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";
    }
    $otsisona=addslashes(stripslashes($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.

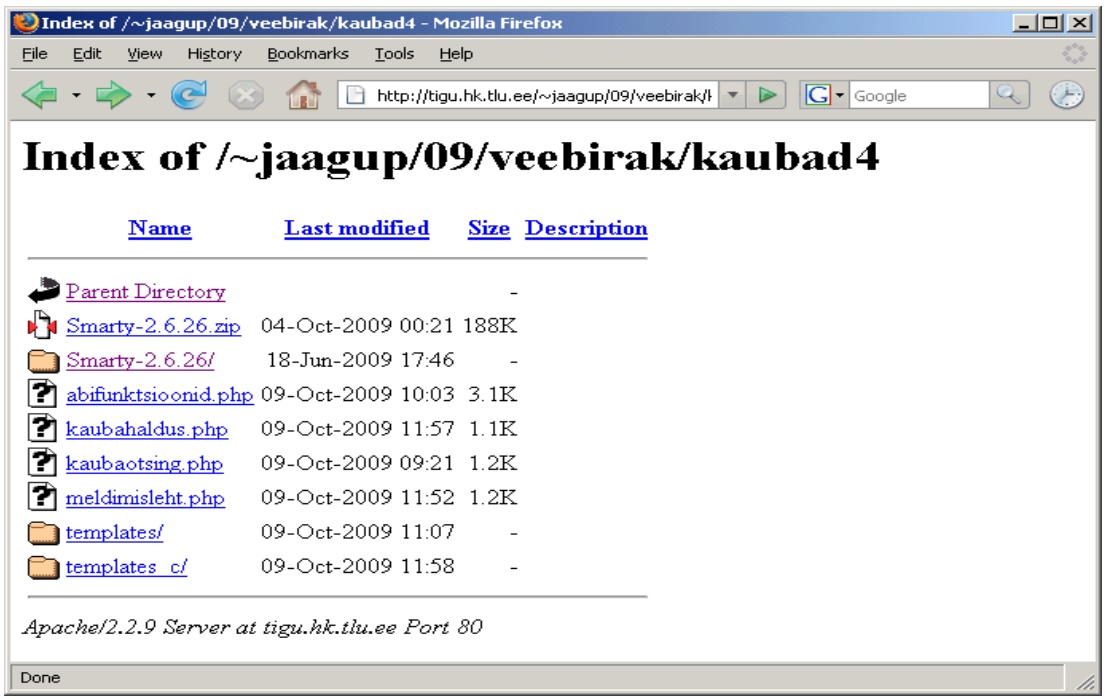
Mõnelgi programmeerijal on tavaks küllalt palju erineva sisu ja kujundusega lehti koondada ühe faili alla. Vahel paistab terve keerukas veebilehestik vaid index.php kaudu. Ka sellisel juhul on lehemallidest märatavalt kasu - eri kujundusega lõigud saab mugavasti eraldi failidesse paigutada.

