LRAP-190

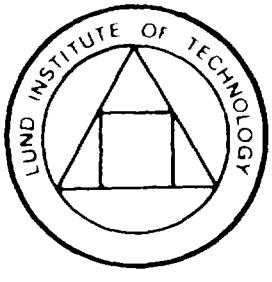
Lund Reports on Atomic Physics

Design of a WWW database server for Atomic Spectroscopy Data

Diploma work by
Alex Contis

Supervisor
Sven Huldt

Lund Reports on Atomic Physics, LRAP-190
Lund, December 1995



Contents

1. Introduction

The department of Atomic Spectroscopy at Lund University produces large amounts of experimental data on energy levels and emission lines for various atomic systems. Most of this data is published in journals. In order to make this data easily available to users outside, this diploma work has been initiated. The purpose of this diploma work is to organise the data in a database and to make the database available on the Internet.

2. Internet , World Wide Web and Databases

The Internet is a collection of more than 10,000 interconnected computer networks around the world that make it possible to share information almost instantly. The Internet allows the users of the system to collaborate easily and quickly through messages, discussion groups, and conferences.

The fastest growing and most exciting part of the Internet is the World Wide Web (WWW, or simply, the Web). Practically the Web is a collection of documents scattered world-wide and connected by electronic references, hypermedia links. Web documents can be text, graphics, video, audio or any digital media. The Web is based on client/server architecture or simply put, a server holds documents which a client can request. On the Web, the client is called a browser, a Web browser takes a user's request, retrieves the document from the Web server, interprets the contents, and presents it to the user. A graphical Web browser, such as Netscape, can display text and GIF graphics, it can also be configured to automatically call external viewers, such as Lview and MPEGPlay, or other applications to properly display specific types of documents.

The other half of the client/server relationship is the server. A server holds the documents and shows them to a Web client when requested. Web servers not only return text and multimedia

documents to browsers, they can also execute special programs that enable them to act as gateways to other applications or information resources. These programs are called Common Gateway Interface (CGI) programs.

A database is a way to store information using a computer. It consists of a software part, the database program, and of a hardware part, the computer. A way to explain how a database works is to make a comparison between a database and a library. In a library the information about the books is stored in a card index. The database stores the information on a hard disk. The librarian is replaced by the computer which accesses, updates and manipulates the information stored.

3. Building the database

The database program used for this project is Watcom SQL. Watcom SQL is a relational database and it runs on both MS-DOS and MS-Windows. Structured Query Language (SQL) is a language used to query relational databases. It consists of various commands which can be put together to form SQL query strings. SQL was standardised in 1986 by the American National Standards Institute (ANSI). Since then, it has been formally adopted as an International Standard by the International Organisation for Standardisation (ISO) and the International Electrotechnical Commission (IEC).The first step in creating a database is designing it, planning the tables needed and determine how the tables are related to each other.

3.1 Designing tables

The basic structure of a relational database is the table, consisting of rows and columns. Data definition includes declaring the name of each table to be included in a database, the names and data types of all columns of each table, constraints on the values in and among columns, and the granting of table manipulation privileges to prospective users. Tables can be accessed by inserting new rows, deleting or updating existing rows, or selecting rows that satisfy a given search condition for the output. Tables can be manipulated to produce new tables by Cartesian

products, unions, intersections, joins on matching columns, or projections on given columns. For example, two tables which have a common column can be joined together on the basis of common values in that column. That is, a given row from the first table will join a given row in the second table to produce a new, wider row, if and only if the two rows in question have a common value.

When the database is initialised the file for storing the database is created and the required system tables are built. The system tables describe the structure of the tables and the columns in the database.

3.2 Creating tables

New tables are created using the SQL command CREATE TABLE. The tables used to store the data of Ag II and Mn II were created using the following commands:

```
create table MnII ( Intensity integer NOT NULL,
                    Wavelength float NOT NULL,
                    Wavenumber float NOT NULL,
                    l_cfg char(8) NOT NULL,
                    l_level char(8) NOT NULL,
                    u_cfg char(8) NOT NULL,
                    u_level char(8) NOT NULL,
                    "References" char(6) NOT NULL);

create table AgIIel (config char(6) NOT NULL,
                     term char(6) NOT NULL,
                     jvalue float NOT NULL,
                     energy double NOT NULL,
                     parity int NOT NULL,
                     "References" char(6) NOT NULL);
```

Thus for transition energy the following data is stored: Intensity, Wavelength, Wavenumber, lower configuration of the transition (l_cfg), lower term of the transition (l_level), upper configuration of the transition (u_cfg), upper term of the transition (u_level) and references.

For the data of the energy levels configuration, term, J value, energy, parity and references is stored in the database.

The tables which store the transition energy data are named:

element+ionisation stage

Those storing data of energy levels are named:

element+ionisation stage+el

Compare with the definition of the tables of AgIIel and MnII, AgIIel is used to store data of energy levels.

Thus for every element and ionisation stage a table must be created. Having tables in the database is not enough, data are needed in order to make it complete. The main idea is to read data into the database from an existing file, this is done with the SQL command INPUT. The general form of the INPUT command is:

INPUT INTO t1 *FROM* file *FORMAT* ascii *DELIMITED BY* delimiter;

"t1" denotes the name of the table in which the data is to be read, "file" denotes the name of the text file containing the data, "ascii" denotes that the input lines are assumed to be ASCII characters, "delimiter" denotes the character which separates the data elements on a row.

The commands used to read data into the tables of MnII and AgIIel are:

INPUT INTO MnII ("FTS intensity", intensity, Wavelength , Wavenumber, Difference, "Ionisation stage", l_cfg , l_level, u_cfg, u_level,"references") *FROM* c:\wsql40\data\mnii.lin

FORMAT ascii *DELIMITED BY* ',';

and

INPUT INTO AgIIel (config, term, jvalue , energy, parity,"references") *FROM* c:\wsql40\data\agii.lev *FORMAT* ascii *DELIMITED BY* ',' ;

The condition required to make a successful input of data into a table is that the number of elements in a row should be the same as the number of columns in the table. A command file (Appendix A) can be created to store the commands describing the database, this file allows the user to recreate the database and it also provides some documentation of the structure of the database.

4. Common Gateway Interface

The WWW server software used on this project is called WebSite. This is a 32-bit application which requires Windows NT or Windows 95. Website supports symmetric multiprocessing which means it can run on a computer with multiple microprocessors. In short, server performance is limited only by the hardware being used. To improve performance one needs to improve the hardware. To make it possible for external user to launch applications on the Web server there is a interface called the CGI. The CGI contains a set of rules of how programs, or gateways must interact with the server. Gateways are programs which handle information requests and return the appropriate result to the server. The server then passes the results to the external user. The CGI program transforms the information, which is originally unreadable to the client, to something readable and thus acts as a gateway between the server and the database. To submit data to the server an HTML fill-out form is used. The information entered in the form is assembled by the browser in a series of name/value pairs. Every input element, text-entry field, pop up menu or checkbox selection in the form is assigned a name and a value. The assigned value is associated with the data entered or the choice made by the user, in case of a checkbox. These name/value pairs are sent to the server which uses them as parameters for the gateway program (figure 1).

