Part 19: Web-DB Interfaces

I have mainly used the following sources:

- Dynamic Information Systems, LLC: ORACLE Web Application Server Handbook. Oracle Press, 1998.
- ORACLE: Oracle Application Server, Overview of Oracle Application Server. Part No. A60115-02, 1998.
- ORACLE: Oracle Application Server, Developer's Guide, Introduction to Applications. Part No. A66957-01, 1998.
- ORACLE: Oracle Application Server, Developer's Guide, PL/SQL and ODBC Applications. No. A66958-01, 1998.
- Rainer Klute: Das World Wide Web (In German). 1996. `http://www.nads.de:82/~klute/WWW-Buch/`
- NCSA: The Common Gateway Interface. `http://hoohoo.ncsa.uiuc.edu/cgi/`
- James Marshall: CGI Made Really Easy. `http://www.jmarshall.com/easy/cgi/`
- Lincoln D. Stein: The World Wide Web Security FAQ. `http://www.w3.org/Security/Faq/`
- Lincoln D. Stein: Web Security, A Step-by-Step Reference Guide. Addison-Wesley, 1998.
- John Paul Ashenfelter: Choosing a Database for your Web Site. Wiley, 1999.
- More links are collected in `http://www2.sis.pitt.edu/~sbrass/db/webdb.html`

Overview

HTTP Protocol (1)

- Suppose a browser wants to access the web page

$$\underbrace{\texttt{http:}}_{\texttt{Protocol}}\texttt{//}\underbrace{\texttt{www2.sis.pitt.edu}}_{\texttt{Server Machine}}\texttt{:}\underbrace{\texttt{80}}_{\texttt{Port}}\underbrace{\texttt{/\~{}sbrass/db/}}_{\texttt{Absolute Path}}$$

- Then it first looks up the server name `www2.sis.pitt.edu` in the domain name service (DNS) to find the numeric IP address: `136.142.116.17`.

  Under UNIX, you can use the command `nslookup` to query the DNS directly.

- Next, it creates a TCP/IP connection to this machine, port 80. The web server waits here for incoming “calls”.

  Port 80 is the default. It does not have to be specified.

HTTP Protocol (2)

- Then it sends an HTTP-request to the web server. This is readable text:

```
GET /~sbrass/db/ HTTP/1.0
Accept:  image/gif, image/jpeg
⟨Empty Line⟩
```

- You can also connect directly to a web server with

```
telnet www2.sis.pitt.edu 80
```

  and enter the request manually.

- `GET` is the “method” of the request. There are others.
- There are many more possible headers besides “`Accept`”.

  RFC 1945 (HTTP/1.0), `http://www.w3.org/Protocols/`

HTTP Protocol (3)

- The configuration file of the web server contains rules what to deliver for the requested path `/~sbrass/db/`. The most common case is that it is translated into a path to a file:

  `/home/sbrass/public_html/db/index.html`

- The web server then answers with a HTTP response:

```
HTTP/1.1 200 OK
Date: Thu, 15 Apr 1999 18:50:29 GMT
Server: Apache/1.3.2 (Unix) ...
Last-Modified: Wed, 14 Apr 1999 13:34:00 GMT
Content-Length: 15407
Content-Type: text/html
⟨Empty Line⟩
... HTML Document follows ...
```


HTTP Protocol (4)

- Of course, a web server can deliver documents / objects / data files of any type. It is not restricted to HTML files.

  The client specifies in the “`Accept`” header which types it can process, and the server specifies in “`Content-Type`” which type is delivered in the body of the response.

- The delivered data does not have to be the contents of a file.

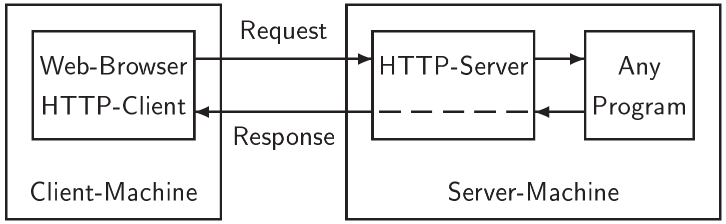
  Any program which speaks the HTTP protocol can act as a web server on the internet. The data could be generated out of a database. But it should be translated into HTML, otherwise the browser needs a plug-in for viewing relations.

- Finally, the TCP/IP connection is disconnected.

  Normally, the server “hangs up” after delivering the data.

Dynamic Documents (1)

- Often, the HTTP server responds to a request by delivering a file (static web page).
- But in general, the server can call any program for computing the response (dynamic web page).



Dynamic Documents (2)

- The client can also attach data to the request, which the server passes to the called program.
- Note that the server configuration must distinguish between programs and data files.

  There is not necessarily something special about the URL. Any request can lead to the execution of a program.

- Without special rules, the HTTP server would deliver the program itself (as a binary file), but would not execute it.
- In our case, `http://www2.sis.pitt.edu/` executes program files ending in “`.cgi`”. Another convention is that such programs reside in a special directory, e.g. “`cgi-bin`”.

Dynamic Documents (3)

- CGI (“Common Gateway Interface”) is an interface between a web server and programs generating dynamic documents.
- CGI defines only environment variables, standard input, output, and command line parameters (seldom used).
- CGI programs run in a different process (not inside the web server) and can be written in any programming language (Perl, C, and shellscripts are common choices).
- The output of a CGI-program looks like an HTTP response: headers, an empty line, data (e.g. an HTML file).
- However, the web server will analyze the headers you print and complete them. Only `Content-Type` is required.

