

* Hangi omale võimalus MySQL-andmebaasile käsklusi saata - olgu siis PHPMyAdmini, käsurea või mõne muu koha kaudu.

* Koosta tabel koerte andmetega: id automaatselt suurenev primaarivõti, koeranimi kuni 30 sümboli pikkune tekst (VARCHAR) ning koera sünniaasta (INT). Lisa mõned andmed.

* Küsi tabelist koerad kord nime, kord sünniaastate järgi järjestatuna välja-
* Näita vaid koeri, kelle sünniaasta on suurem kui 2000.

* Koosta veebileht, kus on näha kõikide andmetabelis olevate koerte nimed ja sünniaastad.
* Kujunda leht kasutades võimalusel koerte pilte.

Teadete valik

Lühema sisuga teadete puhul sobib kõigi andmete korraga välja näitamine täiesti- vajalikud andmed on hea ülevaatlikult leida. Kui aga teated palju või nad pikemad, siis kipub kõige korraga nägemine tülikas olema. Esmaseks abiks sobib, kui algul paistavad välja vaid pealkirjad ning hiljem nendele vajutades saab ühe teate kaupa vaadata teate sisu teksti.

Selleks tuleb lehele andmete vaatamiseks luua kaks suhteliselt eraldi osa: menüü teadete loetelu jaoks ning keskel olev sisuosa ühe valitud teate näitamiseks.

Menüü kood on suhteliselt sarnane eelmise näite teadete loetelu omale. Lihtsalt loetelus näidatakse vaid teate pealkirja, sisu jäetakse näitamata. Pealkirjad muudetakse viideteks siiasamale failile nimega teadetevalik.php. Aadressiribale pannakse küsimärgi järel kaasa lähemalt vaadata soovitava teate id. Selle järgi on võimalik hiljem teate andmed küsida ning neid kasutada.

```
<ul>
  <?php
     $kask=$yhendus->prepare("SELECT id, pealkiri FROM lehed");
     $kask->bind_result($id, $pealkiri);
     $kask->execute();
     while($kask->fetch()){
       echo "<li>  <a href='teadetevalik.php?id=$id'>".
                    htmlspecialchars($pealkiri)."</a></li>";
     }
  ?>
</ul>
```

Teine plokk on `id` järgi konkreetsete andmete küsimine baasist ning näitamine veebilehele. See võetakse ette vaid juhul, kui aadressiribale on lisatud parameeter nimega "id". SELECT-lausele tuleb selle väärtus `bind_param`-käsu kaudu sisse ajada. Jutumärkide vahel olev "i" näitab, et parameetrina etteantavad id-d loetakse integeri ehk täisarvuna. Muude sisendite puhul jõuab päringulausesse selle koha peale lihtsalt arv 0.

Andmete kätte saamiseks määratakse bind_result-käskluse juures muutujate nimed, kuhu iga fetch-käsuga andmed sisestatakse. Endise `while($kask->fetch)` asemel on siin `if($kask->fetch)`, sest mitut kirjet sama id põhjal nagunii küsida ei saa. Kui aga ühtegi ei anta, siis tõenäoliselt on keegi aadressiribale kirjutanud katsetamise huvides olematu id-numbri ning talle on viisakas teatada, et ta andmed on vigased.

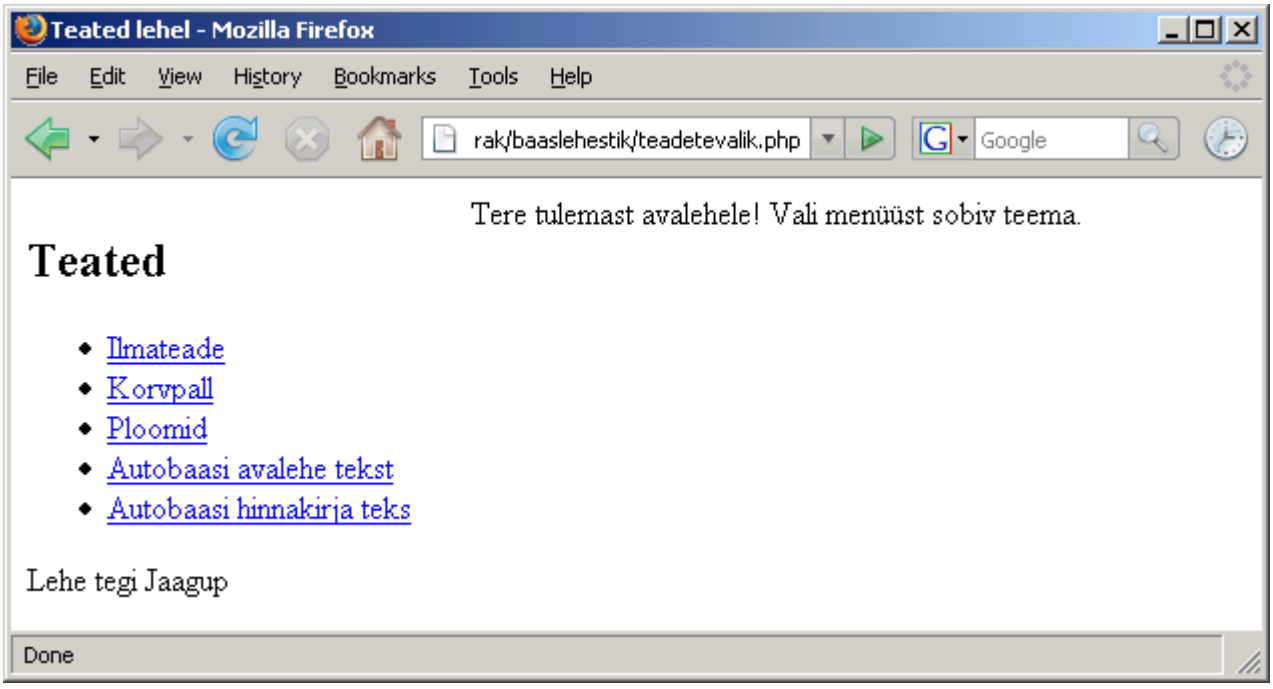
```
<?php
  if(isSet($_REQUEST["id"])){
     $kask=$yhendus->prepare("SELECT id, pealkiri, sisu FROM lehed
        WHERE id=?");
        //Kysim2rgi asemele pannakse aadressiribalt tulnud id,
        //eeldatakse, et ta on tyybist integer (i). (double - d, string - s)
     $kask->bind_param("i", $_REQUEST["id"]);
     $kask->bind_result($id, $pealkiri, $sisu);
     $kask->execute();
     if($kask->fetch()){
       echo "<h2>".htmlspecialchars($pealkiri)."</h2>";
       echo htmlspecialchars($sisu);
     } else {
       echo "Vigased andmed.";
     }
  } else {
    echo "Tere tulemast avalehele! Vali men&uuml;&uuml;st sobiv teema.";
  }
?>
```

Kui `id`-väärtust aadressiribal pole üldse märgitud, siis tõenäoliselt on tegemist lehe esmaavamisega, kus pole veel teate id-d määratud. Viisakuse poolest pannakse avatud ühendus lehe lõpul ka kinni. Kujundust enam eraldi failidesse paigutada pole mõtet, sest kogu andmete näitamise töö teebki ära üks ja seesama fail.

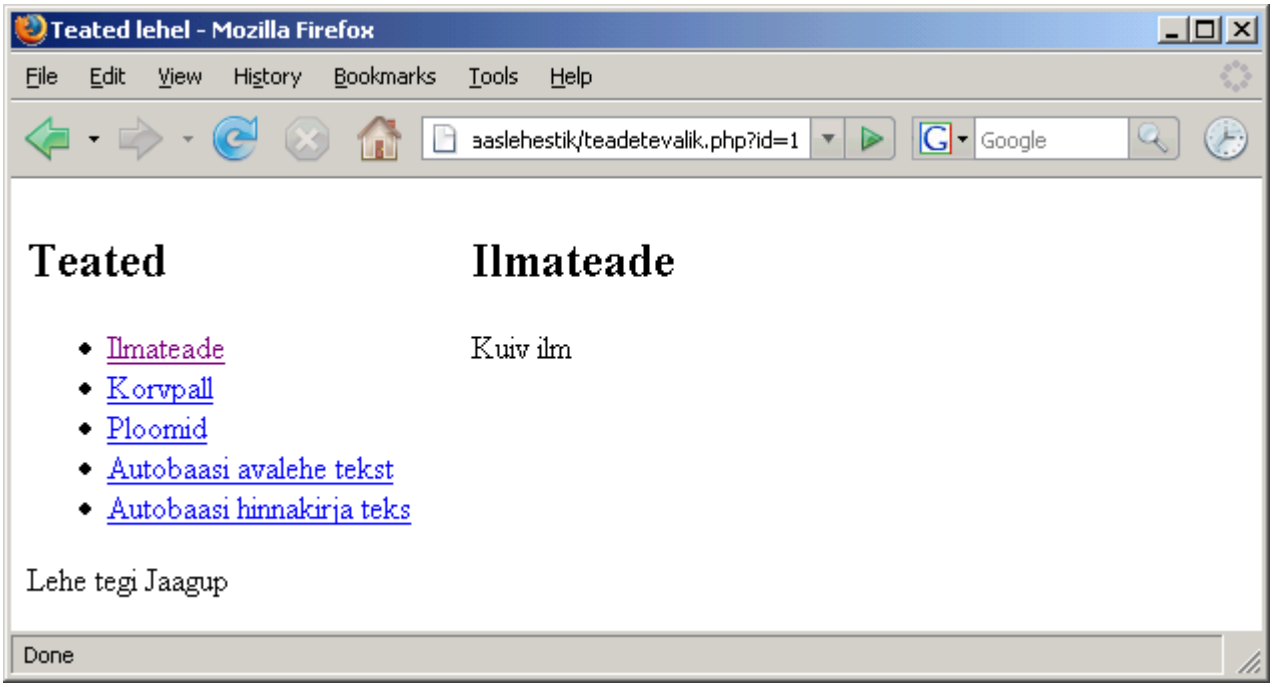
```
<?php
  $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>Teated lehel</title>
    <style type="text/css">
       #menyykiht{
         float: left;
         padding-right: 30px;
```

```
      }
      #sisukiht{
        float:left;
      }
      #jalusekiht{
        clear: left;
      }
    </style>
  </head>
  <body>
    <div id="menyykiht">
        <h2>Teated</h2>
        <ul>
          <?php
             $kask=$yhendus->prepare("SELECT id, pealkiri FROM lehed");
             $kask->bind_result($id, $pealkiri);
             $kask->execute();
             while($kask->fetch()){
               echo "<li><a href='teadetevalik.php?id=$id'>".
                     htmlspecialchars($pealkiri)."</a></li>";
             }
          ?>
        </ul>
    </div>
    <div id="sisukiht">
       <?php
         if(isSet($_REQUEST["id"])){
            $kask=$yhendus->prepare("SELECT id, pealkiri, sisu FROM lehed
               WHERE id=?");
               //Kysim2rgi asemele pannakse aadressiribalt tulnud id,
               //eeldatakse, et ta on tyybist integer (i). (double - d, string - s)
            $kask->bind_param("i", $_REQUEST["id"]);
            $kask->bind_result($id, $pealkiri, $sisu);
            $kask->execute();
            if($kask->fetch()){
              echo "<h2>".htmlspecialchars($pealkiri)."</h2>";
              echo htmlspecialchars($sisu);
            } else {
              echo "Vigased andmed.";
            }
         } else {
           echo "Tere tulemast avalehele! Vali men&uuml;&uuml;st sobiv teema.";
         }
       ?>
    </div>
    <div id="jalusekiht">
       Lehe tegi Jaagup
    </div>
  </body>
</html>
<?php
  $yhendus->close();
?>
```

Avalehel näidatav teatepealkirjade loetelu



Valitud teate andmete kuvamine. Aadressiribal on näha valitud teate id.



Ülesandeid

* Koosta andmetabel koerte andmete hoidmiseks
* Loetelus on näha tabelisse sisestatud koerte nimed
* Koera nimele vajutamisel näidatakse muid andmeid selle koera kohta.

Andmete lisamine ja kustutamine

Baasist tulevaid andmeid on hea ja ilus vaadata, ent kuidagi peavad need andmed baasi saama. Kui andmestik kuigivõrd ei muutu, siis võib ju ka andmebaasi enese haldusliidese abil nad sinna kirja panna. Kui aga tahta, et tavakasutaja saaks mugavasti sättida, mis tal lehe peal kirjas, siis tema hea meelega PHP MyAdmini või otse SQL käsuviiba taha ei lähe. Lihtne arusaadav veebileht andmete sisestamiseks ja muutmiseks sobib palju paremini.

Andmete lisamiseks tuleb kõigepealt toiminguga algust teha: kasutajale antakse pealkirjade menüü lõpus võimalus valida toiming "Lisa ...". Tehniliselt suunatakse kasutaja samale veebilehele, aga andes lehele kaasa parameetri `lisamine=jah`.

```
<a href='<?="$_SERVER[PHP_SELF]?lisamine=jah" ?>'>Lisa ...</a>
```

Selle peale kuvatakse lehel välja sisestusvorm. Vormi kohustuslik parameeter on action, mis näitab lehe aadressi, kuhu andmed jõuavad. Kui kogu haldus toimub sama lehe juures, siis on mugavaks mooduseks kirjutada action'iks `$_SERVER["PHP_SELF"]` - siis pannakse sinna viide lehele enesele. Ning oma juurde saab ligi ka juhul, kui lehe nime vahetatakse. Hilisema toimingu eristamise huvides lisan varjatud väljana (hidden) ka teate uusleht=jah.

```
if(isSet($_REQUEST["lisamine"])){
  ?>
    <form action='<?=$_SERVER["PHP_SELF"] ?>'>
     <input type="hidden" name="uusleht" value="jah" />
     <h2>Uue teate lisamine</h2>
     <dl>
       <dt>Pealkiri:</dt>
       <dd>
        <input type="text" name="pealkiri" />
       </dd>
       <dt>Teate sisu:</dt>
       <dd>
         <textarea rows="20" name="sisu"></textarea>
       </dd>
      </dl>
      <input type="submit" value="sisesta">
    </form>
  <?php
}
```

Selle järgi saab lehe algusosa koodis kontrollida, kas saadeti uued andmed, ehk kas muutuja `$_REQUEST["uusleht"]` on olemas. Tavajuhul seda pole ning plokist minnakse lihtsalt üle. Kui aga lehele saabuti vormist andmeid sisestades, siis varjatud väli andis teada, et sealtkaudu tuldi.

MySQL Improved-laienduse abil tuleb kõigepealt sisestamise SQL-lause kokku panna ning siis sinna andmed sisse paigutada. Pealkiri ja sisu on mõlemad tekstid ehk stringid - sellest ka `bind_param`-käsu parameetrite alguses "ss" ehk kaks stringi. Käsuga `execute` jõuavad andmed baasi.

Iseenesest on siiamaani toimetusega andmed sisestatud. Kui aga kood niimoodi jätta ja pärast `execute`-käsklust muude toimetustega edasi minna, sellisel juhul tekiks refresh-nuppu vajutades üllatus: iga vajutuse korral lisatakse veel kord tabelisse rea jagu samu andmeid. Seda seetõttu, et uuendusnupule vajutades jõuavad aadressireale (või ka post-meetodi kaudu) samad andmed ning nendega tehakse sama toiming ehk andmete lisamine tabelisse.

Soovimatust lisandreast vabanemiseks on üheks võimaluseks suunata lehe näitamine ümber. Päisekäsk `header("Location: failinimi.php")` suunab veebilehitseja edasi nimetatud failile. Kui failinimeks on `$_SERVER["PHP_SELF"]`, siis on tegelikult tegemist sama PHP failiga. Erinevuseks ainult, et kaasa ei anta tabelisse sisestamiseks mõeldud parameetreid. Seda meil aga just vaja ongi.

Pahatihti kiputakse selle päisekäskluse saatmisega piirduma, sest näiliselt oleks just nagu kõik korras. Samas on siiski viisakas andmebaasiühendus kinni panna. Ning mis vahel veel tähtsam - `exit`-käsklusega ülejäänud lehe näitamine katkestada. Tüüpiliseks probleemiks näiteks lehe turvamise juures on, et valede kasutajatunnustega sisenenud inimese puhul saadetakse küll brauserile käsklus lehe vahetamiseks, kuid tegeliku lehe serveerimist ei katkestata. Sellisel juhul on veebilehitseja asemel mõne telneti-sarnase programmiga veebiserveri pordiga suheldes võimalik vabalt ka ülejäänud lehe sisu näha. Nii et meelde jätmiseks - vaatamise ümber suunamisel kindlasti järgi käsklus `exit()`.

```
  if(isSet($_REQUEST["uusleht"])){
    $kask=$yhendus->prepare("INSERT INTO lehed (pealkiri, sisu) VALUES (?, ?)");
    $kask->bind_param("ss", $_REQUEST["pealkiri"], $_REQUEST["sisu"]);
    $kask->execute();
    header("Location: $_SERVER[PHP_SELF]");
    $yhendus->close();
    exit();
  }
```

Andmete kustutamise juures tuleb kõigepealt kasutajale anda võimalus teate kustutamiseks. Sobib see näiteks teate vaatamise lehele, kus vastava teate andmed juba nagunii olemas on. Sealt loon viite samale lehele, andes aadressireal küsimärgi järele parameetri nimega "kustutusid", mille väärtuseks saab vastava teate id-number.

```
            if($kask->fetch()){
              echo "<h2>".htmlspecialchars($pealkiri)."</h2>";
              echo htmlspecialchars($sisu);
              echo "<br /><a href='$_SERVER[PHP_SELF]?kustutusid=$id'>kustuta</a>";
            } else {
              echo "Vigased andmed.";
            }
```

Lehe uuel avamisel saab kontrollida, kas esineb parameeter `kustutusid`. Kui mitte, siis pole vaja kustutamisega tegelda. Kui aga leidub, siis tuleb sobiv DELETE-lause käivitada. Primaarvõtmeks olev id-number siin täisarvuna, sellest ka "i" `bind_param` käskluse esimese parameetrina.

Kustutuse juures praegusel juhul korduva lehe avamise kontrolli ei ole, sest korduval sama id-ga teate kustutamisel ei juhtu midagi - vähemasti senikaua, kui pole eelnevat kontrolli tehtud, kas kustutatav asi üldse olemas on. Kui aga eeldan heauskset kasutajat, kes aadressireale käske ei kirjuta, vaid viisakasti viidete järgi rakendusega suhtleb, siis tal üksi oma lehestikku administreerides ei tohiks sellist üllatust juhtuda.

```
  if(isSet($_REQUEST["kustutusid"])){
    $kask=$yhendus->prepare("DELETE FROM lehed WHERE id=?");
    $kask->bind_param("i", $_REQUEST["kustutusid"]);
    $kask->execute();
  }
```

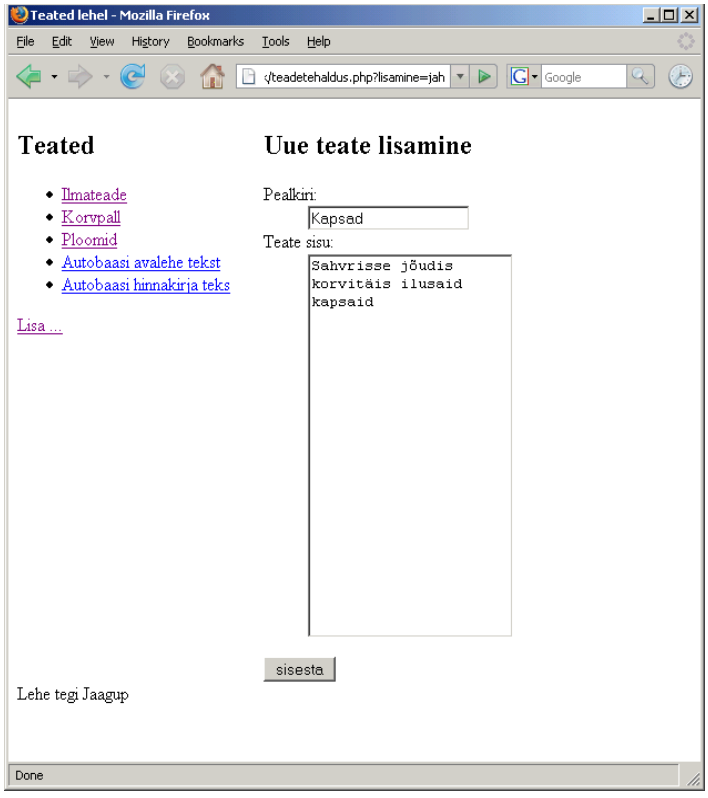
Edasi juba lisamis- ja kustutusvõimelise lehe kood tervikuna.

```
<?php
  $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");
  if(isSet($_REQUEST["uusleht"])){
```

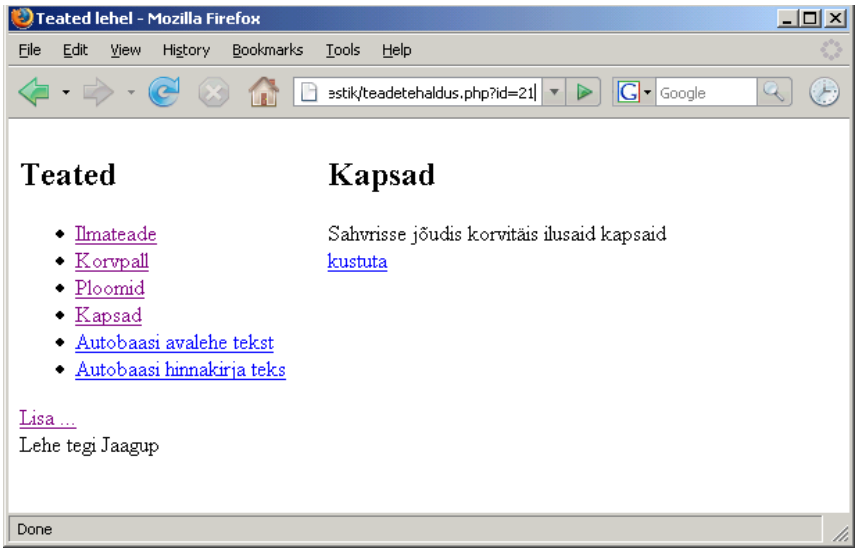
```
    $kask=$yhendus->prepare("INSERT INTO lehed (pealkiri, sisu) VALUES (?, ?)");
    $kask->bind_param("ss", $_REQUEST["pealkiri"], $_REQUEST["sisu"]);
    $kask->execute();
    header("Location: $_SERVER[PHP_SELF]");
    $yhendus->close();
    exit();
  }
  if(isSet($_REQUEST["kustutusid"])){
    $kask=$yhendus->prepare("DELETE FROM lehed WHERE id=?");
    $kask->bind_param("i", $_REQUEST["kustutusid"]);
    $kask->execute();
  }
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>Teated lehel</title>
    <style type="text/css">
       #menyykiht{
         float: left;
         padding-right: 30px;
       }
       #sisukiht{
         float:left;
       }
       #jalusekiht{
         clear: left;
       }
    </style>
  </head>
  <body>
    <div id="menyykiht">
        <h2>Teated</h2>
        <ul>
          <?php
             $kask=$yhendus->prepare("SELECT id, pealkiri FROM lehed");
             $kask->bind_result($id, $pealkiri);
             $kask->execute();
             while($kask->fetch()){
               echo "<li><a href='$_SERVER[PHP_SELF]?id=$id'>".
                           htmlspecialchars($pealkiri)."</a></li>";
             }
          ?>
        </ul>
        <a href='<?="$_SERVER[PHP_SELF]?lisamine=jah" ?>'>Lisa ...</a>
    </div>
    <div id="sisukiht">
       <?php
         if(isSet($_REQUEST["id"])){
            $kask=$yhendus->prepare("SELECT id, pealkiri, sisu FROM lehed
               WHERE id=?");
            $kask->bind_param("i", $_REQUEST["id"]);
            $kask->bind_result($id, $pealkiri, $sisu);
            $kask->execute();
            if($kask->fetch()){
              echo "<h2>".htmlspecialchars($pealkiri)."</h2>";
              echo htmlspecialchars($sisu);
              echo "<br /><a href='$_SERVER[PHP_SELF]?kustutusid=$id'>kustuta</a>";
            } else {
              echo "Vigased andmed.";
            }
         }
         if(isSet($_REQUEST["lisamine"])){
           ?>
             <form action='<?=$_SERVER["PHP_SELF"] ?>'>
              <input type="hidden" name="uusleht" value="jah" />
              <h2>Uue teate lisamine</h2>
              <dl>
                <dt>Pealkiri:</dt>
                <dd>
```

```
                <input type="text" name="pealkiri" />
               </dd>
               <dt>Teate sisu:</dt>
               <dd>
                 <textarea rows="20" name="sisu"></textarea>
               </dd>
              </dl>
              <input type="submit" value="sisesta">
            </form>
          <?php
        }
      ?>
    </div>
    <div id="jalusekiht">
       Lehe tegi Jaagup
    </div>
  </body>
</html>
<?php
  $yhendus->close();
?>
```

Andmete lisamine veebilehel:



Lisatud teate andmete vaatamine:



Ülesandeid

* Koosta/otsi andmetabelipõhine veebileht koerte andmete vaatamiseks
* Võimalda koeri lisada
* Võimalda koeri kustutada

Andmete muutmine veebilehel

Lihtsaim andmete muutmine sageli koosnebki kustutamisest ja uuest lisamisest. Nii on mõnigi kord kergem teha, kui viisakamat muutmisvahendit luua. Pikkade ja keerukate andmete puhul pole aga kasutajad kuigivõrd rahul, kui väikese trükivea või täienduse pärast peaks terve rea jagu andmeid uuesti sisse panema. Samuti on kustutamiseta muutmine tähtis oludes, kus andmeid hakatakse juba omavahel siduma. Kui näiteks koeral on kindel omanik, ehk viide inimeste andmetabeli reale. Siis selle rea juures nt. telefoninumbri muutmisel jääb koera juurde ikka sama omanik. Kui aga telefoninumbri muutmiseks inimese rida ära kustutatakse ja uuesti luuakse, siis on juba keerukam hoolitseda, et koera juures andmebaasis kindel sama omanik oleks.

Järgneva näite pealt näebki, kuidas PHP abil andmeid muuta nõnda, et ei peaks kõike maha kustutama ja uuesti sisestama, vaid saab paranduse paigutada vaid sobivasse kohta.

Kõigepealt lisame kustutamise viite kõrvale viite ka muutmise kohta. Anname lehe uuel avamisel kaasa muudetava teate id, samuti parameetri "muutmine" väärtusega "jah".

```
echo "<br /><a href='$_SERVER[PHP_SELF]?kustutusid=$id'>kustuta</a> ";
echo "<a href='$_SERVER[PHP_SELF]?id=$id&amp;muutmine=jah'>muuda</a>";
```

Nende põhjal saab siis järgmisel avamisel kontrollida, kas on seatud parameeter nimega "muutmine".

```
if(isSet($_REQUEST["muutmine"])){
```

Kui jah, siis luuakse vorm ja pannakse sinna kaasa muutmisid - mille järgi siis muudatuste salvestamise juures teab, milllise teate juures muutus tuleb kirja panna.

```
<input type='hidden' name='muutmisid' value='$id'/>
```

Väljadele antakse sisse olemasolevad väärtused, et neid saaks siis soovi järgi täiendada/parandada.

```
                        <input type='text' name='pealkiri'
value='".htmlspecialchars($pealkiri)."'/>
```

Muutmisvormi loov koodilõik siit tervikuna silma ette. Tekstiväljale ja tekstialale saab vana sisu suhteliselt lihtsalt sisse panna. Rippenüüde puhul tuleb näiteks sobiv rida kavalasti ette kerida - aga sellest juba edaspidi. Ning eks peab jääma ka lõik tavalise teate näitamiseks - ilma muutmata. Ning viisakuse poolest ka kontroll selle kohta, et kui soovitakse olematut teadet - olgu siis aadressirea muutmise tõttu või selle poolest, et keegi vahepeal serverist teate ära kustutas - siis antakse ka sõnadega teada, et nende andmete põhjal teadet kätte ei saa.

```
            if($kask->fetch()){
             if(isSet($_REQUEST["muutmine"])){
                echo "
                   <form action='$_SERVER[PHP_SELF]'>
                     <input type='hidden' name='muutmisid' value='$id'/>
                     <h2>Teate muutmine</h2>
                     <dl>
                       <dt>Pealkiri:</dt>
                       <dd>
                         <input type='text' name='pealkiri'
                                value='".htmlspecialchars($pealkiri)."'/>
                       </dd>
                       <dt>Teate sisu:</dt>
                       <dd>
                         <textarea rows='20' cols='30'
                            name='sisu'>".htmlspecialchars($sisu)."</textarea>
                       </dd>
                     </dl>
                     <input type='submit' value='Muuda' />
                   </form>
                ";
             } else {
              echo "<h2>".htmlspecialchars($pealkiri)."</h2>";
              echo htmlspecialchars($sisu);
              echo "<br /><a href='$_SERVER[PHP_SELF]?kustutusid=$id'>kustuta</a> ";
              echo "<a href='$_SERVER[PHP_SELF]?id=$id&amp;muutmine=jah'>muuda</a>";
             }
            } else {
              echo "Vigased andmed.";
            }
```

Kui uued andmed sees, vajutatakse submit-nuppu ning andmed jõuavad taas lehele. Parameetri "muutmisid" olemasolu järgi teab, et nüüd tasub olemasoleva teate andmeid muutma hakata. Eks jällegi tuleb kokku panna SQL-lause. Kõik muudetavad väärtused saavad SET-osas omale veebilehelt tulnud andmete põhjal uue sisu ning lõpus WHERE osas määratakse, millise teate kohta see muutus käib. Siit kusjuures oht: kui kogemata unustatakse WHERE osa märkimata, siis kirjutatakse nende andmetega üle kõik tabelis leiduvad read ilma midagi eelnevalt küsimata ega kontrollimata.

Parameetrite tüüpideks siin "ssi", sest pealkiri ja sisu on stringid, tabelirida määrav id aga integer. Käsk taas käima ning andmed muudetud. Siingi ei ole lehe ümbersuunamist tehtud, sest kui ka juhtutaks lehe värskendusnupule vajutama, siis kirjutataks uued andmed lihtsalt veel korra tabelisse ning midagi muud sellega ei kaasne.

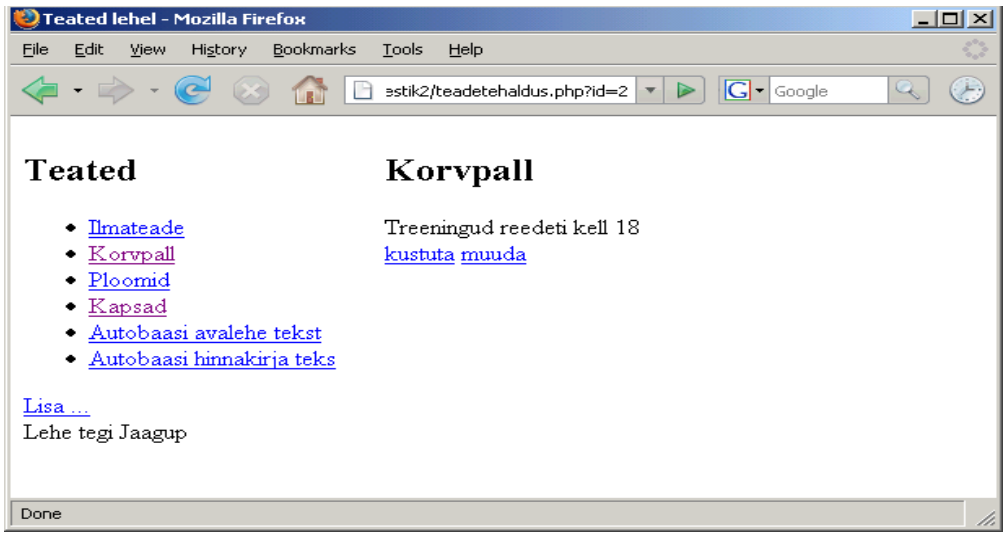
```
  if(isSet($_REQUEST["muutmisid"])){
    $kask=$yhendus->prepare("UPDATE lehed SET pealkiri=?, sisu=? WHERE id=?");
    $kask->bind_param("ssi", $_REQUEST["pealkiri"], $_REQUEST["sisu"],
$_REQUEST["muutmisid"]);
    $kask->execute();
  }
```

Edasi muutmisvõimelise lehe lähtekood tervikuna.

