

Database-baseret Web-publicering, efterår 2001

Forelæsning 8, tirsdag den 23. oktober 2001.

Databaseransktioner via web-forms

- Konstruktion af sites som er databaser
- Datamanipulation via AOLserverens tcl-API
- Eksempel: mailing-list
- ns_db gethandle
- ns_db dml
- set_the_usual_form_variables
- catch
- ns_ora resultrows
- database-to-tcl-string
- ns_db select
- ns_db getrow
- ns_returnredirect

Konstruktion af sites som er databaser

Step 1: Konstruktion af datamodel

- hvilken information skal gemmes og hvordan skal den repræsenteres
- dette er den svære del!

Step 2: Udvikling af legale data-transaktioner

- hvordan indsættes data i databasen
- hvordan udtrækkes data fra databasen

Step 3: Konstruktion af web-forms og site-map

- brugergrænsefladen er html-kode (forms)

Step 4: Konstruktion af database-transaktioner via tcl-scripts

- SQL (Structured Query Language) bruges til de egentlige database-transaktioner

Projekter

Overvej 4-ugers projekter!

Eksempler på nye og gamle projekter:

- Kursservice - brugere får tilsendt email ved kursusudring
- SpecWeb
- HTML Widgets, design, konstruktion og brugerundersøgelse.
- Et lille journalsystem
- Et lille billedarkiv
- ...

Jeg har lavet en projektside (på dansk)

<http://www.itu.dk/courses/W2/E2001/projekter.html>

Eksempel: mailing-list

Vi skal kunne vedligeholde en liste af navne og emails.

Den samme fælles liste vedligeholdes af alle der anvender systemet.

Den eneste information som gemmes er navne og emails.

Man skal kunne oprette, slette og opdatere email-information.

Step 1: Datamodelen (mailing_list.sql)

```
create table mailing_list (  
  email      varchar(100) primary key,  
  name       varchar(100) not null  
);
```

Vi antager, at der ikke er to personer som anvender den samme email, dvs email er primærnøgle.

Vi påtvinger navneoplysninger.

Eksempel: mailing-list

Step 2: Legale transaktioner (mailing_list.sql)

- Indsættelse af nye emails og navne:

```
insert into mailing_list (email,name) values
('phillg@mit.edu','Phillip Greenspun');
insert into mailing_list (email,name) values
('bjergoe@it.edu','Frank Bjergø');
insert into mailing_list (email,name) values
('nh@it.edu','Niels Hallenbergl);
```
- Sletning af email og tilknyttet navn

```
delete from mailing_list where email = 'phillg@mit.edu';
```
- Opdatering af navn for tilknyttet email

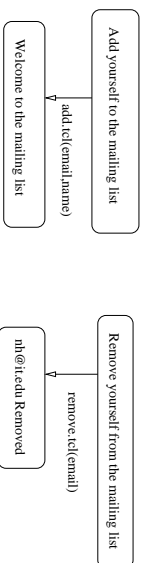
```
update mailing_list set name = 'Kurt Nielsen' where email = 'nh@it.edu';
```

Eksempel: mailing-list - filen add.html

Step 3: Opbygning af web-forms

```
<html>
<head>
<title>Add yourself to the mailing list</title>
</head>
<body bgcolor=#ffffff text=#000000>
<h2>Add yourself to the mailing list</h2>
<form method=post action=add.tcl>
<table>
<tr><td>Name<td><input name=name type=text size=35>
<tr><td>email<td><input name=email type=text size=35>
</table><p>
<input type=submit value="Add Me">
</form>
</body>
</html>
```

Step 3: Konstruktion af web-forms og site-map



Kasserne i diagrammet repræsenterer tilstande for hvilke HTML-kode vises i en brugers browser.

Pile repræsenterer links til en ny tilstand, evt. genereret via et Tcl-script. Nogle pile repræsenterer derfor også transaktioner for hvilke databasen opdateres gennem et Tcl-script.

I parentes angives de formvariable som overføres til Tcl-scriptet, dvs email og name.

Det er en god ide at skrive Tcl-scriptets navn på pilene.