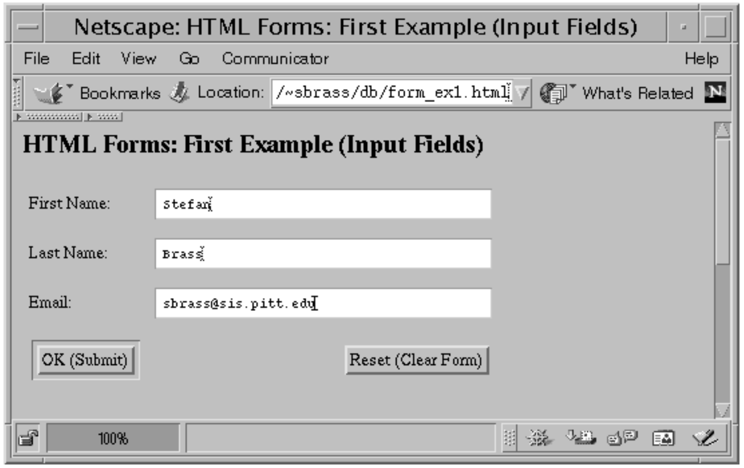
Dynamic Documents (4)

- E.g. store this shellscript in a file “`first.cgi`” in your `public_html` directory:

```
#!/bin/sh
echo "Content-Type:  text/plain"
echo ""
date
```

- Make the file executable, e.g. with `chmod 711 first.cgi`.
- Point your web browser to the URL
  `http://www2.sis.pitt.edu/~`⟨user⟩`/first.cgi`
- The web server will execute your program and deliver a (plain text) document containing the current date.

HTML Forms (1)



HTML Forms (2)

```
<FORM METHOD=POST
 ACTION="http://hoohoo.ncsa.uiuc.edu/cgi-bin/post-query">
<TABLE>
<TR><TD>First Name:  <TD><INPUT NAME="FirstName" SIZE=40>
<TR><TD>Last Name:   <TD><INPUT NAME="LastName" SIZE=40>
<TR><TD>Email:       <TD><INPUT NAME="Email" SIZE=40>
<INPUT TYPE=HIDDEN NAME="hiddenField" VALUE="42">
<TR><TD><INPUT TYPE=SUBMIT VALUE="OK (Submit)">
    <TD ALIGN=right><INPUT TYPE=RESET VALUE="Reset (...)">
</TABLE>
</FORM>
```

HTML Forms (3)

- If the user presses the submit button, an HTTP request will be sent to the address defined in the `ACTION` attribute.
- For the HTTP method `POST`, this request will contain the form data in the body of the request (as one long line):

```
FirstName=Stefan&LastName=Brass&
Email=sbrass%40sis.pitt.edu&hiddenField=42
```

- So the form contents is encoded in this way:

  ⟨Name⟩=⟨Value⟩&...&⟨Name⟩=⟨Value⟩

- A space is replaced by “+”, and certain special characters in the form data (e.g. +, &, =, %) are encoded as “%XX” where XX is the hexadecimal representation of the byte.

HTML Forms (4)

- For the HTTP method `GET`, the string encoding the form contents is appended to the `ACTION` URL (delimited by “?”).

```
http://hoohoo.ncsa.uiuc.edu/
    cgi-bin/query?FirstName=Stefan&...
```

- You can use such URLs in normal hyperlinks, the input data does not necessarily have to be entered each time in a form.
- `GET` should not be used if a large amount of data is submitted to the server (more than several hundered bytes).
  - Data is passed to CGI-program via environment variable.
- `GET`-requests can answered out of a cache. Be careful if the request is supposed to cause a state change on the server.

Overview

1. Basic Web Technology
2. Applications and Design Issues
3. CGI-Programs as Web-DB Interfaces
4. HTML Embedded Languages
5. Oracle Application Server

Data Sources: Files vs. DB (1)

- Suppose that a mail-order company want to publish its product catalog into the web.
- Manually create an HTML file for every product?
- But this will create a redundancy and possible inconsistency.
  - E.g. the price mentioned in the web page is not the same as the price contained in the database for generating invoices. Even a generation of the web pages each night is not a transaction concept.
- Often, all pages have a common structure (header etc.). Thus they anyway have to be generated from the real data.
  - Or else, this would be very redundant, which makes inconsistencies possible and changes very difficult.

Data Sources: Files vs. DB (2)

- The strictly structured data in the database can be searched much better than the text of HTML pages.

  E.g. VCRs for below $150 are hard to find with only keyword search. It is also difficult to distinguish books about Goethe from books authored by Goethe.

- The automatic generation of web pages can be done in different formats according to preferences of the user.

  E.g. compact overview or complete information, one long page or many small ones, different sorting criteria, etc.

- So if there are many similar structured pages, or bigger tables/lists, the pages should be automatically generated out of a database.

Page Generation (1)

- One basic decision is whether you generate pages on-the-fly when they are requested (dynamic pages), or precompute them and store them in standard HTML files (static pages).
- If the number of pages is relatively small and they do not have to be current up-to-the minute, the (re-)generation of all pages every night is probably the simplest solution.
- One option is to record changes to the DB in special tables and use these to generate only the changed pages.

  Triggers on the base tables can be used to record changes.

- You can see the generated pages as views on the data stored in the DB.

  The problem is similar to maintaining materialized views.

Page Generation (2)

- The performance of computing pages on demand is not very good: If the web server is frequently accessed, the load generated by the recomputations might be too high.
- This is especially important if your production DB is used: It might be that your employees are hindered in their work.

  This is anyway a bad idea for security reasons: Web-DB interfaces are not very secure, and you want to publish only part of your data. Also think about a spider accessing your web server at full speed (denial-of-service attack).

- You can use caching schemes to avoid the performance problems (you should delete cached copies when the base data is changed or after a given time period).

Page Generation (3)

- Another decision is whether you want to personalize pages:
  - E.g. you know that the current user has already bought database books. Maybe you want to make him/her a special offer for another DB book on your homepage.
  - The user has requested to have very “classical” pages without frames, applets, colorful graphics.
  - You want to show the user a different advertisement each time he/she visits your site (or even on each page).
- But Bookmarking/link exchanges do not work as expected.

  However, you can do personalization be redirection by to another page which remains more stable.

Page Generation (4)

- One question is where you store the static parts (HTML text) of generated pages.
- Putting common headers etc. as print statements into programs makes later changes more difficult.
  You should rather have them in HTML files.
  Of course you can also store the HTML fragments in the database, but then you have to check them out for editing, and store them back afterwards.
- Think about web designers who are not programmers.
- If you embed program code into web pages, check which HTML editors can work with them.
  Some systems put commands in HTML comments.