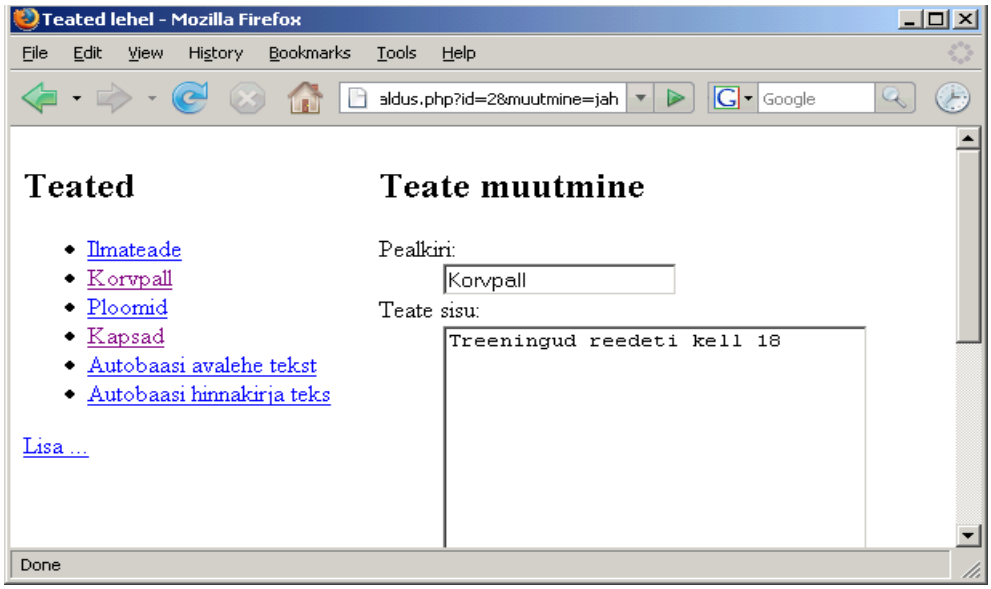
```
<?php
  $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");
  if(isSet($_REQUEST["uusleht"])){
    $kask=$yhendus->prepare("INSERT INTO lehed (pealkiri, sisu) VALUES (?, ?)");
    $kask->bind_param("ss", $_REQUEST["pealkiri"], $_REQUEST["sisu"]);
    $kask->execute();
    header("Location: $_SERVER[PHP_SELF]");
    $yhendus->close();
    exit();
  }
  if(isSet($_REQUEST["kustutusid"])){
    $kask=$yhendus->prepare("DELETE FROM lehed WHERE id=?");
    $kask->bind_param("i", $_REQUEST["kustutusid"]);
    $kask->execute();
  }
  if(isSet($_REQUEST["muutmisid"])){
    $kask=$yhendus->prepare("UPDATE lehed SET pealkiri=?, sisu=? WHERE id=?");
    $kask->bind_param("ssi", $_REQUEST["pealkiri"], $_REQUEST["sisu"],
$_REQUEST["muutmisid"]);
    $kask->execute();
  }
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Teated lehel</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <style type="text/css">
       #menyykiht{
         float: left;
         padding-right: 30px;
       }
       #sisukiht{
         float:left;
       }
       #jalusekiht{
         clear: left;
       }
    </style>
  </head>
  <body>
    <div id="menyykiht">
        <h2>Teated</h2>
        <ul>
          <?php
             $kask=$yhendus->prepare("SELECT id, pealkiri FROM lehed");
             $kask->bind_result($id, $pealkiri);
             $kask->execute();
             while($kask->fetch()){
               echo "<li><a
href='$_SERVER[PHP_SELF]?id=$id'>".htmlspecialchars($pealkiri)."</a></li>";
             }
          ?>
        </ul>
        <a href='<?="$_SERVER[PHP_SELF]?lisamine=jah" ?>'>Lisa ...</a>
    </div>
    <div id="sisukiht">
       <?php
         if(isSet($_REQUEST["id"])){
            $kask=$yhendus->prepare("SELECT id, pealkiri, sisu FROM lehed WHERE id=?");
            $kask->bind_param("i", $_REQUEST["id"]);
            $kask->bind_result($id, $pealkiri, $sisu);
            $kask->execute();
            if($kask->fetch()){
             if(isSet($_REQUEST["muutmine"])){
                echo "
                   <form action='$_SERVER[PHP_SELF]'>
```

```
                    <input type='hidden' name='muutmisid' value='$id'/>
                    <h2>Teate muutmine</h2>
                    <dl>
                      <dt>Pealkiri:</dt>
                      <dd>
                        <input type='text' name='pealkiri' value='".
                                    htmlspecialchars($pealkiri)."'/>
                      </dd>
                      <dt>Teate sisu:</dt>
                      <dd>
                        <textarea rows='20' cols='30' name='sisu'>".
                           htmlspecialchars($sisu)."</textarea>
                      </dd>
                    </dl>
                    <input type='submit' value='Muuda' />
                  </form>
                ";
              } else {
               echo "<h2>".htmlspecialchars($pealkiri)."</h2>";
               echo htmlspecialchars($sisu);
               echo "<br /><a href='$_SERVER[PHP_SELF]?kustutusid=$id'>kustuta</a> ";
               echo "<a href='$_SERVER[PHP_SELF]?id=$id&amp;muutmine=jah'>muuda</a>";
              }
             } else {
               echo "Vigased andmed.";
             }
          }
          if(isSet($_REQUEST["lisamine"])){
            ?>
              <form action='<?=$_SERVER["PHP_SELF"] ?>'>
               <input type="hidden" name="uusleht" value="jah" />
               <h2>Uue teate lisamine</h2>
               <dl>
                 <dt>Pealkiri:</dt>
                 <dd>
                  <input type="text" name="pealkiri" />
                 </dd>
                 <dt>Teate sisu:</dt>
                 <dd>
                   <textarea rows="20" cols="30" name="sisu"></textarea>
                 </dd>
                </dl>
                <input type="submit" value="sisesta" />
              </form>
            <?php
          }
        ?>
    </div>
    <div id="jalusekiht">
       Lehe tegi Jaagup
    </div>
  </body>
</html>
<?php
  $yhendus->close();
?>
```

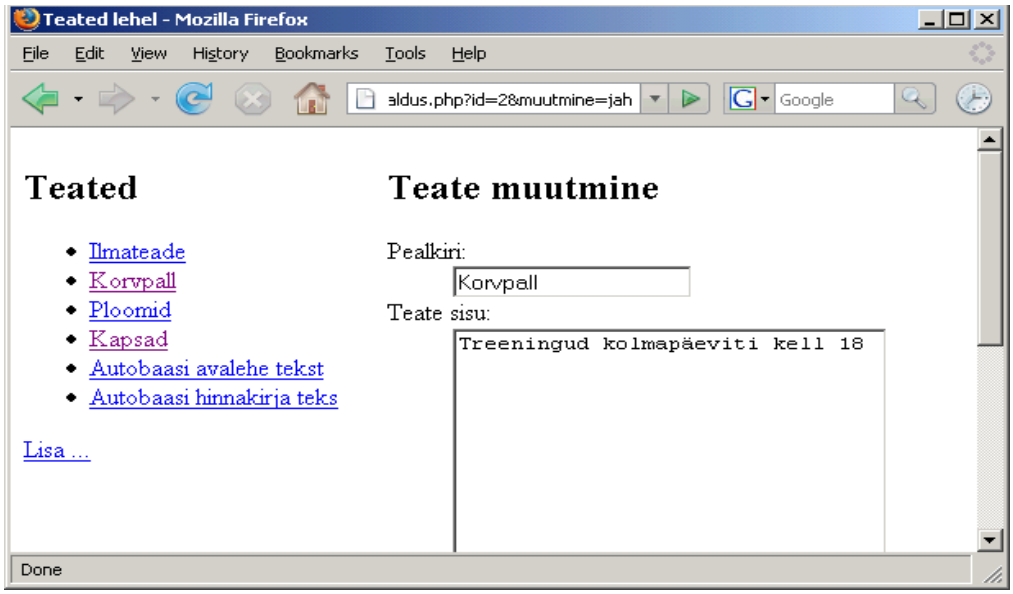
Piltidelt on näha, kuidas kõigepealt avatakse id järgi sobiv teade.



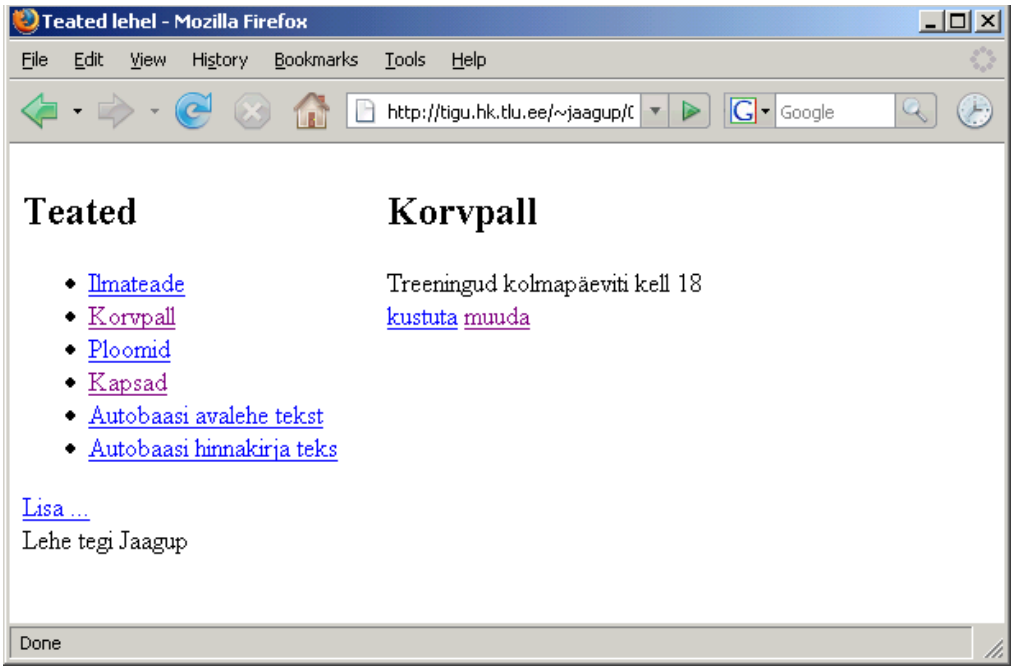
Edasi muutmisviitele vajutades avatakse leht koos vormiga andmete muutmiseks:



Siis saab parandused sisse viia



Ning lõpuks tasub muudetud lehte imetleda.



Ülesandeid

* Tee näide läbi.
* Anna muutmisel hoiatus juhul, kui sisestatud pealkirja pikkus ületab 50 sümbolit

* Loo/otsi koerte andmeid näitav veebirakendus
* Lisa rakendusele andmete muutmise võimalus.

Graafiline tekstiredaktor

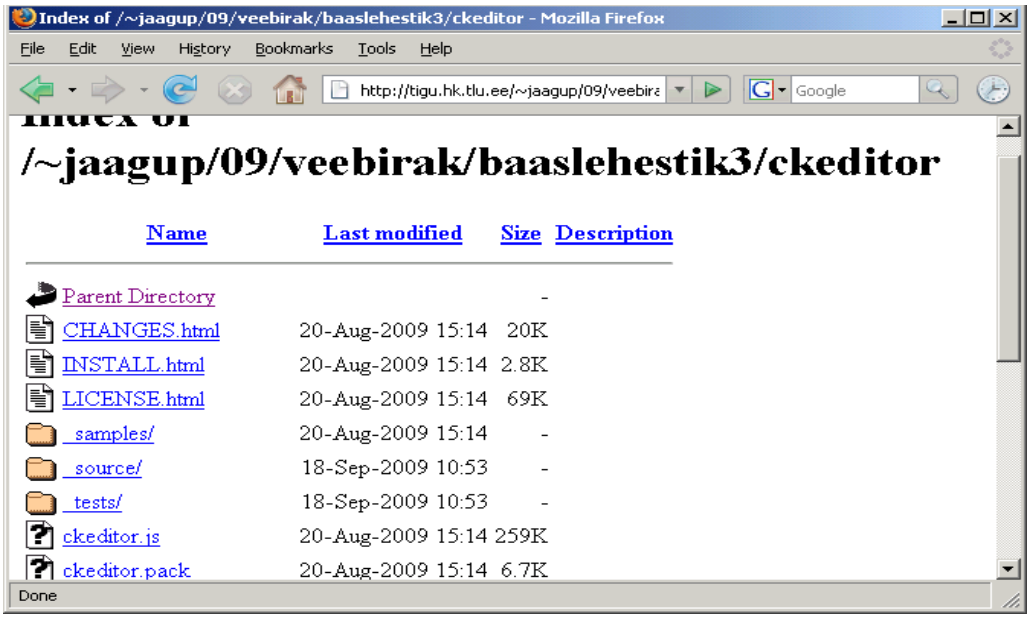
Aastaid oli veebikoostajate märgatavaks osaks tööst lehekülgede kaupa HTMLi kirjutada ja hoolitseda, et selle abil kirja pandud tekst rahvale viisakalt loetav oleks. Eks sellist kujundamist ole praegugi - kui tahtmist oma leht pikslipealt paika joonistada. Aga veebipõhised redaktorid aitavad küllaltki mugavalt tavakasutajatel omaloodud sisu kujundada ilma, et arvutiabilist pidevalt kõrval oleks.

Graafilisi ehk WYSIWYG (What You See Is What You Get) tekstiredaktoreid veebilehel on tekkinud hulgem, tuleb vaid omale sobiv leida. Väiksematel piisab mõnekilobaidisest Javaskripti failist, keerukamatel võib allalaetav skript koos abipiltidega mitmesaja kilobaidini ulatuda. Samuti suudab mõni hakkama saada vaid üksikute brauserite ja versioonidega, teisel on paindlikkust rohkem.

Kirjutise autor on juhtunud kasutama Javaskriptipõhiseid redaktoreid TinyMCE ja Kupu. Ning subjektiivselt parima mulje on jätnud CKEditor (endine FCKEditor). Seda kasutame ka siinses näites.

Redaktor tuleb kõigepealt alla laadida ja lahti pakkida. Tasub vaadata, et kus kaustas asub fail

ckeditor.js - see tuleb varsti meie lehe külge haakida.



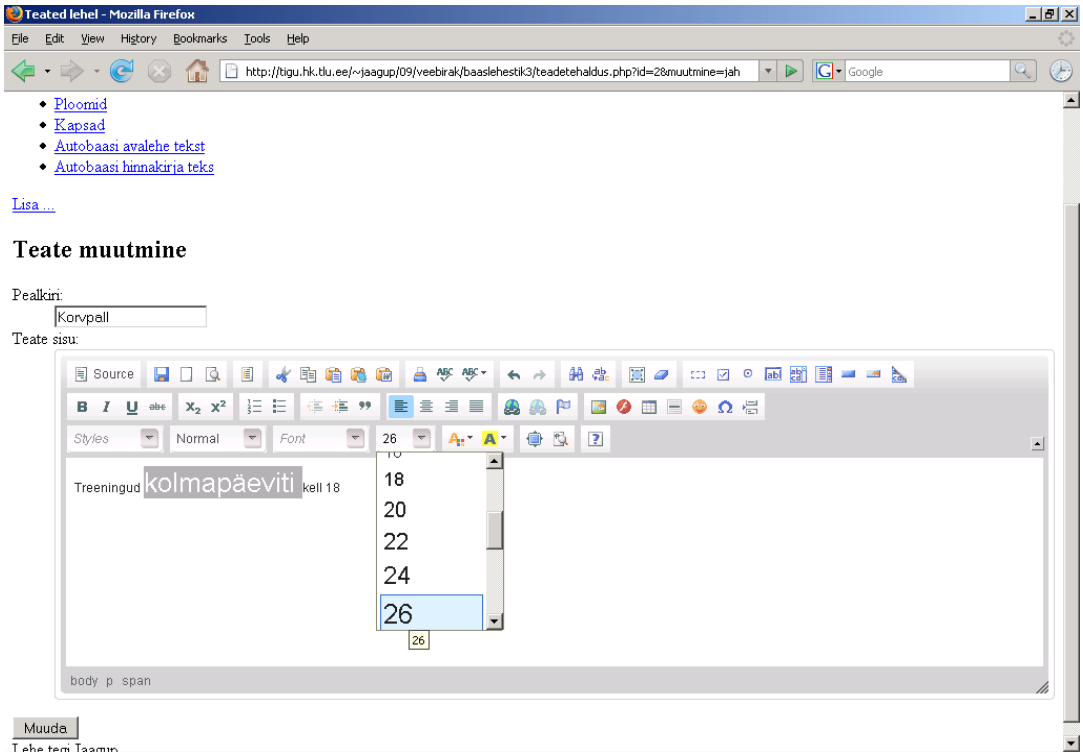
Siin puhul sattusid redaktori failid eraldi alamkausta ckeditor - seega laeme sisse faili aadressilt ckeditor/ckeditor.js.

```
<script type="text/javascript" src="ckeditor/ckeditor.js"></script>
```

Uus versioon on nõnda targaks tehtud, et teksti ala graafiliseks redaktoriks muutmiseks on tarvis vaid talle külge panna atribuut class='ckeditor'

```
<textarea rows='20' cols='30' class='ckeditor' name='sisu'>$sisu</textarea>
```

Ning kui kõik andmed ilusti kätte saadakse, siis võibki rahulikult menüüst kujunduskäske valida ja teksti suurust muuta.



Sama asi ka lisamise juures: tuleb lihtsalt vaadata, kus asub tekstiala, talle külge panna klass ckeditor ning jällegi on elu tunduvalt värvilisem. Kui millegipärast ei ole, siis tasub kõigepealt uurida, et kas viidatud kohast ikkagi javaskripti fail kergesti kätte saadakse. Selle aadressi võib kasvõi brauserireale sisse lüüa ja kontrollida, et mis sealt välja antakse. Samuti tasub kontrollida, et ega veebilehitsejal pole Javaskript kinni keeratud - turvapõhjustel seda mõnikord tehakse.

```
if(isSet($_REQUEST["lisamine"])){
  ?>
    <form action='<?=$_SERVER["PHP_SELF"] ?>'>
     <input type="hidden" name="uusleht" value="jah" />
     <h2>Uue teate lisamine</h2>
     <dl>
       <dt>Pealkiri:</dt>
       <dd>
        <input type="text" name="pealkiri" />
       </dd>
       <dt>Teate sisu:</dt>
       <dd>
         <textarea rows="20" cols="30" class="ckeditor" name="sisu"></textarea>
       </dd>
      </dl>
      <input type="submit" value="sisesta" />
    </form>
  <?php
}
```

Kui niisama lihtsalt panna lehe tekstialadele redaktor külge, siis on küll ilus teksti värviliseks joonistada, kuid suure tõenäosusega pärast vaadates pole muutused korralikult näha. Raskemal juhul võib osa teksti sootuks kadunud olla.

Põhjuseks, et redaktorist tulnud HTMLi varjestab PHP vastavalt serveri konfiguratsioonile ära. MySQL Improved aga ei vaja SQL-lausesse paigutamiseks andmete langjoontega varjestamist ning nõnda jõuavad tavajuhul mitmele poole \-märgid, mis häirivad HTMLi vaatamist.

Liigsetest langjoontest aitab vabaneda funktsioon `stripslashes`, mis siis varjestatud sisu uuesti "tavaliseks" teeb.

```
  if(isSet($_REQUEST["muutmisid"])){
    $kask=$yhendus->prepare("UPDATE lehed SET pealkiri=?, sisu=? WHERE id=?");
    $kask->bind_param("ssi", $_REQUEST["pealkiri"], stripslashes($_REQUEST["sisu"]),
$_REQUEST["muutmisid"]);
    $kask->execute();
  }
```

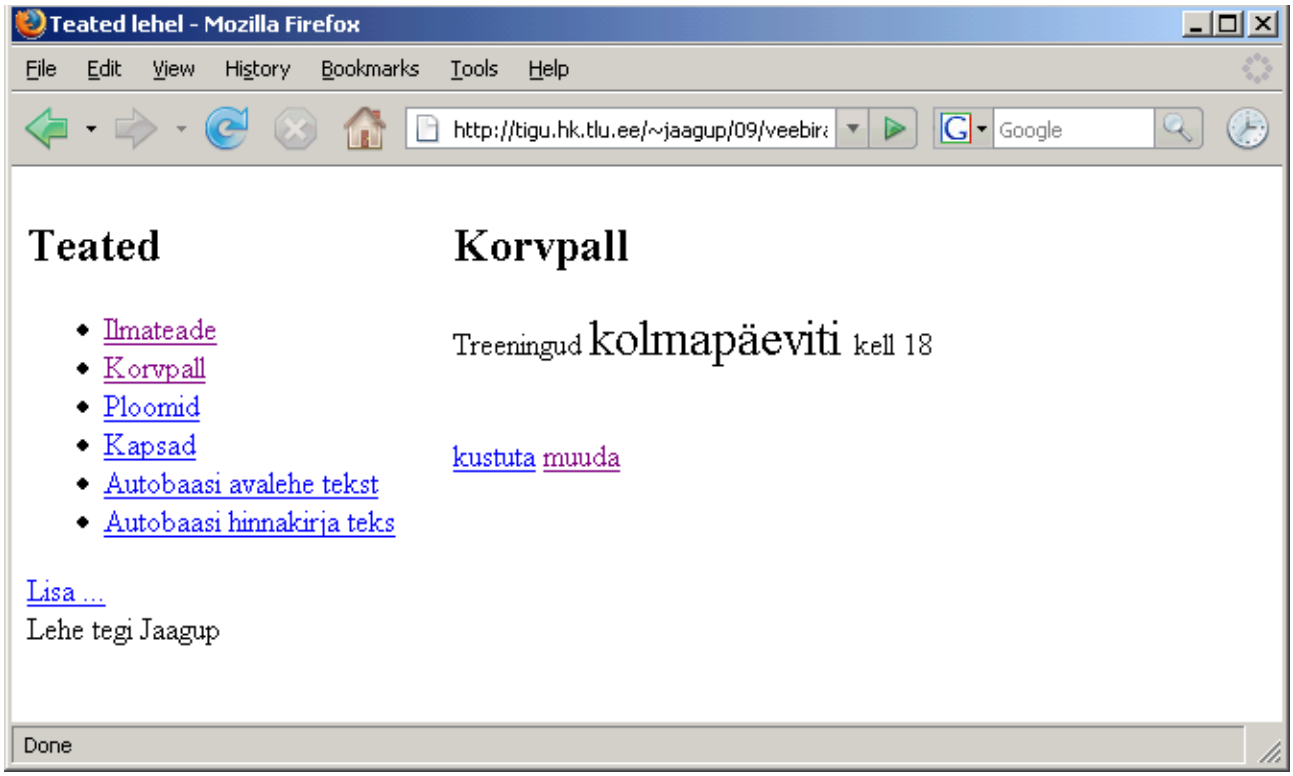
Redaktorit kasutava lehe kood tervikuna.

```
<?php
  $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");
  if(isSet($_REQUEST["uusleht"])){
    $kask=$yhendus->prepare("INSERT INTO lehed (pealkiri, sisu) VALUES (?, ?)");
    $kask->bind_param("ss", $_REQUEST["pealkiri"], stripslashes($_REQUEST["sisu"]));
    $kask->execute();
    header("Location: $_SERVER[PHP_SELF]");
    $yhendus->close();
    exit();
  }
  if(isSet($_REQUEST["kustutusid"])){
    $kask=$yhendus->prepare("DELETE FROM lehed WHERE id=?");
    $kask->bind_param("i", $_REQUEST["kustutusid"]);
    $kask->execute();
  }
  if(isSet($_REQUEST["muutmisid"])){
    $kask=$yhendus->prepare("UPDATE lehed SET pealkiri=?, sisu=? WHERE id=?");
    $kask->bind_param("ssi", $_REQUEST["pealkiri"], stripslashes($_REQUEST["sisu"]),
$_REQUEST["muutmisid"]);
```

```
    $kask->execute();
  }
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Teated lehel</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <style type="text/css">
       #menyykiht{
         float: left;
         padding-right: 30px;
       }
       #sisukiht{
         float:left;
       }
       #jalusekiht{
         clear: left;
       }
    </style>
    <script type="text/javascript" src="ckeditor/ckeditor.js"></script>
  </head>
  <body>
    <div id="menyykiht">
        <h2>Teated</h2>
        <ul>
          <?php
             $kask=$yhendus->prepare("SELECT id, pealkiri FROM lehed");
             $kask->bind_result($id, $pealkiri);
             $kask->execute();
             while($kask->fetch()){
               echo "<li><a href='$_SERVER[PHP_SELF]?id=$id'>".
                    htmlspecialchars($pealkiri)."</a></li>";
             }
          ?>
        </ul>
        <a href='<?="$_SERVER[PHP_SELF]?lisamine=jah" ?>'>Lisa ...</a>
    </div>
    <div id="sisukiht">
       <?php
         if(isSet($_REQUEST["id"])){
            $kask=$yhendus->prepare("SELECT id, pealkiri, sisu FROM lehed WHERE id=?");
            $kask->bind_param("i", $_REQUEST["id"]);
            $kask->bind_result($id, $pealkiri, $sisu);
            $kask->execute();
            if($kask->fetch()){
             if(isSet($_REQUEST["muutmine"])){
                echo "
                   <form action='$_SERVER[PHP_SELF]'>
                     <input type='hidden' name='muutmisid' value='$id'/>
                     <h2>Teate muutmine</h2>
                     <dl>
                       <dt>Pealkiri:</dt>
                       <dd>
                         <input type='text' name='pealkiri' value='".
                                  htmlspecialchars($pealkiri)."'/>
                       </dd>
                       <dt>Teate sisu:</dt>
                       <dd>
                         <textarea rows='20' cols='30'
                             class='ckeditor' name='sisu'>$sisu</textarea>
                       </dd>
                     </dl>
                     <input type='submit' value='Muuda' />
                   </form>
                ";
             } else {
              echo "<h2>".htmlspecialchars($pealkiri)."</h2>";
              echo $sisu;
              echo "<br /><a href='$_SERVER[PHP_SELF]?kustutusid=$id'>kustuta</a> ";
```

```
            echo "<a href='$_SERVER[PHP_SELF]?id=$id&amp;muutmine=jah'>muuda</a>";
           }
          } else {
           echo "Vigased andmed.";
          }
        }
        if(isSet($_REQUEST["lisamine"])){
          ?>
            <form action='<?=$_SERVER["PHP_SELF"] ?>'>
             <input type="hidden" name="uusleht" value="jah" />
             <h2>Uue teate lisamine</h2>
             <dl>
               <dt>Pealkiri:</dt>
               <dd>
                <input type="text" name="pealkiri" />
               </dd>
               <dt>Teate sisu:</dt>
               <dd>
                 <textarea rows="20" cols="30" class="ckeditor" name="sisu"></textarea>
               </dd>
              </dl>
               <input type="submit" value="sisesta" />
            </form>
          <?php
        }
      ?>
    </div>
    <div id="jalusekiht">
       Lehe tegi Jaagup
    </div>
  </body>
</html>
<?php
  $yhendus->close();
?>
```

Ning näide CKEditori abil kujundatud teatest.



Ülesandeid

* Tutvu lehel http://ckeditor.com/ redaktori demoga
* Otsi võimalusel ka teisi Javaskripti-põhiseid redaktoreid ning katseta neid
* Loo või otsi rakendus koerte andmete sisestamiseks ja vaatamiseks
* Võimalda koera kirjeldus kujundada graafilise redaktori abil.

Veebilehestiku lõikude hoidmine andmebaasis

Siiani tehtud näidetes on muudetav sisu olnud lehestiku struktuuri mõttes samas kohas - teateid on kord rohkem kord vähem, kuid nad saab ikka ühest loetelust kätte.

Kui kellegi tarbeks veebilehestikku teha, siis on küllalt tavaline, et sisu soovitakse muuta mitmesugustes kohtades - täiesti erinevatel lehtedel ning ka lehe paigutuse suhtes mitmesugustes kohtades. Üheks võimaluseks sellist lehestikku veebi kaudu hallatavaks teha on hoida kõik muudetavad kohad teadetena andmebaasis ning lehe näitamisel siis vastavad sisud sealt välja võtta. Nõnda on lehe haldus suhteliselt sarnane siinsele teadete haldusele. Lehte sisu poolest näidatakse just sellisena nagu ta tehtud on ning tekstide muutmise võimalus ei sea kuigivõrd piiranguid lehestiku ülesehitusele. Lihtsalt muudetavaid kohti peab olema piiratud hulk- et nende hilisem otsimine teadete menüüst ei osutuks üle jõu käivaks. Viisaka plokkide nimetamise abil saab ka sellest murest üle.

Funktsioonid eraldi failis

Väiksema lehestiku puhul kannatab enamiku vajaminevast ühte faili kokku kirjutada. Mõnda aega on niimoodi hea ülevaatlik. Ning uue rakenduse loomiseks piisab lihtsalt vastava lehe kopeerimisest. Suurema rakenduse puhul aga kipuks nõnda ühes failis olevate koodiridade hulk keerukalt suureks minema. Ligikaudu tuhatkond rida on suurus, mida kannatab hea süsteemi korral veel ühes failis hallata. Üle selle kipub igatmoodi kirjuks minema. Kui aga funktsioonid teemade kaupa eraldi failidesse - põhjalikuma struktueerituse puhul ka klassidesse - paigutada, siis on lootust veel tükk aega rahus elada enne, kui kood päriselt üle pea kasvama kipub.

Samuti on funktsioonide eraldamise eeliseks, et siis on kujundus võimalikult ühes failis ja programmeerimise pool teises. Nii ei teki liialt palju nii kujundajate kui ka arendajate poolt kirutud "spageti-koodi". Erinevalt mõnest muust keelest (ASP.NET, Zope lehemallid) ei ole PHPl kindlat ja üheselt tunnustatud koodi ja kujunduse eraldamise tava välja kujunenud, kuid kodeerimispraktikaid ja mallisüsteeme on loodud päris mitmesuguseid.

Siin näites loodi eraldi fail "sisufunktsioonid.php", kuhu esimeses järjekorras pannakse funktsioon teate sisu küsimiseks vastavalt ploki id-numbrile. Lehtede tabelist küsitakse otsitud teate sisu ning tagastatakse funktsioonist.

sisufunktsioonid.php

```
<?php
   $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");
   function kysiSisu($id){
      global $yhendus;
```

```
      $kask=$yhendus->prepare("SELECT sisu FROM lehed WHERE id=?");
      $kask->bind_param("i", $id);
      $kask->bind_result($sisu);
      $kask->execute();
      if($kask->fetch()){
        return $sisu;
      }
      return "Andmed kadunud";
   }
?>
```

Avalehel - või ka mujal - teadete näitamiseks tuleb hoolitseda, et sisufunktsioonide fail oleks sisse loetud. Edasi saab teate andmed näidata andes funktsioonile ette vastava teatekoha identifikaatori. Kui kord teated valmis teha, siis edaspidi võib neile juba julgesti id järgi viidata. Kuni teadet vaid muudetakse, aga ei kustutata, senikauaks jääb id püsima. Nõnda siis piisabki hallatava sisuploki nägemiseks vaid kahest reast

```
require_once("sisufunktsioonid.php");
echo kysiSisu(16);
```

avaleht.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml>
  <head>
       <title>Autobaasi avaleht</title>
       <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
      <?php require("menyy.php") ?>
      <h1>Tore vahva asjalik autobaas</h1>
      <div>
         Meie uued uudised:
         <?php
           require_once("sisufunktsioonid.php");
           echo kysiSisu(16);
         ?>
      </div>
  </body>
</html>
```

Menüüfail lehtede vahel navigeerimiseks

menyy.php

```
<div>
  <a href="avaleht.php">Avaleht</a> | <a href="hinnakirjaleht.php">Hinnakiri</a>
</div>
```

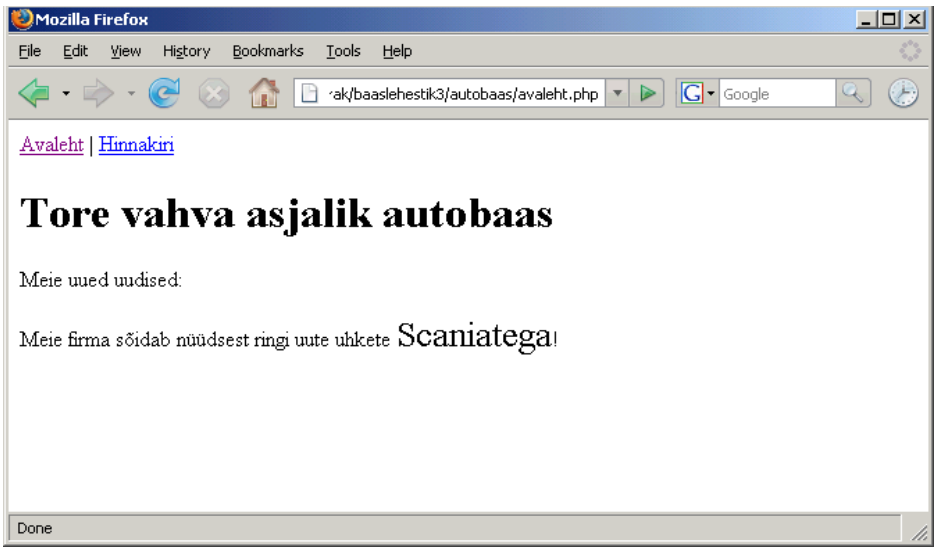
Hinnakirjalehel kasutatakse samuti loodud funktsiooni sisu küsimiseks. Lihtsalt teate kood on erinev.

hinnakirjaleht.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml>
  <head>
       <title>Autobaasi avaleht</title>
       <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
```

```
      <?php require("menyy.php") ?>
      <h1>Hinnakirja leht</h1>
      <div>
         Meie kehtiv hinnakiri:
         <?php
           require_once("sisufunktsioonid.php");
           echo kysiSisu(19);
         ?>
      </div>
  </body>
</html>
```

Kui teated paigas, siis pole enam muud muret kui lehte vaatama asuda. Eks viisakas teemakohane kujundus ka külge - aga see on juba staatilise kujundamise asi millega mõnigi hakkama saab. Kui tegemist on peaasjalikult tutvustava/teatava lehega ning olulist infosüsteemi selle taga pole, siis teadete haldamise oskusest + nende lehel näitamise oskustest piisab peaaegu ükskõik kui ilusa ja keerulise lehestiku kokkupanekust. Autobaasi veebilehe redigeeritava avalehe demo.