Eksempel: mailing-list - filen add.tcl

Step 4: Konstruktion af database-transaktioner via tcl-scripts

```
# set tcl-variables to what the user typed into the form, that is,
# name and email.
set_form_variables 0

if { ![info exists email] || ![regexp {.+@.+} $email] } {
# the REGEXP didn't match
ns_return 200 text/plain "Your email addresss doesn't look right to
us. We need your full Internet address ..."
return
}

# if we got here, that means the email address was OK
if {![info exists name] || [string compare $name "" ] == 0 } {
ns_return 200 text/plain "You didn't give us your name..."
return
}

# Error checking complete; ready to do real work
ns_return 200 text/plain "Here we should (1) insert into database and
(2) return new HTML page."
```

Datamanipulation via AOLservers tcl-API

Vi har brug for en forbindelse til databasen.

ns_db gethandle

- returnerer en *handle* til databasen

- denne handle bruges til at tilgå databasen i efterfølgende kommandoer

```
...
set db [ns_db gethandle]
...
...$db.....
```

Samtidig fortolkning af to tcl-filer får derved forskellige handles til databasen

Handles gives tilbage til webserveren når fortolkning af en tcl-fil afsluttes

ns_db dml

- indsættelse af data - insert

```
...
ns_db dml $db "insert into mailing_list
              (name, email)
              values ($name, $email)"
...
```

Samme fremgangsmåde ved opdatering af data (update) og sletning af data (delete).

Double-Quotes

For at indsætte strengen Ken Mc'Donald skal man "double-quote", dvs ' skrives ' ' :

```
SQL> insert into mailing_list (email, name) values
      ('Ken@mac.org', 'Ken Mc'Donald');
ERROR:
```

```
ORA-01756: quoted string not properly terminated
```

```
SQL> insert into mailing_list (email, name) values
      ('Ken@mac.org', 'Ken Mc''Donald');
1 row created.
```

```
SQL> select name from mailing_list where email='ken@mac.org';
```

```
NAME
```

```
-----
Ken Mc'Donald
```

I stedet for set_form_variables kan vi anvende set_the_usual_form_variables.

I addv2.tcl får vi da fire variable: email, name, qqemail og qqname, hvor qqemail og qqname har alle ' erstatet af ' ' :

Hvis name = Ken Mc'Donald, da vil qqname = Ken Mc ' Donald.

Hvis name = Ken Hansen, da vil qqname = Ken Hansen.

I vores SQL-kommando anvender vi qqemail og qqname.

Eksempel: mailing-list - filen addv2.tcl

```
# set tcl-variables to what the user typed into the form, that is,
# name and email.
set_form_variables 0

set db [ns_db gethandle]

if { ![info exists email] || ![regexp {.+@\.+.+} $email] } { ... }
if { ![info exists name] || [string compare $name ""] == 0 } { ... }

set insert_sql "insert into mailing_list (email, name)
                values ('$email', '$name')"
ns_db dml $db $insert_sql

ns_return 200 text/plain "Welcome to the mailing-list."
```

Håndtaget til databasen gemmes i variabelen db.

SQL-kommandoen gemmes i variabelen insert_sql.

Vi udfører SQL-kommandoen med ns_db dml.

Læg mærke til, at vi anvender ' omkring strenge i SQL.

Læg mærke til, at vi anvender \$email og \$name når vi laver SQL-kommandoen.

Problem: Prøv at indsætte navnet Ken Mc'Donald

Eksempel: mailing-list - filen addv3.tcl

```
# set tcl-variables to what the user typed into the form, that is,
# name, qqname and email, qqemail.
set_the_usual_form_variables 0

set db [ns_db gethandle]

if { ![info exists email] || ![regexp {.+@\.+.+} $email] } { ... }
if { ![info exists name] || [string compare $name ""] == 0 } { ... }

set insert_sql "insert into mailing_list (email, name)
                values ('$qqemail', '$qqname')"
ns_db dml $db $insert_sql

ns_return 200 text/plain "Welcome to the mailing-list."
```

Vi anvender qqemail og qqname når vi bygger SQL-kommandoen.