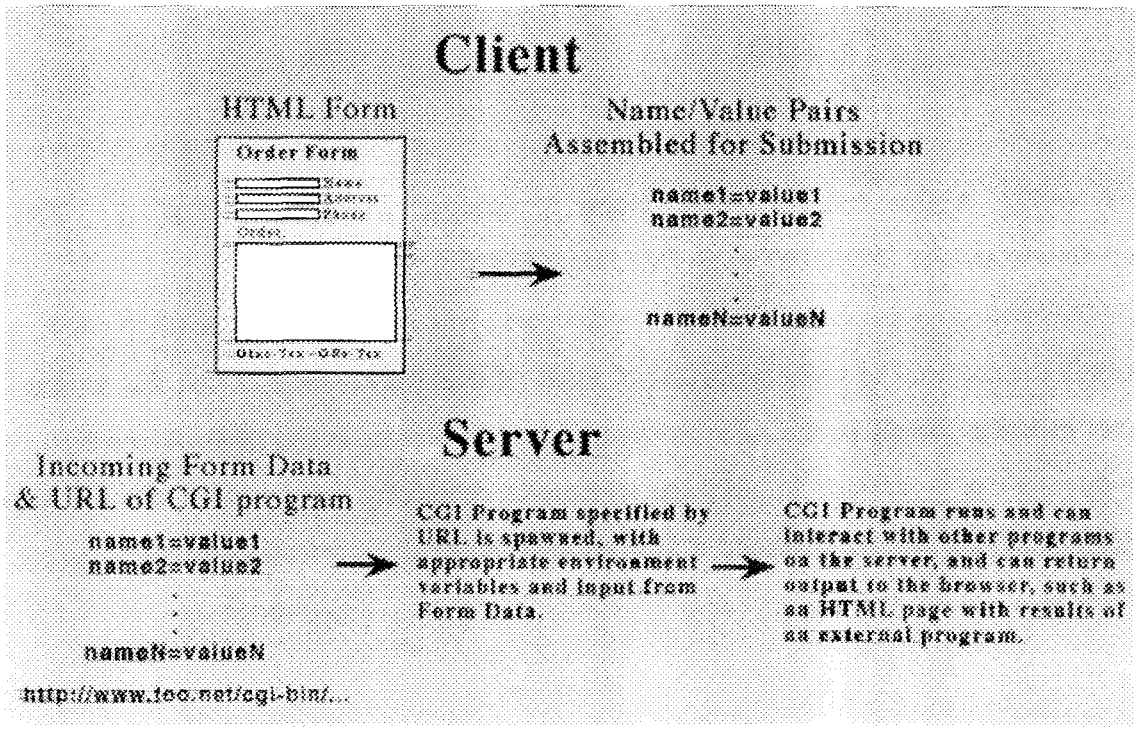


figure 1

The interaction between the client, the server, the gateway and the database program can be illustrated as shown in the figure 2.

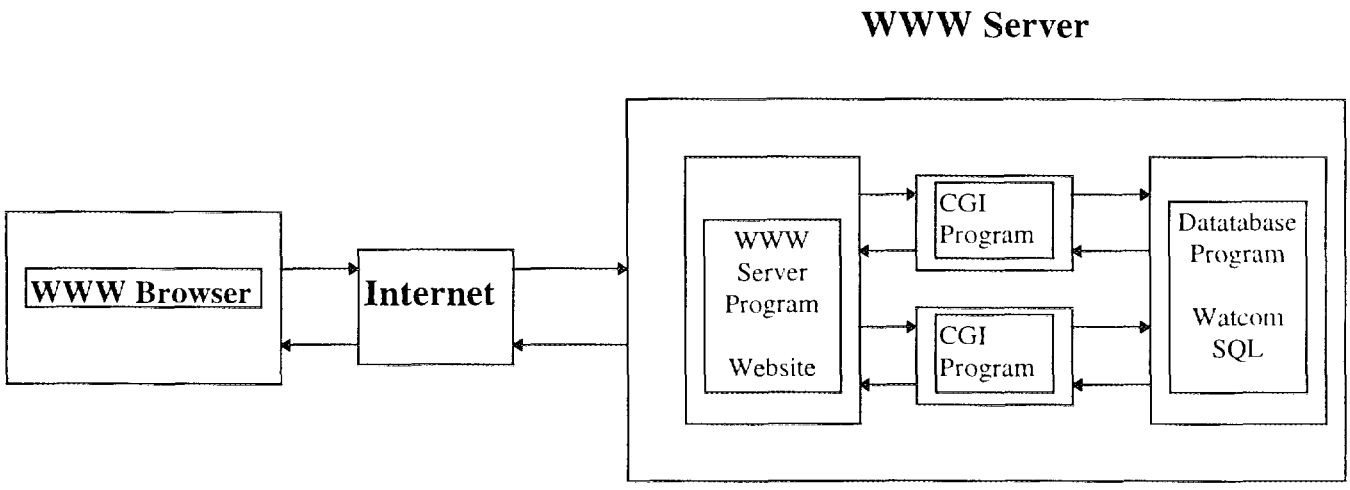


figure 2

5. Programming the Gateway

Gateways are basically executable programs which can be run by themselves. They can be written in any language which produces an executable file. Some of the languages which can be used are:

* C/C++
* Perl
* Visual Basic

The programming language used in this project is C. Since we want to make queries on both energy levels and transition energies data two gateway programs are needed. The source code for each program is divided in two parts which are compiled separately and then linked together. The first part of the source code is common for both gateway programs (mainwin.c, see Appendix B), and is specific for Microsoft Windows and it contains what is necessary to define and open a window. The second part consists of cur5.c and cur6.c (see Appendix C and D), and they are responsible for the interaction with the database, i.e. they open, search, store the results of a query and close the database. cur5.c is the source code for the gateway program used to make queries on transition energy data and cur6.c is the source for the gateway program for energy level data query. They are identical except for those parts which are responsible for creating the SQL command string that is used by the database to perform the query.

The server launches the gateway program using the WinExec() function. Briefly described the Win Exec() function executes a Windows or a non-Windows application with command line parameters. When invoked from the HTML form the gateway program is run by the sever with a command line of the following structure:

gateway_program cgi-path content-file output-file url-args

The most interesting parts of the command line for our purpose is the "cgi-path". "cgi-path" gives the complete path to the CGI data file (Appendix E), which is a text file created by the

server and it contains information that is necessary for the gateway program to perform properly. There are two parameters in the CGI data file that present special interest to us. The first is the QUERY STRING which holds information submitted by the client to the server, for a transition energy query it looks like:

Query String=element=Mn&ionstage=II&unit=%C5&start=0&stop=10000&order=wavelength%2Fwavenumber

and for an energy level query:

Query String=element=ag&ionstage=ii&parity=Both

The second is the OUTPUT FILE which tells the gateway where to put the results. The gateway program reads the data associated with the QUERY STRING of the CGI data file and then it assembles an SQL query and opens the data base. Mostly we get two kinds of result from a query, either we get data as an answer to our request or a system messages. These results are put by the gateway in the file indicated by OUTPUT FILE.

WinMain serves as an entry point for the execution of every Windows application. At the beginning of WinMain the content of the command line is read and the parameters are stored into different strings. The first argument gives the path to the CGI data file and the second gives the path to the Output File. The values of QUERY STRING and OUTPUT FILE are retrieved from the CGI data file and are sent to read_request and mk_file. mk_file creates the file which stores the results of a query.