Ning lehe muutmine käib praegusel juhul täiesti vabalt sama teadete halduse lehestikku kasutades.

Ülesandeid

* Tee autobaasi lehestiku näide läbi
* Kavanda ja koosta temaatiline kujundatud veebilehestik, kus mitmes vajalikus kohas saab sisu veebi kaudu muuta.

Andmed mitmes tabelis

Vaid muudetava tekstilise sisuga lehestikes võib hakkama saada ühe andmetabeliga. Samuti piisab ühest tabelist juhul, kui hoitakse ja näidatakse ühetüübilisi andmeid. Kui aga tahta veebi kaudu tööle panna vähegi mitmekesisemat infosüsteemi, siis jääb varsti ühest tabelist kitsaks. Mõnedki asjad küll saab veel ära ajada mitme eraldiseisva tabeliga. Ning veidi kavaldades õnnestub ka küllalt mitmekülgseid andmeid ühte tabelisse toppida - kui näiteks ei oska andmeid eri tabelitesse jagada ja neid omavahel siduda. Mingist hetkest aga andmete kokkuväänamine enam ei õnnestu ning siis hakkab tabeleid kergesti ja hulgem juurde tulema. Mugav on ju arvestada, kui ühes tabelis on vaid samatüübilised asjad.

Ühe valitava omadusega nähtuse või eseme kirjeldamisel võib hakkama saada kahe andmetabeliga. Kaht tüüpi objekte (nt. inimesi ja reisikohti) mitut moodi omavahel kombineerides kulub vähemasti kolm tabelit (üks inimeste tarbeks, üks kohtade kirjeldamiseks ning kolmandas tabelis on kirjas reisid, kes millal kus käis). Kasutajate, õiguste, pea- ja alateemadega foorumit kokku pannes kulub neli-viis tabelit. Ning lihtsalt võrdluseks, et kooli õppeainetes läbitavaid teemasid mitmelt poolt otsida võimaldav süsteem kasvas paarikümne andmetabeli suuruseks. Kui juures oli veel palga- ja koormusearvestus ning muu personalihaldus, siis üsna pea kasvas infosüsteemi maht saja andmetabelini.

Tabelite rohkust iseenesest ei maksa karta. Kui nad on arusaadavalt nimetatud ning mõne skeemi peal näha, mis kuidas millega suhtleb, siis on paljude selgete tabelitega süsteem siiski päris mugavalt kasutatav. Enamasti rakenduse juures kasutatakse korraga neist ikka väiksemat, mõne tabeliga alamhulka. Vahel harva tekivad päringud, kus korraga on andmeid vaja kümnekonnast tabelist.

Kaubad ja kaubagrupid

Siin alustame taas võimalikult lihtsalt - seome omavahel kokku kaks andmetabelit. Ühes on kirjas olemasolevad kaubagrupid - igal grupil id ja grupinimi. Teises tabelis on tooted, kusjuures iga toode kuulub kindlasse gruppi. Selliselt kirja pannes on võimalik andmeid viisakasti vaadata gruppide kaupa. Pole karta, et üks kirjutab tööriistad väikese, teine suure tähega ning arvuti jaoks võivad vastavates gruppides olevad asjad sootuks erinevatena tunduda. Kui ikka grupi nimi valitakse toote juurde rippmenüüst, siis pole võimalik selle valimise juures enam grupi nime vigaselt kirjutada. Näitrakenduse teemaks tuleb

```
Kaupade otsing poes (tootekataloog)
```

Rakenduse jaoks vajame kahte andmetabelit.

```
kaubad:
```

```
id nimetus kaubagrupi_id hind

kaubagrupid:
id grupinimi
```

Sisuandmed näiteks:

```
kaubagrupid

1 tellised
2 katusematerjal
3 vineer

kaubad

1 ahjutellis      1    8.20
2 fassaaditellis  1    7.50
3 bituumenrull    2  520.00
```

Kaupade tabeli kaubagrupi_id näitab kaubagruppide tabeli id-le. Siit saab siis välja lugda, et ahjutellis ja fassaaditellis kuluvad mõlemad gruppi "tellised" ehk gruppi nr 1. Bituumenrull aga läheb teise grupi ehk katusematerjali alla.

Teeme esimese tabeli SQL-lause abil valmis

```
CREATE TABLE kaubagrupid(
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  grupinimi VARCHAR(255)
);
```

ning lisame mõned andmed sisse.

```
INSERT INTO kaubagrupid(grupinimi) VALUES ('tellised');
INSERT INTO kaubagrupid(grupinimi) VALUES ('katusematerjal');
```

Päringuga kontrollime, et andmed ikka kohale jõudsid. Samuti saame teada tekkinud gruppide id-d.

```
mysql> SELECT * FROM kaubagrupid;
+----+----------------+
| id | grupinimi      |
+----+----------------+
|  1 | tellised       |
|  2 | katusematerjal |
+----+----------------+
2 rows in set (0.00 sec)
```

Kui grupid olemas, on põhjust luua kaupade tabel.

```
CREATE TABLE kaubad(
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  nimetus VARCHAR(255),
  kaubagrupi_id INT,
  hind DECIMAL(10, 2)
);
```

Iseenesest me teame, et kaupade tabeli kaubagrupi_id näitab kaubagruppide tabeli id peale. Seda saaks ka SQL lauses märkida (FOREIGN KEY(kaubagrupi_id) REFERENCES kaubagrupid(id)). Kuna aga MySQL vaikimisi juhul nagunii viiteterviklust ei kontrolli, siis pigem on tähtsam, et oma peas meeles oleks, mis millega ja kuidas seotud.

Andmete lisamiseks SQLi kaudu paneme siis praegu lihtsalt vastavad grupinumbrid lausesse kirja.

```
INSERT INTO kaubad (nimetus, kaubagrupi_id, hind) VALUES ('ahjutellis', 1,
8.20);
INSERT INTO kaubad (nimetus, kaubagrupi_id, hind) VALUES ('fassaaditellis', 1,
7.50);
INSERT INTO kaubad (nimetus, kaubagrupi_id, hind) VALUES ('bituumenrull', 2,
520);
```

SELECTi abil kontrollime järgi, et soovitud andmed ikka baasi kohale jõudsid.

```
mysql> SELECT * FROM kaubad;
+----+---------------+--------------+--------+
| id | nimetus       | kaubagrupi_id | hind   |
+----+---------------+--------------+--------+
|  1 | ahjutellis    |            1 |   8.20 |
|  2 | fassaaditellis |            1 |   7.50 |
|  3 | bituumenrull  |            2 | 520.00 |
+----+---------------+--------------+--------+
```

ID-numbrid on küll programmeerija jaoks toredad, kuid tavainimene armastab andmeid siiski sõnalisel kujul lugeda. Järgmise lause abil saab näha kauba nimetust, sõnalist grupinime (mis võetud kaubagruppide tabelist) ning kauba hinda. Selleks kirjutan soovitud tulbad SELECTi järele. FROM-ossa märgin kasutatavad andmetabelid ning WHERE tingimuses panen kirja, kuidas tabelid omavahel seotud on. Siinsel juhul siis tabeli kaubad tulp kaubagrupi_id näitab tabeli kaubagrupid tulbale id. Võimalikest kummagi tabeli ridade kombinatsioonidest näidatakse välja siis vaid need, kus kaupade tabeli kaubagrupi_id kattub kaubagruppide tabeli id-ga.

```
SELECT nimetus, grupinimi, hind
FROM kaubad, kaubagrupid
WHERE kaubad.kaubagrupi_id=kaubagrupid.id;
```

Mugavuse mõttes on hea SQL-lause enne mõnes tekstiredaktoris valmis kirjutada ning alles siis MySQLi viiba otsa kopeerida. Sellisel juhul on mõne vea puhul kergesti võimalik lauset täiendada/parandada ning uuesti katsetada.

```
mysql> SELECT nimetus, grupinimi, hind
    -> FROM kaubad, kaubagrupid
    -> WHERE kaubad.kaubagrupi_id=kaubagrupid.id;
+---------------+---------------+--------+
| nimetus       | grupinimi     | hind   |
+---------------+---------------+--------+
| ahjutellis    | tellised      |   8.20 |
| fassaaditellis | tellised      |   7.50 |
| bituumenrull  | katusematerjal | 520.00 |
+---------------+---------------+--------+
```

Päringu tulemus paistis täiesti ootuspärane olema: tellised kuuluvad telliste gruppi ning bituumenrull katusele.

Mõned SQL-laused veel. Tingimuste abil saab osa ridu peita, ehk siis soovitud välja näidata. Järgnevalt kaubad, mille tükihind on alla kümne krooni.

```
mysql> SELECT nimetus, hind FROM kaubad WHERE hind<10.00;
+---------------+------+
| nimetus       | hind |
+---------------+------+
| ahjutellis    | 8.20 |
| fassaaditellis | 7.50 |
```

```
+----------------+------+
```

Tekstide puhul on tänuväärne funktsioon nimega LIKE. Päringusõnas tähendab protsent suvalist hulka suvalisi sümboleid. Ehk siis praegu otsitakse kõik kaubad, mille nimes sisaldub sõna tellis.

```
mysql> SELECT nimetus, hind FROM kaubad WHERE nimetus LIKE '%tellis%';
+----------------+------+
| nimetus        | hind |
+----------------+------+
| ahjutellis     | 8.20 |
| fassaaditellis | 7.50 |
+----------------+------+
```

Mitme tabeli ühendamine ning piirang päringu juures võivad kehtida ka üheaegselt.

```
mysql> SELECT nimetus, grupinimi, hind
    -> FROM kaubad, kaubagrupid
    -> WHERE kaubad.kaubagrupi_id=kaubagrupid.id
    ->   AND nimetus LIKE '%tellis%';
+----------------+-----------+------+
| nimetus        | grupinimi | hind |
+----------------+-----------+------+
| ahjutellis     | tellised  | 8.20 |
| fassaaditellis | tellised  | 7.50 |
+----------------+-----------+------+
```

Kui päringusse lisada ka id-tulp, siis juhtub tulema veateade:

```
Column 'id' in field list is ambiguous
```

Kuna mõlemal tabelil on vastava nimega tulp olemas, et teata, kas soovitakse näha kauba või kaubagrupi id-numbrit. Selle vastu aitab, kui tulba ette kirjutada tabeli nimi ehk siis praegusel juhul kaubad.id.

Siin näites püüame samuti hoida kujunduse ja koodi võimalikult lahus. Koodiosa paigutame faili nimega

```
abifunktsioonid.php
```

Eraldi seletust vajab ehk andmete väljastus funktsioonist. Varasemas näites tagastati funktsioonist vaid teate sisu - selleks piisas return-käsust. Siin tuleb aga iga kauba kohta neli väärtust: id, nimetus, grupinimi ja hind. Lisaks püüame funktsioonist korraga kätte saada kõikide kaupade andmed, et neid siis veebilehel mugavasti sobival kujul kuvada saaks.

Mitme samatüübilise väärtuse hoidmiseks aitab massiiv - nii ka siin. Enne andmete läbikäimise tsüklit tehakse uus massiiv nimega $hoidla.

```
    $hoidla=array();
```

Kauba andmete ühe kesta sisse kokku panekuks sobib stdClass-tüüpi objekt. Selle saab new-käsuga luua ning hiljem sinna ükshaaval omadusi külge panna. Olemasolevatest muutujatest võtame andmed ning paneme need kaubaisendi väljadeks. Tekstilisi andmeid töötleme käsuga htmlspecialchars, et teksti sees olevad erisümbolid ei pääseks veebilehe sisu segama.

```
      $kaup=new stdClass();
```

```
      $kaup->id=$id;
      $kaup->nimetus=htmlspecialchars($nimetus);
      $kaup->grupinimi=htmlspecialchars($grupinimi);
      $kaup->hind=$hind;
```

Iga kauba puhul kui selle andmed käes, lisatakse kaubaobjekt massiivi lõppu.

```
      array_push($hoidla, $kaup);
```

Funktsioonist väljudes tagastatakse hoidla koos kaupade andmetega.

```
    return $hoidla;
```

Funktsiooni töö kontrollimiseks on algul hea ta käivitada. Selleks abifunktsioonide faili lõpus vastav lõik. Märgend <pre> määrab, et veebilehel väljastades oleksid kõik tähed (ka tühikud) kindla laiusega (preformatted text). Käsklus print_r (rekursiivne print) trükib etteantud objekti või massiivi andmed sügavuti välja, ehk siis näitab kõik, mis sealt seest saada on.

```
<pre>
<?php
  print_r(kysiKaupadeAndmed());
?>
</pre>
```

Nüüd siis abifunktsioonid.php tervikuna.

```
<?php
  $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");

  function kysiKaupadeAndmed(){
    global $yhendus;
    $kask=$yhendus->prepare("SELECT kaubad.id, nimetus, grupinimi, hind
       FROM kaubad, kaubagrupid
       WHERE kaubad.kaubagrupi_id=kaubagrupid.id");
    //echo $yhendus->error;
    $kask->bind_result($id, $nimetus, $grupinimi, $hind);
    $kask->execute();
    $hoidla=array();
    while($kask->fetch()){
      $kaup=new stdClass();
      $kaup->id=$id;
      $kaup->nimetus=htmlspecialchars($nimetus);
      $kaup->grupinimi=htmlspecialchars($grupinimi);
      $kaup->hind=$hind;
      array_push($hoidla, $kaup);
    }
    return $hoidla;
  }
?>
<pre>
<?php
  print_r(kysiKaupadeAndmed());
?>
</pre>
```

Faili väljund parasjagu baasis olevate andmete põhjal. Käsu print_r tulemus näitab, et väljatrükitud objekt oli massiiv, millel indeksid 0, 1 ja 2. Ning mille igaks elemendiks on objekt klassist stdClass väljadega id, nimetus, grupinimi ja hind parasjagu sissepandud väärtustega.

```
Array
(
    [0] => stdClass Object
        (
            [id] => 1
            [nimetus] => ahjutellis
            [grupinimi] => tellised
            [hind] => 8.20
        )

    [1] => stdClass Object
        (
            [id] => 2
            [nimetus] => fassaaditellis
            [grupinimi] => tellised
            [hind] => 7.50
        )

    [2] => stdClass Object
        (
            [id] => 3
            [nimetus] => bituumenrull
            [grupinimi] => katusematerjal
            [hind] => 520.00
        )

)
```

Kui funktsiooni töö kontrollitud ja õigeks loetud, siis on hea pre-ga tähistatud osa abifunktsioonide failist välja võtta, et see muude failide poolt välja kutsudes segama ei hakkaks.

Järgnevalt kasutame failis kaubaotsing.php loodud abifunktsiooni teenuseid ning näitame tabelis olevad kaubad ekraanile. Kõigepealt tuleb kasutamiseks lugeda require-käsuga abifunktsioonid käivituvasse faili (kaubaotsing.php) sisse. Ning et kaubad oleksid meil lehe väljatrüki juures pidevalt käepärast, selleks küsime funktsiooni poolt väljastatud andmekogumi muutujasse nimega $kaubad.

```
<?php
  require("abifunktsioonid.php");
  $kaubad=kysiKaupadeAndmed();
?>
```

Hiljem andmeid välja trükkides saab siis kaupade massiivi elemendid ükshaaval ette võtta ning iga kauba andmed sobivasse lahtrisse paigutada. Kuna eelnevas funktsioonis on juba kauba nimetusele ja grupinimele pandud ümber htmlspecialchars'i tehtud muudatused, siis võib need väärtused siin rahus otse välja trükkida.

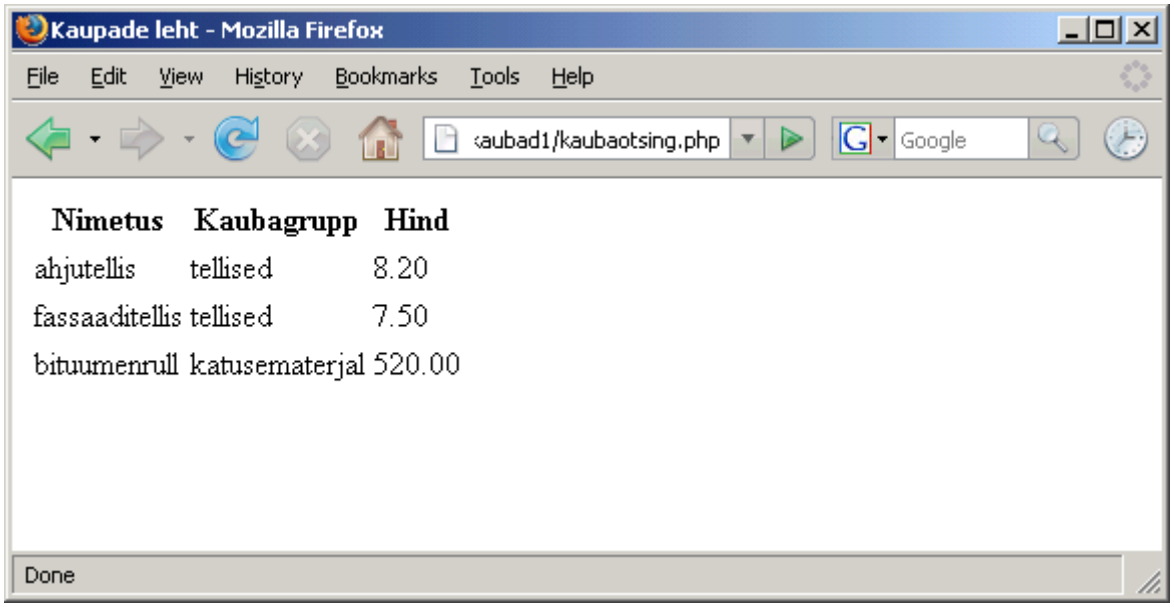
```
       <?php foreach($kaubad as $kaup): ?>
         <tr>
           <td><?=$kaup->nimetus ?></td>
           <td><?=$kaup->grupinimi ?></td>
           <td><?=$kaup->hind ?></td>
         </tr>
       <?php endforeach; ?>
```

Ning kaubaotsingu fail tervikuna

```
<?php
  require("abifunktsioonid.php");
  $kaubad=kysiKaupadeAndmed();
```

```
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
      <title>Kaupade leht</title>
      <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
     <table>
       <tr>
         <th>Nimetus</th>
         <th>Kaubagrupp</th>
         <th>Hind</th>
       </tr>
       <?php foreach($kaubad as $kaup): ?>
         <tr>
           <td><?=$kaup->nimetus ?></td>
           <td><?=$kaup->grupinimi ?></td>
           <td><?=$kaup->hind ?></td>
         </tr>
       <?php endforeach; ?>
     </table>
  </body>
</html>
```

Ja tema töö tulemus arvutiekraanil.



Abifunktsioone on täiendamise korral vaja sageli testida. Pidevalt pre-osa sisse ja välja tõsta on tülikas. Selle asemel võib tingimuslausega kontrollida, et kas abifunktsioonid.php käivitatakse iseseisvalt või mõne teise faili kaudu. Esimesel juhul näidatakse testandmed ekraanile, muidu mitte. Tegeliku veebilehitseja aadressirealt käivitatava faili nime leiab muutujast `$_SERVER["PHP_SELF"]` - koos kogu pika URLiga. Järgnevas käsus tükeldatakse see URL explode abil kaldkriipsu kohtadelt massiivielementideks ning `array_pop` käsu abil võetakse viimane element ehk failinimi ise. Tingimusega kontrollitakse, et kui aadressirealt käivitatava faili nimi on abifunktsioonid.php, siis pannakse pre-märgendi sees olev testosa käima, muidu mitte.

```
  if( array_pop(explode("/", $_SERVER["PHP_SELF"]))=="abifunktsioonid.php"):
?>
<pre>
```

```
<?php
  print_r(kysiKaupadeAndmed());
?>
</pre>
<?php endif ?>
```

Nõnda siis abifunktsioonid.php uuel kujul, kus faili lõpus testosa käivitatakse ainult faili otse veebilehitsejas vaadates. Teise faili kaudu funktsioon sisse lugedes aga muud lehte segavat lõiku ei teki.

```
<?php
  $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");

  function kysiKaupadeAndmed(){
    global $yhendus;
    $kask=$yhendus->prepare("SELECT kaubad.id, nimetus, grupinimi, hind
       FROM kaubad, kaubagrupid
       WHERE kaubad.kaubagrupi_id=kaubagrupid.id");
    //echo $yhendus->error;
    $kask->bind_result($id, $nimetus, $grupinimi, $hind);
    $kask->execute();
    $hoidla=array();
    while($kask->fetch()){
      $kaup=new stdClass();
      $kaup->id=$id;
      $kaup->nimetus=htmlspecialchars($nimetus);
      $kaup->grupinimi=htmlspecialchars($grupinimi);
      $kaup->hind=$hind;
      array_push($hoidla, $kaup);
    }
    return $hoidla;
  }

  //---------------
  if( array_pop(explode("/", $_SERVER["PHP_SELF"]))=="abifunktsioonid.php"):
?>
<pre>
<?php
  print_r(kysiKaupadeAndmed());
?>
</pre>
<?php endif ?>
```

Ülesandeid

* Tee näited läbi
* Koosta tabel maakondade jaoks (id, maakonnanimi, maakonnakeskus)
* Koosta tabel inimeste andmete jaoks, maakonnale viidatakse võõrvõtme abil (id, eesnimi, perekonnanimi, maakonna_id)
* Näita tabelis olevate inimeste andmed koos nende paiknemise maakonna nimedega ekraanile. Andmebaasipäring paiguta eraldi failis olevasse funktsiooni nagu eelnevas näites.

Sortimine

Suuremast andmehulgast sobiva leidmisel aitab sageli andmete sortimine soovitud tunnuse järgi.

Telefoniraamatust näiteks nime järgi numbri leidmine on suhteliselt lihtne toimetus. Numbri järgi kindlaks tegemine, et kellele see kuulub, nõuab aga palju rohkem vaeva. Kõik lihtsalt sellepärast, et telefoniraamat on sorditud nimede järgi.

Järgnevas näites täiendame kaupade loetelu otsinguvõimalusega. Andmebaasipõhises lehestikus ühe tulba järgi sortida on lihtne - tuleb vaid SELECT-lause lõppu kirjutada ORDER BY tulbanimi ning andmed saabuvadki sobivas järjestuses. Siin aga pakume kasutajale võimalust sortida andmed nimetuse, grupinime või hinna järgi - vajutades vastava tulba pealkirjale. Sellisel juhul tuleb vastavalt kasutaja valikule saada SQL-lausesse sobiva tulba nimi.

Koodi täiendades on mugav, kui õnnestub teha nii, et olemasolevad funktsioonid ka vanades kasutuskohtades tööle jäävad. Vanas variandis ei andnud me funktsioonile kysiKaupadeAndmed ette midagi - väljastati lihtsalt baasis leiduvad kaubad. Nüüd aga soovime, et funktsioon väljastaks kaubad sordituna etteantud tulba järgi. Funktsioonile saab andmeid ette anda parameetrite kaudu. Kui lihtsalt lisada funktsioonile kohustuslik parameeter sortimistulba määramiseks, siis ilma selle parameetrita ei saaks funktsiooni enam välja kutsuda. Siin näites pole sellest suurt muret, sest täiendus tuleks viia sisse vaid ühte kohta - sinna kus kaubaotsing.php failis andmeid teise faili järgi küsitakse. Mõnes suuremas rakenduses võidakse sama funktsiooni kasutada kümnetes ja sadades kohtades. Ning vahel kasutatakse abifunktsioonide teeke kogumikuna mõne teise rakenduse juures, millest abifunktsioonide loojal ei pruugi aimugi olla. Sellisel juhul tekitaks funktsioonile parameetrite lisamine paljudes kohtades. Mõnikord pole sellest pääsu - eriti kui jõudluse või turvalisuse huvides funktsioonilt osa oskusi maha võetakse. Parameetrite lisamisel aga saab üldjuhul muudele koodifailidele murede tekitamisest hoiduda pannes loodud parameetrile vaikimisi väärtuse. Siin näiteks siis lisatakse parameeter $sorttulp. Ning juhul, kui seda parameetrit funktsioonile ette ei anta (nt. siiamaani loodud kasutavate failide korral), sellisel juhul antakse sortimistulbale vaikimisi väärtuseks "nimetus", st. et väljastatavad kaubad sorditakse nimetuse järgi tähestikulisse järjekorda.

```
function kysiKaupadeAndmed($sorttulp="nimetus")
```

Veebist saabuvaid andmeid ei või kunagi usaldada - näpuvea või pahatahtlikkuse tõttu võib sealt saabuvates päringutes sisalduda tont-teab-mida. Et saabuvad andmed me rakendust rikkuda ei saaks, tasub ära märkida, milliste tulpade järgi on lubatud sortida - praegu siis pikkus, grupinimi ja hind.

```
    $lubatudtulbad=array("nimetus", "grupinimi", "hind");
```

Kui juhtub, et palutakse andmeid järjestada mõne muu (seni olematu) tunnuse järgi, siis lihtsalt katkestatakse funktsiooni töö veateatega. Kas ja mida väljakutsuv programm selle veateatega peale hakkab, on juba tema mure. Vähemasti ei satu sobimatud andmed edasi me SQL-lausessese andmebaasi sisse pahandust tegema.

```
    if(!in_array($sorttulp, $lubatudtulbad)){
      return "lubamatu tulp";
    }
```

SELECT-lause on lihtne ja suhteliselt sarnane eelmise näite variandiga. Lihtsalt lõppu on tulnud määrang "ORDER BY $sorttulp", mis siis vastavalt päringule asendatakse kas nimetuse, grupinime või hinnaga.

```
    $kask=$yhendus->prepare("SELECT kaubad.id, nimetus, grupinimi, hind
       FROM kaubad, kaubagrupid
       WHERE kaubad.kaubagrupi_id=kaubagrupid.id
       ORDER BY $sorttulp");
```

Väljakommenteeritud rida vea kuvamiseks on mugav vahend jälgimaks, kui miski ei taha tööle hakata. Selle käsu kaudu saab vajadusel välja trükkida MySQLi poolt avastatud vigade selgitused.

Töötavasse lõppversiooni pole koodi soovitatav neid käsklusi sisse jätta - serveripoolsed veateated võivad hõlbustada häkkerite tööl. Enese jaoks arenduse käigus vigade kohta aimu saamiseks on selliste veateadete nägemine aga igati omal kohal.

```
    //echo $yhendus->error;
```

Ülejäänud koodiosa jäi samaks - kergema kasutamise ja kopeerimise huvides on ta aga ka siia pandud.

```
<?php
  $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");

  function kysiKaupadeAndmed($sorttulp="nimetus"){
    global $yhendus;
    $lubatudtulbad=array("nimetus", "grupinimi", "hind");
    if(!in_array($sorttulp, $lubatudtulbad)){
      return "lubamatu tulp";
    }
    $kask=$yhendus->prepare("SELECT kaubad.id, nimetus, grupinimi, hind
       FROM kaubad, kaubagrupid
       WHERE kaubad.kaubagrupi_id=kaubagrupid.id
       ORDER BY $sorttulp");
    //echo $yhendus->error;
    $kask->bind_result($id, $nimetus, $grupinimi, $hind);
    $kask->execute();
    $hoidla=array();
    while($kask->fetch()){
      $kaup=new stdClass();
      $kaup->id=$id;
      $kaup->nimetus=htmlspecialchars($nimetus);
      $kaup->grupinimi=htmlspecialchars($grupinimi);
      $kaup->hind=$hind;
      array_push($hoidla, $kaup);
    }
    return $hoidla;
  }

  //---------------
  if( array_pop(explode("/", $_SERVER["PHP_SELF"]))=="abifunktsioonid.php"):
?>
<pre>
<?php
  print_r(kysiKaupadeAndmed("hind"));
?>
</pre>
<?php endif ?>
```

Loodud abifunktsioone kasutame endiselt andmeotsingu lehel. Lehe algusesse tuleb juurde kontroll, kas aadressireal olevaks sort-parameetriks väärtuseks on saabunud midagi. Kui jah, siis palutakse andmed sordituna selle tunnuse järgi, muul juhul palutakse lihtsalt andmeid, kusjuures praegu siis sorditakse need funktsiooni vaikimisi parameetri ehk nimetuse järgi.

```
  if(isSet($_REQUEST["sort"])){
     $kaubad=kysiKaupadeAndmed($_REQUEST["sort"]);
  } else {
     $kaubad=kysiKaupadeAndmed();
  }
```

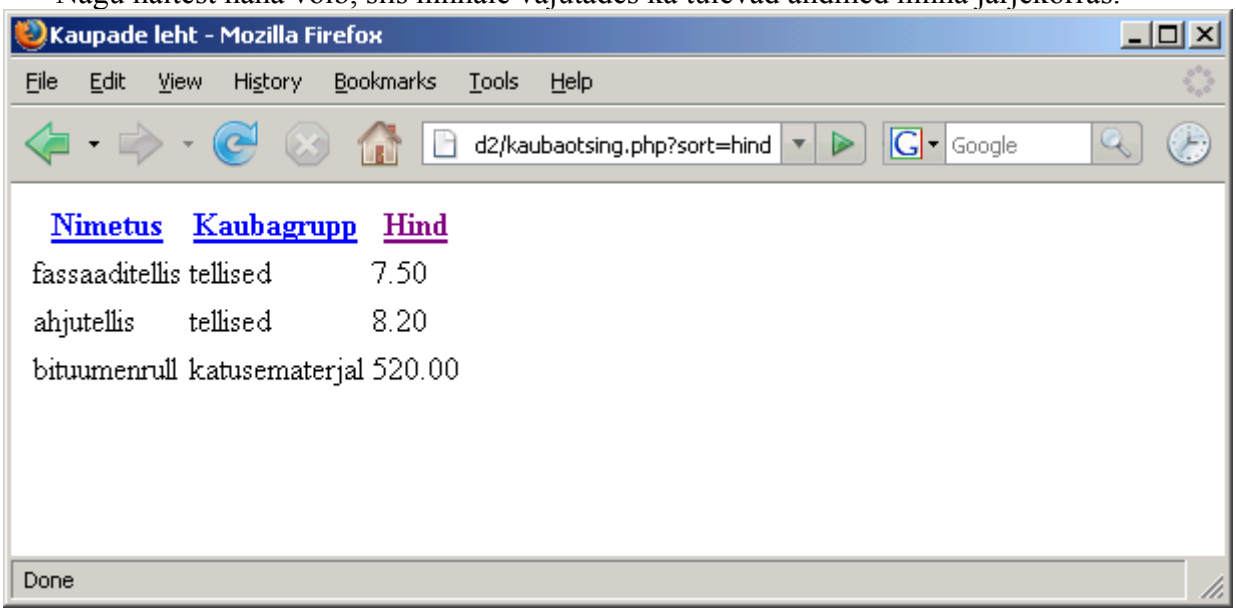
Sortimistulba mugavaks valimiseks tuleb tulbad vastavateks viideteks teha, igaüks saadab kaubaotsing.php lehele parameetri sort sobiva väärtusega.

```
      <tr>
        <th><a href="kaubaotsing.php?sort=nimetus">Nimetus</a></th>
        <th><a href="kaubaotsing.php?sort=grupinimi">Kaubagrupp</a></th>
        <th><a href="kaubaotsing.php?sort=hind">Hind</a></th>
      </tr>
```

Muu fail ikka eelmise näitega sarnane.

```
<?php
  require("abifunktsioonid.php");
  if(isSet($_REQUEST["sort"])){
     $kaubad=kysiKaupadeAndmed($_REQUEST["sort"]);
  } else {
     $kaubad=kysiKaupadeAndmed();
  }
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
      <title>Kaupade leht</title>
      <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
     <table>
       <tr>
         <th><a href="kaubaotsing.php?sort=nimetus">Nimetus</a></th>
         <th><a href="kaubaotsing.php?sort=grupinimi">Kaubagrupp</a></th>
         <th><a href="kaubaotsing.php?sort=hind">Hind</a></th>
       </tr>
       <?php foreach($kaubad as $kaup): ?>
         <tr>
           <td><?=$kaup->nimetus ?></td>
           <td><?=$kaup->grupinimi ?></td>
           <td><?=$kaup->hind ?></td>
         </tr>
       <?php endforeach; ?>
     </table>
  </body>
</html>
```

Nagu näitest näha võib, siis hinnale vajutades ka tulevad andmed hinna järjekorras.