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 paroolivälli 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();
  }
```

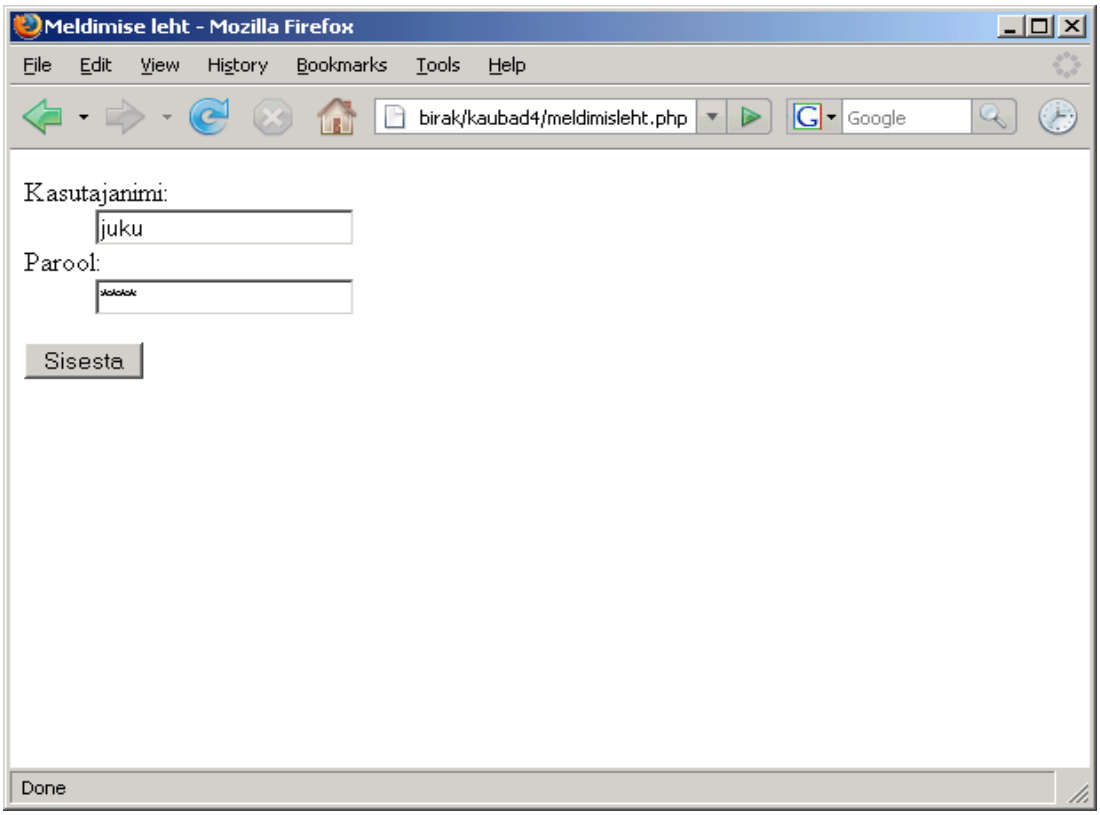
Käsk `session_start()` ikka, et sessioonimuutujaid kasutada saaks. Kui kasutajanime pole, siis saadetakse külastaja lihtsalt edasi meldimiseks mõeldud lehele ning lõpetatakse kaitstud lehe serveerimine. Muul juhul minnakse lehe näitamisega edasi.

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

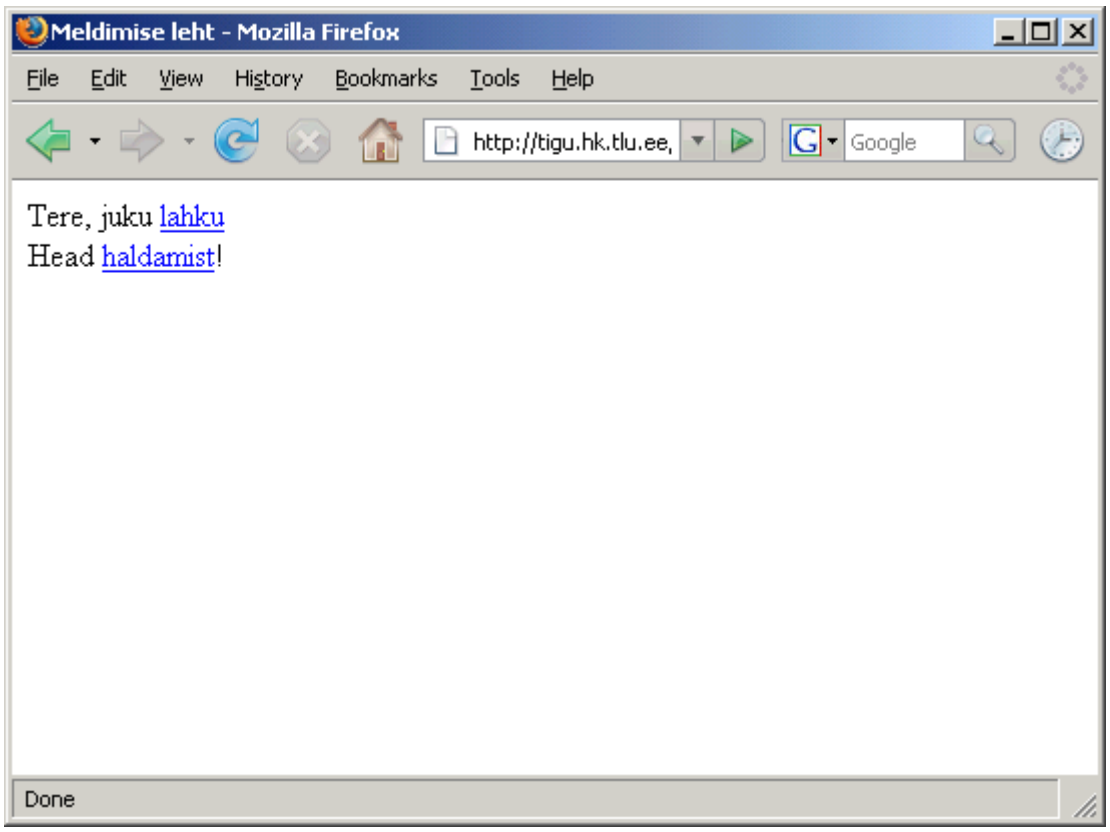
```
<?php
  session_start();
  if(isset($_SESSION["kasutajanimi"])){
     if(isset($_REQUEST["lahku"])){
        unset($_SESSION["kasutajanimi"]);
     }
  } else {
     if($_REQUEST["kasutajanimi"]=="juku" && $_REQUEST["parool"]=="kala"){
        $_SESSION["kasutajanimi"]=$_REQUEST["kasutajanimi"];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