The definition of the window parameters is done in the function InitApplication. The InitInstance creates and places the window on the desktop. The connection to the database is opened by calling WSQLEX_Init. Windows maintains a message queue for each currently running program. When an input event occurs, Windows translates the event into a message that is placed in the message queue of the program. The messages are retrieved from the queue by executing a block of codes known as the message loop:

```
while( GetMessage( &msg, 0, 0, 0 ) )
  {
            TranslateMessage( &msg );
            DispatchMessage( &msg );
  }
  return( msg.wParam );
```

The messages are sent to the function MainWinProc where they are processed and then the program then returns to the message loop. WSQLEX_Process_Command manages the search and storage of the data by calling load_file.

As mentioned before the browser codes the input from an HTML form into name/value pairs. These name/value pairs are used by the gateway to build the SQL command string. Since we can perform two kinds of queries, on energy levels and transition energies, we require two SQL command strings. For an energy level query the string is:

SELECT Config, Term, Energy, Parity, References *FROM* "table" *ORDER BY* Parity *DESC*

and for an emission line query:

SELECT Intensity, "Wavelength or Wavenumber", l_cfg, l_level, u_cfg, u_level, References FROM "table" WHERE "Wavelength or Wavenumber" *BETWEEN* "start" and "end" *ORDER BY* "name of the column" *DESC*

The function read_request reads the name/value pairs from the "Query String" and stores the value in a string. These strings are used by the function make_request_string to form the SQL command string. fetch_row gets the data from the database one row at a time and load_file stores it. When load_data is finished the program returns to WSQLEX_Process_Command which in turn gives the control back to MainWinProc. The only thing left for the program is to clean up by closing the database, with WSQLEX_Finnish and to close the window calling DestroyWindow.

6. Using the database

The address of the database on the WWW is http://sej4.fysik.lu.se. The first page presented to the user is the homepage of the database (figure 3).

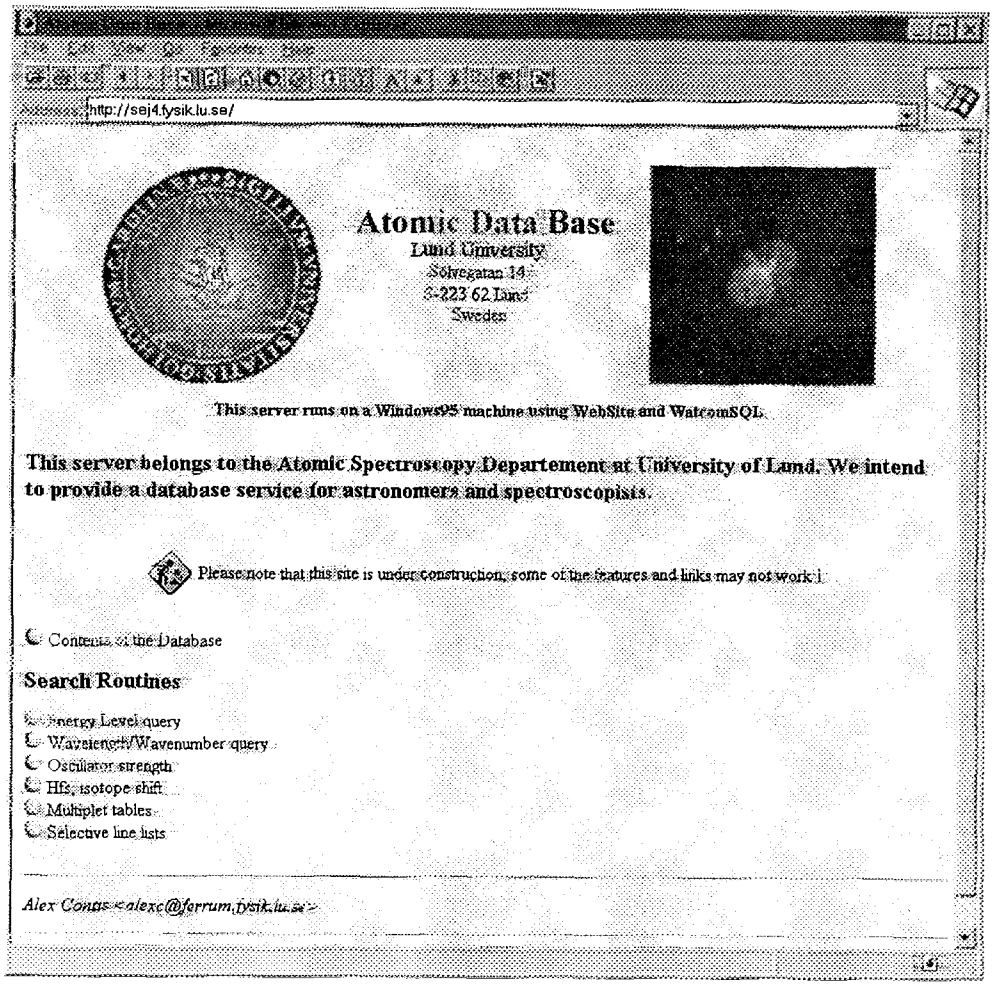


figure 3

There are three links that can be followed, first is the "Contents of the Database", second is "Energy level query" and third is "Wavelength/Wavenumber query". The "Contents of the Database" gives the user information about the stored data and is intended to make it easy to any one to use the database. By choosing "Contents of the Database" we get to a page (figure 4) with three links: "Energy Level query", "Wavelength/Wavenumber query" and "References".

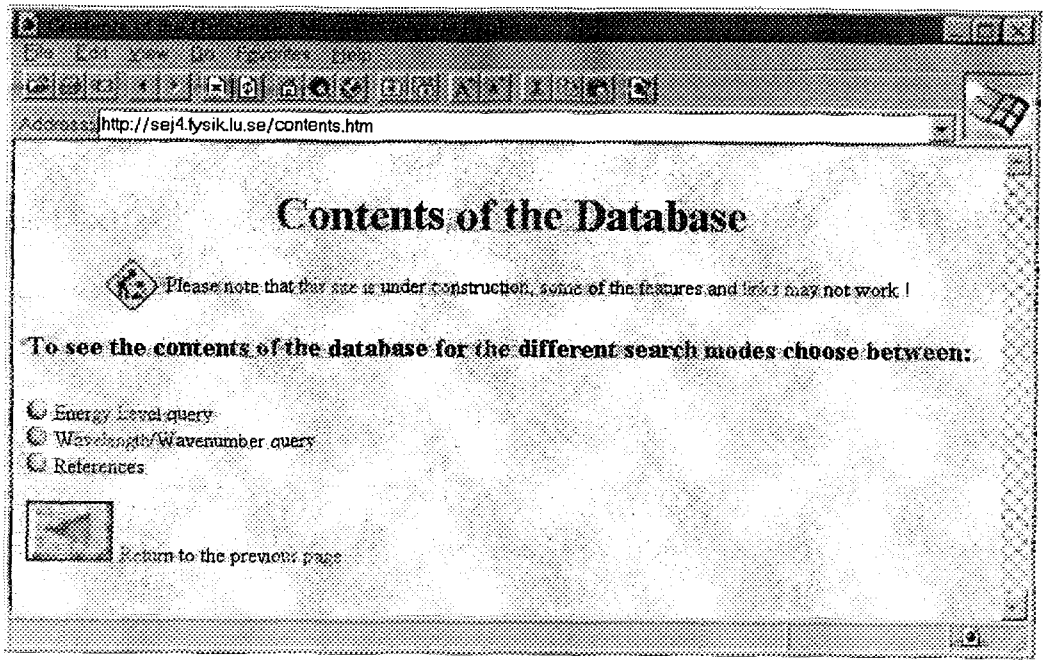


figure 4

"Energy Level query" shows the contents of the database for an Energy level, the elements and the ionisation stages are put into a table. "Wavelength/Wavenumber query" shows the contents of the database for a transition energy query. "References" gives the source of the data.