Ülesandeid

* Tee näide läbi
* Pane sarnane sortimisvõimalus tööle eelneva näite juures tehtud inimeste ja nende paiknemismaakondade rakenduse juures.

Otsimine

Sortimine aitab andmete juures, kus tulba tekst algab otsitava väärtusega. Kui aga otsitav lõik võib asuda teksti sees, siis paljas tekstide või arvude tähestikulisse või kasvavasse järjekorda seadmine ei aita soovitud rida leida. Sellisel puhul on abiliseks otsimisvõimalus. Tänapäevastel veebilehestikel on see paljudel sisse ehitatud. Ning kui ka pole, siis mõnevõrra saab toetuda ka Google otsingule, kirjutades otsingu ette site:masinanimi. Näiteks otsing "site:minitorn.tlu.ee php" annab minitorn.tlu.ee masinas PHPga seotud lehed - niipalju, kui otsimootor neid sealt leidnud on.

Oma särk aga ikka ihule kõige lähemal ning isetehtud otsingu peale võib ka ehk kindlam olla, et ta just vajalikest kohtadest otsib. Funktsioonile tuleb siis juurde järjekordne parameeter. Ning kui otsisõna ei määrata, siis selle väärtuseks saab tühi tekst - ehk osa, mis igas tekstis leidub ning välja kuvatakse kõik andmed.

```
  function kysiKaupadeAndmed($sorttulp="nimetus", $otsisona=""){
```

PHP murelapseks on üle veebi saabuvate sümbolite varjestamine langjoontega. Toimetus oli vajalik saabuvate andmete paigutamiseks MySQLi päringusse. MySQLi seda aga üldjuhul ei vaja ning serveri konfiguratsioonis võib olla erisümbolite varjestus maha võetud. Et küsimärgiga parameeter mugavalt tekstisobivust otsivasse LIKE-lausesse ei lähe, tuleb siin otsitav parameeter otse SQLi paigutada. Kui ei tea, kas varjestavad langjooned on ees või mitte, siis üheks võimaluseks on need maha võõta stripslashes-käsuga ning hiljem uuesti mysql_real_escape_string-käsuga tagasi panna. Puuduva varjestuse puhul pole ka stripslashes-il midagi maha võtta, teine peale paneb ikka, et oleks andmebaasilauses kõik korrektne.

```
    $otsisona=mysql_real_escape_string(stripslashes($otsisona));
```

Otsinguosa saab lause WHERE-ossa olemasolevale tingimusele otsa liita. Siin otsime korraga nimetuse ja grupinime seest - ükskõik kummas otsitav lõik leitakse, sobiv rida kuvatakse ikka tulemusena. Et kaupade tabeli kaubagrupi_id järgi kaubagrupi tabeli id-le viitamist ikka alati kontrollitaks, selleks peab ORidega kahe tulba järgi otsing eraldi sulgudes olema.

```
    $kask=$yhendus->prepare("SELECT kaubad.id, nimetus, grupinimi, hind
       FROM kaubad, kaubagrupid
       WHERE kaubad.kaubagrupi_id=kaubagrupid.id
        AND (nimetus LIKE '%$otsisona%' OR grupinimi LIKE '%$otsisona%')
       ORDER BY $sorttulp");
```

Ning fail tervikuna.

```
<?php
  $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");

  function kysiKaupadeAndmed($sorttulp="nimetus", $otsisona=""){
    global $yhendus;
    $lubatudtulbad=array("nimetus", "grupinimi", "hind");
    if(!in_array($sorttulp, $lubatudtulbad)){
      return "lubamatu tulp";
    }
    $otsisona=mysql_real_escape_string(stripslashes($otsisona));
    $kask=$yhendus->prepare("SELECT kaubad.id, nimetus, grupinimi, hind
       FROM kaubad, kaubagrupid
       WHERE kaubad.kaubagrupi_id=kaubagrupid.id
        AND (nimetus LIKE '%$otsisona%' OR grupinimi LIKE '%$otsisona%')
       ORDER BY $sorttulp");
    //echo $yhendus->error;
    $kask->bind_result($id, $nimetus, $grupinimi, $hind);
    $kask->execute();
```

```
    $hoidla=array();
    while($kask->fetch()){
      $kaup=new stdClass();
      $kaup->id=$id;
      $kaup->nimetus=htmlspecialchars($nimetus);
      $kaup->grupinimi=htmlspecialchars($grupinimi);
      $kaup->hind=$hind;
      array_push($hoidla, $kaup);
    }
    return $hoidla;
  }

  //---------------
  if( array_pop(explode("/", $_SERVER["PHP_SELF"]))=="abifunktsioonid.php"):
?>
<pre>
<?php
  print_r(kysiKaupadeAndmed("hind", "fass\\aad"));
?>
</pre>
<?php endif ?>
```

Kaubaotsingu failis tuleb juurde koht otsinguteksti sisestamiseks. Ning sisestuselemendil peab ümber olema vorm koos action'iga määramaks, kuhu faili sisestatud andmed saata.

```
   <form action="kaubaotsing.php">
     Otsi: <input type="text" name="otsisona" />
        ...
   </form>
```

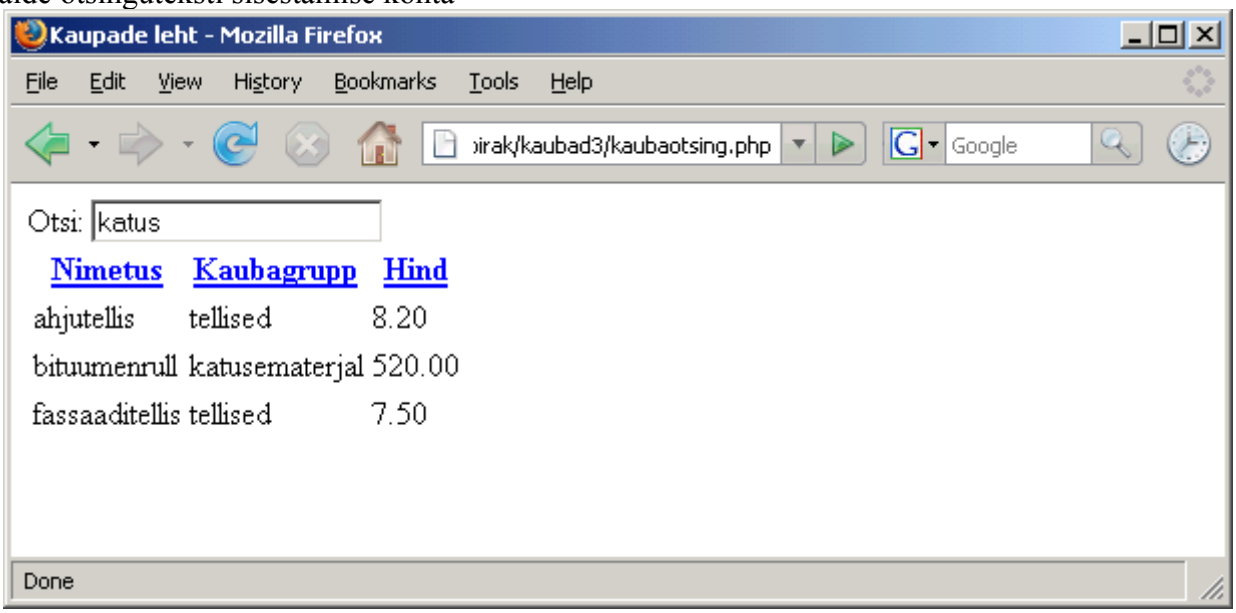
Kuna veebilehe puhul ei ole nüüd enam teada, kas üldse ja kumb parameeter on väärtustatud, siis on heaks mooduseks neile ka PHP-lehel vaikeväärtused anda. Hädapärast võiks töötada ka vorm

`$kaubad=kysiKaupadeAndmed($_REQUEST["sort"], $_REQUEST["otsisona"])`, aga kui vastavad parameetrid lehel puudu (nt. esimest korda avades), siis võib PHP server vastava konfiguratsiooni puhul hakata hoiatusi andma, et küsitakse olematuid väärtusi massiivist. Nõnda siis järgmine lehekuju koos algusega ikka kindlam.

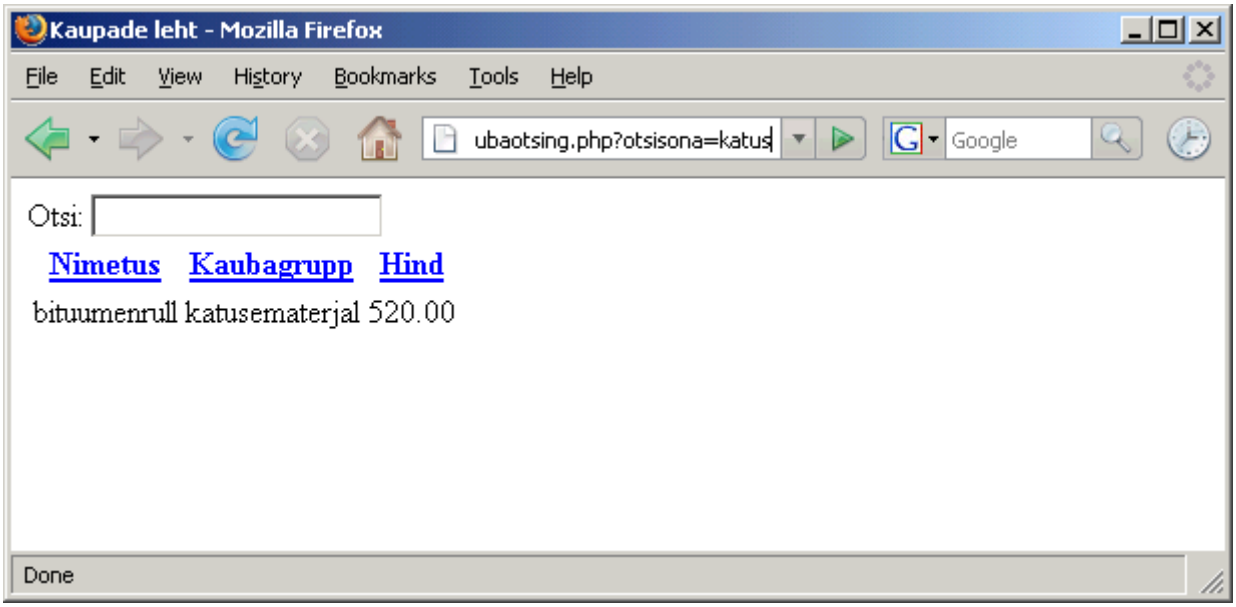
```
<?php
  require("abifunktsioonid.php");
  $sorttulp="nimetus";
  $otsisona="";
  if(isSet($_REQUEST["sort"])){
     $sorttulp=$_REQUEST["sort"];
  }
  if(isSet($_REQUEST["otsisona"])){
     $otsisona=$_REQUEST["otsisona"];
  }
  $kaubad=kysiKaupadeAndmed($sorttulp, $otsisona);
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
      <title>Kaupade leht</title>
      <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
   <form action="kaubaotsing.php">
     Otsi: <input type="text" name="otsisona" />
     <table>
       <tr>
         <th><a href="kaubaotsing.php?sort=nimetus">Nimetus</a></th>
         <th><a href="kaubaotsing.php?sort=grupinimi">Kaubagrupp</a></th>
         <th><a href="kaubaotsing.php?sort=hind">Hind</a></th>
```

```
      </tr>
      <?php foreach($kaubad as $kaup): ?>
        <tr>
          <td><?=$kaup->nimetus ?></td>
          <td><?=$kaup->grupinimi ?></td>
          <td><?=$kaup->hind ?></td>
        </tr>
      <?php endforeach; ?>
    </table>
  </form>
 </body>
</html>
```

Näide otsinguteksti sisestamise kohta



Ning aadressiriba peale jõudnud sõna järgi otsingutulemus.



Ülesandeid

* Tee näide läbi
* Lisa eelnevale inimeste andmete rakendusele otsing teksti järgi

Haldamine

Seni sortisime ja otsisime neid andmeid, mis juba andmetabelis olemas. Kui andmete sisestamine käib sisevõrgus oleva süsteemi kaudu, siis võibki veebiliides nõnda lihtsa osaga piirduda. Kui aga veebi kaudu tuleb ka andmeid lisada ja muuta, siis vaja mõnevõrra rohkem koodi kirjutada.

Kuna kaubagrupid on määratud eraldi tabelis ning võimaluse korral paigutatakse kaup juba olemasolevasse gruppi, siis paremaks mooduseks on kaubagrupi nime mitte tekstina sisse kirjutada, vaid olemasolevate hulgast rippmenüüst valida. Ning kui sobiv grupp puudub, saab selle pigem kusagilt eraldi kohast juurde panna - nii et seda edaspidi kõikide lisatavate kaupade juures kasutada saab.

Kood sellise lisamisvormi loomiseks võiks välja näha nagu allpool. Nimetus ja nind tulevad tekstiväljadest. Tasub märkida, et nii kauba lisamise nupule kui ka grupi lisamise nupule on pandud nimi. Sellisel puhul saab serverisse saadetud andmete põhjalt vaadata, et millisele nupule vajutati. Vormi action-parameetris määratud veebilehele saadetakse kaasa nupu nimega parameeter, mille väärtuseks on nupu peale kirjutatud tekst.

Rippmenüü loomiseks on tehtud eraldi alamprogramm. Kuna tegemist suhteliselt levinud ja korduva toiminguga, siis on otstarbekas SQL-lause põhjal rippmenüü loomine eraldi alamprogrammi kirjutada ning seda hiljem vajalikes kohtades kasutada. Üks põhjus, et nõnda vähem koodi kirjutada. Teine põhjus, et kui nõnda saab menüü loomise korra viisakalt valmis tehtud, siis järgmistel kordadel pole enam vaja karta, et äkki sinna mõni näpuviga kergesti sisse satub.

```
<form action="kaubahaldus.php">
  <h2>Kauba lisamine</h2>
  <dl>
    <dt>Nimetus:</dt>
    <dd><input type="text" name="nimetus" /></dd>
    <dt>Kaubagrupp:</dt>
    <dd><?php
```

```
        echo looRippMenyy("SELECT id, grupinimi FROM kaubagrupid",
                          "kaubagrupi_id");
      ?>
      </dd>
      <dt>Hind:</dt>
      <dd><input type="text" name="hind" /></dd>
    </dl>
    <input type="submit" name="kaubalisamine" value="Lisa kaup" />
    <h2>Grupi lisamine</h2>
    <input type="text" name="uuegrupinimi" />
    <input type="submit" name="grupilisamine" value="Lisa grupp" />
  </form>
```

Rippmenüü loomine ise ikka abifunktsioonide failis. Funktsioonile antakse ette kaks parameetrit. Esimene on SQLi SELECT-lause, millest väljastatud tabel annab rippmenüüle valiku id-d ja nähtavad väärtused. Teiseks parameetriks on select-elemendi nimi HTMLi vormis. HTMLi tekst lihtsalt pannakse jupi kaupa kokku ning salvestatakse muutujas $tulemus. Lõpuks antakse sealt ka funktsioonist välja.

```
function looRippMenyy($sqllause, $valikunimi){
   global $yhendus;
   $kask=$yhendus->prepare($sqllause);
   $kask->bind_result($id, $sisu);
   $kask->execute();
   $tulemus="<select name='$valikunimi'>";
   while($kask->fetch()){
     $tulemus.="<option value='$id'>$sisu</option>";
   }
   $tulemus.="</select>";
   return $tulemus;
}
```

Kuna püüame halduslehe enese võimalikult programmikoodist puhta hoida, tuleb abifunktsioone veel mõned juurde kirjutada. Grupi lisamiseks käsklus ühe grupi lisamiseks kaubagruppide tabelisse.

```
function lisaGrupp($grupinimi){
   global $yhendus;
   $kask=$yhendus->prepare("INSERT INTO kaubagrupid (grupinimi)
                    VALUES (?)");
   $kask->bind_param("s", $grupinimi);
   $kask->execute();
}
```

Kauba lisamiseks käsklus kaupade tabelisse rea panekuks. Kusjuures eeldatakse, et parameetrina juba tuleb kaubagrupi_id, kuhu sisse lisatav kaup kuulub. Kuna me eelnev select-valik sai nõnda tehtud, et näha on grupi nimi, kuid serverisse saadetakse kaubagrupi id, siis on nende andmete salvestamine lihtne.

```
function lisaKaup($nimetus, $kaubagrupi_id, $hind){
   global $yhendus;
   $kask=$yhendus->prepare("INSERT INTO
     kaubad (nimetus, kaubagrupi_id, hind)
     VALUES (?, ?, ?)");
   $kask->bind_param("sid", $nimetus, $kaubagrupi_id, $hind);
   $kask->execute();
}
```

Juurde ka kauba kustutamine. Kusjuures nagu ikka, on viisakas enne küsida, kas ikka tahetakse vastavat kaupa kustutada - et poleks kogemata vajutuse tõttu andmed kaduma läinud. Kõigepealt tuleb siis iga kauba ette vastav kustutusviide teha koos Javaskripti abil küsimisega.

```
          <td><a href="kaubahaldus.php?kustutusid=<?=$kaup->id ?>"
                 onclick="return confirm('Kas ikka soovid kustutada?')">x</a>
          </td>
          <td><?=$kaup->nimetus ?></td>
          <td><?=$kaup->grupinimi ?></td>
          <td><?=$kaup->hind ?></td>
```

Lehe uuel laadimisel kontrollitakse, et kas äkki on saadetud kaasa kustutusid, mille järgi kauba kustutamine otsustada. Kui jah, siis kutsutakse abifunktsioonide alt välja vastav käsklus.

```
if(isSet($_REQUEST["kustutusid"])){
   kustutaKaup($_REQUEST["kustutusid"]);
}
```

Käsklus ise loogiline - tabelist võetakse ära see rida, mille id kattub saadetud kustutatava kauba id-ga.

```
function kustutaKaup($kauba_id){
   global $yhendus;
   $kask=$yhendus->prepare("DELETE FROM kaubad WHERE id=?");
   $kask->bind_param("i", $kauba_id);
   $kask->execute();
}
```

Ülesandeid

* Tee näide läbi
* Võimalda veebi kaudu lisada maakondi.
* Võimalda veebi kaudu lisada inimeste andmeid, valides rippmenüüst, kus maakonnas nad asuvad.

Andmete muutmine

Mitme seotud andmetabeli pealt kokku ehitatud veebihaldusliidese keerukaimaks kohaks on valitava tunnuse muutmine rippmenüü kaudu - et muutmisel oleks olemasolev valik ette keritud ning võimalik muutus ka õigesti kirja läheks.

Muus osas on muutmine suhteliselt sarnane konspekti algusosas oleva teadetetabeli andmete muutmisega. Vaid SQL-päringud on HTMLi koodist välja viidud ning andmeid näidatakse abifunktsioonid.php-failist tulnud muutujate kaudu.

Kaupade loetelu kuvamisel on kõigepealt iga kauba ees viide kustutamiseks või muutmiseks. Andmete kuvamisel vaadatakse, kas aadressiribalt on saabunud parameeter nimega muutmisid ning kas muudetava kauba id kattub parasjagu näidatava kauba id-ga.

```
          <?php if(isSet($_REQUEST["muutmisid"]) &&
intval($_REQUEST["muutmisid"])==$kaup->id): ?>
```

Kui jah, siis kuvatakse rida tavalisega võrreldes erinevalt, sätitakse andmed sellisele kujule, et neid veebilehelt muuta saab. Nimetus ja hind tulevad nähtavale tekstivälja paigutatult. Kaubagrupp aga paigutatakse sellisesse rippmenüüsse, kus sobiv valik on juba ette keeratud. Selleks tuleb vastavat abifunktsiooni mõnevõrra täiendada, millest edaspidi allpool.

```
    <table>
      <tr>
        <th>Haldus</th>
```

```
      <th>Nimetus</th>
      <th>Kaubagrupp</th>
      <th>Hind</th>
    </tr>
    <?php foreach($kaubad as $kaup): ?>
      <tr>
       <?php if(isSet($_REQUEST["muutmisid"]) &&
                    intval($_REQUEST["muutmisid"])==$kaup->id): ?>
        <td>
          <input type="submit" name="muutmine" value="Muuda" />
          <input type="submit" name="katkestus" value="Katkesta" />
          <input type="hidden" name="muudetudid" value="<?=$kaup->id ?>" />
        </td>
        <td><input type="text" name="nimetus" value="<?=$kaup->nimetus ?>" /></td>
        <td><?php
          echo looRippMenyy("SELECT id, grupinimi FROM kaubagrupid",
                            "kaubagrupi_id", $kaup->kaubagrupi_id);
        ?></td>
        <td><input type="text" name="hind" value="<?=$kaup->hind ?>" /></td>
       <?php else: ?>
        <td><a href="kaubahaldus.php?kustutusid=<?=$kaup->id ?>"
               onclick="return confirm('Kas ikka soovid kustutada?')">x</a>
            <a href="kaubahaldus.php?muutmisid=<?=$kaup->id ?>">m</a>
        </td>
        <td><?=$kaup->nimetus ?></td>
        <td><?=$kaup->grupinimi ?></td>
        <td><?=$kaup->hind ?></td>
       <?php endif ?>
      </tr>
    <?php endforeach; ?>
  </table>
```

Kauba andmete näitamise loetellu tuleb parameetriks juurde `kaubagrupi_id`, et oleks võimalik selle abil muutmise ajal rippmenüüst sobivat valikut näidata.

```
function kysiKaupadeAndmed($sorttulp="nimetus", $otsisona=""){
  global $yhendus;
  $lubatudtulbad=array("nimetus", "grupinimi", "hind");
  if(!in_array($sorttulp, $lubatudtulbad)){
    return "lubamatu tulp";
  }
  $otsisona=mysql_real_escape_string(stripslashes($otsisona));
  $kask=$yhendus->prepare("SELECT kaubad.id, nimetus, grupinimi, kaubagrupi_id, hind
     FROM kaubad, kaubagrupid
     WHERE kaubad.kaubagrupi_id=kaubagrupid.id
      AND (nimetus LIKE '%$otsisona%' OR grupinimi LIKE '%$otsisona%')
     ORDER BY $sorttulp");
  $kask->bind_result($id, $nimetus, $grupinimi, $kaubagrupi_id, $hind);
  $kask->execute();
  $hoidla=array();
  while($kask->fetch()){
    $kaup=new stdClass();
    $kaup->id=$id;
    $kaup->nimetus=htmlspecialchars($nimetus);
    $kaup->grupinimi=htmlspecialchars($grupinimi);
    $kaup->kaubagrupi_id=$kaubagrupi_id;
    $kaup->hind=$hind;
    array_push($hoidla, $kaup);
  }
  return $hoidla;
}
```

Rippmenüüle tuleb juurde täiendav parameeter nimega $valitudid. Nagu eelnevalt otsimise juures nii ka siin võimaldab lisandunud parameetrile vaikeväärtuse jätmine kasutada funktsiooni samaaegselt nii senini kehtinud kahe parameetriga kohas (SELECT-lause ning valiku HTMLi sees olev nimi) kui ka uues kohas, kus antakse ette id, millist rida andmete seast ette kerida. Kui id-parameeter puudub, siis antakse sellele vaikimisi väärtuseks tühi tekst. Ning kui sellise väärtusega id-d pole (üldjuhul ei tohiks olla), siis lihtsalt eraldi ei valitagi midagi välja, rippmenüü tuleb

nähtavale nii nagu ennegi. Kui aga funktsioonile anti ettekeritav id ning sarnane id leidub ka näidatavate andmete seas, siis lisatakse vastavale valikule parameeter `selected='selected'`. Koodi mugavamaks kirjutamiseks on see toimetus jagatud kolme ritta. Vaikimisi pannakse lisandit hoidvale muutujale väärtuseks tühi tekst, st. et selle muutuja väärtuse lisamine ei muuda väljundit kuhugi. Järgmise tingimusega vaadatakse, et kas sinna on selected-atribuut põhjust sisse kirjutada. Ning kui kirjutati, siis jõuab see kolmandal real ilusti ka vastava option-rea kirjeldusse.

```
      $lisand="";
      if($id==$valitudid){$lisand=" selected='selected'";}
      $tulemus.="<option value='$id' $lisand >$sisu</option>";
```

Alamprogramm uuel kujul tervikuna:

```
function looRippMenyy($sqllause, $valikunimi, $valitudid=""){
   global $yhendus;
   $kask=$yhendus->prepare($sqllause);
   $kask->bind_result($id, $sisu);
   $kask->execute();
   $tulemus="<select name='$valikunimi'>";
   while($kask->fetch()){
     $lisand="";
     if($id==$valitudid){$lisand=" selected='selected'";}
     $tulemus.="<option value='$id' $lisand >$sisu</option>";
   }
   $tulemus.="</select>";
   return $tulemus;
}
```

Ning nende toimetuste tulemusena tekib siis valitud rida juba muudetaval kujul ekraanile. Muutused küll veel kuhugile ei jõua, aga eks see tuleb siis järgmise sammuna üles tähendada.

Haldus		Nimetus	Kaubagrupp	Hind
Muuda	Katkesta	ahjutellis	tellised	8.20
x m		bituumenrull	katusematerjal	525.00
x m		fassaaditellis	tellised	7.50

Kauba andmete muutmiseks lisatakse taas sobiv funktsioon. Kindlasti tuleb muutmise juurest kaasa saata muudetava kauba `id`. Selleks oli ennist vormi sees rida

```
<input type="hidden" name="muudetudid" value="<?=$kaup->id ?>" />
```

kus see `id` kaasa pandi.

Edasi koostatakse abifunktsioonide alla UPDATE-lause, kus id-numbriga määratud real vanad andmed uutega üle kirjutatakse.

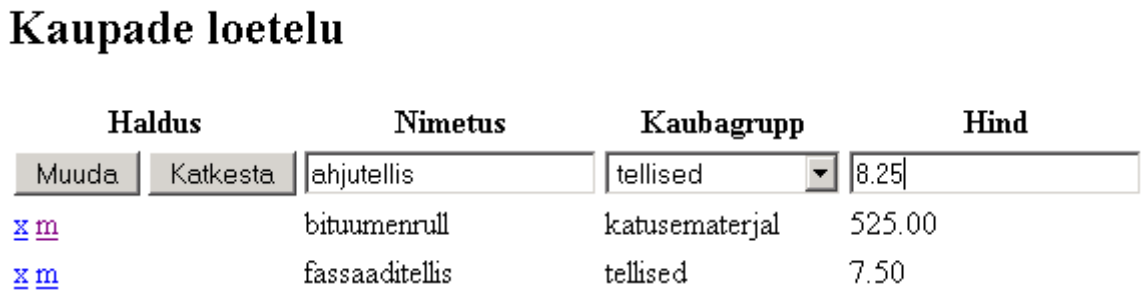
```
function muudaKaup($kauba_id, $nimetus, $kaubagrupi_id, $hind){
   global $yhendus;
   $kask=$yhendus->prepare("UPDATE kaubad SET nimetus=?, kaubagrupi_id=?, hind=?
                    WHERE id=?");
   $kask->bind_param("sidi", $nimetus, $kaubagrupi_id, $hind, $kauba_id);
   $kask->execute();
}
```

Kaupade lehe juures tuleb algusesse juurde kontroll - et kui vajutati muutmisnupule, siis

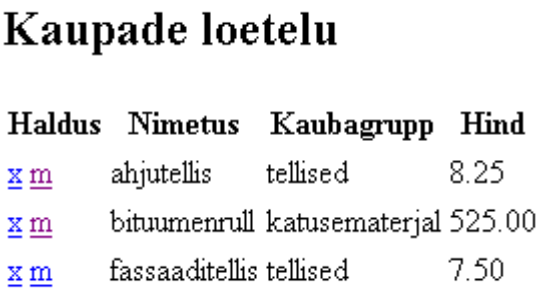
käivitatakse muutmisfunktsioon koos ette antud uute andmetega.

```
  if(isSet($_REQUEST["muutmine"])){
     muudaKaup($_REQUEST["muudetudid"], $_REQUEST["nimetus"],
                          $_REQUEST["kaubagrupi_id"], $_REQUEST["hind"]);
  }
```

Ning võibki uue hinna kirjutada ja muudki parandused vajadusel siise viia.



Muutmisnupu vajutamise järgsel lehe avamisel ongi uuel kujul andmed siis nähtavad.



