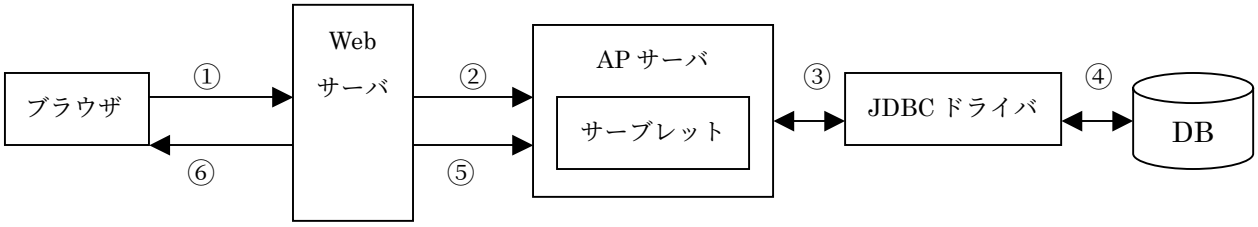
課題実験
「学生名簿検索システムの開発」

高橋研究室
4年　情報工学科　33 番
山道寿美恵
提出日：　2003．2．28（金）

[目的]

・ 1 年生から 4 年生の全学生と、5 年情報工学科の学生の、学籍番号、住所、氏名の検索システムを開発する。

[システムのしくみ]



①② ブラウザから Web サーバを通して、AP サーバのサーブレットを起動する（リクエスト）。
つまり、「学籍番号」、「氏名」、「住所」、「クラス」のどの項目から、どのようなキーワードで検索するかの情報をサーブレットに渡す。

③④ サーブレットはブラウザから渡されたデータをもとに、データベース（DB）にアクセスして、学生名簿のデータを検索して結果を得る（SQL 文を使う）。しかし、サーブレットは直接 DB と対話をすることができないので、JDBC ドライバを通じて DB にアクセスする。

⑤⑥ そして検索結果をサーブレットが Web サーバを通じてブラウザに返す（リスポンス）。

初期画面の操作

1． 入力画面から、「学籍番号、氏名、住所、クラス」のどれから検索するか、チェックマークをつける。

2． 「学籍番号、氏名、住所」のどれかにチェックマークをつけた場合、検索したいキーワードを入力する。

3． 「クラス」にチェックマークをつけた場合、検索したいクラスの「学年」、「学科」にチェックマークをつける。

4． 「送信」ボタンを押すと、検索したいキーワードにヒットしたか、チェックマークをつけたクラスの「学籍番号、氏名、住所」のデータが表示される。

5． 入力画面で「クリア」ボタンを押すと、それまで入力していたものが消えて、初期の状態に戻る。

検索結果画面

- 検索結果が表になって表示される。
- 表の後に、ヒットした件数を表示する。
- 何もヒットしなかった場合は、表の見出しの部分だけと、0 件ヒットしたことが表示される。

[データベースとは]

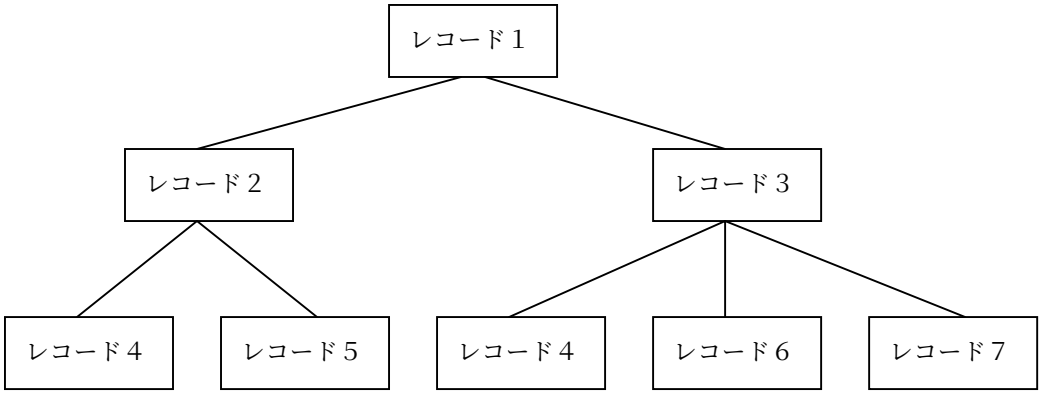
データの基地の意味であり、関連するデータをまとめて体系的、構造的に整理して一元管理し、多くの利用者が使えるようにしたものである。

