

Database-baseret Web-publicering, efterår 2001

Forelæsning 10, tirsdag den 6. november 2001.

Check af form-variable og Online Communities

- Online communities
- Online community features
- ACS - Arsdigita Community System
- Modul 1: Bruger database
- Modul 2: Information om indhold
- Modul 3: Information om hvad en bruger har læst
- Modul 4: Værdien af en bruger
- Modul 5: Hvilke links har brugere fulgt for at komme til sitet
- Modul 6: Banner ads
- Modul 7: Håndtering af tilknytninger mellem administrator og brugere
- OpenACS - ACS baseret på PostgreSQL
- Checking Formvariables using UPDATE
- Introduktion til Øvelse 9

Online community features

Et online community skal:

1. indeholde data om brugere
 2. indeholde information om statske sider knyttet til foreningen - Hvem bidrog med siden? Er siden tilknyttet andre sider?
 3. holde styr på hvilke brugere har set hvilke sider
 4. holde styr på hvilke brugere som koster foreningen penge og tid
 5. holde styr på hvorledes brugere kommer til foreningen og hvilke eksterne links brugere følger
 6. holde styr på banner-ads - og brugerens brug af dem
 7. hjælpe administrator med at kontakte grupper af brugere
- Eksempler på brug:
- Joe med spørgsmål - undersøg om Joe har læst sider og diskussionsforum-artikler om emnet (kræver modul 1 og 3)
 - Eksperten - undersøg om der er nogle huller i de statske sider der trænger til at blive udfyldt (kræver modul 2, 3 og 4)
 - Brugtkøb - en køber skal kunne se sælgers historie i foreningen (kræver modul 1)

Online communities

Online communities (foreninger) har mange fordele:

- Vi kan lære uden at være tilknyttet et universitet
- Vi kan undervise uden at undervise fuldtid
- Vi kan lære uden at sige vores job op
- Vi kan lære uden at forlade vores hjem
- Vi kan samarbejde uden at være i samme bygning
- Vi kan bidrage til projekter uden at arbejde fuldtid
- Vi kan møde ligesindede

ACS - Arsdigita Community System

En samling moduler til opbygning af web-sites med mulighed for bruger-interaktion.

Syv moduler:

1. Bruger database
2. Information om indhold
3. Information om hvad en bruger har læst
4. Værdien af en bruger
5. Hvilke links har brugere fulgt for at komme til sitet
6. Banner ads
7. Håndtering af tilknytninger mellem administrator og brugere

Modul 1: Bruger database

Hjertet i ACS:

```
create table users (
  user_id
  first_names
  last_name
  priv_name
  email
  priv_email
  email_bouncing_p
  password
  url
  on_vacation_until
  last_visit
  second_to_last_visit
  registration_date
)
integer not null primary key,
varchar(100) not null,
varchar(100) not null,
integer default 0,
varchar(100) not null unique,
integer default 5,
char(1) default 'F',
check(email_bouncing_p in ('T','F')),
varchar(30) not null,
varchar(200),
date,
date,
date,
date,
date
```

Modul 2: Information om indhold

Tabeller til at håndtere:

- statiske sider
- brugerindstavede kommentarer
- brugerindstavede relaterede links
- classified ads :
camera dealer: <http://www.photo.net/wtr/thebook/screenshots/>
community-member: <http://www.photo.net/wtr/thebook/screenshots/fotograf.html>
community-member: [community-member.reader.html](http://www.photo.net/wtr/thebook/screenshots/community-member.reader.html)
- chat
- diskussionsgrupper

Nyt og gammelt indhold

Kan vi adskille

”Nyt indhold siden sidste login”

fra

”Indhold du allerede har set”

Når bruger logger på, så sættes

```
last_visit = sysdate.
```

```
second_to_last_visit = last_visit
```

hvis `last_visit` enten er mere end en dag gammel eller slet ikke findes.

Når vi skal præsentere nyheder, så præsenterer vi alt som er nyere end `second_to_last_visit`.

Modul 3: Information om hvad en bruger har læst

Brugbart når en bruger stiller spørgsmål:

- har brugeren læst det relevante materiale

Modul 4: Værdien af en bruger

- hvor tit skal en administrator slette en brugers kommentar eller spørgsmål?
- giver bruger gode svar på spørgsmål stiller at andre brugere?
- emails som ikke virker?