?>
<!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>
```

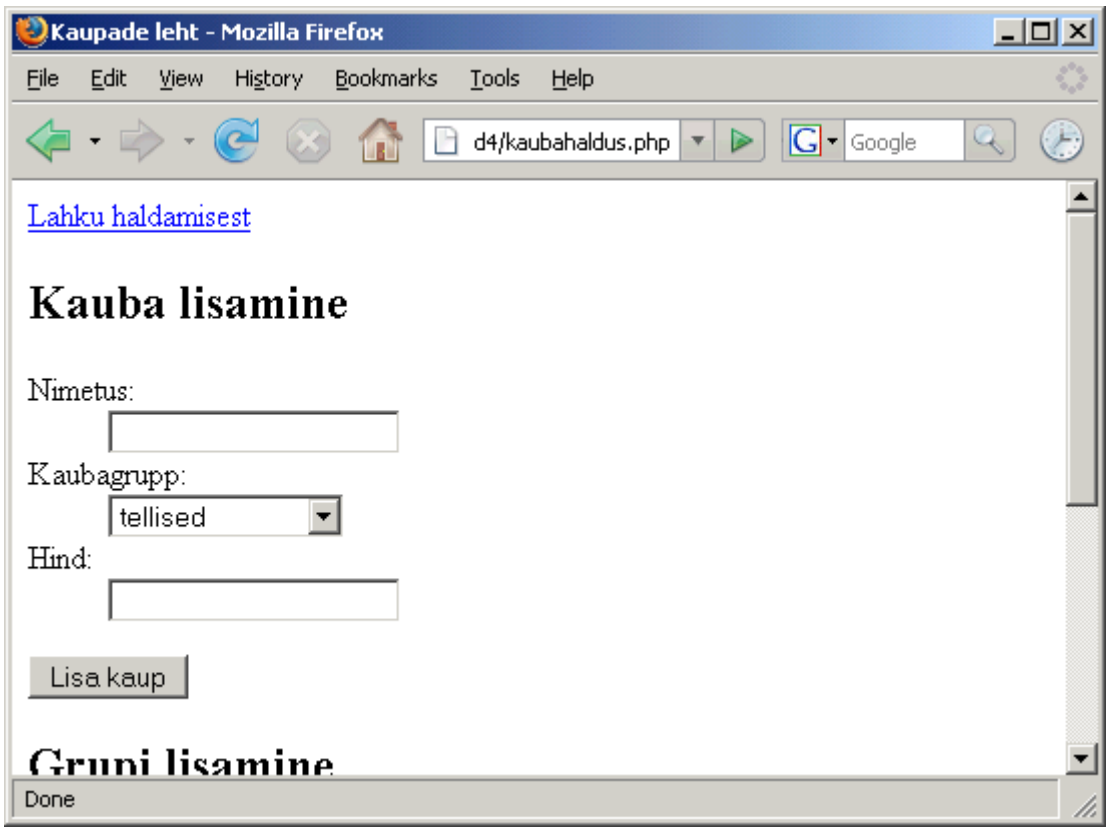
Meldimislehel siis kõigepealt uuritakse kasutajanime ja parooli.



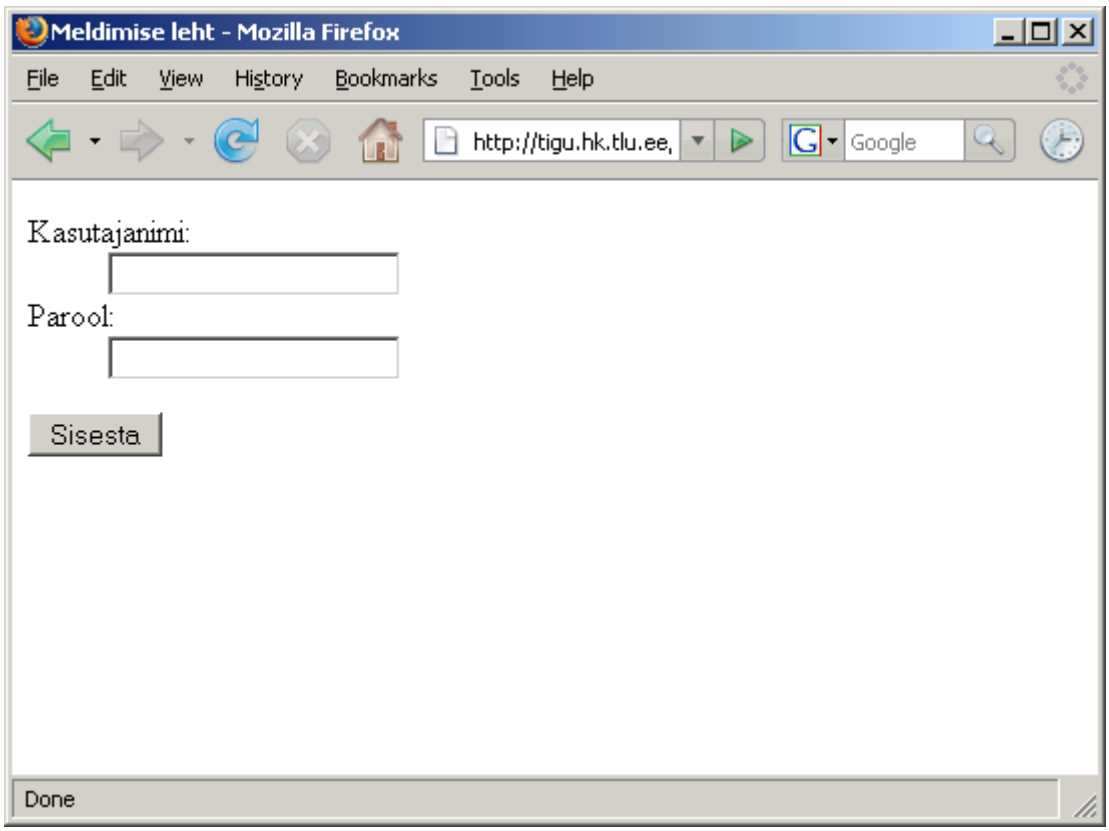
Kui need klapivad, siis pääseb registreeritud kasutajate tarbeks mõeldud viidete juurde.



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.
* Kaitse mõni omaloodud leht (nt. kahe arvu korrutamine) parooliga.
* 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.

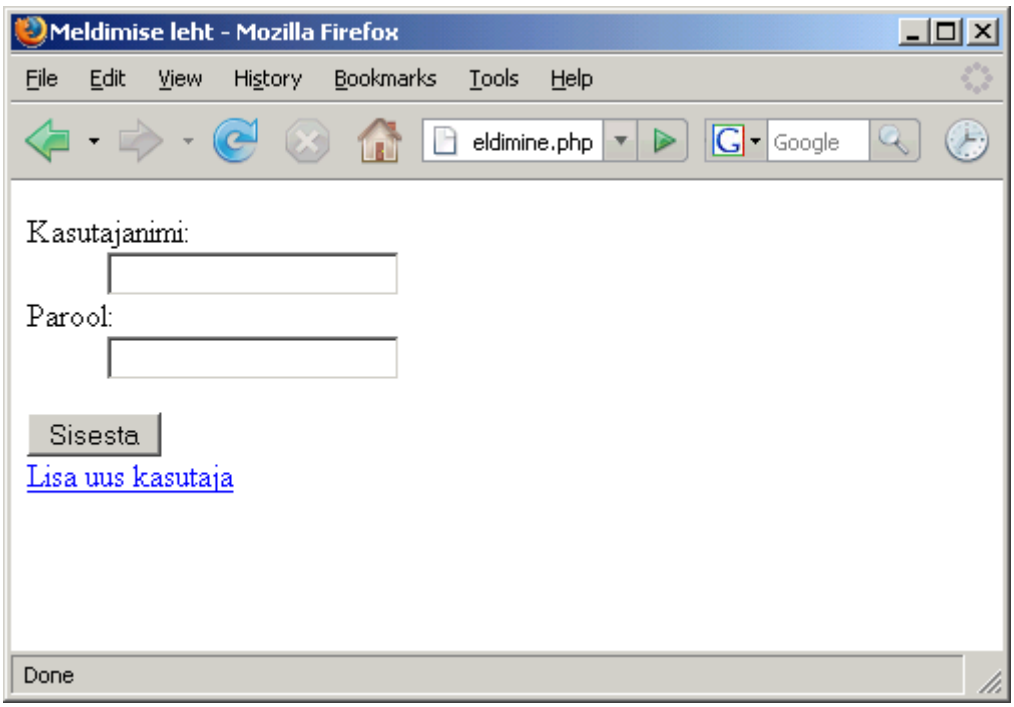
Juhul, kui lehele saabub parool, kuid mitte parool2 - sellisel juhul peab tegemist olema hariliku sisselogimisega. Kasutajahalduse käsklus kontrolliKasutaja teeb kindlaks, kas selliste tunnustega pääseb sisse. Õnnestumise korral jäetakse sessioonimuutujatesse meelde kasutajanimi ning kasutaja id. Viimane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

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

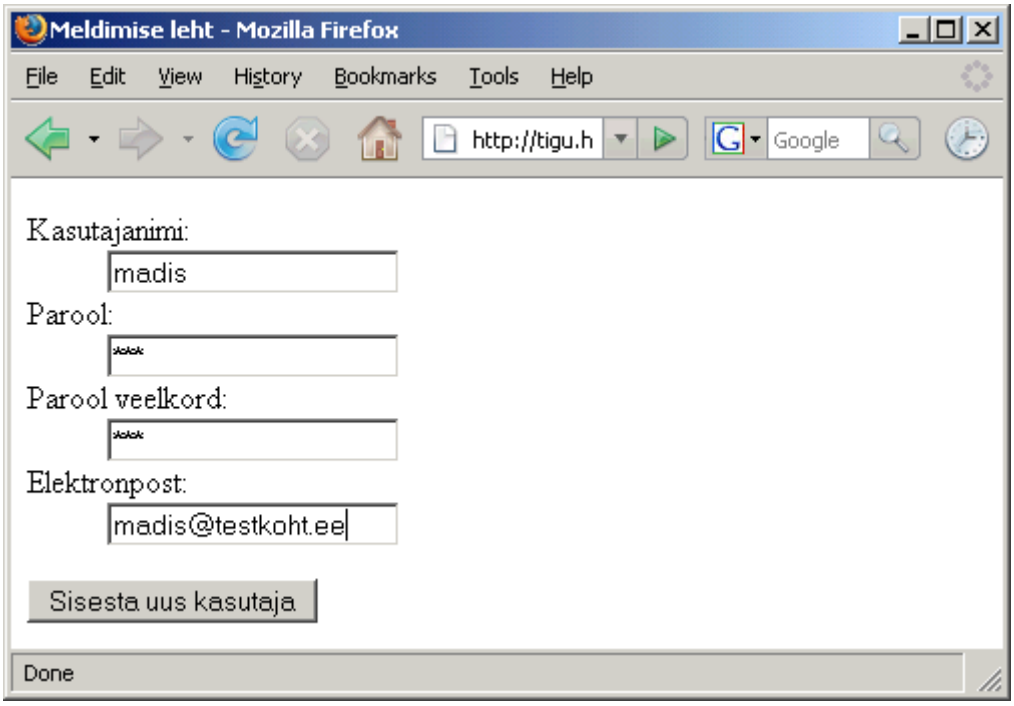
meldimine.php