Query Interfaces: Design Decisions (1)

- Navigating access (via hyperlinks, index: “for items starting with A click here”) or searching access (via forms) or both.
  Access via forms is often simpler and more goal-directed, but navigating access allows it to see in principle all information (important for search engines).
- If there are multiple search conditions, are they connected with “and” or “or”, or can the user select this?
- If there is a search condition for a text field, does the entered string have to be the exact value, a prefix, a substring, a word inside the string? Is the search case sensitive or not? Or can the user select all this?

Query Interfaces: Design Decisions (2)

- If there is a search condition for a numeric or date field, can the user select <, >, `BETWEEN`?
  Which formats are accepted for specifying dates?
- Which sorting criteria will be used?
- What is done if the number of answer tuples is too large?
  E.g. only an error message is printed. Or the first n tuples are shown, with or without the possibility to follow a link to see the next n tuples. Can the user select the limit n?
- Maybe you want ensure that copying the entire database is not too simple.
  E.g. one text search field must be filled out, and there must be more or less an exact match on this field.

Query Interfaces: Design Decisions (3)

- What if the query produces no answer tuples?
  Will the query be automatically weakened (fuzzy logic)?
- Is the user supported in the query formulation?
  E.g. it is possible to generate menues with values which actually appear in the database.
- Is the output format selectable?
  E.g. compact overview vs. complete information. Or the user can select which fields should appear in the output.
- Are field values in the output linked to other DB tuples?
  E.g. from an instructor to his/her courses and vice versa.
  You can put a new query as a link behind output values.
  Drill-down-interface: list of courses linked to detail pages.

Update Interface: Design Decisions

- The classical solution is to show the user a form which is filled out with the current values. The user can modify these values and submit the form.

  Note that e.g. "Increase all salaries by 5%" is very tedious this way. You must prepare such special updates.
- How do you know which row to change? You need some key which the user cannot modify (maybe in a hidden field).
- Note that lost updates can occur.

  One solution is to store the current values of all columns in hidden fields. When the update request arrives with the old and the new values, you check whether the old values are really what is currently stored in the DB.

User Sessions (1)

- HTTP is a stateless protocol: Every request is treated in isolation. There are no "sessions" with login and logout.

  This improves the performance: After the server has sent the response, it can free all memory for this request.
- But then we get back to the times of batch-processing: The request must contain all data, there are no "interactive programs" any more (before Java).
- How do you program a shopping basket where a user selects some goods and then checks out? One solution are cookies.
- Cookies are small pieces of data, which the server can send in the response header together with a web page.

User Sessions (2)

- The client is then supposed to save such cookies and send them together with future requests to the same server.

  One can define that it should be sent to all servers from the domain, and only for specific URLs.
- E.g. `telnet www.infoseek.com 80` gives:

```
   HTTP/1.0 200 OK
   MIME-Version:  1.0
   Date:  Sat, 17 Apr 1999 21:33:25 GMT
   Content-Type:  text/html
   Expires:  Fri, 15 Jan 1999 07:20:05 GMT
→  Set-Cookie:  InfoseekUserId=9A67EB8D...;
                domain=.go.com; path=/; expires=...
   Content-Length:  29097
```


User Sessions (3)

- The client would then send the cookie in a request:

  `Cookie:  InfoseekUserId=9A67EB8D...`
- In this way, the burden to keep a state is moved from the server to the client.

  However, often the cookie contains only a reference to a state stored on the server.
- Cookies can be used for counting the number of distinct users or for keeping user preferences (or a shopping basket?).
- In netscape, look at the file `~/.netscape/cookies`.
- A preliminary specification is available at:

  `http://www.netscape.com/newsref/std/cookie_spec.html`

User Sessions (4)

- http://www2.sis.pitt.edu/~sbrass/set_cookie.cgi:

```
#!/bin/sh
echo ’Content-Type: text/plain’
echo ’Set-Cookie: SBRASSTEST=test; ’ \
     ’domain=www2.sis.pitt.edu; ’ \
     ’path=/~sbrass; ’ \
     ’expires=Tuesday, 31-Dec-2030 23:59:59 GMT’
echo
echo ’Cookie set.’
```

- http://www2.sis.pitt.edu/~sbrass/show_cookie.cgi:

```
#!/bin/sh
echo ’Content-Type: text/plain’
echo
echo ’Cookie: ’ "$HTTP_COOKIE"
```


User Sessions (5)

- Browsers do not have to store cookies.

  There is an option for ignoring them. So you cannot rely entirely on cookies (at least give a good error message).
- If links are followed in a forward direction, it is possible to deliver web pages which contain the cookie embedded in the URLs or in hidden input fields.
- Also the IP address of the computer (and maybe an email address) can relate different requests to one user session.

  This is not really reliable (e.g. connections via proxy).
- Non-active sessions should be aborted after some time.

  Some users might have the last page still in their browser and want to “sleep over it” before finalizing the purchase.

User Sessions (6)

- Try to make sure that nobody can steal other user’s connections or else that it doesn’t do harm.
- E.g. if you use sequential session ID numbers, a hacker could start a session and then try the previous/next number.
- You should check at least that all requests for one session originate from the same computer.
- It cannot do much harm (?) if the credit card number is entered last, and all ordered goods are shown on that form.

  Although you will lose customers who will find e.g. a book in their shopping basket which they never selected.
- But you should not use sequential/easily guessable IDs.

Transactions

- Sometimes it might be important to process several HTTP requests in one database transaction.

  This might be very inefficient, since you must keep the DB connection until the last request is processed (or a timeout occurs). Locks set by the transaction might hinder other users.
- To solve the problem with a price changing while the user submits an order, most companies require to submit the price with the order.

  After all, the price and the ware are part of the contract. The order processing system can then check whether the price is still current or whether it was valid a short time ago and is not too different from the current price.

Web Browser as DB Interface (1)

- Usually, the application programs of a company use a vendor-specific form/screen mask interface (e.g. SQL*Forms).
  - Or they are completely customized programs (C/Windows).
- A web browser might be a good alternative as user interface.
  - For security reasons, these applications will probably first be only available in the intranet of the company.
- Web browers are available for many hardware platforms. No porting is needed, the application runs only on the server.
  - There were many attempts to create portable GUIs, but before the web, no one was generally accepted.
- Simple administration: Everything is stored on the Server.