Problem: Prøv at indsætte den samme post to gange.

Kan vi indsætte en post uden email?

Kan vi indsætte en post uden navn?

Fejlhåndtering med catch

```
catch script errmsg
```

Kommandoen `catch` udfører kommandoerne i `script`, og to ting kan ske:

1. `script` fejler ikke, i hvilket tilfælde `catch` returnerer 0
2. `script` fejler, i hvilket tilfælde `catch` returnerer et tal forskelligt fra 0 og variabelen `errmsg` sættes lig en beskrivelse af fejlen.

Man anvender ofte `catch` sammen med en `if`-kommando, således at man kan reagere på fejl:

```
if { [ catch script errmsg ] } {  
    fejl_kode  
}
```

Koden *fejl_kode* udføres kun, hvis der opstår en fejl når *script* udføres.

I koden *fejl_kode* kan vi henvise til variabelen *errmsg*.

Eksempel:

```
% if { [ catch {expr 4 / 0} errmsg ] } { puts "Fejl: $errmsg" }  
Fejl: divide by zero
```

Håndtering af databasefejl (addtv4.tc1)

Med `catch` kan vi håndtere fejl fra databasen:

```
if { [ catch { ns_db dml $db $insert_sql } errmsg ] } {  
    fejl_kode  
}
```

Vi kan f.eks. returnere en side til brugeren, som forklarer problemet.

```
set the_usual_form_variables 0  
set db [ns_db gethandle]  
if { [ !info exists email ] || [ regexp {.+@.+\.+} $email ] } { ... }  
if { [ !info exists name ] || [ string compare $name "" ] == 0 } { ... }  
  
set insert_sql "insert into mailing_list (email, name)  
                values ('$QOemail', '$QOname')"  
if { [ catch { ns_db dml $db $insert_sql } errmsg ] } {  
    # the insert went wrong; the error description  
    # will be in the Tcl variable ERRMSG  
    ns_return 200 text/plain "The database didn't accept your  
        insert, most likely because your  
        email address is already on  
        the mailing list...  
        The error message: $errmsg"  
} else {  
    ns_return 200 text/plain "Welcome to the mailing-list."  
}
```

Eksempel: mailing-list - filen remove.html

```
<html>  
<head>  
<title>Remove yourself from the mailing list</title>  
</head>  
<body bgcolor=#ffffff text=#000000>  
<h2>Remove yourself from the mailing list</h2>  
<form method=post action=remove.tcl>  
<table>  
<tr><td>email<td><input name=email type=text size=35></tr>  
</table><p>  
<input type=submit value="Remove Me">  
</form>  
</body>  
</html>
```

Bruger indtaster email, som er unik ident på person.

Sletning af data med ns_db dml

- sletning af data - delete

```
ns_db dml $db "delete from mailing_list  
              where email = 'nh@it.edu' "
```

Dette vil normalt ikke fejle medmindre vores SQL-kommando er forkert.

Der er mulighed for at 0, 1 eller flere rækker slettes.

I forbindelse med vores mailingliste

- forventer vi at slette 1 række.
- hvis vi sletter 0 rækker skyldes det at emailen ikke findes i mailinglisten.
- vi kan ikke komme til at slette mere end 1 række, hvorfor?

Antal rækker der slettes – ns_ora resultrows

Med kommandoen `ns_ora resultrows` og vores handle `db` kan vi spørge om antallet af rækker der blev slettet:

```
if { [ ns_ora resultrows $db ] == 0 } {  
    # 0 rows were affected  
} else {  
    # the delete affected at least one row  
}
```

Eksempel: mailing-list - filen remove.tcl

```
set_the_usual_form_variables
set db [ns_db gethandle]

# note that the dual calls to the SQL UPPER function
# ensure that the removal will be case insensitive
set delete_sql "delete from mailing_list
                where upper(email) = upper('$QQemail')"

ns_db dml $db $delete_sql
if { [ns_ora resultrows $db] == 0 } {
    ns_return 200 text/html "...Error..."
} else {
    ns_return 200 text/html "
    <html><head><title>$email Removed</title>
    </head><body bgcolor=#ffffff text=#000000>
    <h2>$email Removed</h2>
    <hr>
    You have been removed from the <a href=/index.html>www.greedy.com</a>
    mailing list.
    <hr>
    <address>webmaster@greedy.com</address>
    </body></html>"
}
```

Opdatering af data med ns_db dml

Helt identisk med sletning af data med ns_db dml.