Eksempler på, hvor betaling kan være på tale:

- download af statiske sider som ikke er gratis.
- brugere up-loader annoncer
- administration, f.eks. at skal slette / rette en annonce
- en bruger rejser et godt spørgsmål
- en bruger kommer med et godt svar
- en bruger kommer med en kommentar
- en administrator skal slette en kommentar

Modul 5: Hvilke links har brugere fulgt for at komme til sitet

– og hvilke lokale sider brugere sendt til fra de eksterne sites!?

```
create table referer_log (  
  -- relative to the PageRoot, includes the leading /  
  local_url varchar(250) not null,  
  -- full URL on the foreign server (including http://)  
  foreign_url varchar(250) not null,  
  -- we count referrals per day  
  entry_date date,  
  click_count integer default 0,  
  primary key ( local_url, foreign_url, entry_date)  
);
```

Eksempel:

```
local_url /photo/  
foreign_url http://www.yahoo.com/Arts/....  
entry_date 1998-06-12  
click_count 24
```

Modul 7: Håndtering af tilknytninger mellem administrator og brugere

Følgende kriterier skal kunne bruges til at kontakte grupper af brugere:

- aktivitet i diskussionsgrupper
- har læst en special statisk side
- har kommenteret en special statisk side
- har skrevet en statisk side
- medlemsid af foreningen
- har udtrykt interesse for et emne

OpenACS - ACS baseret på PostgreSQL

- OpenACS (<http://www.openacs.org>) er baseret på Open Source software:
AOLserver: <http://www.aolserver.com>
PostgreSQL: <http://www.postgresql.org>
- OpenACS kører under Linux
- Site baseret på OpenACS, LinuxLab: <http://linuxlab.dk>

Modul 6: Banner ads

Her er datamodellen for håndtering af banner ads:

```
create table advs (  
  adv_key          varchar(200) primary key,  
  -- picture  
  adv_filename    varchar(200),  
  target_url      varchar(500)  
);  
  
create table adv_log (  
  adv_key          varchar(200) not null references advs,  
  entry_date      date,  
  display_count   integer default 0,  
  click_count     integer default 0,  
  unique(adv_key,entry_date)  
);
```

Checking Form Variables - The Easy Way

We have already seen how to check form variables:

- We use `info exists` to see if a form variable has been defined.
 - If a form variable exists, then we normally use a regular expression to test the contents of the variable.
- Until now, we have copied the code to test form variables into the scripts.

Using the TCL command `upvar` we can build a library of procedures checking both the existence and contents of form variables.

The library is a collection of procedures, each procedure testing one type of form variables, for instance integers or emails.

Given the library, lets call it `Formvar.tcl`, we can have the scripts call one procedure in the library for each form variable to test.

Checking Form Variables - The Easy Way

If the script contains 10 form variables, then the script will contain only 10 procedure calls in order to test the form variables. Without the library, it would take maybe 50 lines of code.

On top of that, we can have the library automatically return an error page, in case a form variable fails a test.

To build the library we must

1. learn about `upvar`
2. build a general error page for our service, file `Formvar.tcl`
3. build procedures testing for various types of form values, file `Formvar.tcl`
4. call the procedures from the scripts

upvar (upvar.tcl)

Consider the procedure

```
proc add4 { var } {  
  upvar $var v  
  set v [expr $v+4]  
}
```

and the program

```
set x 32  
puts "x before add4 $x\n"  
add4 x  
puts "x after add4 $x\n"  
  
set y 2  
puts "y before add4 $y\n"  
add4 y  
puts "y after add4 $y\n"
```

The procedure `add4` accesses the variable with name `$var` in the enclosing procedure call. In `add4` that variable is named `v`

In the first call, the variable `x` is accessed in `add4` as `v`.

In the second call, the variable `y` is accessed in `add4` as `v`.

This works, even if `add4` is located in a separate file!

A Standard Error Page

We write the library procedures in file `Formvars.tcl`

You must define a standard error page, such that all errors are reported consistently.

A simple error page may use the well known back key.

Given an error message `error_msg` we may use the following outline:

We had a problem processing your entry:

- `error_msg`

Please back up using your browser, correct it, and resubmit your entry.