Web Browser as DB Interface (2)

Traditional Client-Server (Two-Tier):

Client	User Interface	Menues, Dialog-Boxes, Forms Online-Help Display of Multimedia Data
	Application Logic	Dialog-Control Integrity Checking Format conversions Complex Aggregations
Server	Database	Searching and Sorting Persistence, Backup/Recovery Multiuser-Access

Web Browser as DB Interface (3)

With Web-Interface (Two-Tier / Three-Tier):

Client	User Interface	Client ("Thin Client")
Server	Application Logic	Application Server
	Database	Database Server

Web Browser as DB Interface (4)

- No extra effort for making applications remotely accessible.
  - Except for careful security checks.
- Multimedia data can be easily displayed.
  - Traditional DB user interfaces are mainly form-based and run also on text terminals.
- Links to documents and external resources can be displayed.
- However, HTML forms are not very expressive ($\rightarrow$ Java?).
  - E.g. if only a number can be entered in a field, an error will only be detected after the form submission (on the server).
- One technology for company-internal applications and the (anyway necessary) representation on the web.

Tools

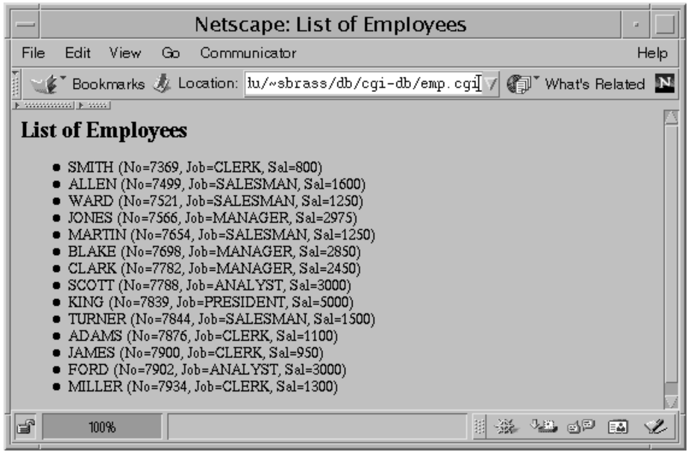
- Conventional programming languages with a DB interface.
  Maybe enhanced by libraries for accessing CGI input.
- Programming languages embedded in HTML.
  These are often more restricted and specialized to the task of producing HTML output.
- Ready-made DB-Interfaces: In the extreme case you specify only the table name, and it gives insert/update/delete and query facilities via the web.
  There are even DBMS which can be entirely controlled over the web, e.g. Filemaker Pro.
- Java applets on the client, accessing the DB server directly via JDBC (not via HTTP and the web server).

Overview

CGI-Programs as Web-DB Interfaces

- A CGI-program can be any program.
- E.g. it can be a C-program with embedded SQL.
  There are also DB-interfaces for other languages, e.g. Perl.
  You can also use a shellscript calling SQL*Plus.
- Note that the CGI program will be started by the HTTP server, and the environment variables will not have the same values as when you directly execute the program.
  In other installations, CGI programs are executed as user “nobody” with very little file access rights. In our case, they run under your account.
- So you must set `ORACLE_HOME`, `TWO_TASK`, and `ORACLE_SID` explicitly in the program.

Example



Example in C (1)

```c
#include <stdio.h>
#include <stdlib.h>
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR user[128];
    VARCHAR passwd[32];
    int empno;
    varchar ename[20];
    varchar job[20];
    int sal;
EXEC SQL END DECLARE SECTION;

```


Example in C (2)

```c
main()
{   char ora_msg[512];
    size_t buf_len = 512; size_t text_len;

    /* Set Environment for Oracle:  */
    putenv("ORACLE_HOME=/opt/local/...");
    putenv("TWO_TASK=sis");
    putenv("ORACLE_SID=sis");

    /* Print header:  */
    printf("Content-Type:  text/html\n");
    printf("\n");
```


Example in C (3)

```c
    /* Print HTML Head:  */
    printf("<!DOCTYPE HTML PUBLIC \"...\">\n");
    printf("<HTML>\n");
    printf("<HEAD>\n");
    printf("<TITLE>List of Employees</TITLE>\n");
    printf("</HEAD>\n");
    printf("<BODY>\n");
    printf("<H1>List of Employees</H1>");

    /* Catch errors:  */
    EXEC SQL WHENEVER SQLERROR GOTO error;

```


Example in C (4)

```c
    /* Log into Oracle:  */
    strcpy(user.arr, "SCOTT");
    user.len = strlen(user.arr);
    strcpy(passwd.arr, "TIGER");
    passwd.len = strlen(passwd.arr);
    EXEC SQL CONNECT :user IDENTIFIED BY :passwd;

    /* Declare a Cursor:  */
    EXEC SQL DECLARE all_emps CURSOR FOR
        SELECT EmpNo, EName, Job, Sal FROM Emp;
    /* Open the Cursor:  */
    EXEC SQL OPEN all_emps;
```

Example in C (5)

```
/* Fetch and print employee data:  */
printf("<UL>\n");
EXEC SQL WHENEVER NOT FOUND GOTO done;
while(1) {
    EXEC SQL FETCH all_emps
        INTO :empno, :ename, :job, :sal;
    ename.arr[ename.len] = 0;
    job.arr[job.len] = 0;
    printf("<LI> %s (No=%d, Job=%s, Sal=%d)\n",
        ename.arr, empno, job.arr, sal);
}
done:
```


Example in C (6)

```
printf("</UL>\n");
EXEC SQL WHENEVER NOT FOUND CONTINUE;

/* Close the cursor:  */
EXEC SQL CLOSE all_emps;

/* Print End of Document:  */
printf("</BODY>\n");
printf("</HTML>\n");

/* End transaction successfully, log off:  */
EXEC SQL COMMIT WORK RELEASE;
```