Järeltegemise hõlbustamiseks siis halduseks vajalike failide koodid tervikuna.

kaubahaldus.php

```
<?php
  require("abifunktsioonid.php");
  if(isSet($_REQUEST["grupilisamine"])){
    lisaGrupp($_REQUEST["uuegrupinimi"]);
    header("Location: kaubahaldus.php");
    exit();
  }
  if(isSet($_REQUEST["kaubalisamine"])){
    lisaKaup($_REQUEST["nimetus"], $_REQUEST["kaubagrupi_id"], $_REQUEST["hind"]);
    header("Location: kaubahaldus.php");
    exit();
  }
  if(isSet($_REQUEST["kustutusid"])){
     kustutaKaup($_REQUEST["kustutusid"]);
  }
  if(isSet($_REQUEST["muutmine"])){
     muudaKaup($_REQUEST["muudetudid"], $_REQUEST["nimetus"],
                               $_REQUEST["kaubagrupi_id"], $_REQUEST["hind"]);
  }
  $kaubad=kysiKaupadeAndmed();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
      <title>Kaupade leht</title>
      <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
```

```
  </head>
  <body>
   <form action="kaubahaldus.php">
     <h2>Kauba lisamine</h2>
     <dl>
       <dt>Nimetus:</dt>
       <dd><input type="text" name="nimetus" /></dd>
       <dt>Kaubagrupp:</dt>
       <dd><?php
         echo looRippMenyy("SELECT id, grupinimi FROM kaubagrupid",
                           "kaubagrupi_id");
       ?>
       </dd>
       <dt>Hind:</dt>
       <dd><input type="text" name="hind" /></dd>
     </dl>
     <input type="submit" name="kaubalisamine" value="Lisa kaup" />
     <h2>Grupi lisamine</h2>
     <input type="text" name="uuegrupinimi" />
     <input type="submit" name="grupilisamine" value="Lisa grupp" />
   </form>
   <form action="kaubahaldus.php">
     <h2>Kaupade loetelu</h2>
     <table>
       <tr>
         <th>Haldus</th>
         <th>Nimetus</th>
         <th>Kaubagrupp</th>
         <th>Hind</th>
       </tr>
       <?php foreach($kaubad as $kaup): ?>
         <tr>
          <?php if(isSet($_REQUEST["muutmisid"]) &&
                            intval($_REQUEST["muutmisid"])==$kaup->id): ?>
           <td>
             <input type="submit" name="muutmine" value="Muuda" />
             <input type="submit" name="katkestus" value="Katkesta" />
             <input type="hidden" name="muudetudid" value="<?=$kaup->id ?>" />
           </td>
           <td><input type="text" name="nimetus" value="<?=$kaup->nimetus ?>" /></td>
           <td><?php
             echo looRippMenyy("SELECT id, grupinimi FROM kaubagrupid",
                                        "kaubagrupi_id", $kaup->kaubagrupi_id);
           ?></td>
           <td><input type="text" name="hind" value="<?=$kaup->hind ?>" /></td>
          <?php else: ?>
           <td><a href="kaubahaldus.php?kustutusid=<?=$kaup->id ?>"
                  onclick="return confirm('Kas ikka soovid kustutada?')">x</a>
               <a href="kaubahaldus.php?muutmisid=<?=$kaup->id ?>">m</a>
           </td>
           <td><?=$kaup->nimetus ?></td>
           <td><?=$kaup->grupinimi ?></td>
           <td><?=$kaup->hind ?></td>
          <?php endif ?>
         </tr>
       <?php endforeach; ?>
     </table>
   </form>
  </body>
</html>
```

ning abifunktsioonid.php

```
<?php
  $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");

  function kysiKaupadeAndmed($sorttulp="nimetus", $otsisona=""){
    global $yhendus;
```

```
    $lubatudtulbad=array("nimetus", "grupinimi", "hind");
    if(!in_array($sorttulp, $lubatudtulbad)){
      return "lubamatu tulp";
    }
    $otsisona=mysql_real_escape_string(stripslashes($otsisona));
    $kask=$yhendus->prepare("SELECT kaubad.id, nimetus, grupinimi, kaubagrupi_id, hind
       FROM kaubad, kaubagrupid
       WHERE kaubad.kaubagrupi_id=kaubagrupid.id
        AND (nimetus LIKE '%$otsisona%' OR grupinimi LIKE '%$otsisona%')
       ORDER BY $sorttulp");
    //echo $yhendus->error;
    $kask->bind_result($id, $nimetus, $grupinimi, $kaubagrupi_id, $hind);
    $kask->execute();
    $hoidla=array();
    while($kask->fetch()){
      $kaup=new stdClass();
      $kaup->id=$id;
      $kaup->nimetus=htmlspecialchars($nimetus);
      $kaup->grupinimi=htmlspecialchars($grupinimi);
      $kaup->kaubagrupi_id=$kaubagrupi_id;
      $kaup->hind=$hind;
      array_push($hoidla, $kaup);
    }
    return $hoidla;
  }

  /**
  * Luuakse HTML select-valik, kus v6etakse v22rtuseks sqllausest tulnud
  * esimene tulp ning n2idatakse teise tulba oma.
  */

  function looRippMenyy($sqllause, $valikunimi, $valitudid=""){
     global $yhendus;
     $kask=$yhendus->prepare($sqllause);
     $kask->bind_result($id, $sisu);
     $kask->execute();
     $tulemus="<select name='$valikunimi'>";
     while($kask->fetch()){
       $lisand="";
       if($id==$valitudid){$lisand=" selected='selected'";}
       $tulemus.="<option value='$id' $lisand >$sisu</option>";
     }
     $tulemus.="</select>";
     return $tulemus;
  }

/*
  function looRippMenyy($sqllause, $valikunimi){
     global $yhendus;
     $kask=$yhendus->prepare($sqllause);
     $kask->bind_result($id, $sisu);
     $kask->execute();
     $tulemus="<select name='$valikunimi'>";
     while($kask->fetch()){
       $tulemus.="<option value='$id'>$sisu</option>";
     }
     $tulemus.="</select>";
     return $tulemus;
  }
*/

  function lisaGrupp($grupinimi){
     global $yhendus;
     $kask=$yhendus->prepare("INSERT INTO kaubagrupid (grupinimi)
                     VALUES (?)");
     $kask->bind_param("s", $grupinimi);
     $kask->execute();
  }

  function lisaKaup($nimetus, $kaubagrupi_id, $hind){
     global $yhendus;
```

```
    $kask=$yhendus->prepare("INSERT INTO
      kaubad (nimetus, kaubagrupi_id, hind)
      VALUES (?, ?, ?)");
    $kask->bind_param("sid", $nimetus, $kaubagrupi_id, $hind);
    $kask->execute();
  }

  function kustutaKaup($kauba_id){
    global $yhendus;
    $kask=$yhendus->prepare("DELETE FROM kaubad WHERE id=?");
    $kask->bind_param("i", $kauba_id);
    $kask->execute();
  }

  function muudaKaup($kauba_id, $nimetus, $kaubagrupi_id, $hind){
    global $yhendus;
    $kask=$yhendus->prepare("UPDATE kaubad SET nimetus=?, kaubagrupi_id=?, hind=?
                    WHERE id=?");
    $kask->bind_param("sidi", $nimetus, $kaubagrupi_id, $hind, $kauba_id);
    $kask->execute();
  }

  //---------------
  if( array_pop(explode("/", $_SERVER["PHP_SELF"]))=="abifunktsioonid.php"):
?>
<pre>
<?php
  print_r(kysiKaupadeAndmed("hind", "fass\\aad"));
?>
</pre>
<?php endif ?>
```

Ülesandeid

* Tee näited läbi
* Võimalda muuta inimeste andmeid, kaasa arvatud maakonda, kus inimene paikneb.

* Koosta kommenteeritavate uudistega veebilehestik. Administraatorilehel saab lisada uudiseid.
* Vaatajalehel näeb uudiseid ning neid saab kommenteerida. Kommentaaride tabelis on viide uudise id-le.

Funktsioonid klassis

Mõne või mõnekümne loodava funktsiooni puhul saab nad mugavalt paigutada abifunktsioonide faili (või ka mitmesse). Ning kui funktsioone arusaadavalt nimetada, siis nendega ei tohiks erilisi probleeme tekkida. Kui aga loodud funktsioonide arv jõuab sajani (või isegi tuhandeni), siis funktsioonide nimesid leida ja meenutada ja eristada läheb juba päris tülikaks. Samuti faili sees õige koha üles leidmine on küllalt suur katsumus. Samas - vähegi suurema ja mitmekülgsema lehestiku juures tekib tuhatkond funktsiooni üsna pea - eriti kui veel kasutatakse rakenduse juures mõnd varem valmis tehtud ning samuti hulgem funktsioone sisaldavat juppi. Kui veel eri moodulid tegelevad küllalt sarnaste teemadega (mis ikka ühte valdkonda suunatud rakenduse juures tarvilik on), siis on segadused funktsioonide nimedega kerged tulema.

Programmeerimisteoreetikud on selleks puhuks juba aastakümnete tagant välja mõelnud objektorienteeritud programmeerimise, mis on tarkvaraarenduse juures suhteliselt valdavaks

muutunud. Tema häid külgi saab kasutada ka PHP juures. PHP 5. versiooni mootor kirjutati suurelt jaolt just seetõttu üsna nullist uuesti, et objektimajandus ladusamalt välja tuleks.

Ega esimeses lähenduses klassid ja objektid midagi väga maagilist olegi. Tegemist on ühe kapseldumiskihiga. Klassi ehk objektitüübi juures kirjeldatakse ära, millised muutujad (väljad) ja käsklused (meetodid) objekti juurde kuuluvad. Klassi põhjal luuakse üks või mitu isendit ehk objekti ehk eksemplari, kel siis vastavad käsklused sees ning mis saavad isendi piires ühiseid muutujaid kasutada.

Näitena loome siin klassi Kaubad, mille ülesandeks on koondada enese sisse kaupade haldamisega seotud toimingud. Ainsaks muutujaks klassi sees jääb praegu toimingute teostamiseks vajalik andmebaasiühendus, tähistatuna praegu privaatmuutujana nimega $ab. Privaatmuutujatele pääseb ligi ainult sama klassi käskluste ehk meetodite seest.

Kui eelmistes näidetes oli meil globaalmuutuja nimega $yhendus ning igas seda kasutavas funktsioonis tuli sellele ligipääs deklareerida käskluse

```
global $yhendus
```

kaudu, siis nüüd on muutuja $ab kogu klassi alamprogrammide sees vabalt kasutatav. Endise $yhendus->prepare asemel tuleb lihtsalt kirjutada $this->ab->prepare. Muutuja $this abil saab pöörduda konkreetse objekti muutujate külge tema enese funktsioonide seest.

Klassi põhjal eksemplari loomise juures saab algväärtustamistoimingud panna konstruktorisse. PHP 5st alates on selleks funktsioon nimega __construct. Siinses näites eeldatakse, et Kaubad-klassile antakse selle eksemplaari loomisel ette avatud andmebaasiühendus sobivasse kohta. See jäetakse tema privaatmuutujasse meelde.

```
  function __construct($yhendus){
    $this->ab=$yhendus;
  }
```

Klassi üldiselt niisama ja otse ei kasutata. Selle asemel luuakse tema põhjal objekt ja antakse talle vajalikud algväärtused.

```
  $kaubahaldur=new Kaubad($yhendus);
```

Hilisemad toimingud tehakse siis juba loodud kaubahalduri objekti kaudu. Kui ennist võisin otse käivitada funktsiooni kysiKaupadeAndmed, siis nüüd peab sel objekti nimi ees olema. Ehk siis käivituskujuks on $kaubahaldur->kysiKaupadeAndmed. Kirjaviis läheb küll pikemaks. Kuid selle eest pole muret, kui keegi kusagil mujal loob ka käskluse kysiKaupadeAndmed. Kumbki tuleb välja lihtsalt eraldi objekti kaudu ning käsklused üksteist ei sega.

```
  print_r($kaubahaldur->kysiKaupadeAndmed("hind"));
```

Rippmenüüga seotud käsklused jäid kaupade klassist välja, sest neid võib vaja minna ka mujal kui kaupade juures.

Sageli paigutatakse vajaminevad klassid eraldi omanimelistesse failidesse. St. et klass Kaubad võiks olla failis nimega Kaubad.class.php. Iseenesest hea tava - eriti kui klasse on rohkem ja neid vaja leida ja nende vahel orienteeruda. Siin näiteks aga jääme parem kujule, kus kõik abivahendid on failis nimega abifunktsioonid.php

```
<?php
 $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");
```

```
class Kaubad{
 private $ab;
 function __construct($yhendus){
   $this->ab=$yhendus;
 }
 function kysiKaupadeAndmed($sorttulp="nimetus", $otsisona=""){
   $lubatudtulbad=array("nimetus", "grupinimi", "hind");
   if(!in_array($sorttulp, $lubatudtulbad)){
     return "lubamatu tulp";
   }
   $otsisona=mysql_real_escape_string(stripslashes($otsisona));
   $kask=$this->ab->prepare("SELECT kaubad.id, nimetus, grupinimi, kaubagrupi_id, hind
      FROM kaubad, kaubagrupid
      WHERE kaubad.kaubagrupi_id=kaubagrupid.id
       AND (nimetus LIKE '%$otsisona%' OR grupinimi LIKE '%$otsisona%')
      ORDER BY $sorttulp");
   $kask->bind_result($id, $nimetus, $grupinimi, $kaubagrupi_id, $hind);
   $kask->execute();
   $hoidla=array();
   while($kask->fetch()){
     $kaup=new stdClass();
     $kaup->id=$id;
     $kaup->nimetus=htmlspecialchars($nimetus);
     $kaup->grupinimi=htmlspecialchars($grupinimi);
     $kaup->kaubagrupi_id=$kaubagrupi_id;
     $kaup->hind=$hind;
     array_push($hoidla, $kaup);
   }
   return $hoidla;
 }


 function lisaGrupp($grupinimi){
    $kask=$this->ab->prepare("INSERT INTO kaubagrupid (grupinimi)
                     VALUES (?)");
    $kask->bind_param("s", $grupinimi);
    $kask->execute();
 }

 function lisaKaup($nimetus, $kaubagrupi_id, $hind){
    $kask=$this->ab->prepare("INSERT INTO
      kaubad (nimetus, kaubagrupi_id, hind)
      VALUES (?, ?, ?)");
    $kask->bind_param("sid", $nimetus, $kaubagrupi_id, $hind);
    $kask->execute();
 }

 function kustutaKaup($kauba_id){
    $kask=$this->ab->prepare("DELETE FROM kaubad WHERE id=?");
    $kask->bind_param("i", $kauba_id);
    $kask->execute();
 }

 function muudaKaup($kauba_id, $nimetus, $kaubagrupi_id, $hind){
    $kask=$this->ab->prepare("UPDATE kaubad SET nimetus=?, kaubagrupi_id=?, hind=?
                     WHERE id=?");
    $kask->bind_param("sidi", $nimetus, $kaubagrupi_id, $hind, $kauba_id);
    $kask->execute();
 }
}

 /**
 * Luuakse HTML select-valik, kus v6etakse v22rtuseks sqllausest tulnud
 * esimene tulp ning n2idatakse teise tulba oma.
 */

 function looRippMenyy($sqllause, $valikunimi, $valitudid=""){
    global $yhendus;
    $kask=$yhendus->prepare($sqllause);
    $kask->bind_result($id, $sisu);
    $kask->execute();
```

```
    $tulemus="<select name='$valikunimi'>";
    while($kask->fetch()){
      $lisand="";
      if($id==$valitudid){$lisand=" selected='selected'";}
      $tulemus.="<option value='$id' $lisand >$sisu</option>";
    }
    $tulemus.="</select>";
    return $tulemus;
  }

  function rippMenyyAndmed($sqllause){
    global $yhendus;
    $kask=$yhendus->prepare($sqllause);
    $kask->bind_result($id, $sisu);
    $kask->execute();
    $hoidla=array();
    while($kask->fetch()){
      $hoidla[$id]=$sisu;
    }
    return $hoidla;
  }

  $kaubahaldur=new Kaubad($yhendus);

  //---------------
  if( array_pop(explode("/", $_SERVER["PHP_SELF"]))=="abifunktsioonid.php"):
?>
<pre>
<?php
  print_r($kaubahaldur->kysiKaupadeAndmed("hind"));
?>
</pre>
<?php endif ?>
```

Ülesandeid

* Paiguta eelnevalt loodud inimeste andmeid haldavas rakenduses inimestega seotud funktsioonid eraldi klassi.
* Muuda halduslehel funktsioonide väljakutse kuju selliselt, et haldusleht suudaks kasutada eraldi klassis leiduvad inimeste andmetega tegelevaid funktsioone ning leht töötaks.

Smarty lehemallid

Programmeeritavate veebilehestike juures on juba algusest peale püütud programmiosa ja kujundust võimalikult eraldada - et kujundajad saaksid rahus lehe väljanägemise kallal töötada ning et programmikood ei muutuks väljatrükitavate kujunduslõikude tõttu pikaks ja halvasti loetavaks. Samuti võimaldab selline eraldatus sama sisu ja funktsionaalsusega lehe kujundust märgatavalt muuta (näiteks ühildada veebilehestikke kahe firma ühinemisel) ilma, et peaks andmehalduse ja arvutuste poolt kuigivõrd muutma.

Mõnelgi programmeerijal on tavaks küllalt palju erineva sisu ja kujundusega lehti koondada ühe faili alla. Vahel paistab terve keerukas veebilehestik vaid index.php kaudu. Ka sellisel juhul on lehemallidest märatavalt kasu - eri kujundusega lõigud saab mugavasti eraldi failidesse paigutada.

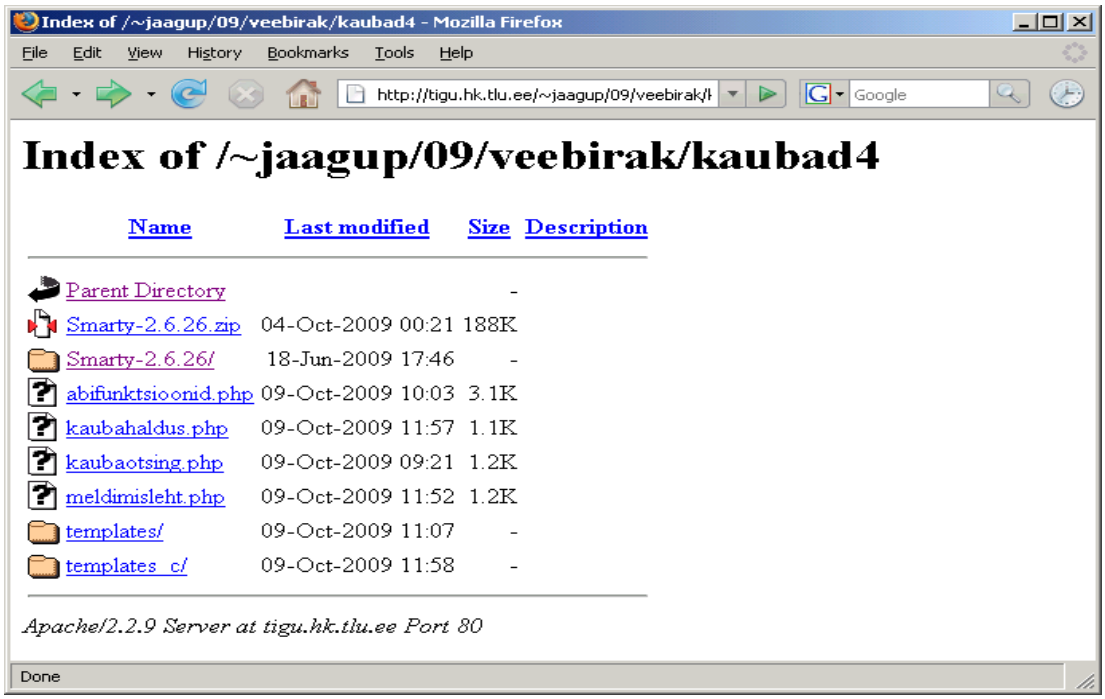
Võrreldes mõne muu veebitehnoloogiaga (nt. Java Servletid, PERL) saab PHP koodi enesegagi küllalt hästi lehemalli rolli täita. Siingi materjalis tehtud näidetes on eraldi abifunktsioonid ja eraldi kujundusleht ning mõningane sisu ja vormi eraldatus on saavutatud. Seetõttu mõnigi väidab, et PHP jaoks pole eraldi mallide (template) süsteemi vaja, et PHP oma lehe loomise vahendid on selleks

eraldatuseks kõige selgemad ja paremad. Ei oska talle vastu vaielda, kuid vaidlejaid siiski leidub, kes on vastava süsteemi kokku pannud. Tuntumad lehemallide süsteemid ehk Smarty, phpBB ja PatTemplate, kuid erisuguseid enese ja või oma firma jaoks aretatud PHP lehemallide süsteeme on maailma peal kindlasti tuhandeid. Siinse kirjutise autor näiteks teab Eesti pealt päris mitut inimest ja firmat, kes kõik oma lehtede mugavamaks haldamiseks on selle tarvis sarnase süsteemi kirjutanud.

Siin tutvustame lehemallinduse võimalus Smarty näitel. Projekti koduleht on http://www.smarty.net/, kust saab ka abiinfot ning lähtekoodi mallsüsteemi paigaldamiseks oma serverisse.

Tüüpiline Smarty abil lehe näitamise moodus koosneb vähemalt kahest lehest: tpl-laiendiga lehemall, kus HTMLi sees kirjas muutujad ja mõningad piiratud süntaksiga programmikäsud ning käivitav PHP- leht, kes annab tpl-failile kaasa vajalikud andmed ning palub siis mallilehte nende andmetega näidata.

Smarty lähtekoodi lahtipakkimisel tuleb silmas pidada, kus kataloogis asub fail nimega Smarty.class.php. Siinses versioonis leiab ta asukohast `Smarty-2.6.26/libs/Smarty.class.php` TPL-failis tuleb näidatavate muutujate andmed kirjutada looksulgude sisse, nagu näiteks `{$kaup->id }`. Samuti on looksulgude sees käsklused.



Näitena kohandame kaubahalduse faili ümber lehemalli kasutama. Smarty lehemalli kasutuseks tuleb luua jooksvale kaustale alamkataloogid nimedega templates ning templates_c. Esimesse neist pannakse loodud tpl-fail, teise kompileerib Smarty käivitatava vahetulemuse. Kaustale templates_c tuleb anda kõik õigused, et Smarty pääseks sinna sisu kompileerima ja pärast käivitama.

HTML-fail hakkab peale nagu tavaliselt. Mõnikord pannakse sinna kommentaar, et leht tehtud Smarty abil, aga see pole kohustuslik.

Esimene tähelepanu äratav koht on kaubagruppide rippmenüü loomine. Smarty käsk html_options võimaldab teha rippmenüü. Talle antakse praegusel juhul ette massiiv nimega $gruppide_andmed, kus massiivi iga elemendi indeksiks on näidatava kaubagrupi id ning väärtuseks selle grupi nimi. Select-valiku nimi tuleb parameetrist name, ehk kaubagrupi_id samuti nagu vahal PHP-lehel tehti.

```
        {html_options name=kaubagrupi_id options=$gruppide_andmed}
```

Järgmine suurem märkamist vajava koht on kaupade andmete läbikäimine. Tsükkel foreach käib läbi etteantud kaubaobjektide massiivi nimega $kaupade_andmed. Iga ringi peal saab konkreetse kauba andmed kätte muutujast nimega kaup - just nii nagu item-pärameetris näidatud. Valikulausega if kontrollitakse, kas tegemist muudetavate andmetega kaubaga. Kui jah, siis antakse andmerida muudetavate sisestuselementidega. Rippmenüü puhul saab ette anda ka valitud väärtuse - sarnaselt nagu me omaloodud rippemenüü loomise funktsioonid. Ning veebilehel siis ka keeratakse selle väärtusega valik ette.

Andmete niisama näitamine jääb else-ossa. Kaubaobjektilt saab andmeid küsida sarnaselt nagu tavalise PHP juures. St., et hind näidatakse välja kujul `{$kaup->hind }`

```
       {foreach from=$kaupade_andmed item=kaup}
         <tr>
          {if $muutmisid==$kaup->id}
           <td>
             <input type="submit" name="muutmine" value="Muuda" />
              ...
           {html_options name=kaubagrupi_id options=$gruppide_andmed
                         selected=$kaup->kaubagrupi_id}
             ...

          {else}
             ...
           <td>{$kaup->hind }</td>
          {/if}
         </tr>
       {/foreach}
```

Ning fail tervikuna.

templates/kaubahaldus.tpl

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
      <title>Kaupade leht</title>
      <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
   <a href="meldimisleht.php?lahku=jah">Lahku haldamisest</a>
   <form action="kaubahaldus.php">
     <h2>Kauba lisamine</h2>
     <dl>
       <dt>Nimetus:</dt>
       <dd><input type="text" name="nimetus" /></dd>
       <dt>Kaubagrupp:</dt>
       <dd>
         {html_options name=kaubagrupi_id options=$gruppide_andmed}
       </dd>
       <dt>Hind:</dt>
       <dd><input type="text" name="hind" /></dd>
     </dl>
     <input type="submit" name="kaubalisamine" value="Lisa kaup" />
     <h2>Grupi lisamine</h2>
     <input type="text" name="uuegrupinimi" />
     <input type="submit" name="grupilisamine" value="Lisa grupp" />
   </form>
  <form action="kaubahaldus.php">
     <h2>Kaupade loetelu</h2>
     <table>
       <tr>
         <th>Haldus</th>
```

```
        <th>Nimetus</th>
        <th>Kaubagrupp</th>
        <th>Hind</th>
      </tr>
      {foreach from=$kaupade_andmed item=kaup}
        <tr>
         {if $muutmisid==$kaup->id}
          <td>
            <input type="submit" name="muutmine" value="Muuda" />
            <input type="submit" name="katkestus" value="Katkesta" />
            <input type="hidden" name="muudetudid" value="{$kaup->id }" />
          </td>
         <td><input type="text" name="nimetus" value="{$kaup->nimetus }" /></td>
         <td>
          {html_options name=kaubagrupi_id options=$gruppide_andmed
                                    selected=$kaup->kaubagrupi_id}
         </td>
         <td><input type="text" name="hind" value="{$kaup->hind}" /></td>

         {else}
          <td><a href="kaubahaldus.php?kustutusid={$kaup->id }"
                 onclick="return confirm('Kas ikka soovid kustutada?')">x</a>
              <a href="kaubahaldus.php?muutmisid={$kaup->id }">m</a>
          </td>
          <td>{$kaup->nimetus }</td>
          <td>{$kaup->grupinimi }</td>
          <td>{$kaup->hind }</td>

         {/if}
        </tr>
      {/foreach}
     </table>
  </form>
 </body>
</html>
```

Mallifail iseseisvalt ei käivitu. Tema väljakutseks peab olema PHP-fail. Sinna sisse saab panna/jätta kõik muud koodilõigud, mis muidu kippusid lehe kujunduse kokkupanekut häirima. Samuti tuleb siin Smarty lehemallile kõik muutujatest pärinevad väärtused ette anda, et mallil neid kusagilt võtta oleks.

Käsklus `session_start()` ning kasutajanime kontroll on tulnud juurde - et igaüks siia lehele niisama vabalt sisse ei saaks. Meldimisleht ise näha veidi tagapool. Lühiseletuseks, et kui sessioonimuutujasse pole salvestatud kasutajanime, siis sellest võib järeldada, et inimene ei ole administreerimislehele saamiseks sisse loginud ning saadetakse edasi meldimislehele.

Grupi lisamine, kauba lisamine, kustutamine ja muutmine on sarnased nagu ennegi. Uus osa hakkab Smartyga seoses. Kõigepealt loetakse sisse Smarty klassi fail, mis ise sealt ülejäänud vajalikud asjad enesele külge haagib. Sealt sisseloetud klassi põhjal tehakse uus Smarty-tüüpi objekt, mille kaudu siis edasine suhtlus lehemalliga käib. Käsuga assign määratakse üksikud andmed Smarty-mallilehe muutujate külge. Lisamise rippmenüü jaoks gruppide andmed, kaupade tabeli näitamiseks kaupade andmed. Ning ühe kaubarea näitamiseks muudetaval kujul selle rea muutmisid. Kusjuures siin hoolitsetakse, et selle muutuja väärtus veateadete vältimiseks ikka olemas oleks. Kui ka tegelikult muuta ei soovita ning `$_REQUEST["muutmisid"]` puudub, siis pannakse selleks väärtuseks võimatu väärtus: -1.

Edasi juba käsklus `display` soovitud malli näitamiseks ning võibki lehte imetleda.

```
require("Smarty-2.6.26/libs/Smarty.class.php");
$smarty=new Smarty;
$smarty->assign("gruppide_andmed",
              rippMenyyAndmed("SELECT id, grupinimi FROM kaubagrupid"));
$smarty->assign("kaupade_andmed", $kaubahaldur->kysiKaupadeAndmed());
```

```
  $smarty->assign("muutmisid",
                isSet($_REQUEST["muutmisid"])?intval($_REQUEST["muutmisid"]):-1);
  $smarty->display('kaubahaldus.tpl');
  $yhendus->close();
```

Ning mallilehte käivitav PHP-leht tervikuna - saabuvate andmete püüdmine, nendele reageerimine ning lõpus mallilehe näitamiseks vajalikud toimingud.

```
<?php
  session_start();
  if(!isSet($_SESSION["kasutajanimi"])){
    header("Location: meldimisleht.php");
    exit();
  }
  require("abifunktsioonid.php");

  if(isSet($_REQUEST["grupilisamine"])){
    $kaubahaldur->lisaGrupp($_REQUEST["uuegrupinimi"]);
    header("Location: kaubahaldus.php");
    exit();
  }
  if(isSet($_REQUEST["kaubalisamine"])){
    $kaubahaldur->lisaKaup($_REQUEST["nimetus"],
               $_REQUEST["kaubagrupi_id"], $_REQUEST["hind"]);
    header("Location: kaubahaldus.php");
    exit();
  }
  if(isSet($_REQUEST["kustutusid"])){
     $kaubahaldur->kustutaKaup($_REQUEST["kustutusid"]);
  }
  if(isSet($_REQUEST["muutmine"])){
     $kaubahaldur->muudaKaup($_REQUEST["muudetudid"],
                $_REQUEST["nimetus"], $_REQUEST["kaubagrupi_id"], $_REQUEST["hind"]);
  }
  require("Smarty-2.6.26/libs/Smarty.class.php");
  $smarty=new Smarty;
  $smarty->assign("gruppide_andmed",
                 rippMenyyAndmed("SELECT id, grupinimi FROM kaubagrupid"));
  $smarty->assign("kaupade_andmed", $kaubahaldur->kysiKaupadeAndmed());
  $smarty->assign("muutmisid",
                 isSet($_REQUEST["muutmisid"])?intval($_REQUEST["muutmisid"]):-1);
  $smarty->display('kaubahaldus.tpl');
  $yhendus->close();
?>
```

Ülesandeid

* Koosta lihtne tervitav Smarty-leht, käivita PHP kaudu
* Anna PHP kaudu edasi tervituslause
* Smarty-lehel saab sisestada kaks arvu. Sinnasamasse väljastatakse nende korrutis.
* Koosta massiiv maakondade loeteluga. Näita nad Smarty-lehel nummedamata loetleus <ul>.
* Koosta Smarty-lehe tarvis rippmenüü maakondade loeteluga. Eelneva rakenduse käigus valminud inimeste tabelist näita välja nende nimed, kes elavad valitud maakonnas.

Sessioonimuutujaga meldimine

Kasutajate ligipääsu haldamine on tänapäevastes veebirakendustes peaaegu kohustuslik osa - nõnda on julgem rohkem toimetusi lubada veebist ette võtta. Lihtsaimas lahenduses kontrollitakse kasutajanime ja parooli vaid koodi sees ning kasutajanimi jäetakse veebilehitseja lahtiolekuajaks meelde sessioonimuutujasse - erilisse kohta, kustkaudu on võimalik andmeid hoida ja küsida eri

PHP-lehtedel liikudes. Nõnda on ühel lehel sessioonimuutujasse pandud kasutajanimi kättesaadav ka sama rakenduse teistel lehtedel. Ning välja logides kasutajanime sessioonimuutuja kustutamise teel pole seda muutujat enam olemas ka muude lehtede jaoks ning kasutajat sinna enam ei lasta, kui vastav piirang peal on.

Sessioonimuutujate kasutamiseks peab olema lehe päises käivitatud käsklus session_start(). Selle abil saadetakse veebilehitsejasse päiserida (küpsis), mille kaudu saab server hiljem kontrollida, et tegemist on sama vaatava brauseriga.

```
if(isset($_SESSION["kasutajanimi"]))
```

kontrollib, kas vastava nimega muutuja on olemas. Kui jah- siis järelikult on kasutaja juba varasemast sisse loginud.

```
    if(isset($_REQUEST["lahku"])){
       unset($_SESSION["kasutajanimi"]);
    }
```

Sellisel puhul kontrollitakse kõigepealt, et ega kasutaja kogemata pole lahkumissoovi avaldanud. Kui aasressiribale on saabunud parameeter nimega "lahku", siis eemaldatakse kasutajanime sessioonimuutuja.

Kui aga kasutaja pole veel sees, siis kontrollitakse, kas ta äkki tahab ja saab siia tulla. Praegusel juhul võrreldakse vaid koodi sisse kirjutatud nime ja parooli, kuid siinse kontrolli saab edaspidi tunduvalt asjalikumaks muuta.

```
} else {
   if($_REQUEST["kasutajanimi"]=="juku" && $_REQUEST["parool"]=="kala"){
      $_SESSION["kasutajanimi"]=$_REQUEST["kasutajanimi"];
   }
}
```

Allpool siis käitumine lehel sõltuvalt sellest, kas kasutaja on sisse loginud või mitte. Ehk siis kas leidub sessioonimuutuja `$_SESSION["kasutajanimi"]`. Kui see olemas, siis saab kasutajat tervitada ning talle pakkuda viiteid, mis registreeritud kasutajane kohane. Kui aga kasutajanime pole, siis on paras koht selle küsimiseks. Kusjuures tasub rõhutada, et vormi juurde kuuluks method="post". Muul juhul oleks parool selgesti aadressireal nähtav - sõltumata sellest, et selle sissekirjutamiseks eraldi parooliväli loodud on.