Thank you

```
proc return_page { title body } {  
  ns_return 200 text/html "  
  <html>  
  <title> $title </title>  
  <body>  
  $body  
  </body>  
  </html>"  
}
```

Code for a very simple error page

```
proc return_error { err_msg } {  
  set title "Problem with Your Input"  
  set body "We had a problem processing your entry:  
  <ul>\n"  
  append body "<li>$err_msg\n"  
  append body "</ul>  
  Please back up using your browser, correct it, and  
  resubmit your entry.  
<p>  
  Thank you.  
  return_page $title $body  
  exit  
}
```

Checking Email, file formvars.tcl

Because we are only writing a test procedure for each form type once, we can actually put some effort into reporting good error messages.

A procedure chk_email for checking an email:

```
proc chk_email {form_var {error_msg ""}} {
    upvar $form_var email
    if {[info exists email]} {
        !regexp "\[^\t ]+@[^\.\t]+\.[^\.\n ]+$" $email}} {
        if {[empty_string $error_msg]} {
            if {[info exists email]} {
                set error_msg "You must enter an <b>email</b> address"
            } else {
                set error_msg "You must enter a valid <b>email</b> address -
                    '$email' is not one!"
            }
        }
        append error_msg "<blockquote> Example email addresses are
        <ul>
        <li>user@supernet.com
        <li>FirstLastname@very.big.compagny.com
        </ul></blockquote>"
        return_error $error_msg
    }
    return
}
```

Checking sequences, file formvars.tcl

Hidden form variables such as sequences should never fail, unless the user is hacking the system.

```
proc return_panic_error { err_msg } {
    set title "Problem with this service"
    set body "Sorry, but it seems that something is wrong with the
    supplied values. The error is logged,
    and the system administrator has been notified.
    <p>
    Please try again later."
    return_page $title $body
    # Log error, and notify the system administrator
    exit
}
```

```
# If form_var is non-existing 0 or negative then it is mal formed.
proc chk_seq_id {form_var {err_msg ""}} {
    upvar $form_var seq_id
    if {[info exists seq_id]} {
        if {[empty_string $err_msg]} {
            set err_msg "Sequence id $form_var is not valid"
        }
        return_panic_error $err_msg
    }
    return
}
```

Check Email

In chk_email, the variable email accesses the form variable \$form_var in the enclosing procedure call, that is, the script.

In the script, where we check the form variable email, we write

```
source "/web/nh/www/formvars/formvars.tcl"
set_form_variables 0
```

```
chk_email email
```

#Now we can safely use the form variable email

Try http://hug.it.edu:8077/formvars/chk_formvars.tcl

Checking a range [a...b], file formvars.tcl

A procedure checking that two form variables a and b forms a range [a...b], where a < b

```
proc chk_range {form_var1 form_var2} {
    error_msg "You must enter a <b>range</b> \[a,...,b\], where a < b" } {
    upvar $form_var1 first
    upvar $form_var2 last
    if {[info exists first]} {
        !regexp {^(0|([1-9]([0-9]*)))$} $first} {
            !info exists last} {
                !regexp {^(0|([1-9]([0-9]*)))$} $last} {
                    ($last - $first < 0)} {
                        return_error $error_msg
                    }
                return
            }
        }
    }
}
```

In the script we write:

```
source "/web/nh/www/formvars/formvars.tcl"
set_form_variables 0
```

```
chk_range a b
```

#Now, I can safely use a and b

where the two form variables are named a and b.

Checking Enumerations, file Formvars.tcl

Say the user must enter / select one item from a list of items.

Here are some examples of enumerations:

1. Yes, No
2. Jan, Feb, Mar, ..., Dec
3. Very Good, Good, Bad, Very Bad
4. 1,2,3,4,5
5. New, Read, Done

In the script we write:

```
source "/web/nh/www/formvars/formvars.tcl"
set_form_variables 0

chk_enum enum1 [list "Yes" "No"]
chk_enum enum2 [list "Very Good" "Good" "Bad" "Very Bad"]

# Now, we can safely use the two enumerations enum1 and enum2.
```