Example in C (7)

```
    return(0);

    /* In case of errors:  */
    error:
        EXEC SQL WHENEVER SQLERROR CONTINUE;
            sqlglm(ora_msg, &buf_len, &text_len);
            ora_msg[text_len] = 0;
            printf(Oracle Error:  %s\n", ora_msg);
        }
        EXEC SQL ROLLBACK WORK RELEASE;
        exit(1);
}
```


Example with Shellscript (1)

```
#!/bin/sh

# Set environment variables for Oracle:
ORACLE_HOME=/opt/local/...
export ORACLE_HOME
TWO_TASK=sis
export TWO_TASK
ORACLE_SID=sis
export ORACLE_SID
PATH=/usr/bin:$ORACLE_HOME/bin
export PATH
```

Example with Shellscript (2)

```
# Print Header:
echo "Content-Type:  text/html"
echo ""
# Print HTML Head:
echo "<!DOCTYPE HTML PUBLIC \"...\">"
echo "<HTML>"
echo "<HEAD>"
echo "<TITLE>List of Employees</TITLE>"
echo "</HEAD>"
echo "<BODY>"
echo "<H1>List of Employees</H1>"
```


Example with Shellscript (3)

```
echo "<UL>"

# Call SQL*Plus to get the data:
sqlplus scott/tiger @emp_list >errors 2>&1
cat all_emps.lst

# Print End of Document:
echo "</UL>"
echo "</BODY>"
echo "</HTML>"
```


Example with Shellscript (4)

SQL*Plus Script "`emp_list.sql`":

```
set pagesize 0
set feedback off
set echo off
set termout off
spool all_emps
SELECT '<LI> ' || EName || ' (No=' || EmpNo
    ', Job=' || Job || ')' FROM Emp;
spool off
quit
```


Debugging of CGI Programs (1)

- It is simpler to debug the program first as far as you can by directly calling it (not via the HTTP deamon).

  This is also the only possibility to uses debuggers.
- In order to do this, write a small shellscript which sets up typical values for the environment variables and calls your program with an example input string.
- Output on `stderr` (standard error channel) does not appear when the program is executed by the HTTP deamon.

  So make sure that all output goes to `stdout` or a file. If you really need it, the server log file may contain `stderr` output. It is also prossible to catch exceptions such as "segmentation fault" and print an error message to `stdout`.

Debugging of CGI Programs (2)

- Print the HTTP header and the start of the HTML output as early in your program as you can, so that later error output becomes visible.

  If you print error messages before the `Content-Type` header etc., the HTTP deamon does not know what to do with them. Print error messages in HTML format.

- Keep in mind that your program later runs in a different environment (without your settings from `.bashrc`) and maybe under a different user account.

  E.g. print helpful error messages if you can’t open files.

- The HTTP deamon can start different instances of your program concurrently (if requests arrive at the same time).

Input to CGI Programs (1)

Method `POST`:

- The string encoding the field values can be read from the standard input channel.

  The length of this string is contained in the environment variable `CONTENT_LENGTH`. You must exactly that much characters from `stdin`, you cannot read until the EOF.

- You might want to check that the environment variable `REQUEST_METHOD` is really `POST`, and that `CONTENT_TYPE` is really “`application/x-www-form-urlencoded`”.

- There are library procedures for C, Perl, and other languages to decode the field values. But this is anyway not difficult.

Input to CGI Programs (2)

Method `GET`:

- The string encoding the field values (everything after the “?” in the requested URL) is contained in the environment variable `QUERY_STRING` (`REQUEST_METHOD` will be “`GET`”).

  If `QUERY_STRING` contains no “=”, its words (separated by “+”) are passed in addition as command line arguments. Cannot happen with forms, but see HTML tag `ISINDEX`.

- The URL might contain additional components after the path identifying the CGI-program (for `GET` and `POST`). These are passed in the environment variable `PATH_INFO`.

  E.g. `http://.../db/cgi-bin/cgi_test.cgi/ab/c?de` sets `QUERY_STRING` to “`de`” and `PATH_INFO` to “`/ab/c`”.

Input to CGI Programs (3)

Environment-Variables:

- The server makes some HTTP headers from the request available in environment variables called `HTTP_*`.

  E.g. `HTTP_ACCEPT`, `HTTP_USER_AGENT`.

- `REMOTE_ADDR` is the IP address of the machine from which the request came. This can also be a proxy server (?).

  Some web servers set `REMOTE_HOST` to the name of the machine. Many browsers transfer the user’s email address in the request, but this is often not passed to CGI-programs.

- `SERVER_NAME`, `SERVER_PORT`, `SCRIPT_NAME` allow to construct the URL under which the CGI-program was called.

Security (1)

- With CGI-programs, you allow the whole world to execute programs on your machine.
- It can happen relatively easily that you create security holes.

  Some universities do not allow their students to write CGI-programs. Some execute CGI-programs as a user “nobody” with small file access rights. (Never run them as root!) Some ensure that CGI-programs may never access files outside the `public_html` folder (with “change root”).
- Even experienced programmers have made such mistakes.
- The problem is that a program might do under special circumstances things which you did not think that it could.

Security (2)

- E.g. we want to make manual pages available in the web. The user fills the program name into a form, we get the value into a variable `$prog`, and print the page (in Perl) with:

  ```
  system("man $prog");
  ```
- A malicious user can enter “`cp; rm -rf /`” into the form, so the string “`man cp; rm -rf /`” will be passed to the shell for execution.
- Never pass input values to a shell without very careful testing. E.g. check that `$prog` is a sequence of letters.

  Such a positive test is much safer than a negative test (Does it contain any special characters?).

Security (3)

- Suppose you want to do the same thing, but only for 10 possible programs. So you do you not use an input field in the HTML form, but a menu with only these 10 possible choices. Does this help? (This answer is no, but why?)
- Even hidden fields can be read by the client (show document source) and set arbitrarily in requests.
- Also the following command is dangerous:

  ```
  open(MANPAGE, "/usr/man/man1/$prog");
  ```

  How would a hacker be able to see `/etc/passwd` instead?
- Don't make the source-code of your CGI-programs available via the web — the hacker can then simpler search for holes.