When the user chooses "Energy levels" from the homepage he/she is required to write several parameters into the query form. The parameters required are the element in question, the stage of ionisation (in Roman numbers) and the parity - default is both (odd and even). If the search is successful the result is presented in six columns in the following order: Config, Term, Level, J, Parity, and References.

The headings for the columns are:

Config: Electronic configuration.

Term: Term configuration.

Energy: Energy level in cm^{-1}.

J: J value.

Parity: Parity, 0 if odd, 1 if even.

References: Initials representing the source of the data.

By choosing from the homepage the link for a wavelength or wavenumber query the user is shown the periodical table of the elements (figure 5). Every element represents a link to an HTML form which can be used to enter the parameters for the query.

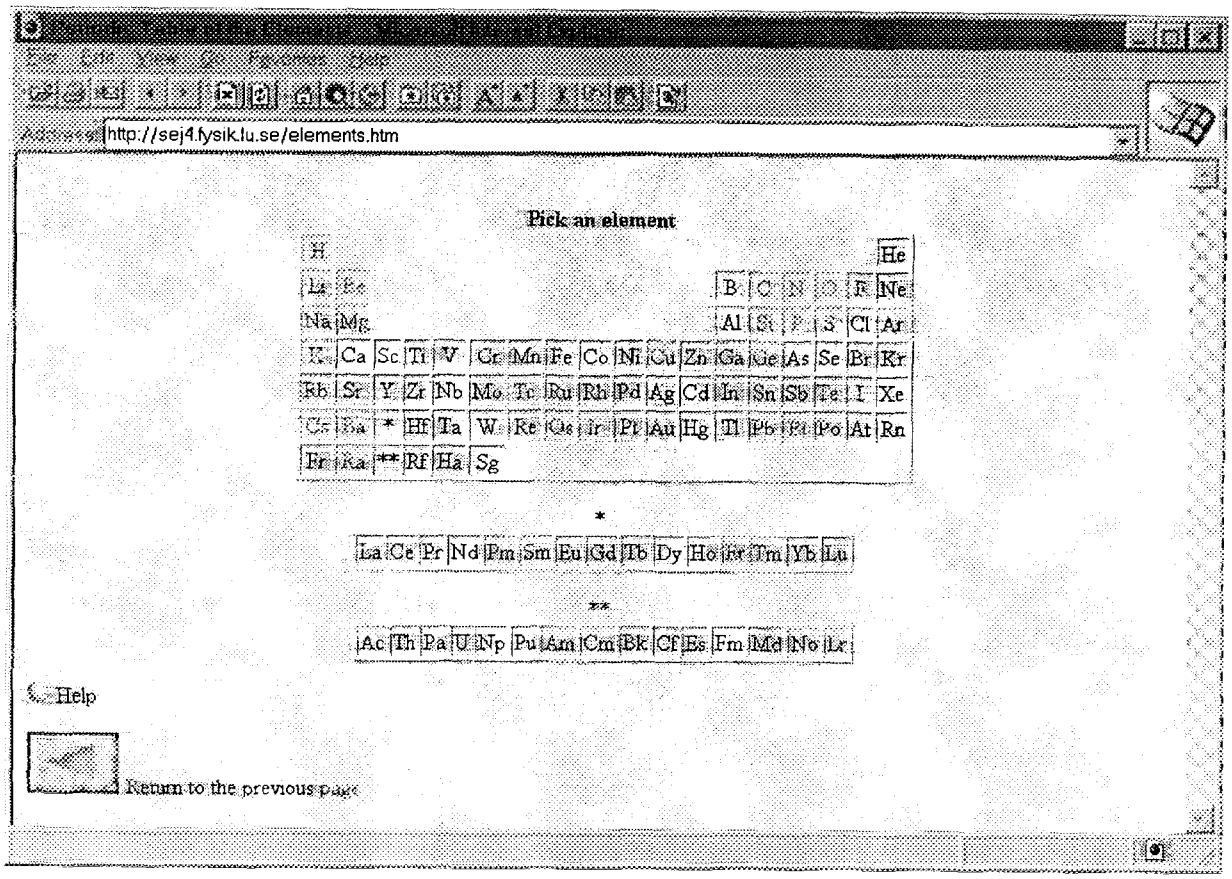


figure 5

For a search on wavelengths or wavenumbers the user is required to enter the following parameters into the query form. The ionisation stage must be given, (in Roman numbers), choosing the unit between Å or cm^{-1} automatically decides if the search is made in a wavelength or a wavenumber interval. Next the boundaries for the search must be entered in ascending order. The "order by" item selects the way in which the results are to be ordered. Selecting "wavenumber/wavelength" orders the data in the wavelength or wavenumber column in descending order. If "intensity" is selected the data will be arranged in descending order according to intensity.

If the search is successful the result is presented in six columns in the following order: intensity, wavelength or wavenumber, l_cfg, l_level, u_cfg, and u_level.

The headings for the columns are:

Intensity: Relative intensity.
Wavelength or Wavenumber: Observed wavelengths or wavenumbers. Wavelengths below 2000 Å are vacuum values.
l_cfg: Lower configuration of the transition.
l_level: Lower term of the transition.
u_cfg: Upper configuration of the transition.
u_level: Upper term of the transition.
References: Initials representing the source of the data.

7. Conclusion

To further develop this project it is important to decide what kind of data of the energy levels and transition energies are to be stored in the database. The database structure and the columns in the tables can be manipulated very easily. If new columns are introduced in the tables which store data on energy levels or emission lines, changes must be made on the text files which contain the data to be read in the database. New search routines can be implemented to make queries on oscillator strength data. The gateway program must be modified, the part which is responsible for assembling the QUERY STRING (make_request_string) must be rewritten.

References

Watcom SQL User's Guide 4th Edition, Watcom International Corporation Waterloo, Ontario, Canada 1994

C. Petzold, Programming Windows 3rd Edition, MICROSOFT PRESS, 1992

C. J. Date, An Introduction to Database Systems 6th Edition, ADDISON-WESLEY, 1994

H. Schildt, C the complete reference, Osborne McGRAW-HILL, 1987

Appendix A: Database Command File

```
create table Mnii (Intensity integer NOT NULL,
                   Wavelength float NOT NULL,
                   Wavenumber float NOT NULL,
                   l_cfg char(8) NOT NULL,
                   l_level char(8) NOT NULL,
                   u_cfg char(8) NOT NULL,
                   u_level char(8) NOT NULL,
                   "references" char(6) NOT NULL
);

create table Agiiel (config char(6) NOT NULL,
                     term char(6) NOT NULL,
                     jvalue float NOT NULL,
                     energy double NOT NULL,
                     parity int NOT NULL,
                     "references" char(6) NOT NULL
);

input into MnII ("FTS intensity", intensity, Wavelength , Wavenumber, Difference, "Ionisation stage", l_cfg ,
l_level, u_cfg, u_level,"references" )  from c:\wsql40\data\mnii.lin  format ascii delimited by ',';

input into AgIIel (config, term, jvalue , energy, parity,"references" )  from c:\wsql40\data\agii.lev  format ascii
delimited by ',' ;
```