- opdatering af data - update

```
ns_db dml $db "update mailing_list set name = 'Kurt Nielsen'
                where email = 'nh@it.edu'"
```

Dette vil normalt ikke fejle medmindre vores SQL-kommando er forkert.

Der er mulighed for at 0, 1 eller flere rækker opdateres.

I forbindelse med vores mailingliste

- forventer vi at opdatere 1 række.
- hvis vi opdaterer 0 rækker skyldes det at emailen ikke findes i mailinglisten.
- vi kan ikke komme til at opdatere mere end 1 række, hvorfor?

Antal rækker der opdateres – ns_ora resultrows

Med kommandoen ns_ora resultrows og vores handle db kan vi spørge om antallet af rækker der blev opdateret:

```
if { [ns_ora resultrows $db] == 0 } {
    # 0 rows were affected
} else {
    # the delete affected at least one row
}
```

Eksempel: mailing-list - filen update.html

```
<html>
<head>
<title>update yourself on the mailing list</title>
</head>
<body bgcolor=#ffffff text=#000000>
<h2>Update name</h2>
<form method=post action=update.tcl>
<table>
<tr><td>Name</td><input name=name type=text size=35>
<tr><td>email</td><input name=email type=text size=35>
</table><p>
<input type=submit value="Update Name">
</form>
</body>
</html>
```

Eksempel: mailing-list - filen update.tcl

```
set_the_usual_form_variables 0
set db [ns_db gethandle]
if { [info exists email] || [regexp {+@+\.\.+} $email] } { ... }
if { [info exists name] || [string compare $name ""] == 0 } { ... }

set update_sql "update mailing_list set name = '$QQname'
                where upper(email) = upper('$QQemail'"
ns_db dml $db $update_sql

if { [ns_ora resultrows $db] == 0 } {
    ns_return 200 text/html "... Error ..."
} else {
    # the update affected at least one row so update must
    # have been successful
    ns_return 200 text/html "
    <html><head><title>Name for $email Updated</title>
    </head><body bgcolor=#ffffff text=#000000>
    <h2>Name for $email Updated</h2>
    <hr>
    You have been updated on the <a href=/index.html>www.greedy.com</a>
    mailing list.
    <hr>
    <address>webmaster@greedy.com</address>
    </body></html>"
}
```

Eksempel: mailing-list - filen search.html

```
<html>
<head>
<title>Search the mailing list</title>
</head>
<body bgcolor=#ffffff text=#000000>
<h2>Search the mailing list</h2>
<form method=post action=search.tcl>
<table>
<tr><td>email<td><input name=email type=text size=35>
</td><td><input type=submit value="Search">
</form>
</body>
</html>
```

Eksempel: mailing-list - filen search.tcl

```
set_the_usual_form_variables
set db [ns_db gethandle]

set query "select name
           from mailing_list
           where upper(email) = upper('$qqemail')"

if { [catch {set name [database_to_tcl_string $db $query]} errmsg] } {
  ns_return 200 text/html "... Error ..."
} else {
  ns_return 200 text/html "
<html><head><title>$email Found</title>
</head><body bgcolor=#ffffff text=#000000>
<h2>$email Found</h2>
<hr>
We found the name $name for email $email on the <a
href=/index.html>www.greedy.com</a>
mailing list.
<hr>
<address>webmaster@greedy.com</address>
</body></html>"
}
```

Dataudtrækning med database_to_tcl_string db sql – en celle i en række

- udtækning 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
}
```

Dataudtrækning – 0, 1 eller flere rækker med ns_db select

ns_db select db sql

hvor

db er vores database handle og

sql er den forespørgsel som resulterer i 0, 1 eller flere rækker.

ns_db select returnerer en selection:

```
set query "select name, email from mailing_list"
set selection [ns_db select $db $query]
```

ns_db getrow for at hente de enkelte rækker

ns_db getrow db selection

hvor selection indeholder resultatet af et kald til ns_db select

```
while { [ns_db getrow $db $selection] } {
  # set tcl-variables corresponding to the selected columns
  set_variables_after_query
}
```

ns_db getrow returnerer 0, når der ikke er flere rækker.

set-variables-after-query skaber variable svarende til feltene i forespørgslen, f.eks. email og name.

Oversigt over mailinglisten (list.tcl)

```
proc home_page { title body } { ... }

set db [ns_db gethandle]

set query "select name, email from mailing_list"
set selection [ns_db select $db $query]

set tab "<table>\n<tr><th>Name</th><th>Email</th></tr>\n"
while { [ns_db getrow $db $selection] } {
    # set tcl-variables corresponding to the selected columns,
    # i.e., name and email
    set_variables_after_query

    append tab "<tr><td>$name</td><td>$email</td></tr>\n"
}
append tab "</table>\n"

ns_return 200 text/html [home_page "Mailing list" "This is the
mailing list: <br>$tab"]
```

Vi kan kode form-variable direkte i en url

Tcl-scriptet remove.tcl fjerner posten med den email som er angivet i form-variablen email.

Vi kan derfor skrive, f.eks.

```
remove.tcl?email=nh@it.edu
```

Vi adskiller form-variable med &, f.eks: remove.tcl?email=nh@it.edu&name=Kurt

Oversigt over mailinglisten - med mulighed for at slette emails (listv2.tcl)

```
proc home_page { title body } { ... }
set db [ns_db gethandle]
set query "select name, email from mailing_list"

set selection [ns_db select $db $query]
set tab "<table>\n<tr><th>Name</th><th>Email</th></tr>\n"
while { [ns_db getrow $db $selection] } {
    # set tcl-variables corresponding to the selected columns,
    # i.e., name and email
    set_variables_after_query
    append tab "<tr><td>$name</td><td>$email</td><td>
<a href='\"removev2.tcl?email=$email\">remove</a></td></tr>\n"
}
append tab "</table>\n"

ns_return 200 text/html [home_page "Mailing list" "This is
the mailing list: <br>$tab"]
```

ns_returnredirect (removev2.tcl)

Fra et Tcl-script kan vi returnere resultatet af at kalde et andet Tcl-script.

Fra listv2.tcl kalder vi removev2.tcl.

removev2.tcl sletter en email fra databasen.

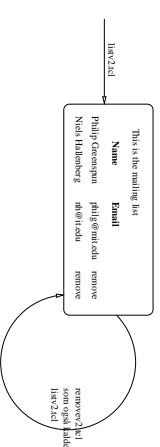
Deretter ønsker vi at se mailinglisten igen, dvs. resultatet af at køre listv2.tcl igen.

removev2.tcl:

```
set_the_usual_form_variables

set db [ns_db gethandle]
set delete_sql "delete from mailing_list
where upper(email) = upper('\"$email')"
```

ns_db dml \$db \$delete_sql
ns_returnredirect listv2.tcl



Introduktion til Øvelse 7

- Kommentarservice – *present multiple truths!*
- Brug mailing-list eksemplet som referenc

Sekvenser

En sekvens i Oracle er en tæller, som returnerer et unikt tal, der ikke tidligere har været anvendt.

Vi opretter en sekvens comment-sequence med

```
create sequence comment_sequence start with 2;
```

Sekvensens første tal er 2, og derefter returneres fortløbende numre.

Vi får det næste nummer i rækken ved at skrive

```
comment_sequence.nextval
```

Dette kan f.eks. skrives i en insert-kommando:

```
insert into comments (id, url, ...)
values (comment_sequence.nextval, ...);
```

Hvilke Oracle-tabeller er oprettet

```
select table_name from user_tables;
```