Security (4)

- Since it was a bit difficult to start CGI-Programs written in Perl under Windows NT, the Netscape hotline has suggested (for a few days) to make the Perl-interpreter available as CGI-program, and call your program as

  ```
  http://poor.machine/cgi-bin/perl.cgi?myprog.pl
  ```

  What do you think about this?
- Some servers support “server side includes” (SSI). This means the server will parse the HTML file before delivering it, and will process commands it it (including arbitary shell commands). It has happened that guestbooks (where any user can enter text) had the SSI feature activated.

Security (5)

- Don't think that if you run CGI-programs as "nobody", it wouldn't do any harm if a hacker somehow gets a shell.

  He can fetch `/etc/passwd` and other files and search for security holes. He can use your computer to start elsewhere attacks. He can start many processes to make your machine slower. He can change the log file of the web server.

- Even C-programs which do not check array boundaries before copying data from the form are a security hole.

  First the hacker simply crashes your program. Then he fetches the core file. With a careful analysis, he can devise an input string which overwrites a return address on the stack and jumps to "`system`"-call with any command.

Security (6)

- You should store passwords in encrypted form.

  Users often have the same password for multiple WWW shops. If you store the password in readable form, your employees might use the password for other shops. Also, if a hacker gets access to your database, the damage is bigger.

- Do not give the user the possibility to store credit card information for future visits of your shop.

  If the password of this user becomes somehow known to hackers, there is no further protection.

Problems of CGI

- The programmer needs to know the programming language, CGI, HTML, SQL, and the DB interface.

  He/she has to work with three languages at the same time.

- If one wants to construct a mainly static document with only small portions generated out of the database, the "print" commands are clumsy.
- Performance problems: CGI generates for every request a new process. This is already an expensive operation.
- But in our case, the process will connect to the database and open it. This might take several seconds.

  Many short living DB application processes also make certain kinds of cashes useless.

Solutions (1)

- One can avoid to open the database again for every request if one has an "application server" process, which keeps running and does the real work for the CGI program.

  So the web server starts a CGI-program, but this is small and only forwards requests/responses to a server process.

- One can avoid to start a new process for every request by not using CGI but instead making the program for generating the dynamic document part of the web server process.

  Every web server has its own API (application program interface) for exchanging data which procedures linked to it. If you use this solution, you can keep the DB connection open all the time the web server runs.

Solutions (2)

- If the web server has an interpreter e.g. for Perl built-in, it does not have to start a new process for executing CGI-programs written in Perl.
- The problem of mainly static documents with only small dynamic portions are often solved by allowing to embed program code into HTML (similar to server side includes).
- The problem of having to know CGI is solved by library procedures or interpreters which map field names into variables/parameters.
- Java Servlets might replace CGI programs in the future. See next page, `http://java.sun.com/products/servlet/`.

Java Servlets

- Java Servlets are Java programs which the web server can execute in order to compute the response to a request.
- So they are in principle similar to CGI programs written in Java, however:
  - The Java virtual machine for executing them is part of the web server, so no new process has to be created.
  - A servlet is started only once and then can process many requests. E.g. a DB connection can stay open all the time.

    In contrast, a CGI program terminates itself once it has processed a single request.
- Servlets are already nearly as portable as CGI programs. Most major web servers support them.

Overview

Example: HotSQL (1)

- HotSQL (`http://www.chimerasoft.com/hotsql/`) is an example of a language which extends HTML by special tags for accessing the database.

  Using of HotSQL as an example here does not mean that I would recommend it more than other products. It is only a good simple example. There is a variety of products and I do not have sufficient practical experience yet to recommend any specific product.
- HotSQL is an interpreter which can be called as a CGI program. (So it does not solve the performance problems.)
- Suppose our task is to print a list of employees for a department chosen by the user.

Example: HotSQL (2)

- So we would create a normal HTML file with a form which allows the user to choose a department:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML><HEAD>
<TITLE>Employee Database (Query Form)</TITLE>
</HEAD><BODY>
<H1>Employee Database (Query Form)</H1>
<FORM METHOD="post"
    ACTION="/cgi-bin/hotsql.exe/emp_print">
Department Number:  <INPUT NAME="Dept" SIZE=2>
<INPUT TYPE=SUBMIT VALUE="Execute Query">
</FORM></BODY></HTML>
```


Example: HotSQL (3)

- In the action of this form, we call the HotSQL interpreter with the following input file:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML> <HEAD> <TITLE>Employee-DB (Query Result)</TITLE>
</HEAD> <BODY>
<!-- hsql print "<H1> Department $Dept</H1>" -->
<UL>
<!-- hsql CONNECT sis, scott, tiger -->
<!-- hsql SELECT ENAME, JOB FROM DEPT D
          WHERE DEPTNO = $Dept -->
<!-- hsql LIST "<LI> $1 ($2)" -->
</UL>
</BODY> </HTML>
```


Example: HotSQL (4)

- HotSQL commands are written as HTML comments of the form <!-- hsql ... -->
- This allows to use standard HTML browsers and editors for developing the file.
- All other input text is simply printed to the output.
- HotSQL parses the CGI input and makes the input fields available as $field.
- The LIST command prints the result of the preceding SELECT query in the given format.

  $i is replaced by the i-th result column.
- You can add a limit (maximal number of rows) to LIST.

Example: HotSQL (5)

- Be careful with textual replacement in SQL queries.

  E.g. a user can enter 10 OR 1=1 to see all employees.

  It might be that HotSQL excludes this (I have not tried it). The problem is always when data becomes program code.
- Note that the interpreter is supposed to be stored in the cgi-bin directory. This is potentionally dangerous.

  You must check that program code is not accepted from standard input. It seems like all documents on your web server can potentially be executed.
- HotSQL source code contains database passwords.

  You should make sure that it can never be seen by anybody you don't trust (especially not via the web).