Appendix B: mainwin.c

```
/* MAINWIN.C    Windows specific routines for all example programs
*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <malloc.h>
#include <conio.h>
#include "example.h"
#include <windows.h>



#define _exportkwd  __export
#define _exportkwd  _export


long FAR _exportkwd PASCAL MainWndProc( HWND, unsigned, unsigned, long );

static HANDLE       hInst;
static HWND         hWnd;
static int          yText;
static int          CharHeight;
static int          CharWidth;
static BOOL         WindowCreated = FALSE;
extern int                  StartedDB;
static LPSTR                Prompt;
static LPSTR        PromptString;
static int          StringLen;
int                         PageSize;
MSG                                     msg;
char                    input[150], input1[150];
char            cgidata[400], arg1[40], arg2[40], arg3[40], arg4[40], arg5[40];

extern int Displaytext( int col, char *fmt, ... )
{
  RECT            hWndRect;
  int           x;
  int                                   len;
  va_list         arg_ptr;
  char            buffer[ 100 ];
  HDC             hDC;

  va_start( arg_ptr, fmt );
  vsprintf( buffer, fmt, arg_ptr );
  va_end( arg_ptr );

  len = strlen( buffer );
  if( buffer[len - 1] == '\n' ) {
            if( col == 0 && len == 1 ) return FALSE;
            buffer[len - 1] = '\0';
  }
```

```
    hDC = GetDC( hWnd );
    GetClientRect( hWnd, &hWndRect );
    x = col * CharWidth;
    if( x == 0 ) {
    if( yText + CharHeight > hWndRect.bottom - CharHeight ) {
                        ScrollWindow( hWnd, 0, -CharHeight, NULL, NULL );
                        UpdateWindow( hWnd );
                    } else {
                        yText += CharHeight;
                    }
    }
    OemToAnsi( buffer, buffer );
    TextOut( hDC, x, yText, buffer, strlen( buffer ) );
    ReleaseDC( hWnd, hDC );
    return( TRUE );
}

BOOL FAR _exportkwd PASCAL get_string_rtn( HWND hDlg, unsigned message,
                                                                    unsigned wParam, long
lParam )
{
    lParam = lParam;

    switch( message ) {
    case WM_INITDIALOG:
                    SetWindowText( hDlg, (LPSTR) Prompt );
                    return( TRUE );

    case WM_COMMAND:
                    switch( LOWORD( wParam ) ) {
                    case IDOK:
                        GetDlgItemText( hDlg, IDE_STRING_EDIT,
                                                (LPSTR) PromptString, StringLen );

                    case IDCANCEL:
                        EndDialog( hDlg, TRUE );
                    }
    }
    return( FALSE );
}

static void prompt_for_string( LPSTR prompt, LPSTR buff, int len )
{
    FARPROC         proc;

    buff[0] = '\0';
    Prompt = prompt;
    PromptString = buff;
    StringLen = len;
    proc = MakeProcInstance( (FARPROC)get_string_rtn, hInst );
    DialogBox( hInst, "DLG_PROMPT_STRING", hWnd, proc );
    FreeProcInstance( proc );
}

extern void Getvalue( char *prompt, char *buff, int len)
{
    prompt_for_string( (LPSTR) prompt, (LPSTR) buff, len );
}
```

```
BOOL InitApplication( HANDLE hInstance )
{
  WNDCLASS        wc;

  wc.style = 0;
  wc.lpfnWndProc = (WNDPROC) MainWndProc;
  wc.cbClsExtra = 0;
  wc.cbWndExtra = 0;
  wc.hInstance = hInstance;
  wc.hIcon = LoadIcon( 0, IDI_APPLICATION );
  wc.hCursor = LoadCursor( 0, IDC_ARROW );
  wc.hbrBackground = GetStockObject( WHITE_BRUSH );
  wc.lpszMenuName = "ExampleMenu";
  wc.lpszClassName = "ExampleWClass";
  return( RegisterClass( &wc ) );
}

BOOL InitInstance( HANDLE hInstance, int nCmdShow )
{

  hInst = hInstance;
  hWnd = CreateWindow( "ExampleWClass", "Example", WS_OVERLAPPEDWINDOW,
                                          10, 50, 300, 500, 0, 0, hInstance, 0 );
  if( !hWnd ) {
            return( FALSE );
  }
  ShowWindow( hWnd, nCmdShow );
  UpdateWindow( hWnd );
  return( TRUE );
}

extern void Display_systemerror( char *message )
{
  MessageBox( hWnd, message, "System Error", MB_OK );
}

int PASCAL WinMain( HANDLE hInstance, HANDLE hPrevInstance,
                                LPSTR lpCmdLine, int nCmdShow )
{
  MSG           msg;

lstrcpy( cgidata, lpCmdLine );
sscanf(  cgidata,"%s%s%s%s%s%s", arg1, arg2, arg3, arg4, arg5 );

GetPrivateProfileString("CGI","Query String"," ", input, 140, arg1 );
read_request( input );

GetPrivateProfileString("System","Output File"," ", input1, 140, arg1 );
mk_file( input1 );

  if( !hPrevInstance ) {
            if( !InitApplication( hInstance ) ) {
              return( FALSE );
            }
  }
  if( !InitInstance( hInstance, nCmdShow ) ) {
            return( FALSE );
```

```
    }
    if( !WSQLEX_Init() ) {
                return( FALSE );
    }

    SendMessage( hWnd, WM_KEYDOWN, 'M', 0 );

    while( GetMessage( &msg, 0, 0, 0 ) ) {
                TranslateMessage( &msg );
                DispatchMessage( &msg );
    }
    return( msg.wParam );
}


long FAR _exportkwd PASCAL MainWndProc( HWND hWnd, unsigned message,
                                                                    unsigned wParam, long
lParam )
{
    HDC             hDC;
    TEXTMETRIC      tm;
    PAINTSTRUCT     ps;

    switch( message ) {
    case WM_KEYDOWN:

                    WSQLEX_Process_Command( wParam );
                    WSQLEX_Finish();
                    DestroyWindow( hWnd );
                    break;


    case WM_CREATE:
                hDC = GetDC( hWnd );
                GetTextMetrics( hDC, &tm );
                CharHeight = tm.tmHeight + tm.tmExternalLeading;
                CharWidth = tm.tmAveCharWidth;
                ReleaseDC( hWnd, hDC );
                break;

    case WM_SIZE:
                yText = 0;
                PageSize = HIWORD( lParam ) / CharHeight - 2;
                if( PageSize <= 0 ) {
                   PageSize = 1;
                }
                if( WindowCreated ) {
                   InvalidateRect( hWnd, NULL, TRUE );
                   UpdateWindow( hWnd );
                }
                else {
                   WindowCreated = TRUE;
                }
                break;

    case WM_PAINT:
                BeginPaint( hWnd, &ps );
```

```
                EndPaint( hWnd, &ps );
                break;

    case WM_DESTROY:
                PostQuitMessage( 0 );
                break;

    default:
                return( DefWindowProc( hWnd, message, wParam, lParam ) );
    }
    return( 0 );
}
```