```
<?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>
```

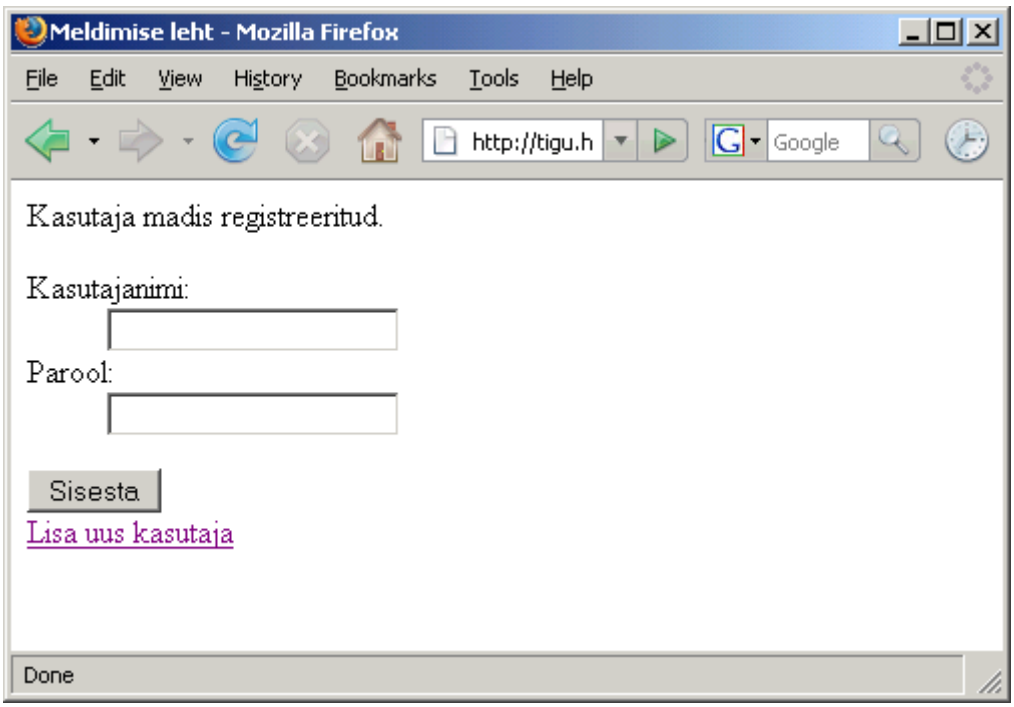
```
    <title>Meldimise leht</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
    <?php if(!isSet($_SESSION["knimi"]) AND !isSet($_REQUEST["lisauus"]) AND !
isSet($epost)): ?>
       <form action="meldimine.php" method="post">
          <?=$sisenemisteade ?>
          <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