Example: HotSQL (6)

- Instead of `LIST`, you can use `LOOP`.
  This allows to format the result rows with `IF` statements.
  HotSQL commands (complete list):
  `CONNECT` (to ODBC data source), `EXEC` (SQL-command), `FETCH` (result row), `LOOP` (over result set), `PRINT` (a string), `LIST` (query result), `IF` (condition), `SET` (a variable).
- Exercise: Use HotSQL to print a form for selecting Departments with a menu of the departments in the DB:

```
<SELECT NAME="Dept">
    <OPTION VALUE="10">Accounting
    <OPTION VALUE="20">Research
    <OPTION VALUE="30">Sales
</SELECT>
```


Example: Cold Fusion (1)

- ColdFusion is a Web-DB interface tool which
  - is well known (standard topic in every Web-DB book)
  - has a much bigger functionality than HotSQL,
  - has professional support and professional prices,
    ColdFusion Server 4.0 for Windows costs $1295,
  - runs on Windows, Solaris, HP/UX, Linux,
  - runs not only as a CGI program, but can also be linked to a number of web servers (for improved performance),
  - supports not only ODBC connections, but also native connections to Oracle, DB2, Sybase, and Informix.
- See: `http://www.allaire.com/Products/ColdFusion/`

Example: Cold Fusion (2)

- Cold Fusion embeds its commands as new tags into HTML.
  All new tags start with “`CF`”. The extended language is called CFML (Cold Fusion Markup Language). Allaire offers a special HTML editor which understands these tags.
- Variable references must be enclosed in `#`. Example:

```
<CFQUERY NAME="emps" DATASOURCE="scotts_db">
    SELECT ENAME, JOB FROM EMP
</CFQUERY>
<TABLE>
    <CFOUTPUT QUERY="emps">
        <TR><TD>#ENAME# <TD>#JOB#
    </CFOUTPUT>
</TABLE>
```


Example: Cold Fusion (3)

- DB account and password are normally not defined in the CFML program, but in the server configuration file.
  This probably improves the safety.
- Input fields from HTML forms are also made available as variables.
- E.g. suppose that “`dept`” is an input field. You can access its value as “`#dept#`” or as “`#FORM.dept#`”.
  The prefix is only needed if there are name clashes.
  You can also reference an output column from a query by prefixing the query name, e.g. “`#emps.ENAME`”.
- In SQL statements, Cold Fusion will automatically double single quotes when a variable name is replaced by its value.

Example: Cold Fusion (4)

- You can optionally declare variables from the HTML form:

```
<CFPARAM NAME="FORM.dept" DEFAULT="10">
```

- If you do not specify a default value, an error message will be printed if the variable is undefined.
- You can also append parameters to the URL, e.g.

```
http://www.xyz.com/example.cfm?dept=10
```

- When `example.cfm` is processed, the variable “`URL.dept`” (or simply “`dept`”) will have the value 10.
- Cookies and CGI parameters are also made available as variables.

Example: Cold Fusion (5)

- Cold Fusion can also manage client variables, which are kept persistently for every user who accesses the website.

  So these variables keep their value between different page requests.
- This is normally done by means of cookies.

  The values can be stored in the Windows Registry, in a database, or in the cookie (on the client side).
- Session variables are similar but are kept only for the duration of a session.

  E.g. they are deleted after 20 minutes of inactivity.

Example: Cold Fusion (6)

- You can check whether a query returned a result:

```
<CFIF #emps.RecordCount# IS 0>
    <P>Department does not exist!
<CFELSE>
    ... print query result ...
</CFIF>
```

- You can execute any SQL statement:

```
<CFQUERY NAME="InsDEPT" DATASOURCE="scotts_db">
    INSERT INTO DEPT VALUES(#FORM.deptno#,
        '#FORM.dname#', '#FORM.loc#')
</CFQUERY>
```

- However, Cold Fusion has a simpler way to do this.

Other HTML-Embedded Languages

- PHP is a free and powerful language embedded in HTML: `http://www.php.net/`
  - It is claimed that PHP is used on nearly one million servers.
  - To access Oracle from PHP, you have to use the OCI functions (Oracle Call Interface).

    It is not as easy as embedded SQL.
  - PHP can be linked as a Module to the Apache Server (gives better performance than CGI).
- Oracle LiveHTML (Perl, see below).
- Active Server Pages (Visual Basic Script).

Overview

Architecture (1)

- The purpose of the Oracle Application Server (OAS) is to manage server-side application programs, i.e. to act as the middle tier of the three-tier computing model.
- It supports HTTP clients (web browsers) and CORBA clients (e.g. Java applets using JCORBA).
- The OAS comes with its own HTTP server ("HTTP Listener"), but it is also possible to use other web servers.
- This "HTTP Listener" is responsible for delivering static documents (and for calling standard CGI-programs).
- However, requests can also be forwarded via a CORBA object request broker (ORB) to "cartridges".

Architecture (2)

- Applications can be developed in a variety of different languages, e.g. PL/SQL, Java, C, Perl, LiveHTML.
- Oracle uses the term "Cartridge" ambigiously. It can be
  - An interpreter for a specific language, e.g. the JWeb Cartridge is a Java virtual machine and the PL/SQL cartridge executes stored procedures in the database.
  - Part of an application, e.g. a PL/SQL interpreter plus the configuration information to access procedures under a specific DB account.

    An application can consist of several cartridges.
  - A process/thread which executes your program.

Architecture (3)

- Oracle avoids the performance problems of CGI by using cartridge servers which run all the time.

  The number is configurable, and you can also allow that new cartridge servers are started when the others are busy, and killed again after being idle for some time.
- The dispatcher (called by the HTTP listener) forwards an incoming request to a free cartridge server.

  In fact, cartridge servers can have multiple threads, so they can process more than one request at the same time.
- Cartridge servers are bound to applications — they can only execute programs from one application (which must all be written in the same language).

Architecture (4)

- The OAS also has many untility processes, e.g. for authentication and logging.
- The OAS comes also with program development tools (Oracle JDeveloper for Java, Oracle Developer for PL/SQL) and tools for analyzing server statistics.
- The OAS can run on multiple machines with load balancing.
- The OAS can manage sessions (series of requests processed by the same cartridge which can keep a state).

  The requests can even be processed in one transaction.
- The OAS is administrated via the “OAS Manager” which can be accessed with any Java capable browser.

Example: PL/SQL (1)

- The PL/SQL cartridge can execute stored procedures.
- The values for the input fields in the form are automatically converted to parameters of this procedure.
- There are library packages for writing HTML, processing cookies, and more.