Appendix C: cur5.c

```
#define _SQL_OS_WINDOWS
#define TRUE   1
#define FALSE  0

#include "stdlib.h"
#include "string.h"
#include <stdio.h>
#include <ctype.h>
#include "sqldef.h"
#include "sqlca.h"
#include "sqlda.h"



extern SQLCA      sqlca;
extern SQLCA _fd_ *sqlcaptr;
extern short int _ESQL_Version4_;
extern short int _ESQL_OS_WINDOWS_;


extern int     Displaytext( int, char*, ... );
extern void    Display_systemerror( char * );
extern char    arg1, arg2, arg3, arg4, arg5;
extern int     PageSize;

int    numind=0, valind=0,index=0,vindex=0,i=0,ch, ii=1;
char   name[10][20], value[7][25],str[200],*pp,*of, slask[300];
FILE   *fd,*outfile;

static struct {
      char       _sqlfar *userid;
      char       _sqlfar *password;
      short int  num;
      SQLCA      _fd_ * _sqlfar *sqlca;
      short int  _sqlfar *sqlpp_version;
      short int  _sqlfar *sqlpp_os;
} __SQLV_cur4_1 = { SQLFARNULL, SQLFARNULL, 0, &sqlcaptr, &_ESQL_Version4_,
&_ESQL_OS_WINDOWS_ };

static char  __SQLV_cur4_2[] = "DBA";
static char  __SQLV_cur4_3[] = "SQL";
char  __SQLV_cur4_4[250];
static char        *__SQL_ProgName = {"cur4"};
static a_sql_statement_number  __SQLV_cur4_5 = 0;
static char  __SQLV_cur4_6[] = "C1";

static struct {
   unsigned char sqldaid[8];
   long int sqldabc;
   short int sqln;
   short int sqld;
   SQLDA_VARIABLE sqlvar[7];
} __SQLV_cur4_7
   = { {'S','Q','L','D','A',' ',' ',' '},
```

```
    sizeof(SQLDA)+(7-1)*sizeof(SQLDA_VARIABLE),
    7, 7, {
     { 500, 2, 0L, 0L, {0,0}},
     { 482, 4, 0L, 0L, {0,0}},
     { 460, 32767, 0L, 0L, {0,0}},
     { 460, 32767, 0L, 0L, {0,0}},
     { 460, 32767, 0L, 0L, {0,0}},
     { 460, 32767, 0L, 0L, {0,0}},
     { 460, 32767, 0L, 0L, {0,0}}
    } };



typedef struct a_spek {
    int         fwhm;
    float       wline;
    char        l_cfg[8];
    char        l_level[8];
    char        u_cfg[8];
    char        u_level[8];
    char        references[8];
} a_spek;


static void printSQLError()
{
    char            buffer[ 200 ];

    if( SQLCODE==SQLE_SYNTAX_ERROR || SQLCODE==SQLE_COLUMN_NOT_FOUND )
    {
    fprintf(outfile,"Wrong input data, please verify the entry fields\n");
    fprintf(outfile,"for the interval of the search");
    }
    else if( SQLCODE==SQLE_TABLE_NOT_FOUND )
    {
    fprintf(outfile,"No data available on %s %s\n", value[0],value[1]);
    }
    else
    {
    fprintf(outfile, "SQL error -- %s\n", sqlerror_message( &sqlca, buffer, sizeof( buffer ) ) );
    }
}


static int warning( char *msg )
{
    if( SQLCODE == SQLE_NOTFOUND)
    {
       if(ii)
       fprintf(outfile, "Not found - past count %ld -- %s\n", SQLCOUNT,msg );
       else
       return( TRUE );
    }
    else
    {
       fprintf(outfile, "Unexpected warning %ld -- %s\n", SQLCODE, msg );
    }
    return( TRUE );
```

```
}



static int connect()
{

    {
    dbpp_connect_40( (sqlcaptr), &__SQLV_cur4_1, __SQLV_cur4_2, __SQLV_cur4_3, SQLNULL,
SQLNULL, SQLNULL );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
    }

  return( TRUE );

}


static int release()
{

    {
    dbpp_rollback( (sqlcaptr), &__SQLV_cur4_1, 0 );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
    }


    {
    dbpp_disconnect( (sqlcaptr), &__SQLV_cur4_1, SQLNULL );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
    }

  db_fini( &sqlca );
  return( TRUE );
}


static int open_cursor()
{


    {
    {
    dbpp_prepare_into( (sqlcaptr), &__SQLV_cur4_1, SQLNULL,
_sql_ptrtypechk_(__SQL_ProgName,char), &__SQLV_cur4_5, __SQLV_cur4_4, (SQLDA _fd_ *)0, (SQLDA
_fd_ *)0, 2 );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
    }
    dbpp_open( (sqlcaptr), &__SQLV_cur4_1, __SQLV_cur4_6, SQLNULL,
_sql_ptrtypechk_(__SQL_ProgName,char), &__SQLV_cur4_5, (SQLDA _fd_ *)0, 0, 0, 1 );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
    }

  return( TRUE );
}
```

```
static int close_cursor()
{

    {
    dbpp_close( (sqlcaptr), &__SQLV_cur4_1, __SQLV_cur4_6 );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
    }

  return( TRUE );
}



static int fetch_row(

 int *fwhm,
 float *wline,
 char *l_cfg,
 char *l_level,
 char *u_cfg,
 char *u_level,
char *references
)


{

    {
((SQLDA  *)&__SQLV_cur4_7)->sqlvar[0].sqldata = (void _sqldafar *)((fwhm));
((SQLDA  *)&__SQLV_cur4_7)->sqlvar[1].sqldata = (void _sqldafar *)((wline));
((SQLDA  *)&__SQLV_cur4_7)->sqlvar[2].sqldata = (void _sqldafar *)((l_cfg));