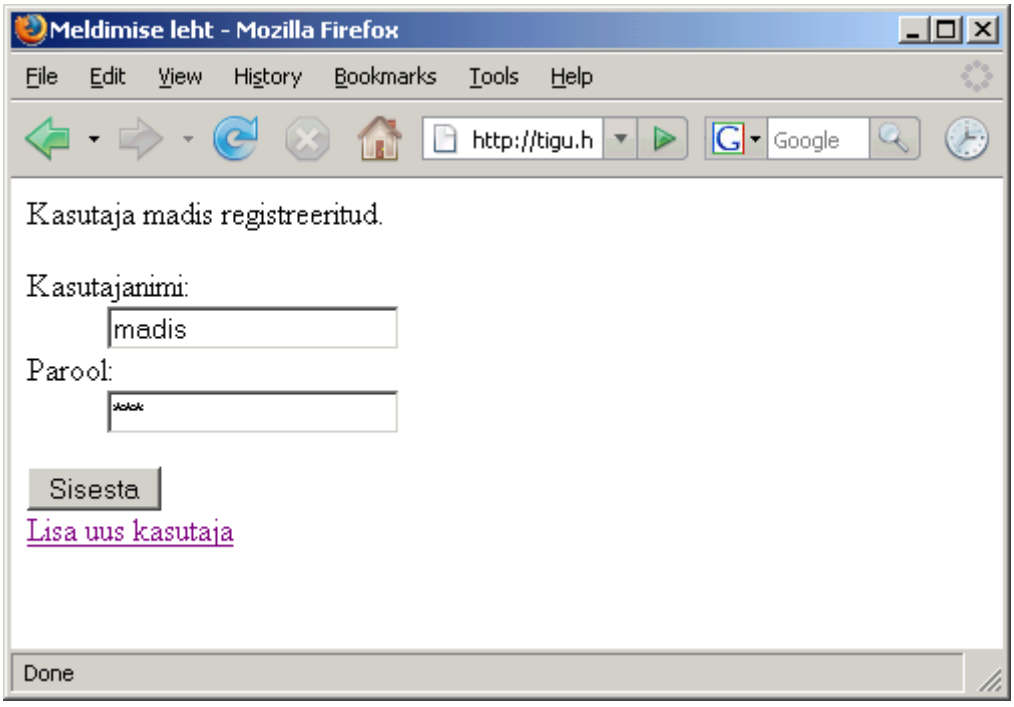
Kasutaja lisamine



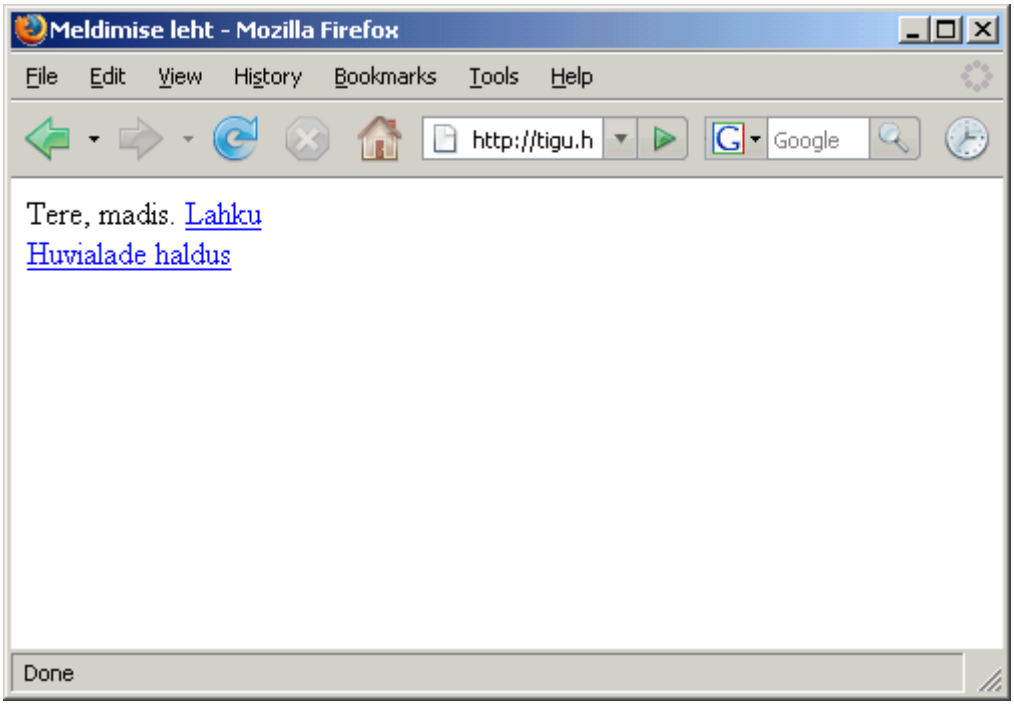
Lisamisteade

```
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.

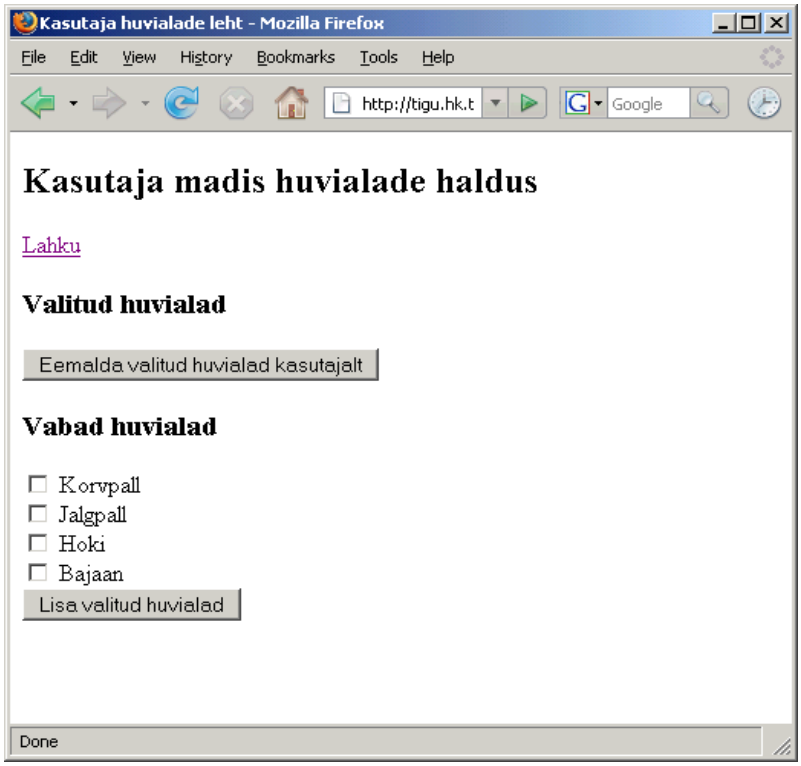
```
    function eemaldaHuvialad($seoste_idd){
       $kask=$this->ab->prepare("DELETE FROM kasutajad_huvialad WHERE id=?");
       foreach($seoste_idd as $seose_id){
          $kask->bind_param("i", $seose_id);
          $kask->execute();
       }
    }
```

Ning kasutaja huvialade haldamise leht tervikuna.

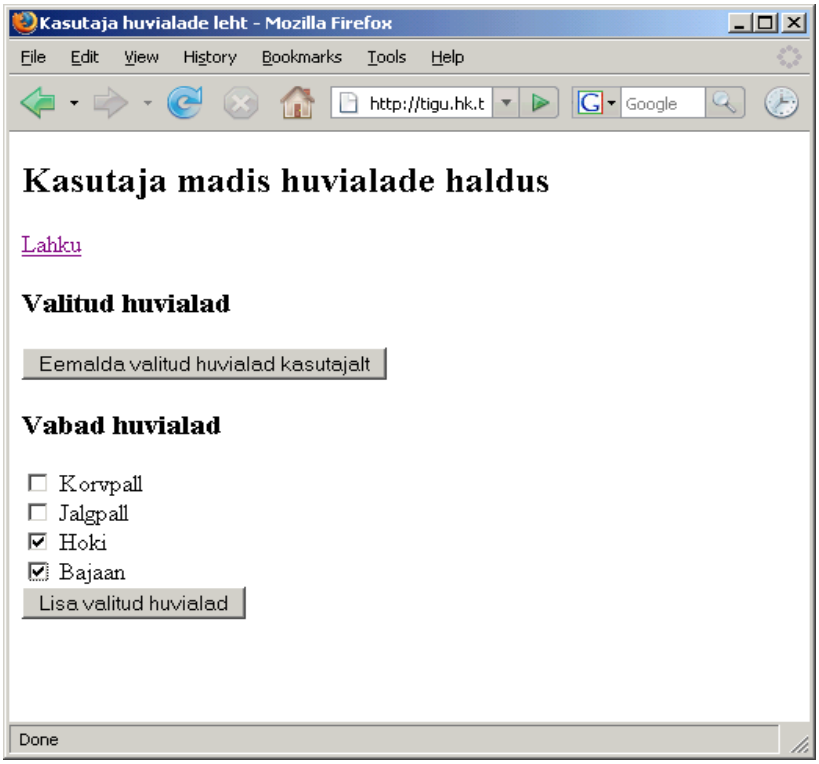
kasutajahuvialad.php