Checking Enumerations, file Formvars.tcl

```
proc chk_enum {form_var enums {error_msg ""}} {
    upvar $form_var enum
    if [lempty_string $error_msg] {
        if {[info exists enum]} {
            append error_msg "You entered form variable $form_var
                with value $enum.<p>\n"
        } else {
            append error_msg "Form variable $form_var is missing.<p>\n"
        }
        append error_msg "You must enter one of the following values:
            <blockquote>\n"
            append error_msg [join $enums ", "]
            append error_msg "\n</blockquote>\n"
        if {[info exists enum]} {
            return_error $error_msg
        } else {
            foreach e $enums {
                if {[string compare $enum $e] == 0} { return }
            }
            # we did not find a match
            return_error $error_msg
        }
    }
    return
}
```

Site-design ved anvendelse af procedurer

Et site-design kan holdes konsistent ved at benytte en procedure:

```
proc return_page { title body } {
    ns_return 200 text/html "<html>
    <title>$title</title><body bgcolor=white>
    $body
    <hr>
    <address>
    <a href=\"mailto:nh@it.edu\">nh@it.edu</a>
    </address>
    </body>
    </html>"
}
```

```
proc return_page_with_title { title body } {
    return_page $title "<h2>$title</h2> $body"
}
```

Proceduren return_page_with_title kan nu bruges i alle Tcl-programmer der genererer html-sider:

```
# source the utility procedures
source "/web/mael/www/rating/utfl.tcl"

return_page_with_title "About" $text
```

Introduktion til Øvelse 9

Konstruktion af et klassificeringssystem:

<http://www.it.edu/courses/W2/F2001/Lb/1b9.html>

- select-funktionaliteten group by
- strukturering - tilstandsdiagram for klassificeringssystemet
- procedure til generering af stjerner.

Eksempel: Statistik af logins

```
create table login (
    email varchar(100),
    clicks integer
);

insert into login (email, clicks) values ('nh@it.edu', 5);
insert into login (email, clicks) values ('kenneth@it.edu', 3);
insert into login (email, clicks) values ('nh@it.edu', 9);
insert into login (email, clicks) values ('kenneth@it.edu', 8);
insert into login (email, clicks) values ('kenneth@it.edu', 3);
insert into login (email, clicks) values ('nh@it.edu', 4);
insert into login (email, clicks) values ('kenneth@it.edu', 17);
```

select-funktionaliteten group by

```
select email, count(*) from login group by email;
select email, sum(clicks) from login group by email;
select email, avg(clicks) from login group by email;
select email, max(clicks) from login group by email;
I øvelsen bruges group by til beregning af den gennemsnitlige rating af restauranterne
```

Tiden brugt i opgave C - Webstrukturen - tjener sig hurtigt ind.

Planlæg hvilke form-variabler der skal overføres fra en tilstand til en anden.

Procedure til generering af stjerner

- Til at generere stjerner benytter vi følgende *rekursive* procedure:

```
proc genstars { n } {
  if { $n == 0 } {
    return ""
  } else {
    return "*"genstars [expr $n - 1]]"
  }
}
```
- Når proceduren kaldes med tallet 0 returneres den tomme streng ("")
- Når proceduren kaldes med et andet tal end 0, f.eks. 4, returneres en streng bestående af en stjerne (*) sammensat med resultatet af at kalde proceduren med tallet 3, som er lig 4-1

Datautrækning med database_to_tcl_string db sql – en celle i en række

- udrækning af en celle i en tabel
- fejler hvis der returneres andet end en celle

Eksempel:

```
...
set query "select name
            from mailing_list
            where email = 'mh@it.edu'"
set name [database_to_tcl_string $db $query]
...
```

Kommandoen database_to_tcl_string fejler, hvis der ikke findes præcis en række med en celle.

Derfor anvender vi catch:

```
if { [catch {set name [database_to_tcl_string $db $query]} errmsg] } {
  fejl_kode
} else {
  variabel name indeholder cellen fra databasen
}
```

Sekvenser

Med en sekvens i Oracle (*sequence*) kan vi generere en række af unikke løbenumre (tal).

```
SQL> create sequence id_seq start with 5;
Sequence created.
```

```
SQL> select id_seq.nextval from dual;
NEXTVAL
-----
5
```

```
SQL> select id_seq.nextval from dual;
NEXTVAL
-----
6
```

```
SQL> select id_seq.currval from dual;
CURRVAL
-----
6
```

```
SQL> drop sequence id_seq;
Sequence dropped.
```

Tabellen dual er en tom tabel som vi f.eks. kan anvende i vores SQL-kommando når syntaksen dikterer, at vi skal angive en tabel uden egenligt at have brug for tabellen.