((SQLDA  *)&__SQLV_cur4_7)->sqlvar[3].sqldata = (void _sqldafar *)((l_level));
((SQLDA  *)&__SQLV_cur4_7)->sqlvar[4].sqldata = (void _sqldafar *)((u_cfg));
((SQLDA  *)&__SQLV_cur4_7)->sqlvar[5].sqldata = (void _sqldafar *)((u_level));
((SQLDA  *)&__SQLV_cur4_7)->sqlvar[6].sqldata = (void _sqldafar *)((references));
    dbpp_fetch( (sqlcaptr), &__SQLV_cur4_1, __SQLV_cur4_6, 2, (long int)1, ((SQLDA
*)&__SQLV_cur4_7), 0, 0 );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
    }


  if( SQLCODE ) {
    warning( "Fetching" );
    return( FALSE );
  } else {
    return( TRUE );
  }
}



static int top()
{

    {
    dbpp_fetch( (sqlcaptr), &__SQLV_cur4_1, __SQLV_cur4_6, 1, (long int)0, (SQLDA _fd_ *)0, 0, 0 );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
```

```
    }

  return( TRUE );
}



extern int WSQLEX_Init()
{
  if( !db_init( &sqlca ) ) {
    Display_systemerror("Unable to initialize database interface\n" );
    return( FALSE );
  }
  if( !db_find_engine( &sqlca, NULL ) ) {
    Display_systemerror("Database Engine/Station not running\n" );
    return( FALSE );
  }
  if( !connect() ) {
    Display_systemerror("Could not connect\n" );
    return( FALSE );
  }
  if( !open_cursor() )
  {
    return( FALSE );
  }

return( TRUE );

}


void ff()

{
    if(index)
    value[vindex][valind++]=ch;
    else
    name[vindex][numind++]=ch;
}



make_request_string()

{

char s1[]={" select intensity,"};
char s2[14];
char s3[]={" ,l_cfg, l_level, u_cfg, u_level, \42references\42 from "};
char s4[14];
char s5[4];
char s6[]={" where "};
char s7[14];
char s8[]={" between "};
char s9[14];
char s10[]={" and  "};
char s11[14];
char s12[]={" order by "};
```

```
char s13[14];
char s14[]={" desc "};

   strcpy(s4,value[0]);
   strcpy(s5,value[1]);
   strcpy(s9,value[3]);
   strcpy(s11,value[4]);

   if(value[2][0]=='%')
   {
      strcpy(s2," wavelength");
      strcpy(s7," wavelength");
   }
   else
   {
      strcpy(s2," wavenumber");
      strcpy(s7," wavenumber");
   }

   if( value[5][0]=='w')
   strcpy(s13,s2);
   else
   strcpy(s13,value[5]);

   strcpy(__SQLV_cur4_4,s1);
   strcat(__SQLV_cur4_4,s2);
   strcat(__SQLV_cur4_4,s3);
   strcat(__SQLV_cur4_4,s4);
   strcat(__SQLV_cur4_4,s5);
   strcat(__SQLV_cur4_4,s6);
   strcat(__SQLV_cur4_4,s7);
   strcat(__SQLV_cur4_4,s8);
   strcat(__SQLV_cur4_4,s9);
   strcat(__SQLV_cur4_4,s10);
   strcat(__SQLV_cur4_4,s11);
   strcat(__SQLV_cur4_4,s12);
   strcat(__SQLV_cur4_4,s13);
   strcat(__SQLV_cur4_4,s14);
}


 int sgets()

{
   char cc;

   if((cc=*(pp++))=='\0')
   return(EOF);
   else return(cc);

}


void read_request( char sinput[150] )

{
pp=sinput;
strcpy(slask,sinput);
```

```
while((ch=sgets())!=EOF)
{
      if(isalnum(ch)  || ch==',' || ch=='.' || ch=='%' || ch=='-' || ch=='_') ff();
      else if(ch=='=')
      {
      index=1;valind=0;name[vindex][numind]='\0';
      }
      else if(ch=='&')
      {
      value[vindex][valind]='\0';
      index=0;numind=0;vindex++;
      }
}
value[vindex][valind]='\0';
make_request_string();

}


void mk_file( char sinput1[150] )

{

   if((outfile=fopen( sinput1,"w"))==NULL)
   {
      exit(-1);
   }
fprintf(outfile,"Content-type: text/plain%c%c",10,10);
}



extern load_file()
{
   a_spek        s;
   int          status;
   char          wline[12];


if(value[2][0]=='%')
   strcpy(wline,"Wavelength");
   else
   strcpy(wline,"Wavenumber");

   top();
   for(;; ) {
      status = fetch_row( &s.fwhm, &s.wline, s.l_cfg, s.l_level, s.u_cfg, s.u_level, s.references );

      if( status ) {
         if(ii){
            fprintf(outfile,"   Intensity %s   l_cfg  l_level  u_cfg   u_level   References\n\n",wline);
            ii=0;
         }
         fprintf(outfile,"%10i",s.fwhm);
         fprintf(outfile,"%14.5f",s.wline);
         fprintf(outfile, "%9s",s.l_cfg);
         fprintf(outfile, "%9s",s.l_level);
         fprintf(outfile, "%9s",s.u_cfg);
```

```
            fprintf(outfile, "%9s",s.u_level);
            fprintf(outfile, "%10s\n",s.references);

        } else {
            break;
        }

    }
}



extern void WSQLEX_Process_Command( int selection )
{
    switch( tolower( selection ) ) {


        case 'm':
                load_file();
                break;

        default:    Displaytext( 0, "Invalid command, press 'h' for help\n" );
    }
}



extern int WSQLEX_Finish()
{
    close_cursor();
    release();
    fclose(fd);
    fclose(outfile);
    return( TRUE );
}
```