```
<?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 <?=$_SESSION["knimi"] ?> huvialade haldus</h2>
    <a href="meldimine.php?lahku=jah">Lahku</a>
    <h3>Valitud huvialad</h3>
    <form action="kasutajahuvialad.php" method="post">
      <?php foreach($valitud as $h): ?>
          <input type="checkbox" name="eemaldatav[]" value="<?=$h->seose_id ?>" />
               <?=$h->huviala ?><br />
      <?php endforeach ?>
      <input type="submit" name="eemaldamine" value="Eemalda valitud huvialad kasutajalt"
/>
    </form>
    <h3>Vabad huvialad</h3>
    <form action="kasutajahuvialad.php" method="post">
      <?php foreach($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>
```

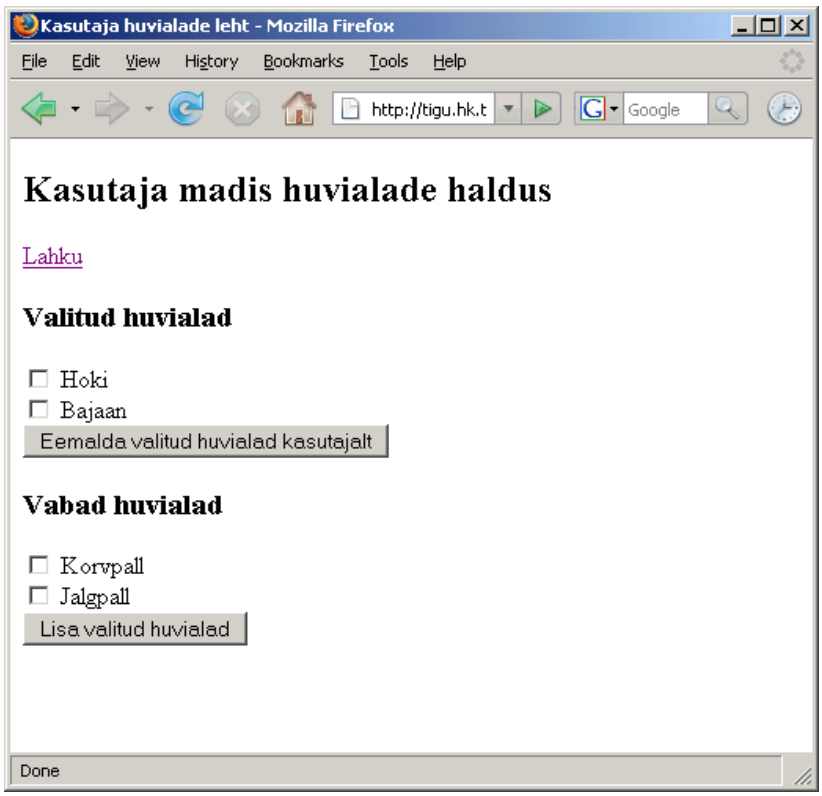
Lehe avamisel pole kasutajal veel ühtki huviala märgitud, kõik on vabad.



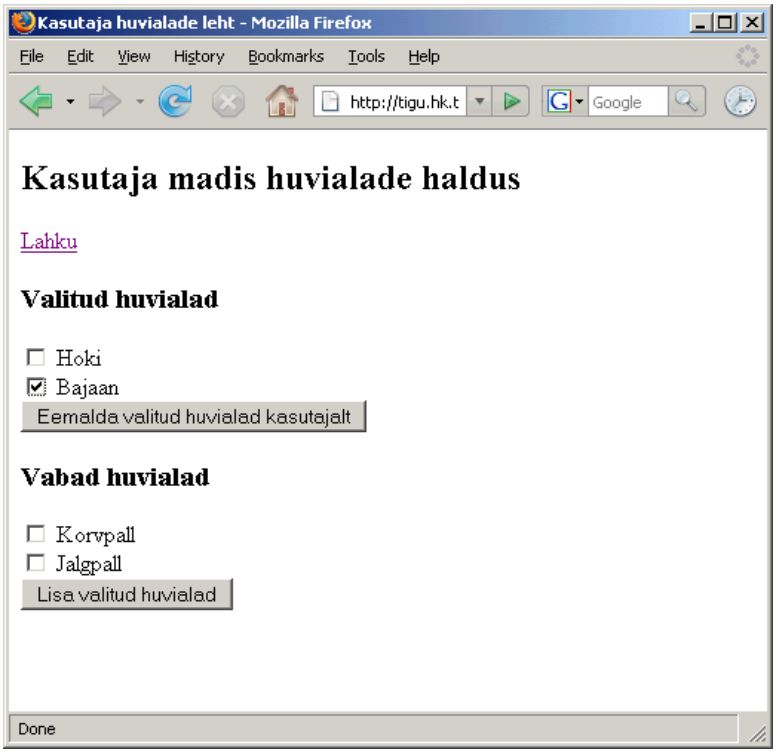
"Linnukestega" saab sobivad ära märkida.



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.

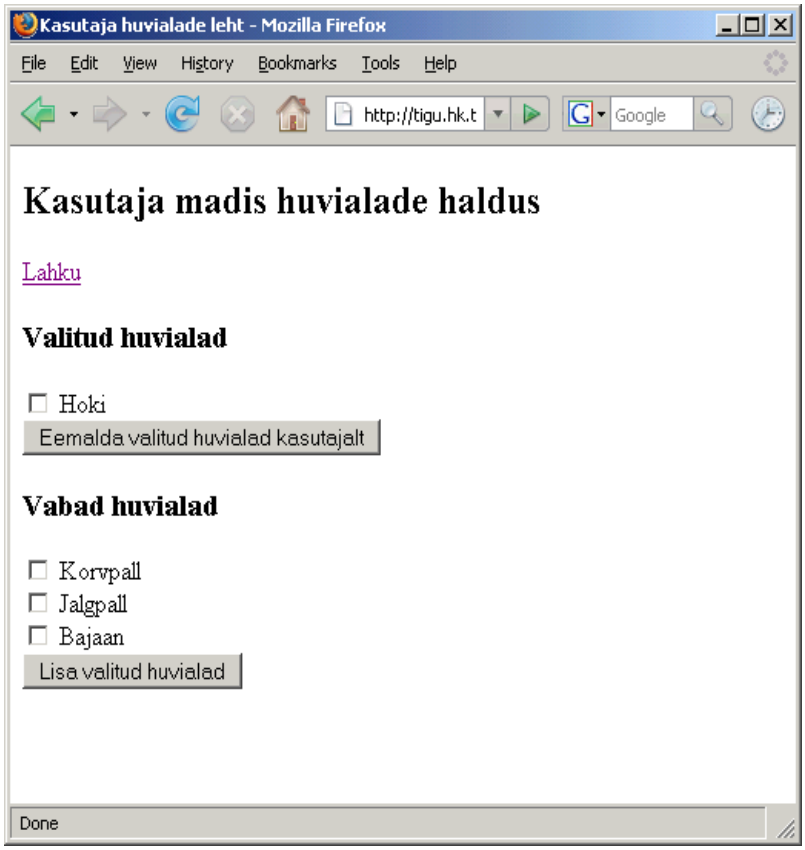


Andmebaasist kontrollides näeb, et kasutajale on jäänud vaid üks huviala, id-ga 3 ehk Hoki.