データベースはそのデータモデルによって、階層型、構造型、関係の 3 つの種類がある。

- 階層型データベース

  階層型データモデルとして木構造を使ったもの。レコードとレコードが親子関係にある構造である。なので、上位のレコードに対して複数の下位レコードが存在する。

  複数の上位レコードに所属する下位レコードは、重複して登録しなくてはいけない。

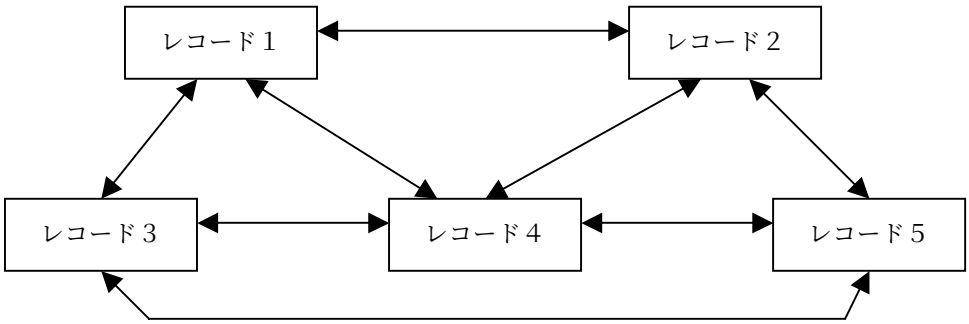


レコード４がレコード２と３に重複してある。

- 構造型データベース

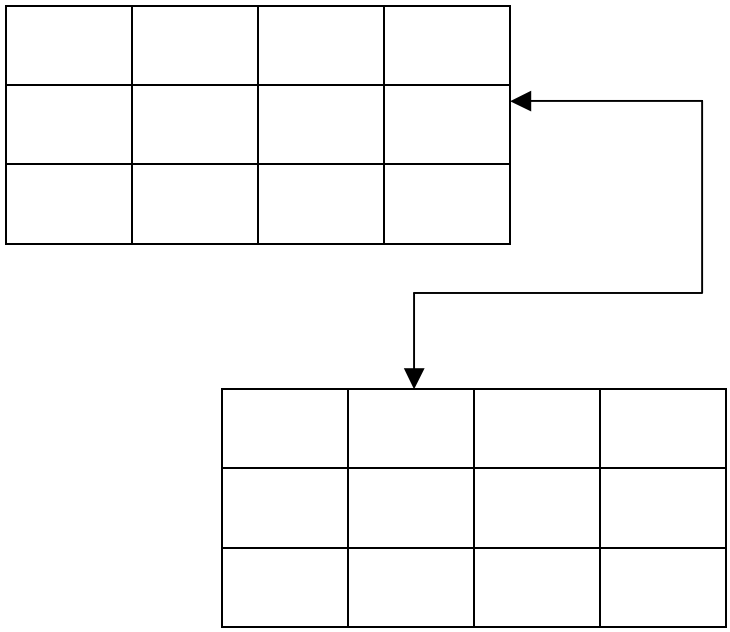
  構造型データモデルとして網構造を使ったデータベースで、ネットワーク型データベースとも呼ばれる。これは、上位レコードに対する下位レコードが複数ありうる。また、下位レコードに対する上位レコードも複数ありうる。つまり、レコードとレコードの関係は網のようになり、どのレコードも対応付けられる。