Appendix D: cur6.c

```
#define _SQL_OS_WINDOWS
/* Energy levels */

#include "stdlib.h"
#include "string.h"
#include <stdio.h>
#include <ctype.h>
#include "sqldef.h"
#include "example.h"
#include "sqlda.h"
#include "sqlca.h"

extern SQLCA sqlca;
extern SQLCA _fd_ *sqlcaptr;
extern short int _ESQL_Version4_;
extern short int _ESQL_OS_WINDOWS_;


extern int     Displaytext( int, char*, ... );
extern void    Display_systemerror( char * );
extern char    arg1, arg2, arg3, arg4, arg5;
extern int     PageSize;

int    numind=0, valind=0,index=0,vindex=0,i=0,ch, ii=1;
char   name[10][20], value[6][25],str[200],*pp,*of;
FILE   *fd,*outfile, *fp;

static struct {
      char      _sqlfar *userid;
      char      _sqlfar *password;
      short int  num;
      SQLCA      _fd_ * _sqlfar *sqlca;
      short int  _sqlfar *sqlpp_version;
      short int  _sqlfar *sqlpp_os;
} __SQLV_cur6_1 = { SQLFARNULL, SQLFARNULL, 0, &sqlcaptr, &_ESQL_Version4_,
&_ESQL_OS_WINDOWS_ };
static char __SQLV_cur6_2[] = "DBA";
static char __SQLV_cur6_3[] = "SQL";
static char __SQLV_cur6_4[120];
static char       *__SQL_ProgName = {"cur6"};
static a_sql_statement_number __SQLV_cur6_5 = 0;
static char __SQLV_cur6_6[] = "C1";

static struct {
   unsigned char sqldaid[8];
   long int sqldabc;
   short int sqln;
   short int sqld;
   SQLDA_VARIABLE sqlvar[6];
} __SQLV_cur6_7
   = { {'S','Q','L','D','A',' ',' ',' '},
   sizeof(SQLDA)+(6-1)*sizeof(SQLDA_VARIABLE),
   6, 6, {
     { 460, 32767, 0L, 0L, {0,0}},
```

```
    { 460, 32767, 0L, 0L, {0,0}},
    { 482, 4, 0L, 0L, {0,0}},
    { 482, 4, 0L, 0L, {0,0}},
    { 500, 2, 0L, 0L, {0,0}},
    { 460, 32767, 0L, 0L, {0,0}}
   } };


typedef struct a_spek {
   char          config[10];
   char          term[6];
   float         jvalue;
   float         energy;
   int           parity;
   char                        references[8];
} a_spek;

static void printSQLError()
{
   char          buffer[ 200 ];

   if( SQLCODE==SQLE_SYNTAX_ERROR || SQLCODE==SQLE_COLUMN_NOT_FOUND )
   {
   fprintf(outfile,"Wrong input data, please verify the entry fields\n");
   fprintf(outfile,"for the interval of the search");
   }
   else if( SQLCODE==SQLE_TABLE_NOT_FOUND )
   {
   fprintf(outfile,"No data available on %s %s\n", value[0],value[1]);
   }
   else
   {
   fprintf(outfile, "SQL error -- %s\n", sqlerror_message( &sqlca, buffer, sizeof( buffer ) ) );
   }
}


static int warning( char *msg )
{
   if( SQLCODE == SQLE_NOTFOUND)
   {
      if(ii)
      fprintf(outfile, "Not found - past count %ld -- %s\n", SQLCOUNT,msg );
      else
      return( TRUE );
   }
   else
   {
      fprintf(outfile, "Unexpected warning %ld -- %s\n", SQLCODE, msg );
   }
   return( TRUE );
}



static int connect()
{
```

```
                {
                dbpp_connect_40( (sqlcaptr), &__SQLV_cur6_1, __SQLV_cur6_2, __SQLV_cur6_3,
SQLNULL, SQLNULL, SQLNULL );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
                }

   return( TRUE );

}

static int release()
{

                {
                dbpp_rollback( (sqlcaptr), &__SQLV_cur6_1, 0 );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
                }


                {
                dbpp_disconnect( (sqlcaptr), &__SQLV_cur6_1, SQLNULL );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
                }

   db_fini( &sqlca );
   return( TRUE );
}

static int open_cursor()
{

                {
                {
                dbpp_prepare_into( (sqlcaptr), &__SQLV_cur6_1, SQLNULL,
_sql_ptrtypechk_(__SQL_ProgName,char), &__SQLV_cur6_5, __SQLV_cur6_4, (SQLDA _fd_ *)0, (SQLDA
_fd_ *)0, 2 );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError();return( FALSE ); };
                }
                dbpp_open( (sqlcaptr), &__SQLV_cur6_1, __SQLV_cur6_6, SQLNULL,
_sql_ptrtypechk_(__SQL_ProgName,char), &__SQLV_cur6_5, (SQLDA _fd_ *)0, 0, 0, 1 );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError();return( FALSE ); };
                }

   return( TRUE );
}

static int close_cursor()
{

                {
                dbpp_close( (sqlcaptr), &__SQLV_cur6_1, __SQLV_cur6_6 );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
                }

   return( TRUE );
}

static int fetch_row( char *config, char *term, float *jvalue, float *energy, int *parity, char *references)
```

```
{

                {
((SQLDA  *)&__SQLV_cur6_7)->sqlvar[0].sqldata = (void _sqldafar *)((config));
((SQLDA  *)&__SQLV_cur6_7)->sqlvar[1].sqldata = (void _sqldafar *)((term));
((SQLDA  *)&__SQLV_cur6_7)->sqlvar[2].sqldata = (void _sqldafar *)((jvalue));
((SQLDA  *)&__SQLV_cur6_7)->sqlvar[3].sqldata = (void _sqldafar *)((energy));
((SQLDA  *)&__SQLV_cur6_7)->sqlvar[4].sqldata = (void _sqldafar *)((parity));
((SQLDA  *)&__SQLV_cur6_7)->sqlvar[5].sqldata = (void _sqldafar *)((references));
                dbpp_fetch( (sqlcaptr), &__SQLV_cur6_1, __SQLV_cur6_6, 2, (long int)1, ((SQLDA
*)&__SQLV_cur6_7), 0, 0 );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
                }


   if( SQLCODE ) {
      warning( "Fetching" );
      return( FALSE );
   } else {
      return( TRUE );
   }
}



static int top()
{

                {
                dbpp_fetch( (sqlcaptr), &__SQLV_cur6_1, __SQLV_cur6_6, 1, (long int)0, (SQLDA _fd_ *)0,
0, 0 );
if( (sqlcaptr)->sqlcode < 0 ) { printSQLError(); return( FALSE ); };
                }

   return( TRUE );
}



extern int WSQLEX_Init()
{
   if( !db_init( &sqlca ) ) {
      Display_systemerror(
         "Unable to initialize database interface\n" );
      return( FALSE );
   }
   if( !db_find_engine( &sqlca, NULL ) ) {
      Display_systemerror( "Database Engine/Station not running" );
      return( FALSE );
   }
   if( !connect() ) {
      Display_systemerror( "Could not connect" );
      return( FALSE );
   }

   if(!open_cursor())
   {
```

```
        return (FALSE);
    }

    return( TRUE );
}

void ff()

{
        if(index)
        value[vindex][valind++]=ch;
        else
        name[vindex][numind++]=ch;
}



make_request_string()

{

char s1[]={" select config, term, jvalue, energy, parity , \42references\42 from "};
char s2[]={"el where parity=1 "};
char s3[]={"el where parity=0 "};
char s4[]={"el order by parity desc "};

    strcpy(__SQLV_cur6_4,s1);

    strcat(__SQLV_cur6_4,value[0]);
    strcat(__SQLV_cur6_4,value[1]);

if( value[2][0] =='B' )
    strcat( __SQLV_cur6_4,s4 );
else if ( value[2][0]=='O' )
    strcat( __SQLV_cur6_4,s3 );
else strcat( __SQLV_cur6_4,s2 );

}


 int sgets()

{
    char cc;

    if((cc=*(pp++))=='\0')
    return(EOF);
    else return(cc);

}


void read_request(  char sinput[150] )

{
pp=sinput;

while((ch=sgets())!=EOF)
```

```
{
      if(isalnum(ch)  || ch==',' || ch=='.' || ch=='%' || ch=='-' || ch=='_') ff();
      else if(ch=='=')
      {
      index=1;valind=0;name[vindex][numind]='\0';
      }
      else if(ch=='&')
      {
      value[vindex][valind]='\0';
      index=0;numind=0;vindex++;
      }
}
value[vindex][valind]='\0';
make_request_string();

}



void mk_file( char sinput1[150]  )

{

   if((outfile=fopen( sinput1,"w"))==NULL)
   {
      exit(-1);
   }
 fprintf(outfile,"Content-type: text/plain%c%c",10,10);


}



static void load_file()
{
   a_spek        s;
   int           status;

   top();
   for(;; ) {
    status = fetch_row( s.config, s.term, &s.jvalue, &s.energy, &s.parity, s.references );
      if( status ) {
         if(ii){
            fprintf(outfile,"  Config Term    J     Energy      Parity References\n\n" );
            ii=0;
         }
         fprintf(outfile,"%8s",s.config);
         fprintf(outfile,"%7s",s.term);
         fprintf(outfile,"%8.1f",s.jvalue);
         fprintf(outfile,"%14.4f",s.energy);
         fprintf(outfile,"%10i",s.parity);
                  fprintf(outfile,"%10s\n",s.references);
      }
         else
       {
         break;
      }
```

```
    }
}


extern void WSQLEX_Process_Command( int selection )
{
    switch( tolower( selection ) ) {


        case 'm':     load_file();
                      break;

        default:      Displaytext( 0, "Invalid command, press 'h' for help\n" );
    }
}



extern int WSQLEX_Finish()
{
    close_cursor();
    release();
    fclose(fd);
    fclose(outfile);
    return( TRUE );
}
```

Appendix E: CGI data file

[CGI]
Request Protocol=HTTP/1.0
Request Method=GET
Executable Path=/cgi-win/cur5win.exe
Query String=element=Mn&ionstage=II&unit=%C5&start=0&stop=10000&order=
wavelength%2Fwavenumber
Server Software=WebSite/1.0c (demo version)
Server Name=sej4.fysik.lu.se
Server Port=80
Server Admin=alexc@ferrum.fysik.lu.se
CGI Version=CGI/1.2 (Win)
Remote Address=130.235.92.70
Referer=http://sej4.fysik.lu.se/table/form/mnform.htm
Authentication Method=Basic
Authentication Realm=Web Server

[System]
GMT Offset=3600
Debug Mode=No
Output File=c:\temp\c1ws.out

[Accept]
image/gif=Yes
image/x-xbitmap=Yes
image/jpeg=Yes
image/pjpeg=Yes
/=Yes

[Extra Headers]
Connection=Keep-Alive
User-Agent=Mozilla/2.0b2 (Windows; I; 16bit)
Host=sej4.fysik.lu.se