  E.g. `owa_util.tablePrint` takes a table name and outputs this database table as an HTML table.
- Suppose we have a form in which the user can enter the name of a department (or a substring) and we want to show a list of all employees in this department.
- We write a PL/SQL procedure for this task (→ next pages).

Example: PL/SQL (2)

```
CREATE OR REPLACE PROCEDURE
   emp_by_dept(d_name varchar)
IS
   CURSOR c(d varchar) IS
      SELECT * FROM Emp WHERE deptno IN
         (SELECT x.deptno FROM dept x
          WHERE UPPER(dname) LIKE
                ’%’||UPPER(d)|| ’%’);
   e c%ROWTYPE;
   result_empty boolean := true;
BEGIN
```


Example: PL/SQL (3)

```
   htp.htmlOpen;
   htp.headOpen;
   htp.title(’Employee List’);
   htp.headClose;
   htp.bodyOpen;
   htp.heading(1, ’Employees’);
   htp.ulistOpen;
   FOR e IN c(d_name) LOOP
      htp.listItem(e.ename ||
                ’ (’ || e.job || ’)’ );
      result_empty := false;
```

Example: PL/SQL (4)

```
      END LOOP;
      htp.ulistClose;
      IF result_empty THEN
         htp.print('Sorry, no employees.');
      END IF;
      htp.hr;
      htp.print('Generated:  '  ||
         TO_CHAR(SYSDATE,'DD-MON-YY, HH:MI'));
      htp.bodyClose;
      htp.htmlClose;
   END;
```


Example: PL/SQL (5)

- Then we use the OAS Manager to create a new application, e.g. `staff`. We choose the type “PL/SQL”.
- Next, we create a new cartridge for this application, say “`cart1`”.
- We must define a “database access descriptor” for this cartridge, this is username, password, and the location of the database on the network (it can run on another machine).

  Username and password can be left blank in which case the user will be asked when he/she tries to access an URL processed by this cartridge.
- Alternatively, the web server can restrict accesses to URLs.

Example: PL/SQL (6)

- Only a special “admin” user can create new cartridges and afterwards certain system processes must be restarted.
- A cartridge can run different stored procedures, e.g.

  `http://.../staff/cart1/emp_by_dept?d_name=SALES`

  will access our procedure.

  The “virtual path” for a cartridge is configurable, one can choose “`/staff`” instead of “`/staff/cart1`”. Multiple cartridges are only needed for accessing different DB accounts or for special load balancing requirements.
- You can also specify what happens in case of an error (e.g. how much information is shown to the client).

LiveHTML (1)

- LiveHTML is Oracle’s implementation of server-side includes (SSI). It also allows to embed Perl scripts in HTML pages.
- Server-side includes are useful for mostly static web pages. One can write normal HTML, but embed a few commands in it, which are executed and replaced by their output before the document is delivered.
- An SSI command has the following form (HTML comment):

  ```
  <!--#⟨command⟩ ⟨Parameter⟩="⟨Value⟩" ...>
  ```
- E.g.

  ```
  <!--#include file="foot.html">
  ```

  would be replaced by the contents of the file “`foot.html`”

LiveHTML (2)

- `<!--#echo var="⟨Variable⟩">`
  allows to print environment variables.
  CGI variables and some new ones, e.g. DATE_LOCAL.
- `<!--#flastmod>`
  prints the date of the last modification of the file.
  You can specify any file with the parameter “file”.
- `<!--#exec cmd="/bin/who">`
  executes /bin/who and pastes the result at this point in the HTML-file (probably inside a <PRE> tag).
  Any shell command can be executed in this way.
  It is also possible to call CGI-programs (cgi="...").

LiveHTML (3)

- The command #request is an Oracle extension to SSI.
  It allows to access other cartridges.
- E.g. it is a bit clumsy to construct a complicated HTML page by procedure calls in PL/SQL.
- Suppose we reduce the procedure emp_by_dept only to the main loop, which prints the employee data.
- Then we could construct a normal HTML file and embed the command

```
<!--#request
      URL="http://.../emp_by_dept?d_name=SALES">
```

at the point where the employee data should appear.

LiveHTML (4)

- This also works if the LifeHTML page itself was accessed with parameters: E.g. “http://...?dept=SALES”.
- Then we can pass the parameter to the PL/SQL procedure:

```
<!--#request
      URL="http://.../emp_by_dept?d_name=$dept">
```

- Oracle also supplies an “ODBC Cartridge” which processes an SQL command passed as a parameter of the URL:

```
http://.../odbc/tableprint?dsn=ora&username=scott
          password=tiger&maxrows=10&d=20&
          sql=select+*+from+emp+where+deptno=:d
```


LiveHTML (5)

- It is possible to embed pieces of Perl-code in a LiveHTML document. The LiveHTML cartridge will execute this program code and replace it by its output.
- The program code is embedded between “<%” and “%>”:

```
<% if($result_empty) { %>
      <P>Please check department name.
<% } else { %>
      <H1>Employee List</H1> ...
<% } %>
```

- HTML text between scripts is treated like write statements.
  All script parts together are executed as one program.

Oracle Developer (1)

- Oracle Developer is Oracles main suite of application development tools. It consists of:
  - Form Builder
  - Report Builder
  - Graphics Builder
  - Project Builder
  - Procedure Builder
  - Translation Builder
  - Schema Builder
  - Query Builder

Oracle Developer (2)

- E.g. the Form Builder allows to construct form/screen mask based application programs which can be understood as specialized relation editors.

  They allow to: insert and delete rows, update rows currently displayed in the form, query relations. Forms can also access several relations (master-detail).
- There are interpreters which can execute such form definitions on several hardware platforms (Forms Runtime).
- Oracle also has a forms interpreter written as a Java applet which allows you to deploy Forms Applications via the web.

  You need to purchase the Developer Server in addition to Oracle Developer if you want to do this.

Oracle Developer (3)

- Reports created with the Report Builder can print the output as HTML and PDF.
- So these applications can also be deployed via the web (again you need the Developer Server).
- In the same way, graphics (e.g. tort diagrams, bar charts) created with the Graphics Builder can be embedded into HTML pages.