```
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 - (tunnen huvi, tegelen aktiivselt, tippsportlane). ”Ala tundmatu” tähendab, et kasutajal vastava huvialaga seos puudub.

* Koosta peatuste tabel (id, peatusenimi). Lisa mõned peatused
* Koosta liinide tabel (liini_nr, liininimetus). Lisa mõned liinid
* Koosta peatumisaegade tabel (id, liini_nr, peatuse_id, aeg_algpeatusest). Lisa mõned peatumised
* Koosta reiside tabel(id, liini_nr, algusaeg, suund) (1-edasi, 2-tagasi).
* Koosta leht ühe bussiliini peatumisandmete näitamiseks. Andmed sorditakse peatusesse jõudmise aja järgi.
* Kasutaja sisestab väljumisaja, lehel näidatakse bussi peatumise tegelikud kellaajad.

* 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.

* Looge tabel isikud(id, eesnimi, synniaasta, isa_id, ema_id)
* Lisage andmed. Kui vanema isikut pole tabelis, siis selle koha peal tühiväärtus NULL
* Väljastage isikute eesnimed
* Väljastage isikute ees- ja isanimed kui võimalik
* Väljastage inimeste ees-, isa- ja emanimed kui võimalik.
* Väljastage kõikide eesnimed, isanimed nendel kel teada (LEFT JOIN)
* Väljastage, kui vana olid isa ja ema siis kui trükitav isik sündis.
* Väljastage inimesed, kelle isa on vanem kui ema

* Loo/otsi eelmise ploki ülesandes olnud tabelid peatuste, bussiliinide, peatumiste ja reiside kohta.
* Näita bussiliini juures, millistele liinidele ümberistumised sellelt liinilt on võimalikud (liinid kasutavad sama peatust)
* Näita iga ümberistutava liini juures ümberistumiseks kõlbuliku peatuse id.
* Näita iga ümberistutava liini juures ümberistumiseks kõlbuliku peatuse nimi ja liini nimi.