```
<?php if(isSet($_SESSION["kasutajanimi"])): ?>
   Tere, <?=$_SESSION["kasutajanimi"] ?>
   <a href="<?="$_SERVER[PHP_SELF]?lahku=jah"; ?>">lahku</a><br />
   Head <a href="kaubahaldus.php">haldamist</a>!
<?php else: ?>
   <form action="<?=$_SERVER["PHP_SELF"] ?>" method="post">
      <dl>
         <dt>Kasutajanimi:</dt>
         <dd><input type="text" name="kasutajanimi" /></dd>
         <dt>Parool:</dt>
         <dd><input type="password" name="parool" /></dd>
      </dl>
      <input type="submit" value="Sisesta" />
   </form>
<?php endif ?>
```

Muudel lehtedel piisab sisenemiskontrolliks paarist reast:

```
session_start();
if(!isSet($_SESSION["kasutajanimi"])){
  header("Location: meldimisleht.php");
```

```
    exit();
  }
```

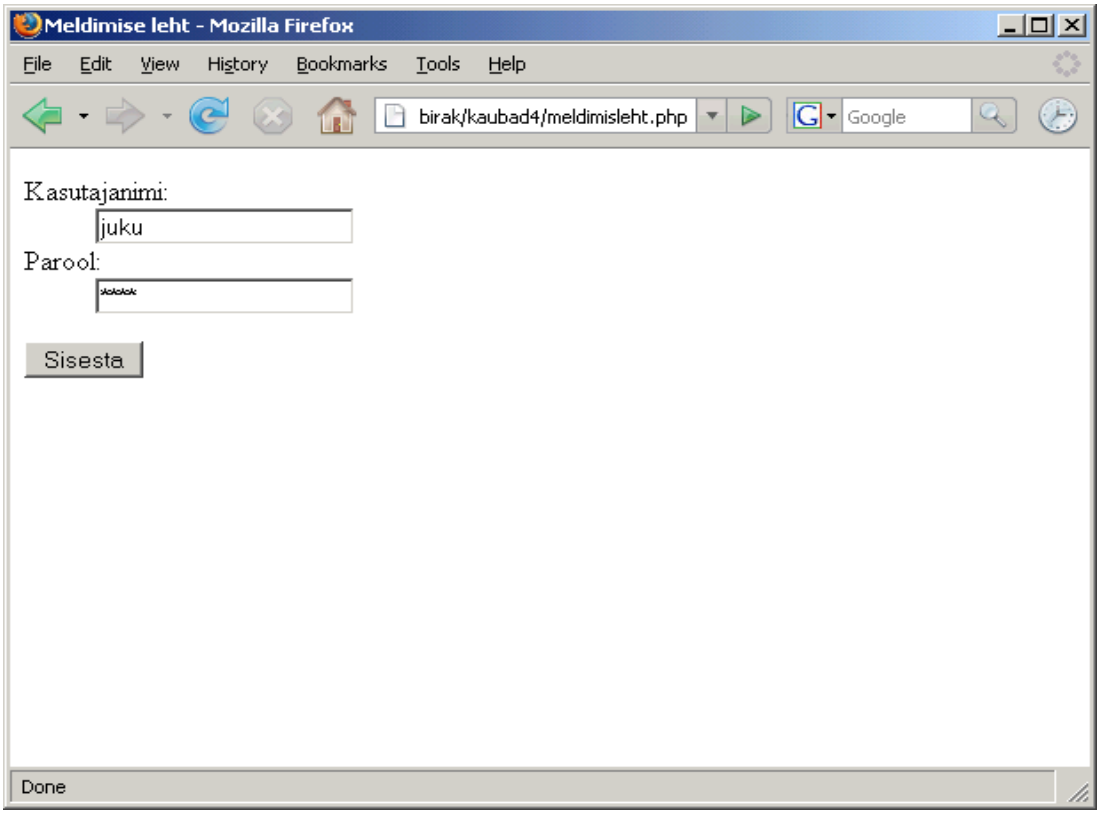
Käsk `session_start()` ikka, et sessioonimuutujaid kasutada saaks. Kui kasutajanime pole, siis saadetakse külastaja lihtsalt edasi meldimiseks mõeldud lehele ning lõpetatakse kaitstud lehe serveerimine. Muul juhul minnakse lehe näitamisega edasi.

Ning nüüd meldimislehe kood tervikuna - suhteliselt universaalselt kasutatav mitmesuguste rakenduste juures. Vaid kasutajanime ja parooli kontroll on pärisrakenduste juures põhjust asjalikumad teha.

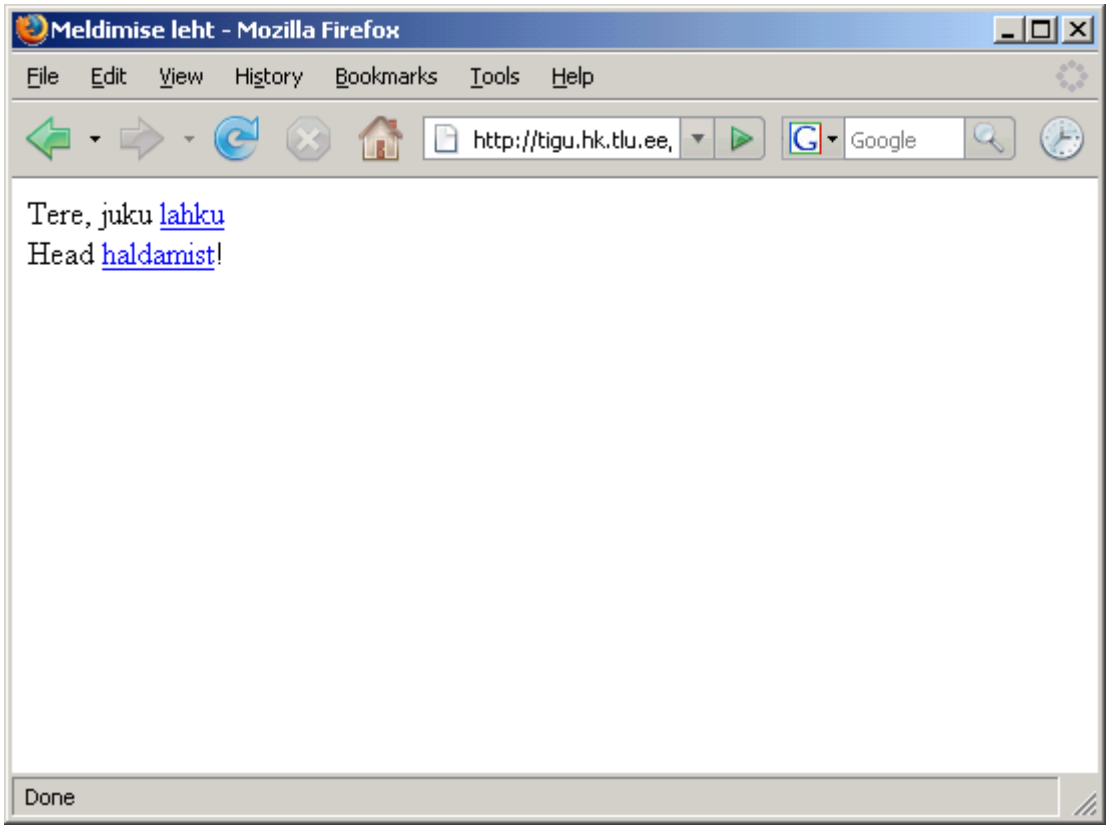
```
<?php
  session_start();
  if(isset($_SESSION["kasutajanimi"])){
     if(isset($_REQUEST["lahku"])){
        unset($_SESSION["kasutajanimi"]);
     }
  } else {
     if($_REQUEST["kasutajanimi"]=="juku" && $_REQUEST["parool"]=="kala"){
        $_SESSION["kasutajanimi"]=$_REQUEST["kasutajanimi"];
     }
  }

?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
        <title>Meldimise leht</title>
        <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
    <?php if(isSet($_SESSION["kasutajanimi"])): ?>
       Tere, <?=$_SESSION["kasutajanimi"] ?>
       <a href="<?="$_SERVER[PHP_SELF]?lahku=jah"; ?>">lahku</a><br />
       Head <a href="kaubahaldus.php">haldamist</a>!
    <?php else: ?>
       <form action="<?=$_SERVER["PHP_SELF"] ?>" method="post">
          <dl>
             <dt>Kasutajanimi:</dt>
             <dd><input type="text" name="kasutajanimi" /></dd>
             <dt>Parool:</dt>
             <dd><input type="password" name="parool" /></dd>
          </dl>
          <input type="submit" value="Sisesta" />
       </form>
    <?php endif ?>
  </body>
</html>
```

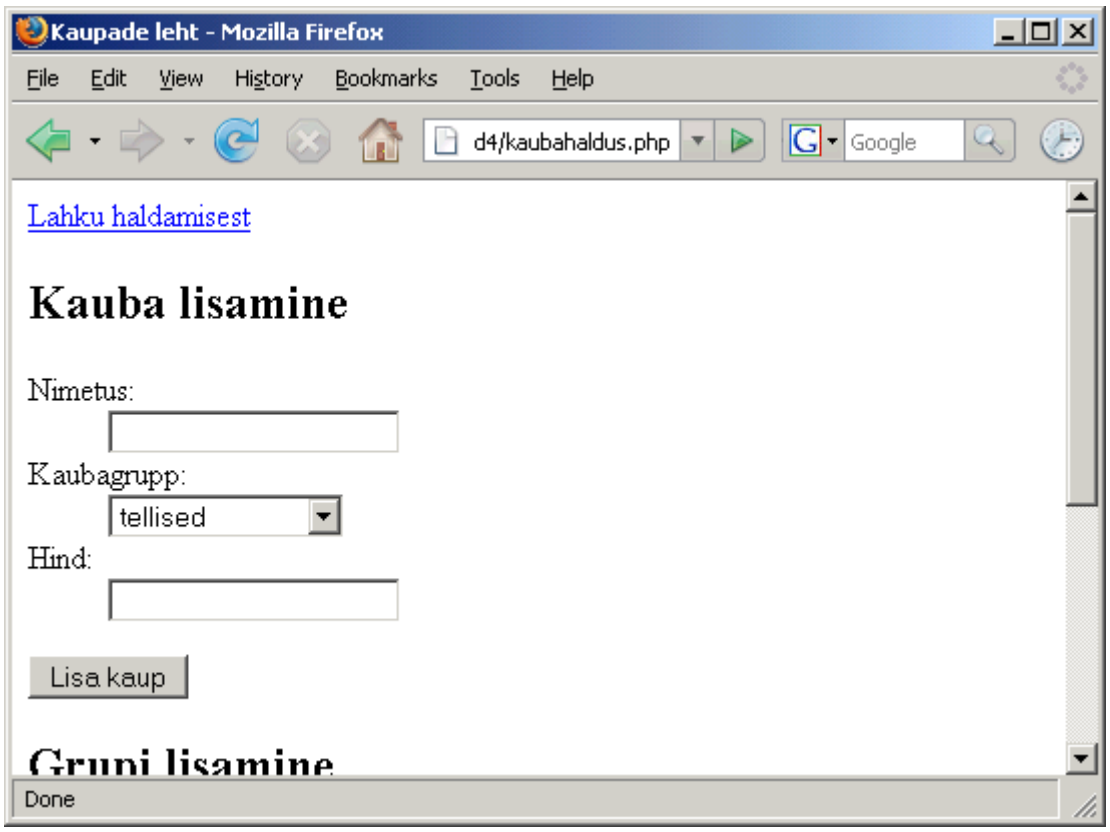
Meldimislehel siis kõigepealt uuritakse kasutajanime ja parooli.



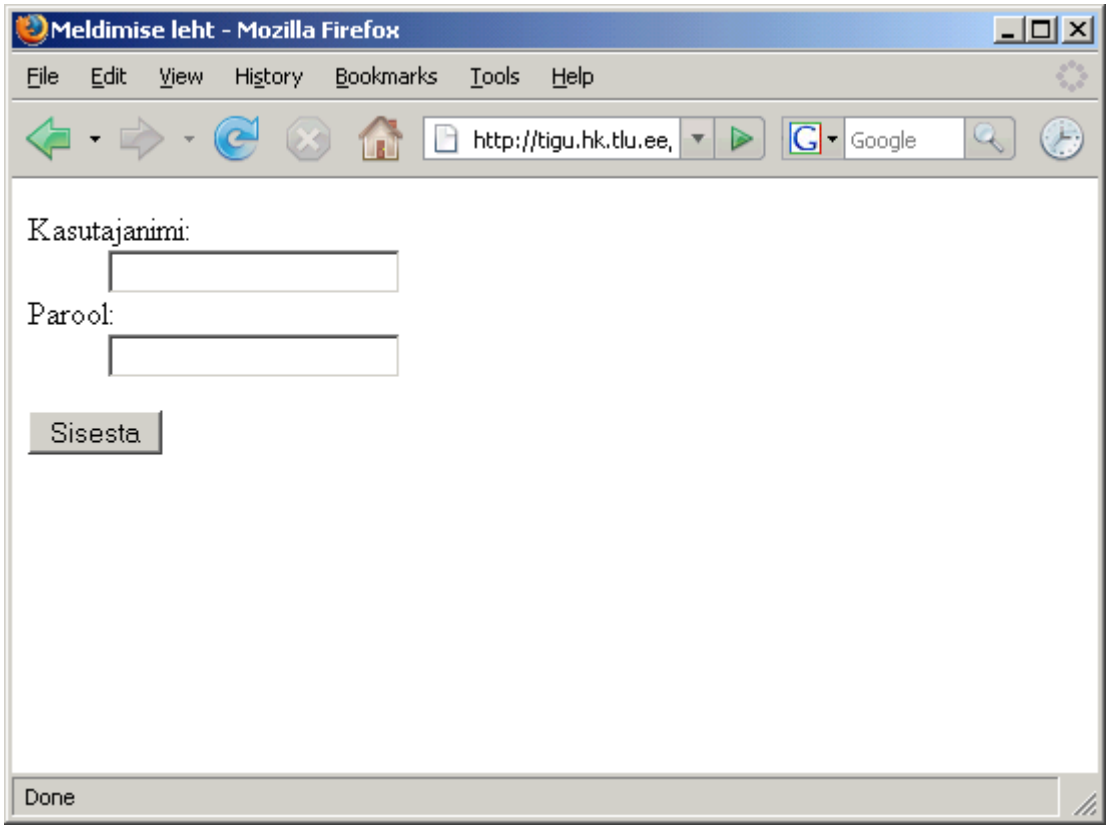
Kui need klapivad, siis pääseb registreeritud kasutajate tarbeks mõeldud viidete juurde.



Ja sealt juba omakorda tegelikku lehestikku haldama. Kuna kasutajanimi sessioonimuutujas kirjas, siis võib rahumeeli pikemat aega lehe peal toimetada.



Kui tööd tehtud, tasub vajutada viitele "Lahku haldamisest" ning jõuamegi taas meldimislehele tagasi, kust ilma kasutajanime ja parooli teadmata enam kuhugi edasi ei pääse.



Ülesandeid

* Tee kasutaja sisselogimise näide läbi. Muuda lubatud kasutajanime ja parooli.
* Koosta väike külalisraamat, kus nime ja parooliga kasutaja saab teateid lisada, teised ainult vaadata.

* Lisa eraldi andmetabel kasutajanimede ja paroolide tarbeks. Luba sisse vaid tegelased, kes tabelis kirjas.

Mitu mitmele seos ehk kolm seotud tabelit

Eelnevas näites seoti kaks tabelit. Ehk siis andmebaasi projekteerijate keeles oli "üks mitmele seos", kus ühte kaubagruppi võis kuuluda suvaline arv kaupu (kaasa arvatud 0 või 1). Ning konkreetne kaup on alati seotud ühe kaubagrupiga. Jooniste puhul siis "üks" on kaubagrupi poolel ning "mitu" kaupade poolel - tähistatagu neid milliste noolekeste või märkidega tahes.

Selliseid tunnuseid on mugav põhitabeli külge linkida - põhitabelis on lihtsalt vastav id ning viidatavast tabelist saab siis selle id järgi kätte vajaliku rea andmed. Sarnaselt võiks kauba külge siduda näiteks tema paiknemise maakonna, valmistamise riigi jm. tunnuse, millel on üks konkreetne väärtus. Mõnevõrra tülikaks osutub, kui vastav väärtus on puudu või teadmata, kuid ka sellisel juhul on võimalik mure kahe tabeliga ära lahendada. Üheks võimaluseks on lisada riikide tabelisse riigid nimedega "muu" ja "teadmata". Kui viidete terviklust (et viidatavas tabeli veerus oleks viidatav väärtus olemas) ei kontrollita, siis saab panna kauba tabelisse riigi kohale ka tühiväärtuse NULL, mis siis annab teada, et kauba tootmise kohta selget riiki määrata ei saa. Võimalusel on aga parem sellistest NULL-väärtustest hoiduda, sest need muudavad korrektsete keerukamate päringute tegemise tüki maad tülikamaks. Näiteks võib eeltoodud tabelite sidumiste moel kergesti juhtuda, et määramata grupiga kaubad lähevad üldse kaupade loetelust kaduma.

Mõnegi seose puhul aga pole lootustki kahe tabeliga hakkama saada. Kui näiteks leidub võimalus, kus kaup võiks üheaegselt kuuluda mitmesse gruppi, siis ei piisa enam grupi id-st kaupade tabelis. Ühte tabeli lahtrisse mitme erisuguse väärtuse kirjutamist ei soosita enam ammu. Sarnane olukord tekib, kui on vaja üles märkida, millised riigid on kaup läbinud, millised kliendid on millises koguses kaupu ostnud, milliste õpetajate juures on milline õpilane õppinud ...

Sellisel juhul aitab kolme tabeli abil üles tähendatud "mitu mitmele" seos. Üldjuhul on siis kaks tabelit olemite ehk kirjeldatavate objektide (nt. klient, kaup) kohta. Kolmandas tabelis on igal real kirjas id viitamaks ühele ning teine id viitamaks teisele tabelile. Sellisel juhul on võimalik põhitabelite vahelisi seoseid kolmandas tabelis luua täpselt nii palju, kui just parajasti vaja on. Kaupade ja klientide puhul oleks tõenäoliselt kolmanda tabeli nimeks "ostud". Ning iga ostu korral pannakse sinna tabelisse kirja, millise id-ga klient ostis millise kauba. Vajadusel saab sinna tabelisse vastavatesse veergudesse lisada ka näiteks korraga ostetud kaupade koguse või tehingu sooritamise aja.

Siit muidugi on varsti tähtsustamise küsimus, et kas tähtsamaks tabeliteks osutuvad tegelikke objekte kirjeldavad klient ja kaup, või saab vaadata tähtsaima loeteluna pigem ostude tabelit, kuhu iga ostu juurde lihtsalt abitabelitest andmed juurde haagitud. Kliendihaldur tõenäoliselt vaatab andmeid klientide kaupa ning tunneb huvi, milliseid tooteid ja kui palju ta ostnud on kasutades ostude tabelit lihtsalt sidujana, abivahendina ostetud kaupade andmete kätte saamiseks. Kaupade tootja tunneb huvi, kes ja kui palju tema tooteid on ostnud - kasutades samuti ostude tabelit vaid

siduva abivahendina. Müügimehe jaoks on aga tõenäoliselt kõige armsam koht ostude tabel, kus ta müügitöö tulemused kirjas. Kõrvalt tabelitest saab lihtsalt igaks juhuks toetavaid andmed vaadata, et kellele ja mida ta siiski müüs.

Siinses näites võtame kolmanda tabeli abil kokku seotavateks üksusteks ette veebilehe kasutajad ning nende huvialad. Kuna seosetabelile ei oska head selget ja arusaadavat nime välja mõelda (hädapärast ehk sobiks nimi "huvitub"), siis saab kolmanda tabeli nimeks lihtsalt kasutajad_huvialad. Ehk siis kokku seotuna mõlema tabeli nimed, mis mitmelgi pool selliste seoste tegemisel tavaks on. Kuna kasutajate tabeli kirjed peavad aitama ka kasutajatel end veebilehele sisse meldida ja sealse kasutajana käituda, siis ei piirduta vaid kasutajanimega, vaid on ka mõned täiendavad väljad. Huvialade tabel seevastu võimaikult lihtne - vaid loetelu kirjapandud huvialadest ning viitamiseks nende jaoks id-d.

Seosetabel kasutajad_huvialad on kirjeldatud veidi põhjalikumalt kui hädapärast tarvis. Kasutaja id ja huviala id on kindlasti vajalikud. Tuleviku rakenduste tarbeks maha kirjutamise huvides on aga lisatud ka tabeli enese ridu eristav id. Sest kasutajate puhul pole põhjust ehk ühte huviala mitu korda lisada. Üks klient võib aga sama koodiga toodet osta eri aegadel korduvalt. Samuti pole võõrvõtmete (foreign key) märkimine MySQLis kohustuslik. Tavaolukorras isegi viidete õigsust ei kontrollita (selle saab peale lülitada, hoides andmeid InnoDB-le vastaval kujul). Samas on tabeli kirjeldust lugedes hea vaadata, et kuhu täpselt miski tulba andmetega viidatakse. Programmeerijal peab see oma peas aga nagunii teada olema. Et kogemata sama huviala ühele kasutajale korduvalt ei lisataks, selleks lisati kasutajad_huvialad -tabelisse piirang UNIQUE(kasutaja_id, huviala_id) - see ei luba sama kombinatsiooni mitu korda korraga tabelis hoida.

```
CREATE TABLE kasutajad(
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  knimi VARCHAR(50) UNIQUE,
  paroolir2si CHAR(40) NOT NULL,
  epost VARCHAR(50)
);

CREATE TABLE huvialad(
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  huviala VARCHAR(50)
);

CREATE TABLE kasutajad_huvialad(
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  kasutaja_id INT,
  huviala_id INT,
  FOREIGN KEY(kasutaja_id) REFERENCES kasutajad(id),
  FOREIGN KEY(huviala_id) REFERENCES huvialad(id),
  UNIQUE (kasutaja_id, huviala_id)
);

INSERT INTO huvialad(huviala) VALUES ('Korvpall');
....

mysql> SELECT * FROM huvialad;
+----+----------+
| id | huviala  |
+----+----------+
|  1 | Korvpall |
|  2 | Jalgpall |
|  3 | Hoki     |
|  4 | Bajaan   |
+----+----------+
```

Kasutajate haldus

Püüame luua viisaka mooduse kasutajate registreerimiseks ning sisse-väljameldimiseks. Kasutajate tabeli tulpadeks said id, knimi, paroolir2si ja epost. Iseenesest piisaks lihtsamal juhul kasutaja tuvastamiseks ka vaid ühest tunnusest ja paroolist, aga siin püüame veidi põhjalikumad olla. Parooli asemel hoitakse selle räsikoodi seetõttu, et ootamatul andmete lekkimisel ei satuks paroolid avalikult nähtavaks. Nagu hilisematest andmetest paistab, siis räsikoodist on peale vaatamisel raske midagi välja lugeda - ning nii peabki olema. Allolev pikk räsi on tegelikult vaid mõnetähelise parooli põhjal arvutatud. Kuna aga räsi pikkus ei sõltu algandmete pikkusest, siis ka nt. terve videofaili põhjal arvutatud räsi näeb sama pikk ja ligikaudu sama segane välja.

```
mysql> SELECT * FROM kasutajad;
+----+-------+------------------------------------------+-------------------+
| id | knimi | paroolir2si                              | epost             |
+----+-------+------------------------------------------+-------------------+
| 25 | madis | 054b8a97845b86a485ee89beb31ff3447379e38c | madis@testkoht.ee |
+----+-------+------------------------------------------+-------------------+
```

Iseenesest piisaks kasutaja eristamiseks vaid kasutajanimest - ning seda mõnikord tehaksegi. Aga siiski kipuvad süsteemid panema sinna lisaks veel id-tulba. Et kui praegune madis süsteemist lahkub ning mõne aasta pärast järgmine samanimeline tuleb, siis on siiski võimalik eristada, kumma tegeliku inimesega mõnede säilinud andmete juures pistmist on. Samuti on id-number tabelite sidumisel ehk lühemini võrreldav, kui mõnikord pikaks kiskuv kasutajanimi. Või kui Google id või Windows Live id tavasid järgida, siis võib kasutaja eristamiseks piisata ka ainult elektronpostist - et see unikaalne tunnus ongi kasutajatunnuseks. Saab mitut moodi, aga siin valiti suhteliselt keerulisem tee.

Koodi struktueerimiseks jagatakse suuremad iseseisvad lõigud eraldi klassidesse ning neist igaüks ka omanimelisse faili. Kui funktsioone kipub rohkem saama, siis klasside kaupa grupeeritutena on neid hiljem mugavam otsida ja testida. Samuti on vahel hea klassis kokkukuuluvatel funktsioonidel ühiseid salvestatud andmeid kasutada nagu siin viita andmebaasiühendusele või hiljem kasutaja huvialasid uurides ka kasutaja id-le.

Nagu varasemas klassinäites, nii ka siin tuleb konstruktoris ehk klassikirjelduse põhjal eksemplari loomisel anda ette avatud andmebaasiühendus. Järgnevad kaks funktsiooni - lisaKasutaja uue kasutaja lisamiseks baasi ning kontrolliKasutaja kindlaks tegemiseks, kas vastava kasutajanime ja parooliga isik võib siseneda.

Kasutaja lisamisel kõigepealt kontrollitakse, et ega sellise kasutajanimega kasutajat juba baasis pole. Kui on, siis katkestatakse lisamistoiming. ID-de korduvust pole mõtet karta, sest see arvutatakse lisamisel nagunii uus.

Parool pannakse tabelisse räsina. Muundamiseks kasutatakse siin käsklust

```
        $r2si=sha1($parool);
```

ehk siis räsi loomisel kasutatakse algoritmi sha1, mis annab 40-tähelise kuueteistkümnendsüsteemi numbrite jada. Vastavalt sai valitud ka tabeli tulba andmete pikkus

```
  paroolir2si CHAR(40) NOT NULL
```

Kui kindel pikkus teada, siis töötab tüüp CHAR andmebaasis rutem, samuti ei vajata enam lisabaite tegeliku pikkuse salvestamiseks nagu VARCHARi puhul.

Registreeritud kasutajale saadetakse sellekohane kiri:

```
        mail($epost, "Registreeritud",
              "Registreerid end huvialade andmebaasi kasutajana $knimi.");
```

Kirja saatmine PHPs tõesti lihtne toiming - muidugi, kui PHP installeerimisel on kirjasaatmiseks vajalik konfigureeritud. Käsklus mail ning parameetriteks saaja aadress, pealkiri ja sisu. Kui tahta ka saatja aadressi määrata - et selleks ei tuleks apache@localhost või midagi muud sarnast kahtlast, siis saab lisaparameetritesse juurde lisada kirja päiseid. Nagu näiteks siin, kus saatja ja vastuse ootaja aadress eraldi märgitud

```
<?php
  mail("jaagup@tlu.ee", "tervist", "tere",
    "From: J.K <jaagup@minitorn.tlu.ee>\nReply-to: testpisi@hot.ee");
?>
```

Päiste üle kirjutamisel tasub aga sellega ettevaatlik olla, et mõned kirjaserverid vähemalt 2009ndal aastal suhtuvad sellisesse muudetud päistega kirjadesse ettevaatlikult ning saadavad suurema spämmikontrolli juurde.

Kasutaja ja parooli kombinatsiooni sobivuse kontrolliks tuleb samuti parool enne võrdlemist räsiks teha. Ning praegusel juhul saadetakse leitud ja sobiva parooliga kasutaja puhul vastuseks tema id-number, muul juhul saadetakse andmete sobimatuse näitamiseks tagasi arv 0.

kasutajahaldus.class.php

```
<?php
  class Kasutajahaldus{
     private $ab;
     function __construct($yhendus){
       $this->ab=$yhendus;
     }
     function lisaKasutaja($knimi, $parool, $epost){
        $kask=$this->ab->prepare("SELECT id FROM kasutajad WHERE knimi=?");
        $kask->bind_param("s", $knimi);
        $kask->execute();
        if($kask->fetch()){
          return "Kasutaja $knimi juba olemas";
        }
        $r2si=sha1($parool);
        $kask=$this->ab->prepare("INSERT INTO kasutajad (knimi, paroolir2si, epost)
                VALUES (?, ?, ?)");
        $kask->bind_param("sss", $knimi, $r2si, $epost);
        $kask->execute();
        mail($epost, "Registreeritud",
              "Registreerid end huvialade andmebaasi kasutajana $knimi.");
        return ""; //veatekst puudub;
     }
     function kontrolliKasutaja($knimi, $parool){
        $r2si=sha1($parool);
        $kask=$this->ab->prepare("SELECT id FROM kasutajad WHERE knimi=?
                              AND paroolir2si=?");
        $kask->bind_param("ss", $knimi, $r2si);
        $kask->bind_result($id);
        $kask->execute();
        if($kask->fetch()){
          return $id;
        }
        return 0;
     }
  }
?>
```

Kuna rakenduse juures ei piirduta tulevikus vaid kasutajate lisamise ja kontrollimisega, siis sai siia

ikka funktsioonide ohjamiseks tehtud fail abifunktsioonid.php. Seal loetakse loodud koodifail sisse ning tehakse selle klassi põhjal üks eksemplar. Edasine toimetus käibki objektiga, mil nimeks $khaldus ning mille oskused Kasutajahalduse klassis kirjeldatud. Siin näiteks piisab klassi põhjal tehtud ühest objekti eksemplarist. Kui aga allpool asub objekt toimetama ühe kasutaja huvialadega, siis võidakse samaaegselt ühe klassi põhjal teha mitu aktiivset objekti- näiteks ühe iga aktiivse kasutaja kohta. Ning kasutaja andmed on loodud objektil juba küljes.

Abifunktsioonide failis siis saadetakse session_start-käsuga teele sessioonimuutujate meelespidamiseks vajalikud päiseread. Edasi loetakse sisse loodud kasutajahalduse klassi fail ning luuakse selle põhjal kasutajahalduse suhtes tegutsemisvõimeline objekt. Seda objekti saab igal pool kasutada, kus on sisse loetud fail abifunktsioonid.php

abifunktsioonid.php

```
<?php
  session_start();
  require("kasutajahaldus.class.php");
  $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");
  $khaldus=new Kasutajahaldus($yhendus);

?>
```

Edasi siis tegelikult veebis nähtav fail, kus võetakse ette kasutaja registreerimise ning meldimisega seotud toimingud. Aadressireal või POST-meetodiga saabuvate parameetrite järgi otsustatakse, milline toiming on vaja parajasti ette võtta.

Kõigepealt loetakse sisse abifunktsioonide fail, et oleks sealtkaudu kättesaadav kasutajahalduse objekt. Samuti saadetakse seal algul välja sessioonimuutujate salvestamiseks tarvilikud päised. Järgnev sisenemisteate muutuja võimaldab kokku korjata toimingute käigus tekkivad teated ning need viisakalt oma lehele näitamiseks sobilikku piirkonda kokku korjata.