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 * FROM 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.

* Loo/otsi tabel isikud(id, eesnimi, synniaasta, isa_id, ema_id)
* Leia tabelis suurim sünniaasta
* Leia, mitu inimest on sündinud enne aastat 2000
* Väljasta inimeste nimed sündimise aastakümnete kaupa.

* Leia iga inimese laste arv
* Leia iga inimese vanima lapse sünniaasta.

* Loo/otsi peatuste, bussiliinide, peatumiste ja reiside tabelid.
* Näita etteantud reisi peatused (peatuse_id) ja peatumise ajad (edasisuunas)
* Näita etteantud peatuse kohta sealt väljuvate busside liinid ja ajad.
* Näita ümberistumiseks sobivad liinid ja väljumisajad nendes peatustes.

* Ümberistumist loetakse võimalikuks, kui ümberistutava liini peatumiskellaaeg on hilisem kui peatusesse saabumiseks kasutatava liini peatumisaeg.

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.

Lihtsaim näide pildi üleslaadimise kohta. Kogu toimetus on jäetud samale veebilehele. Piltide jaoks on tehtud php-faili suhtes alamkaust nimega pildid ning antud sellele veebiserveri õiguses kasutaja jaoks kõik õigused.

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.

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.

Järgnevas näites kirjutatakse salvestava faili nime ette: kaustas pildid fail nimega pilt1.png. Kasutajapoolne veebikaustas asuva faili nime määramine oleks ohtlik: kui näiteks ta laeks üles php-laiendiga faili ja saaks selle veebi kaudu kirja panna, siis pääseks veebi kaudu kergesti masinasse igasuguseid segadusi tekitama. Siin aga nimi ja koht kindel ning vastavaid muresid pole.

```
<?php
  if(isSet($_FILES["pildifail"])){
    move_uploaded_file($_FILES["pildifail"]["tmp_name"],
      "pildid/pilt1.png");
  }
?>
<!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>Faili laadimine</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
     <h1>Pildifaili laadimine</h1>
     <form action='<?=$_SERVER["PHP_SELF"] ?>'
          method="post" enctype="multipart/form-data">
        <input type="file" name="pildifail" />
        <input type="submit" value="OK" />
     </form>
     <?php
       if(file_exists("pildid/pilt1.png")){
         echo "<img src='pildid/pilt1.png' alt='' />";
       }
     ?>
  </body>
</html>
```

Kasutajate ja huvialade rakendust täiendavas näites 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.

kasutajaandmed.class.php

```
<?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;
    }
}
?>
```

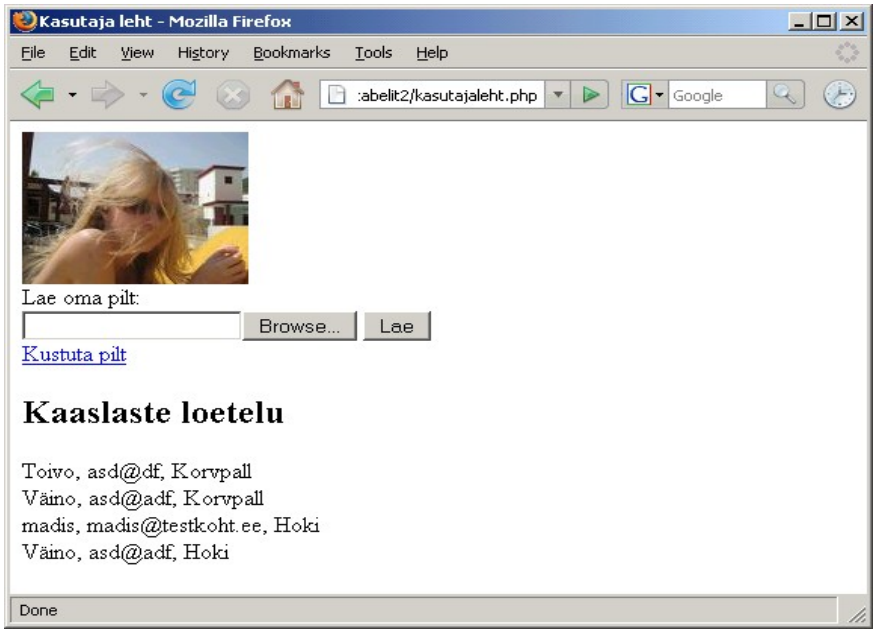
Kasutaja leht ise küllalt lihtne. Päises kontrollitakse, et oleks kindla kasutajanimega sisse logitud - võõraid pilte haldama ja kustutama ei lasta. KasutajaAndmete klassi põhjal luuakse andmebaasiühenduse ja praeguse kasutaja id-ga objekt, mille kaudu siis tema pildifaili hallata. Saabuv fail kirjutatakse 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 näited läbi.
* Võimalda lehelt üles laadida ja vaadata kahte eraldi pilti.
* Jäta üleslaadimisel faili nimi samaks. Luba ainult piiratud hulk laiendeid (txt, doc, rtf).

* Koosta pildigalerii. Failid laetakse üles eraldi kataloogi ning sealt näidatakse veebilehele. Failid nummerdatakse 1.png, 2.png vastavalt kirje id-le andmebaasitabelis. Piltide juurde kuulub pealkiri.
* 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.
* Pilte näidatakse lehel viie kaupa, nuppudega viited edasi ja tagasi.
* Pildid saab paigutada eri albumitesse.
* Piltide küljes on märksõnad, mille järgi neid otsida saab.