```
  if(isSet($_REQUEST["parool2"])){
```

kontrollib kordusparooli sisestamist - ehk siis toimingut, mis vajalik vaid uue kasutaja registreerimisel. Hilisemal sisenemisel piisab ühekordsest sisestusest. Järelikult - kui saabub parameeter nimega parool2, siis tuleb ette võtta uue kasutaja lisamisega seotud toimingud. Proovitakse käivitada Kasutajahalduse klassi eksemplari käsklus lisaKasutaja. Kui tuli veateade, siis jäetakse sisestatud kasutajanimi ja elektronpost meelde muutujatesse $uusknimi ja $epost, et kasutaja ei peaks neid tühjalt kohalt uuesti sisestama hakkama. Samuti jäetakse kasutajanimi ja elektronpostiaadress meelde juhul kui sisestatud paroolid ei kattunud - sellisel juhul tuleb kasutajal lihtsalt paroolid uuesti kirja panna - neid kaasa ei panda.

Kui registreerimine õnnestus, siis session_destroy kaotab igasugused meelde jäänud sessioonimuutujad, et võiks rahumeeli puhta lehena sisse meldida.

Juhul, kui lehele saabub parool, kuid mitte parool2 - sellisel juhul peab tegemist olema hariliku sisselogimisega. Kasutajahalduse käsklus kontrolliKasutaja teeb kindlaks, kas selliste tunnustega pääseb sisse. Õnnestumise korral jäetakse sessioonimuutujatesse meelde kasutajanimi ning kasutaja id. Viimane põhjusel, et selle kaudu on mugavam seosetabelitest hiljem andmeid hankida.

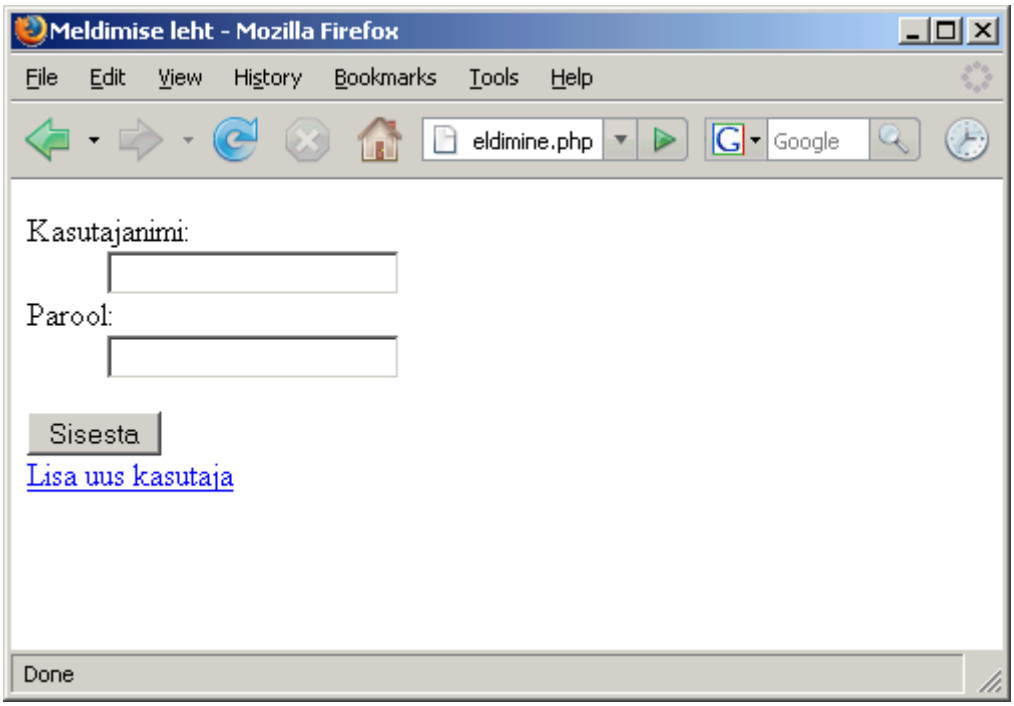
Välja logimisel piisab session_destroy-käsklusest, mis platsi puhtaks teeb.

meldimine.php

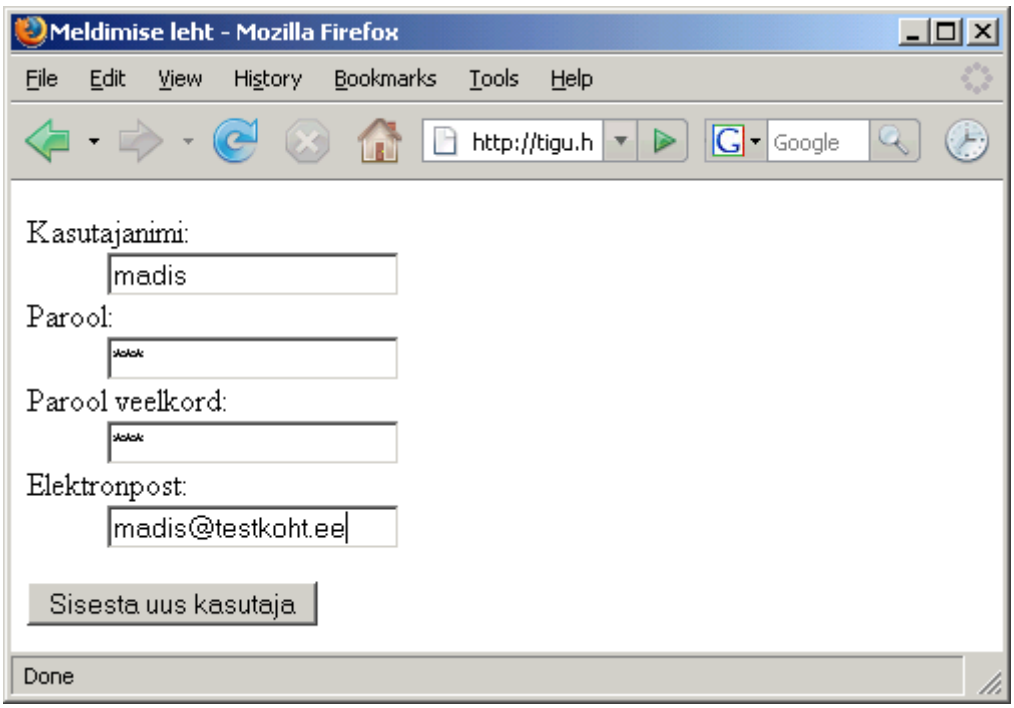
```
<?php
  require("abifunktsioonid.php");
  $sisenemisteade="";
  if(isSet($_REQUEST["parool2"])){
     if($_REQUEST["parool"]!=$_REQUEST["parool2"]){
        $uusknimi=$_REQUEST["kasutajanimi"];
        $epost=$_REQUEST["epost"];
        $sisenemisteade="Paroolid ei kattu";
     } else {
        $sisenemisteade=$khaldus->lisaKasutaja($_REQUEST["kasutajanimi"],
                          $_REQUEST["parool"], $_REQUEST["epost"]);
        if($sisenemisteade!=""){
          $epost=$_REQUEST["epost"];
        }  else {
          session_destroy();
          $sisenemisteade="Kasutaja $_REQUEST[kasutajanimi] registreeritud.";
        }
     }
  }
  if(isSet($_REQUEST["parool"]) AND !isSet($_REQUEST["parool2"])){
     $id=$khaldus->kontrolliKasutaja($_REQUEST["kasutajanimi"], $_REQUEST["parool"]);
     if($id!=0){
       $_SESSION["knimi"]=$_REQUEST["kasutajanimi"];
       $_SESSION["kid"]=$id;
     } else {
       $sisenemisteade="Vigane meldimine";
     }
  }
  if(isSet($_REQUEST["lahku"])){
    session_destroy();
    header("Location: meldimine.php");
  }
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Meldimise leht</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
    <?php if(!isSet($_SESSION["knimi"]) AND !isSet($_REQUEST["lisauus"]) AND !
isSet($epost)): ?>
       <form action="meldimine.php" method="post">
          <?=$sisenemisteade ?>
          <dl>
             <dt>Kasutajanimi:</dt>

             <dd><input type="text" name="kasutajanimi" /></dd>
             <dt>Parool:</dt>
             <dd><input type="password" name="parool" /></dd>
          </dl>
          <input type="submit" value="Sisesta" /><br />
          <a href="meldimine.php?lisauus=jah">Lisa uus kasutaja</a>
       </form>
    <?php endif ?>
    <?php if(isSet($_REQUEST["lisauus"]) OR isSet($epost)): ?>
       <form action="meldimine.php" method="post">
          <?=$sisenemisteade ?>
          <dl>
             <dt>Kasutajanimi:</dt>
             <dd><input type="text" name="kasutajanimi"  value=
                  "<?=((isSet($uusknimi))?$uusknimi:"") ?>" />
             </dd>
             <dt>Parool:</dt>
             <dd><input type="password" name="parool" /></dd>
             <dt>Parool veelkord:</dt>
             <dd><input type="password" name="parool2" /></dd>
             <dt>Elektronpost:</dt>
             <dd><input type="text" name="epost" value=
```

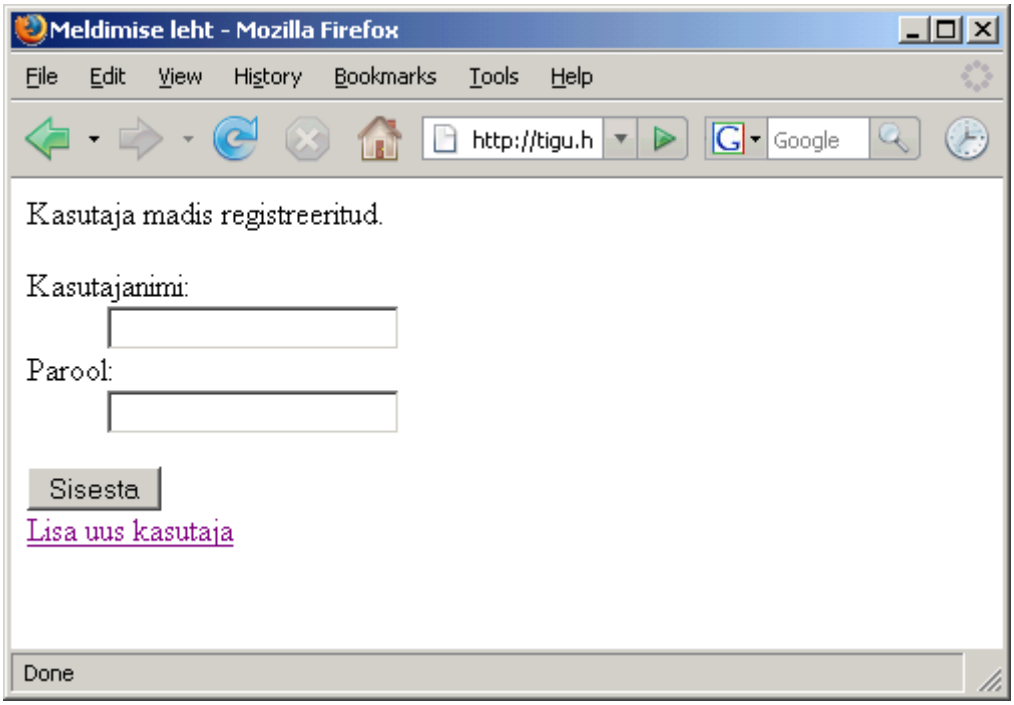
```
                "<?=((isSet($epost))?$epost:"") ?>"/></dd>
          </dl>
          <input type="submit" value="Sisesta uus kasutaja" /><br />
       </form>
    <?php endif ?>
    <?php if(isSet($_SESSION["knimi"])): ?>
         Tere, <?=$_SESSION["knimi"] ?>. <a href="meldimine.php?lahku=jah">Lahku</a> <br
/>
         <a href="kasutajahuvialad.php">Huvialade haldus</a>
    <?php endif ?>
  </body>
</html>
```



Tühi avaleht



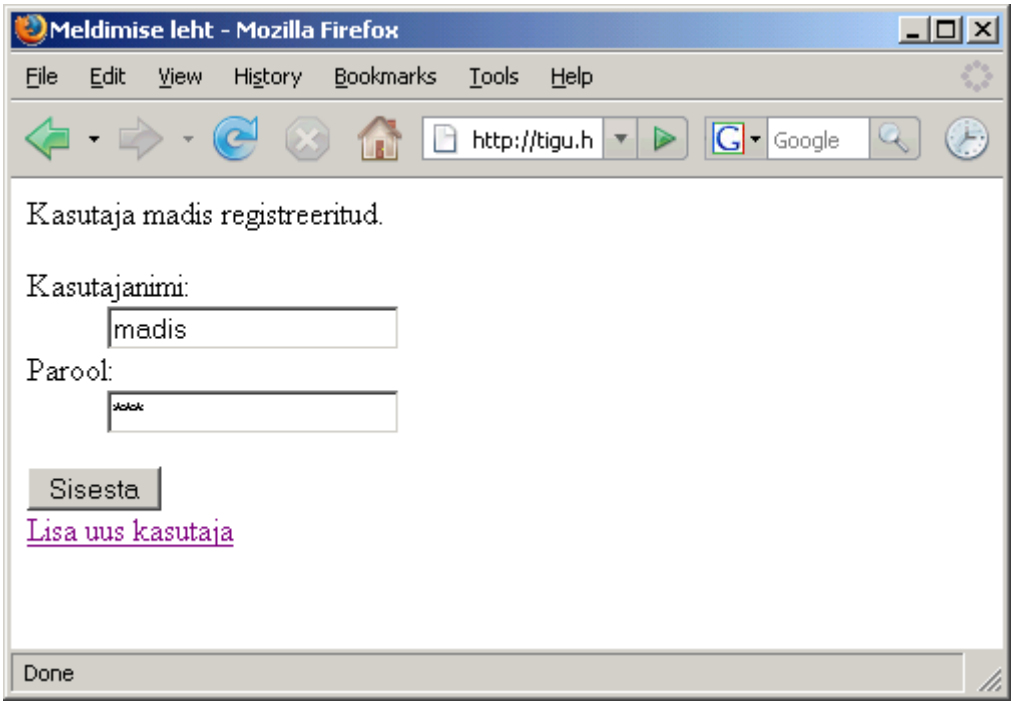
Kasutaja lisamine



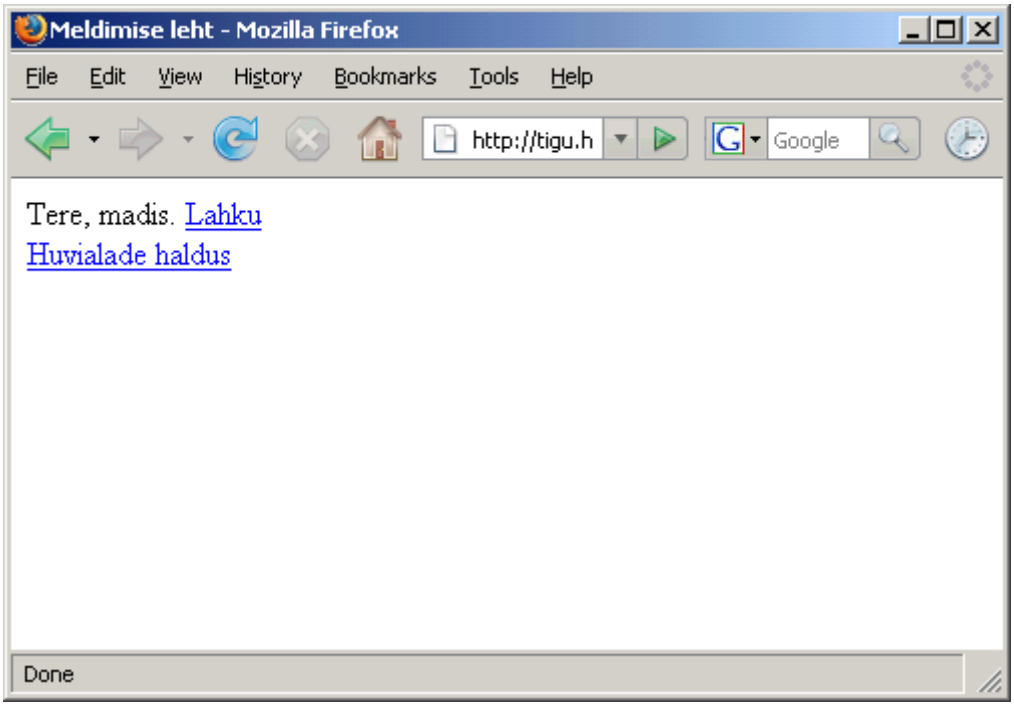
Lisamisteade

```
mysql> SELECT * FROM kasutajad;
+----+-------+------------------------------------------+-------------------+
| id | knimi | paroolir2si                              | epost             |
+----+-------+------------------------------------------+-------------------+
| 25 | madis | 054b8a97845b86a485ee89beb31ff3447379e38c | madis@testkoht.ee |
+----+-------+------------------------------------------+-------------------+
1 row in set (0.00 sec)
```

Andmebaasitabelisse tekkinud kirje. Nagu näha, siis parooli koha peal on parajalt krüptiline räsi. Selle järgi saab küll teadaoleva/katsetatava parooli kõlbulikkust kontrollida, kuid mitte tagurpidi parooli ennast leida.



Kord registreeritud kasutajaga saab end sisse möllida.



Kes juba sees, siis võimalik liikuda tegeliku asjaliku lehe ehk huvialada halduse juurde.

Kasutaja huvialade vaatamiseks ja haldamiseks loodi taas klass omaette failis. Taas koondatakse siia käsud, mis muidu veebilehe enese koodi kipuksid segakseks muutma - siin eraldi koodifailis on aga nad täiesti omal kohal.

Kasutaja lehele tuleb loetelu selle kasutaja huvialade kohta. Samuti tuleb teine loetelu huvialade kohta, mida kasutajal veel ei ole - et ta võiks sealt soovi korral enesele sobivad valida.

Selgitus kulub ära ehk vabade huvialade leidmise päringu kohta:

```
SELECT id, huviala FROM huvialad WHERE id NOT IN
         (SELECT huviala_id FROM kasutajad_huvialad WHERE kasutaja_id=?)
```

Tegemist piirangu poolelt siis alampäringuga, kus antakse ette kõik vastava id-ga kasutaja huvialad. Ning põhipäringus küsitakse välja kõik huvialad, millest siis alampäringu abil arvatakse välja need huvialad, mis kasutajale juba märgitud on - tulemuseks jäävadki selle kasutaja jaoks veel vabad huvialad.

Valitud huvialad koos huviala nimega tabeleid ühendavas päringus näeb välja järgmine:

```
SELECT kasutajad_huvialad.id as seose_id,
              huvialad.id as h_id, huviala
              FROM kasutajad_huvialad, huvialad
              WHERE kasutajad_huvialad.huviala_id=huvialad.id
                AND kasutajad_huvialad.kasutaja_id=?
```

Iseenesest kannatanuks selle ka alampäringuga kokku panna nii nagu eelmise SQL-lause, kus NOT IN-i asemel jääks lihtsalt IN. Aga siin soovime ka kasutajad_huvialad tabeli seose id-d näha, et hiljem oleks selle järgi mugav seost muuta või kustutada. Siis võetakse ette päringusse kaks tabelit

ning seotakse nad sobiva välja kaudu kokku: kasutajad_huvialad.huviala_id näitab huvialade tabeli id peale.

Kasutajale huviala lisamisel lisatakse see huviala_id kaudu. Kasutaja ja huviala vahelise seose eemaldamiseks aga antakse ette seose id. Iseenesest siin näiteks piisaks ka huviala idst, sest kasutajal saab huviala olla vaid ühe korra. Tuleviku peale mõeldes aga, kus ei ühendata mitte kasutajaid ja huvialasid, vaid näiteks kliente ja tooteid ostudeks, siis võib kergesti juhtuda, et tahetakse pöörduda vaid ühe ostu andmete poole ning teiste sama kliendi sama toote ostudega parajasti mitte tegelda - sellisel juhul on seose id kaudu lähenemine kindlasti vajalik.

Edasi kasutaja huvialasid haldav koodilõik tervikuna.

kasutajahuvialad.class.php

```
<?php
  class KasutajaHuvialad{
    private $ab;
    private $kid;
    function __construct($yhendus, $kasutaja_id){
      $this->ab=$yhendus;
      $this->kid=$kasutaja_id;
    }
    function vabadHuvialad(){
      $kask=$this->ab->prepare("SELECT id, huviala FROM huvialad WHERE id NOT IN
         (SELECT huviala_id FROM kasutajad_huvialad WHERE kasutaja_id=?)");
      $kask->bind_param("i", $this->kid);
      $kask->bind_result($id, $huviala);
      $kask->execute();
      $hoidla=array();
      while($kask->fetch()){
         $h=new stdClass();
         $h->id=$id;
         $h->huviala=$huviala;
         array_push($hoidla, $h);
      }
      return $hoidla;
    }
    function valitudHuvialad(){
      $kask=$this->ab->prepare("SELECT kasutajad_huvialad.id as seose_id,
              huvialad.id as h_id, huviala
              FROM kasutajad_huvialad, huvialad
              WHERE kasutajad_huvialad.huviala_id=huvialad.id
                AND kasutajad_huvialad.kasutaja_id=?");
      $kask->bind_param("i", $this->kid);
      $kask->bind_result($seose_id, $h_id, $huviala);
      $kask->execute();
      $hoidla=array();
      while($kask->fetch()){
         $h=new stdClass();
         $h->seose_id=$seose_id;
         $h->h_id=$h_id;
         $h->huviala=$huviala;
         array_push($hoidla, $h);
      }
      return $hoidla;
    }
    function lisaHuvialad($huviala_idd){
       $kask=$this->ab->prepare("INSERT INTO kasutajad_huvialad (kasutaja_id,
          huviala_id) VALUES (?, ?)");
       foreach($huviala_idd as $hid){
         $kask->bind_param("ii", $this->kid,  $hid);
         $kask->execute();
       }
    }
    function eemaldaHuvialad($seoste_idd){
       $kask=$this->ab->prepare("DELETE FROM kasutajad_huvialad WHERE id=?");
       foreach($seoste_idd as $seose_id){
```

```
            $kask->bind_param("i", $seose_id);
            $kask->execute();
        }
    }
}
?>
```

Abifunktsioonide failis loetakse huvialade klass lihtsalt sisse. Klassi põhjal eksemplari aga siin veel ei tehta - otstarbekam on see luua alles seal, kus ta teeneid ka tegelikult vaja on.

abifunktsioonid.php

```
<?php
  session_start();
  require("kasutajahaldus.class.php");
  require("kasutajahuvialad.class.php");
  $yhendus=new mysqli("localhost", "jaagup", "xxxxxx", "jaagup");
  $khaldus=new Kasutajahaldus($yhendus);
?>
```

Sobiv koht siis failis, mis kohe eraldi mõeldud kasutaja huvialade haldamiseks. Et tegemist vaid kasutajale enesele haldustöödeks mõeldud lehega, siis sisse logimata edasi ei lasta, vaid suunatakse ümber meldimislehele.

```
  if(!isSet($_SESSION["knimi"])){
    header("Location: meldimine.php");
    exit();
  }
```

Kui kasutaja juba sees, siis on põhjust tema huvialade küsimiseks ja haldamiseks ka vastav objekt teha. Ette antakse andmebaasiühendus ning sisseloginud kasutaja id - neid läheb objektil oma käskluste käivitamise juures vaja.

```
  $khuvihaldus=new KasutajaHuvialad($yhendus, $_SESSION["kid"]);
```

Kasutajate huvialade halduse objekti käest küsitakse ühte muutujasse juba tema valitud huvialad, teise muutujasse need, mis veel loetelust tema jaoks vabad on.

```
  $valitud=$khuvihaldus->valitudHuvialad();
  $vabad=$khuvihaldus->vabadHuvialad();
```

Uus tehniline lahendus tärkab silma eemaldatavate huvialade märkimise juures. Tavapäraselt on igal veebivormi elemendil oma nimi. Ning andmed saab selle järgi aadressirealt või post-meetodiga lehe avamise juurest küsida. Kui aga huvialasid on teadmata arv - neist aga vaid osa märgitud, siis ei tundu mugav igaühele neist eraldi nimega elementi teha. Selleks puhuks aitab PHP puhul, kui elemendi nime lõppu lisada kantsulud []. Sellisel juhul jõuavad selle elemendi külge pandud andmed vastuvõtva PHP lehe juurde massiivina. Elementide väärtused saab siis juba vastava massiivi seest kätte. Olenevalt elemendist: tekstivälja puhul saadetakse väärtus igal juhul - ka siis kui tekstiväli on tühi, ehk sees tekst pikkusega 0 sümbolit. Märkeruutude puhul aga saadetakse elemendile määratud väärtus serverisse vaid juhul, kui ruut on märgitud. Siin näiteks pannakse väärtusena kaasa eemaldatava seose id-number.

```
    <form action="kasutajahuvialad" method="post">
      <?php foreach($valitud as $h): ?>
          <input type="checkbox" name="eemaldatav[]" value="<?=$h->seose_id ?>" />
               <?=$h->huviala ?><br />
      <?php endforeach ?>
      <input type="submit" name="eemaldamine"
                               value="Eemalda valitud huvialad kasutajalt" />
    </form>
```

PHP pool saab seda kasutada täiesti tavalise massiivina ning siin näites antakse see ette huvialade

alguse klassi eksemplari meetodile eemaldaHuvialad.

```
  if(isSet($_REQUEST["eemaldamine"])){
    $khuvihaldus->eemaldaHuvialad($_REQUEST["eemaldatav"]);
  }
```

Seal käiakse seoste id-d läbi ning eemaldatakse vastavad seosed.

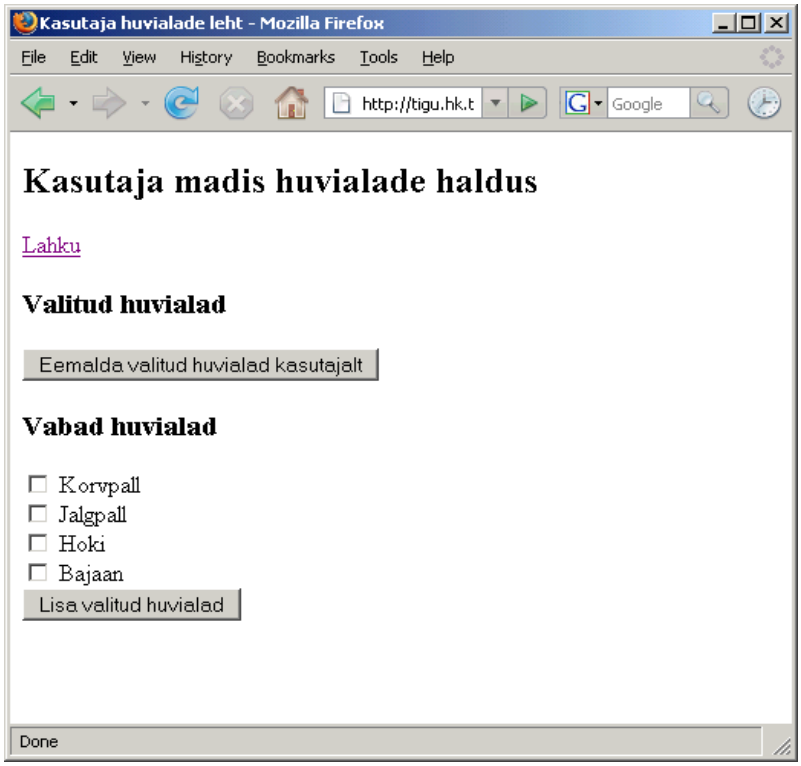
```
    function eemaldaHuvialad($seoste_idd){
       $kask=$this->ab->prepare("DELETE FROM kasutajad_huvialad WHERE id=?");
       foreach($seoste_idd as $seose_id){
          $kask->bind_param("i", $seose_id);
          $kask->execute();
       }
    }
```

Ning kasutaja huvialade haldamise leht tervikuna.

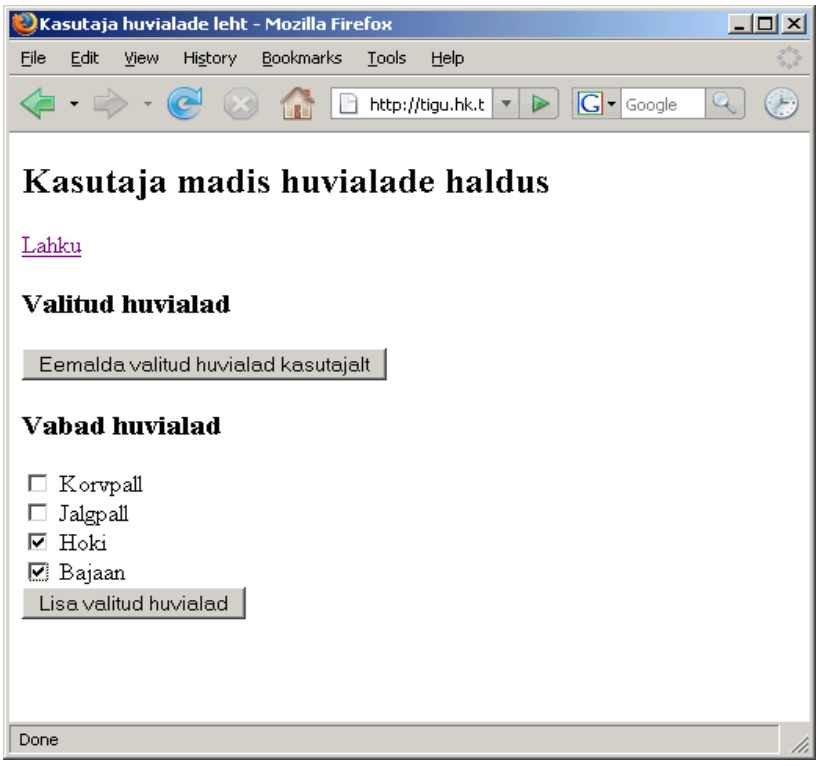
kasutajahuvialad.php

```
<?php
  require("abifunktsioonid.php");
  if(!isSet($_SESSION["knimi"])){
    header("Location: meldimine.php");
    exit();
  }
  $khuvihaldus=new KasutajaHuvialad($yhendus, $_SESSION["kid"]);
  if(isSet($_REQUEST["lisamine"])){
    $khuvihaldus->lisaHuvialad($_REQUEST["lisatav"]);
    header("Location: kasutajahuvialad.php");
    exit();
  }
  if(isSet($_REQUEST["eemaldamine"])){
    $khuvihaldus->eemaldaHuvialad($_REQUEST["eemaldatav"]);
  }
  $valitud=$khuvihaldus->valitudHuvialad();
  $vabad=$khuvihaldus->vabadHuvialad();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Kasutaja huvialade leht</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
    <h2>Kasutaja <?=$_SESSION["knimi"] ?> huvialade haldus</h2>
    <a href="meldimine.php?lahku=jah">Lahku</a>
    <h3>Valitud huvialad</h3>
    <form action="kasutajahuvialad" method="post">
      <?php foreach($valitud as $h): ?>
          <input type="checkbox" name="eemaldatav[]" value="<?=$h->seose_id ?>" />
               <?=$h->huviala ?><br />
      <?php endforeach ?>
      <input type="submit" name="eemaldamine" value="Eemalda valitud huvialad kasutajalt"
/>
    </form>
    <h3>Vabad huvialad</h3>
    <form action="kasutajahuvialad" method="post">
      <?php foreach($vabad as $h): ?>
          <input type="checkbox" name="lisatav[]" value="<?=$h->id ?>" />
               <?=$h->huviala ?><br />
      <?php endforeach ?>
      <input type="submit" name="lisamine" value="Lisa valitud huvialad" />
    </form>
  </body>
</html>
```

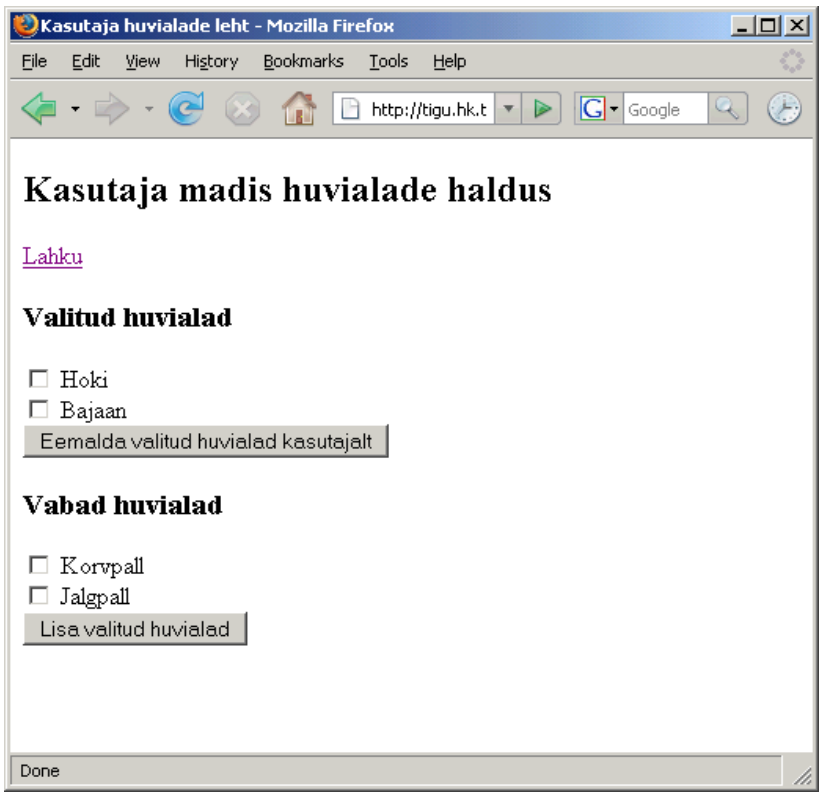
Lehe avamisel pole kasutajal veel ühtki huviala märgitud, kõik on vabad.



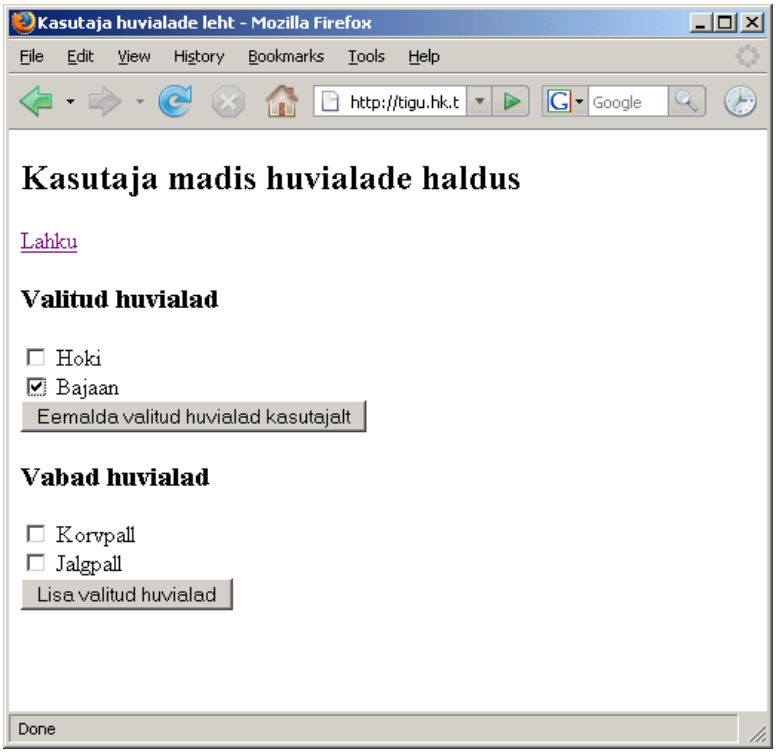
"Linnukestega" saab sobivad ära märkida.



Lisamisnupu peale kirjutatakse nende huvialade id-d kasutajate ja huvialade seosetabelisse ning lehe avamisel paistab välja, millised huvialad kasutaja juures märgitud on.



Soovides huvialadest vabaneda, tasub see lihtsalt märgistada ja vajutada eemaldusnuppu.



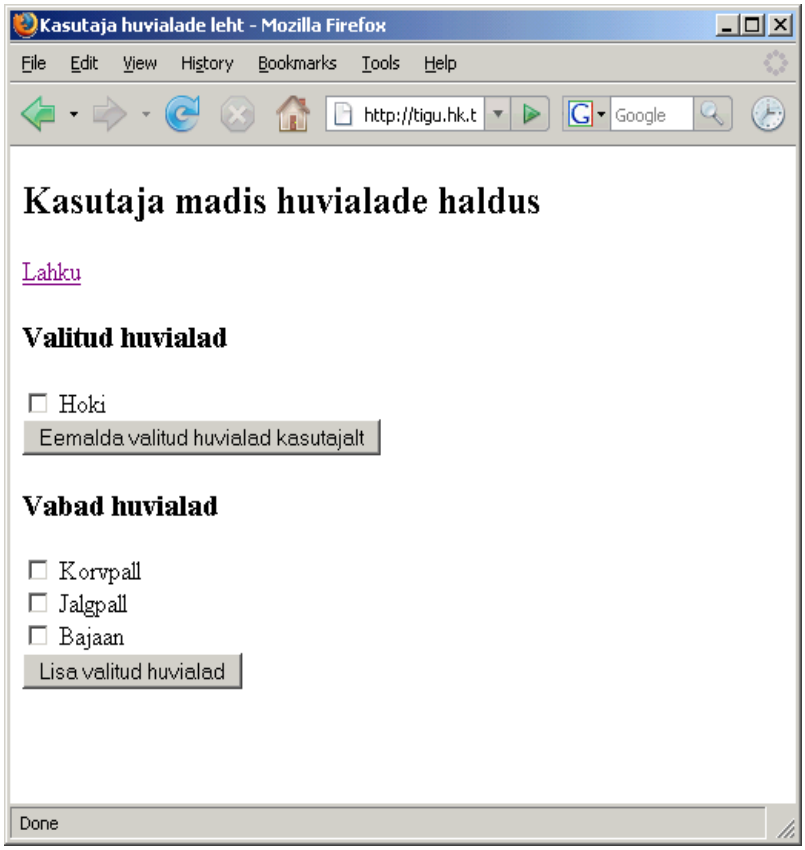
Andmebaasist kontrollides näeb, et kasutajale on jäänud vaid üks huviala, id-ga 3 ehk Hoki.

```
mysql> SELECT * FROM huvialad;
+----+----------+
| id | huviala  |
+----+----------+
|  1 | Korvpall |
```

```
|  2 | Jalgpall |
|  3 | Hoki     |
|  4 | Bajaan   |
+----+----------+

mysql> SELECT * FROM kasutajad_huvialad;
+----+-------------+------------+
| id | kasutaja_id | huviala_id |
+----+-------------+------------+
|  7 |          25 |          3 |
+----+-------------+------------+
1 row in set (0.00 sec)
```

Sama tulemus paistab ka kasutaja huvialasid näitavalt lehelt.



Ülesandeid

* Tee näide läbi
* Lisa kasutajale tulp näitamaks tema viimast sisselogimise aega.
* Loenda kasutaja juures, mitu korda ta on sisse loginud.
* Lisa kasutajale huvialade juurde valik - (ala tundmatu, tunnen huvi, tegelen aktiivselt)

* Koosta bussipeatuste tabel (id, peatusenimi). Lisa mõned peatused
* Koosta bussiliinide tabel (liini_nr, bussi_suurus). Lisa mõned liinid
* Koosta peatumisaegade tabel (id, liini_id, peatuse_id, kellaaeg). Lisa mõned peatumised
* Koosta leht ühe bussiliini peatumisandmete näitamiseks. Andmed sorditakse kellaaja järgi.

* Koosta leht peatumiste lisamiseks bussiliinile. Rippmenüüst saab valida peatuse, käsitsi sisestatakse kellaeg.
* Liinilt saab peatumisi kustutada.

* Ühe liini juures olevaid peatumiste kellaaegu saab korraga ühel lehel muuta.

Andmetabel päringus mitme koopiana

Seni on selge olnud, millisest tabelist me andmed võtame, või millise tabeli teise tabeli kõrvale ühendame. Keerukamate infosüsteemide puhul tekib aga kergesti olukordi, kus samast tabelist on vaja mitmes kontekstis olevaid andmeid võtta. Näiteks, kui tabelis on inimeste andmed. Ning tabelis olevatel inimestel on id-de kaudu viited ka nende isa ja ema andmetele samas inimeste tabelis, siis inimese ja tema vanemate eesnimede kättesaamiseks läheb vaja pöördumist sama tabeli poole kolmes erinevas kontekstis: uuritava inimese andmete saamiseks ning eraldi tema isa ja ema andmete saamiseks. Sellegipoolest pole üldjuhul vaja eraldi teha õppurite tabelit, isade tabelit ja emade tabelit, vaid õnnestub päringu ajal vähemalt näiliselt luua sellest tabelist kolm koopiat ning seoste abil määrata, millisest reast just millised andmed välja lugeda tuleb.

Siinse rakenduse juures võib tekkida tahtmine inimesel leida enesega kattuvate huvialadega kaaslasi. Ka sellisel juhul läheb kasutajaid ja huvialasid ühendav tabel käiku kahel korral: kõigepealt saab kasutaja id järgi teada temaga seotud huvialade id-d. Edasi on nende huvialade koodide järgi võimalik leida inimesed, kes on vastavate huvialadega seotud. Ehk siis lahendatava ülesande kirjeldus näeks välja:

Leia kasutajad, kel on etteantuga vähemasti üks kattuv huviala

Ning koostatav päring tuleb järgmine:

```
SELECT *
  FROM kasutajad_huvialad AS kh1, kasutajad_huvialad AS kh2
  WHERE kh1.huviala_id=kh2.huviala_id AND kh1.kasutaja_id=25;
```

Tabel kasutajad_huvialad võetakse päringusse kahel korral. Ühel korral anname talle nime kh1, teisel korral kh2. Tagapool oleva tingimusega määrame, et otsime kaaslasi kasutajale, kel id-ks 25. Ning teise tingimusega määrame, et kasutajale 25 leitud huvialad peavad olema samad, mis ülejäänud näidatavatel ridadel seosetabelist.

```
mysql> SELECT *
    ->   FROM kasutajad_huvialad AS kh1, kasutajad_huvialad AS kh2
    ->   WHERE kh1.huviala_id=kh2.huviala_id AND kh1.kasutaja_id=25;
+----+-------------+------------+----+-------------+------------+
| id | kasutaja_id | huviala_id | id | kasutaja_id | huviala_id |
+----+-------------+------------+----+-------------+------------+
|  7 |          25 |          3 |  7 |          25 |          3 |
|  7 |          25 |          3 | 10 |          27 |          3 |
+----+-------------+------------+----+-------------+------------+
```

Vastust lähemalt analüüsides selgub, et kõige lähemaks kaaslaseks kasutajale 25 leitakse kasutaja 25 ise :-). Ehk siis seos nr. 7 määrab, et kasutajal 25 on huviala nr. 3. Ning teistpidi kirjeid otsides on huviala 3 olemas kasutajal 25 seose nr. 7 kaudu. Mis on igati loomulik - iseasi, kas me seda just päringust ootasime. Aga peale iseenesele leidsime kasutajale ikka ühe kaaslase ka: huviala 3 on muuhulgas olemas ka kasutajal 27.

Kui tahta, et kasutaja ei peaks saatma iseenesele hulgem kirju adressaadiga "Muhv, nõudmiseni", siis võib määrata, et tabeli teise koopia kaudu leitud kasutaja id ei kattuks esimese koopia kasutaja id-ga. Ehk siis otsitakse kasutajaid, kel on küll sama huviala id, kuid mitte iseenesega sama kasutaja id. Suurem-väiksem märgid koos tähendavad, et väärtused ei oleks võrdsed.

```
SELECT *
  FROM kasutajad_huvialad AS kh1, kasutajad_huvialad AS kh2
  WHERE kh1.huviala_id=kh2.huviala_id AND kh1.kasutaja_id=25
     AND kh1.kasutaja_id <> kh2.kasutaja_id;
```

Kui päring tööle panna, siis on näha, et iseennast enam enesele sobivaks kaaslaseks ei loeta. Kasutajale 25 loetakse huviala 3 kaudu sobivaks kaaslaseks kasutaja numbriga 27.

```
mysql> SELECT *
    ->   FROM kasutajad_huvialad AS kh1, kasutajad_huvialad AS kh2
    ->   WHERE kh1.huviala_id=kh2.huviala_id AND kh1.kasutaja_id=25
    ->      AND kh1.kasutaja_id <> kh2.kasutaja_id;
+----+-------------+------------+----+-------------+------------+
| id | kasutaja_id | huviala_id | id | kasutaja_id | huviala_id |
+----+-------------+------------+----+-------------+------------+
|  7 |          25 |          3 | 10 |          27 |          3 |
+----+-------------+------------+----+-------------+------------+
```

Programmeerijad saavad paljaste id numbritega päris hästi hakkama. Tavakasutajad aga eelistavad kasutajanimesid lugeda. Kaaslasele kirja saatmiseks on hea teada ka ta elektronpostiaadressi. Koodi järgi kasutaja teada saamiseks tuleb päringusse juurde liita kasutajate tabel. Kus siis seosetabeli teise koopia ehk otsitava kaaslase rea juures oleva kasutaja id järgi võetakse kasutajate tabelist välja sobiv rida. Nii tuleb välja, et me 25ndale kasutajale sobivaks kaaslaseks on kasutaja nimega kati.

```
SELECT knimi, epost
  FROM kasutajad_huvialad AS kh1, kasutajad_huvialad AS kh2,
       kasutajad
  WHERE kh1.huviala_id=kh2.huviala_id
     AND kh1.kasutaja_id <> kh2.kasutaja_id
     AND kh2.kasutaja_id=kasutajad.id
     AND kh1.kasutaja_id=25;
```

```
+-------+--------------------+
| knimi | epost              |
+-------+--------------------+
| kati  | kati@testserver.ee |
+-------+--------------------+
```

Ilus oleks juurde vaadata ka huviala nime, mis neid siis kokku seob. Ega muud kui päringusse jälle üks tabel juurde. Nüüd pole enam tähtis, kas see seotakse esimese või teise seosetabeli huviala id tulbaga, sest need kohe esimese tingimusena ju sama huvialaga inimeste otsingu juures kattuvad. Edasi nagu näha vaadatakse, et teise seosetabeli koopia kasutaja_id näitaks kasutajate tabeli id-e ning samuti praegu teise seosetabeli huviala_id näitaks huvialade tabeli primaarvõtmele. Lõppu veel piirang, et näidatagu ainult neid seoseid, kus seosetabeli esimese koopia juures on kasutaja id-ks 25. Seetõttu siis tema huvialad (need, mis kellelgi teisel ka on) ja sealtkaudu ka kaaslased.

```
SELECT knimi, epost, huviala
  FROM kasutajad_huvialad AS kh1, kasutajad_huvialad AS kh2,
       kasutajad, huvialad
  WHERE kh1.huviala_id=kh2.huviala_id
     AND kh1.kasutaja_id <> kh2.kasutaja_id
     AND kh2.kasutaja_id=kasutajad.id
     AND kh2.huviala_id=huvialad.id
     AND kh1.kasutaja_id=25;
```

```
+-------+--------------------+---------+
| knimi | epost              | huviala |
+-------+--------------------+---------+
| kati  | kati@testserver.ee | Hoki    |
+-------+--------------------+---------+
```

Eelmise tabeli andmed olid mõeldud näitamiseks/kasutamiseks ühe konkreetse kasutaja lehel. Kui tahta sobivustest kokkuvõtet teha, siis ei pea enam kasutaja id-d piirama - näidatagu kõiki, kel leidub omale sobiva huvialaga kaaslasi. Kuna vaadatavad tulbad (kasutajanimi ja elektronpost) kipuksid päringus korduma, siis nimetame nad ümber. Vaadeldava kasutajanime tulbaks määrame "kes" ning tema kaaslaseks leitu kasutajanime tulba juurde kirjutame sõna "kellega". Samuti eposti tulpade nimed lähevad muutusesse, et hilisema päringu juures segadust ei tekiks. Mõnes süsteemis saab tulpade andmed küll ka tulba järjekorranumbri järgi kätte, aga erinev pealkiri ikka kindlam.

```
SELECT k1.knimi as kes, k1.epost as epost1, huviala,
       k2.knimi as kellega, k2.epost as epost2
  FROM kasutajad_huvialad AS kh1, kasutajad_huvialad AS kh2,
       kasutajad as k1, kasutajad as k2, huvialad
  WHERE kh1.huviala_id=kh2.huviala_id
     AND kh1.kasutaja_id <> kh2.kasutaja_id
     AND kh1.kasutaja_id=k1.id
     AND kh2.kasutaja_id=k2.id
     AND kh2.huviala_id=huvialad.id
```

```
+-------+--------------------+---------+---------+--------------------+
| kes   | epost1             | huviala | kellega | epost2             |
+-------+--------------------+---------+---------+--------------------+
| kati  | kati@testserver.ee | Hoki    | madis   | madis@testkoht.ee  |
| madis | madis@testkoht.ee  | Hoki    | kati    | kati@testserver.ee |
+-------+--------------------+---------+---------+--------------------+
```

Nii on ilusti näha, kes kellega hokit mängida suudab. Samas koorub aga välja, et sama seos on väljundis tegelikult kaks korda kirjas. Kati võib mängida hokit Madisega ning Madis Katiga. Jällegi loogiline, aga võibolla mitte see, mida ootasime. Kui määrata, et väljundtabelis peab vasakul pool olevas tulbas paiknev kasutajanimi olema tähestikus eespool paremal pool asuva kasutaja nimest, siis selle peale korduvad nimed eemaldatakse.

```
SELECT k1.knimi as kes, k1.epost as epost1, huviala,
       k2.knimi as kellega, k2.epost as epost2
  FROM kasutajad_huvialad AS kh1, kasutajad_huvialad AS kh2,
       kasutajad as k1, kasutajad as k2, huvialad
  WHERE kh1.huviala_id=kh2.huviala_id
     AND kh1.kasutaja_id <> kh2.kasutaja_id
     AND kh1.kasutaja_id=k1.id
     AND kh2.kasutaja_id=k2.id
     AND kh2.huviala_id=huvialad.id
     AND k1.knimi < k2.knimi
```

```
+------+--------------------+---------+---------+--------------------+
| kes  | epost1             | huviala | kellega | epost2             |
+------+--------------------+---------+---------+--------------------+
| kati | kati@testserver.ee | Hoki    | madis   | madis@testkoht.ee  |
+------+--------------------+---------+---------+--------------------+
```

Nõnda võibki pealtnäha suhteliselt väheste tabelite juurest küllalt palju välja lugeda.

Ülesandeid

* Tee näited läbi.
* Katseta vastuseid rohkemate kasutajate ja seoste juures.
* Sorteeri viimatise päringu vastused kord uuritava kasutajanime, kord huviala järgi.

* Loo/otsi eelmise ploki ülesandes olnud tabelid peatuste, bussidliinide ja peatumiste kohta.
* Näita bussiliini juures, millistele liinidele ümberistumised sellelt liinilt on võimalikud (liinid

kasutavad sama peatust)
* Näita iga ümberistutava liini juures ka ümberistumiseks kõlbuliku peatuse nimi.
* Ümberistumist loetakse võimalikuks, kui ümberistutava liini peatumiskellaaeg on hilisem kui peatusesse saabumiseks kasutatava liini peatumisaeg.

Agregaatfunktsioonid, grupeerimine

SQLi sees on tänuväärsed võimalused andmetest kokkuvõtete tegemiseks. PHP koodi kaudu saab iseenesest pea kõike arvutada, aga samas andmebaasi oma vahenditega võivad selleised päringud olla märgatavalt lihtsamad ja kiiremad.

Levinumaks agregaat- ehk kokkuvõttefunktsiooniks on tõenäoliselt COUNT. Loetakse kokku, mitu rida päringu tulemusena väljastatakse. SELECT * FFOM kasutajad väljastanuks meil praegu kolme kasutaja andmed. SELECT COUNT(*) FROM kasutajad väljastab aga viisakalt vaid ühe arvu, näitamaks, kui palju registreeritud kasutajaid tabelis on.

```
SELECT COUNT(*) FROM kasutajad;

+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
```

Tahtes tekkinud tulbale viisaka nime anda, aitab ümbernimetamine AS- i abil. Eriti tähtis see oludes, kus tabelitulpade nimed lähevad kohe automaatselt otse veebi peale nähtavaks.

```
SELECT COUNT(*) as kasutajate_arv FROM kasutajad;

+----------------+
| kasutajate_arv |
+----------------+
|              3 |
+----------------+
```

Tuntumad agregaatfunktsioonid veel MAX, MIN ja AVG - kasutades tuleb lihtsalt ette anda tulba nimi, milles olevatest väärtustest siis tuleb suurim, vähim ja keskmine välja leida.

Kui soovitakse aga huvialade arv kasutajate kaupa kindlaks teha, sis aitab funktsioon GROUP BY. Sellise statistika saab muidugi vajadusel PHP oma massiivide ja tsüklite abil korraldada, kuid SQLi abil tuleb enamasti tunduvalt lühem ja mugavam. Tahtes iga kasutaja juurde kokku lugeda, mitu huviala tal kirjas, tasub saada huvialade seosetabelist kätte erinevad kasutaja id-d. Grupeerimiskäsu puhul näidatakse neist iga erinevat vaid ühe korra. Sarnase tulemuse annaks ka päringus olev DISTINCT. GROUP BY aga lubab lisaks erinevate väärtuste kuvamisele ka samasse gruppi kuuluvate grupeerimistulba ühesuguste väärtustega ridade peale agregeerimisfunktsioone rakendada - ehk siis neid ridu loendada, neist suuremaid, väiksemaid või keskmisi võtta. Siin loetakse siis iga huvialadega kasutaja kohta kokku tema huvialade kogus.

```
SELECT kasutaja_id, COUNT(*) as huvialade_kogus
  FROM kasutajad_huvialad
  GROUP BY kasutaja_id;

+-------------+-----------------+
| kasutaja_id | huvialade_kogus |
+-------------+-----------------+
|          25 |               1 |
|          27 |               2 |
```

```
+-------------+-----------------+
```

MySQLil on mitme väärtuse ühes lahtris näitamiseks mugav funktsioon nimega GROUP_CONCAT - gruppi jäävad väärtused väljastatakse järjestikuse tekstina, vaikimisi eraldajaks on koma.

```
SELECT kasutaja_id, GROUP_CONCAT(huviala_id) as huvialanumbrid
  FROM kasutajad_huvialad
  GROUP BY kasutaja_id;

+-------------+-----------------+
| kasutaja_id | huvialanumbrid |
+-------------+-----------------+
|          25 | 3               |
|          27 | 1,3             |
+-------------+-----------------+
```

Kui ei piisa ainult numbritest, vaid tahetakse ka kasutajanimesid ja huvialade nimetusi näha, siis tuleb lisada vastavad tabelid. Ning id-de järgi kokku ühendada, et milline tulp kuhu viitab. Ikka seosetabeli kasutajad_huvialad tulp kasutaja_id viitab kasutajate tabeli id-tulbale ning seosetabeli huviala_id viitab huvialade tabeli id-tulbale. Nii saab ilusa loetelu nimedest ja huvialadest.

```
SELECT knimi, huviala
  FROM kasutajad, kasutajad_huvialad, huvialad
  WHERE kasutajad_huvialad.huviala_id=huvialad.id
    AND kasutajad_huvialad.kasutaja_id=kasutajad.id;

+-------+----------+
| knimi | huviala  |
+-------+----------+
| madis | Hoki     |
| kati  | Korvpall |
| kati  | Hoki     |
+-------+----------+
```

Ka siin sobib grupeerimiskäsklust kasutada. Kasutajanime järgi grupeerides on igal real üks kasutajanimi. GROUP_CONCAT-käsuga saab kõrvaltulpa ilusti koondada selle kasutaja huvialade nimetused.

```
SELECT knimi, GROUP_CONCAT(huviala) AS huvialanimed
  FROM kasutajad, kasutajad_huvialad, huvialad
  WHERE kasutajad_huvialad.huviala_id=huvialad.id
    AND kasutajad_huvialad.kasutaja_id=kasutajad.id
  GROUP BY knimi;

+-------+---------------+
| knimi | huvialanimed  |
+-------+---------------+
| kati  | Korvpall,Hoki |
| madis | Hoki          |
+-------+---------------+
```

Ülesandeid

* Tee näited läbi
* Koosta päring, kus iga huviala juures näidatakse ära sellega seotud kasutajate arv.
* Koosta päring, kus iga huviala juures näidatakse ära kõik kasutajanimed, kes selle huvialaga seotud on.

Failide üleslaadimine

Märgatav osa veebi kaudu sisu haldamisega seotud rakendusi jõuavad millalgi ka failide üleslaadimiseni - olgu selleks siis näidatavad dokumendid, pildid või hoopis saadetavad XML-andmed. Kinnipüütud failidega on kolm tüüpilist võimalust: nad kas pannakse ootele kuhugi kataloogi, andmebaasitabeli tulpa, või töödeldakse läbi, eraldatakse vajalikud andmed ning algset faili ei pruugigi salvestada.

Siin näites mängime läbi kõige lihtsama mooduse - kasutajapoolse pildi üleslaadimise. Pilte hoitakse omaette kaustas nimega pildid. Pildi nimeks määratakse kasutaja id ning laiendiks png.

Faili üleslaadimiseks peab olema vormi saatmismeetodiks määratud "post". Samuti on nõutav lisaparameeter enctype="multipart/form-data". Muul juhul faili andmed lihtsalt ei lähe serveri poole teele. Faili valimiseks on input-sisestuselement tüübist file. Kujundus sõltub brauserist, kuid üldjuhul on seal nupp failidialoogi avamiseks ning tekstiväli, kus laetava faili aadressi näha saab. Turvakaalutlustel sinna andmeid ette panna pole võimalik, samuti ei kannata kujundust, nt. nupul olevat teksti muuta. Ikka selleks, et kasutaja enesele kogemata mõnda faili veebi rändama ei saadaks.

```
<form action="<?=$_SERVER['PHP_SELF'] ?>" method="post"
      enctype="multipart/form-data">
    Lae oma pilt: <br />
       <input type="file" name="pildifail" />
    <input type="submit" value="Lae" />
</form>
```

Üleslaetud failid jõuavad PHP nägemispiirkonda muutuja $_FILES kaudu. Siin näites oli sisestuselemendi nimi "pildifail", selle tõttu jõuavad ka andmed kohale muutuja $_FILES["pildifail"] kaudu. $_FILES["pildifail"]["tmp_name"] näitab üleslaetud faili ajutist asukohta. Sealt omale vajalikku kohta paigutamiseks sobib käsklus move_uploaded_file. Siin näiteks on kasutaja failiga seotud toimingud usaldatud eraldi klassi eksemplari kätte, sestap siis vormilt saabuvaid andmeid püüdva faili päises vaid muutuja olemasolu kontroll ning $kandmed-nimelise objekti kaudu käskluse väljakutse, mis peaks pildifaili sisu sobivasse kohta paigutama.

```
if(isSet($_FILES["pildifail"])){
  $kandmed->lisaPilt($_FILES["pildifail"]);
}
```

KasutajaAndmed ise eraldi klassina eraldi failis. Sisesteks muutujateks viide andmebaasiühendusele, kasutaja id (kid) ning pildikausta nimi. Konstruktoris eeldatakse, et klassi põhjal eksemplari loomisel antakse ette andmebaasiühenduse viide ning kasutaja id. Käsklus pildi_nimi liidab osadest kokku näidatava/salvestatava pildi suhtelise aadressi. Praegusel juhul eeldatakse laiendiks png-d, kuid muidu üldiselt on veebis viisakalt kasutatavad ja gif ja jpeg-vormingud. Mitme laiendiga hakkama saaval rakendusel oleks viisakas piltide laiendid salvestada ning vaatamise ajal nad siis sellistena ette anda. Siin on piirdutud aga lihtsama variandiga.

Pildi lisamise juures käsklus move_uploaded_file salvestab faili ajutisest asukohast püsivasse kohta ootama. Kataloogil, kuhu fail salvestatakse, peavad olema nii lugemise, kirjutamise kui kataloogi sisu vaatamise õigused - muul juhul kas ei saada andmeid lugeda, salvestada või ei leita faili kataloogist üles. See aga ühekordne seadistus, mille saab teha chmod - käsu või mõne kopeerimisprogrammi (nt WinSCP) abil (kausta omadused).

Faili kustutamiseks on PHPs käsklus unlink. Harjumuspärast "delete" või "remove"-kõlaga käsklust pole. Põhjuseks Unixi loogika, kus samale failile võidakse viidata mitmest kataloogist. Ning alles pärast seda, kui viimane viide on lahti lingitud, kustutatakse vastav fail ka tegelikult kettalt.

Kasutaja andmete klassi sai juurde tehtud ka käsklus näitamaks kõiki kaaslasi, kel me lehe omanikust kasutajaga vähemalt üks sarnane huviala.

```
<?php
  class KasutajaAndmed{
    private $ab;
    private $kid;
    private $pildikaust="pildid/";
    function __construct($yhendus, $kasutaja_id){
      $this->ab=$yhendus;
      $this->kid=$kasutaja_id;
    }
    function lisaPilt($pildiandmed){
       move_uploaded_file($pildiandmed["tmp_name"], $this->pildiNimi());
    }
    function kasKasutajalPilt(){
       return file_exists($this->pildiNimi());
    }
    function kustutaPilt(){
       unlink($this->pildiNimi());
    }
    function pildiNimi(){
       return $this->pildikaust.$this->kid.".png";
    }

    function kaaslasteAndmed(){
       $kask=$this->ab->prepare("
 SELECT knimi, epost, huviala
  FROM kasutajad_huvialad AS kh1, kasutajad_huvialad AS kh2,
       kasutajad, huvialad
  WHERE kh1.huviala_id=kh2.huviala_id
     AND kh1.kasutaja_id <> kh2.kasutaja_id
     AND kh2.kasutaja_id=kasutajad.id
     AND kh2.huviala_id=huvialad.id
     AND kh1.kasutaja_id=?");
       $kask->bind_param("i", $this->kid);
       $kask->bind_result($knimi, $epost, $huviala);
       $kask->execute();
       $hoidla=array();
       while($kask->fetch()){
          $k=new stdClass();
          $k->knimi=$knimi;
          $k->epost=$epost;
          $k->huviala=$huviala;
          array_push($hoidla, $k);
       }
       return $hoidla;
    }
}
?>
```

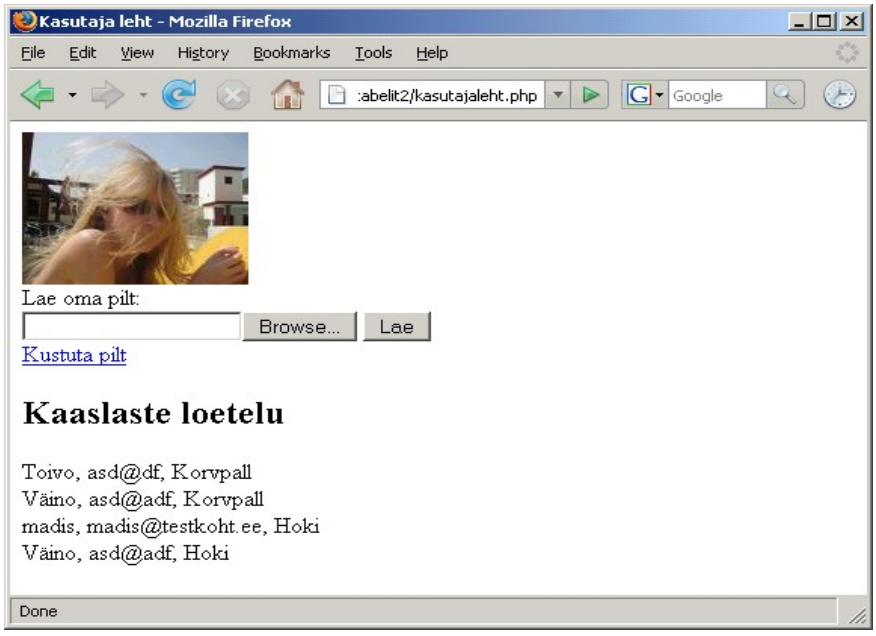
Kasutaja leht ise küllalt lihtne. Päises kontrollitakse, et oleks kindla kasutajanimega sisse logitud - võõraid pilte haldama ja kustutama ei lasta. KasutajaAndmete klassi põhjal luuakse andmebaasiühenduse ja praeguse kasutaja id-ga objekt, mille kaudu siis tema pildifaili hallata. Saabuv fail kirjutatakse piltide kataloogi - vajadusel kirjutatakse seal olemasolev sama kasutaja id-ga fail üle.

Kustutamiseks eraldi viide. Sest muidu tekib kergesti olukord, kus pilt küll lisatakse ja näidatakse, aga sellest lahti kuidagi kergel moel ei saa. Kui aga siin antakse aadressiribalt viitega kaasa, et pildikustutus=jah ning selle peale unlink-käsklus käivitatakse, siis saab üleslaetud pildist soovi korral lahti ka.

Pildi näitamise juures kontrollitakse, et kas piltide kaustas ikka kasutaja id-ga kattuvat pilti olemas on - ainult sel juhul lisatakse pildi välja meelitav koodilõik ka veebilehe teksti sisse.

```
<?php
  require("abifunktsioonid.php");
  if(!isSet($_SESSION["knimi"])){
    header("Location: meldimine.php");
    exit();
  }
  $kandmed=new KasutajaAndmed($yhendus, $_SESSION["kid"]);
  if(isSet($_FILES["pildifail"])){
    $kandmed->lisaPilt($_FILES["pildifail"]);
  }
  if(isSet($_REQUEST["pildikustutus"])){
    $kandmed->kustutaPilt();
  }
  $kaaslased=$kandmed->kaaslasteAndmed();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Kasutaja leht</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
     <?php if($kandmed->kasKasutajalPilt()): ?>
        <img src="pildid/<?=$_SESSION[kid]?>.png" />
     <?php endif ?>
     <form action="<?=$_SERVER['PHP_SELF'] ?>" method="post"
           enctype="multipart/form-data">
         Lae oma pilt: <br />
            <input type="file" name="pildifail" />
         <input type="submit" value="Lae" />
     </form>
     <a href="<?=$_SERVER['PHP_SELF']?>?pildikustutus=jah">Kustuta pilt</a><br />
     <h2>Kaaslaste loetelu</h2>
     <?php
        foreach($kaaslased as $k){
          echo "$k->knimi, $k->epost, $k->huviala<br />";
        }
     ?>
  </body>
</html>
```

Edasi polegi muud, kui oma üleslaetud pilti imetleda ning nimekirjast vaadata, milliste kaaslastega oleks põhjust/lootust lähemalt suhelda.



Ülesandeid

* Tee näide läbi
* Koosta pildigalerii. Failid laetakse üles eraldi kataloogi ning sealt näidatakse veebilehele.
* Iga pildi juurde üles laadides panna pealkirja. Vaatamislehel näidatakse pildid koos pealkirjadega.
* Salvestatakse pildi üleslaadimise ajal olnud laiend (gif, jpg/jpeg või png). Näitamise ajal on pildil õige laiend küljes, failinimeks on ikka piltide tabeli vastava pildi id-number.