  しかし、レコードを検索する入り口、経路も複雑になるので、レコードの追加、削除が難しくなる。



・ 関係データベース

　リレーショナルデータベースともいい、２次元の表形式でデータを表す。ひょうの１行がレコードに、１列がフィールドに該当する。データの検索、更新、追加、削除などの操作が簡単である。データの独立性が高いことから、普及している。



今回はリレーショナルデータベースを作成する。

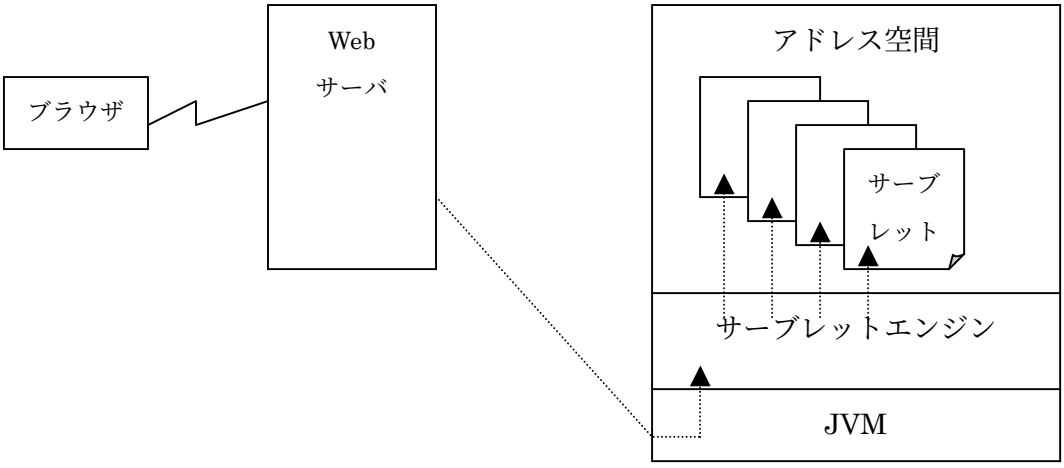
今回作成したテーブルは以下のとおりである。

No(学籍番号)	Shimei（氏名）	Juusho（住所）
02101	秋葉　桂輔	北海道函館市･･･
02102	家保　充里	北海道茅部郡･･･
02103	石井　貴幸	北海道函館市･･･
・	・	・
・	・	・
・	・	・

学籍番号を主キーとする。

[サーブレットとは]

- JAVA で作成し動作する、サーバサイドプログラムのこと。
- ブラウザからリクエストがあると、JVM（JAVA の動作環境）上で、「インスタンス化」される。
- 「インスタンス化」とは、JAVA プログラムがメモリ空間上にロードされ、いつでも稼動できるように JVM の管理下に置かれること。
- 一度「インスタンス化」されると、サーブレットが終了しても、「インスタンス化」されたまま、AP サーバ上に残る。つまり、一度「インスタンス化」されてしまえば、次回からはロードといった余分な作業を行わずに、すばやく動かすことができる。
- 複数のブラウザからの要求にたいして、同時実行（マルチスレッド）させて稼動させることができる。

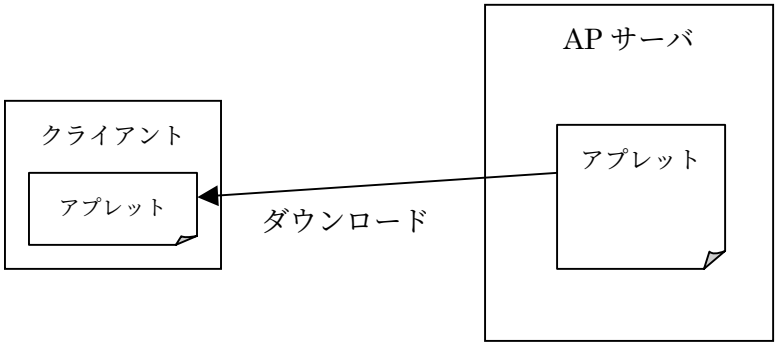


サーブレットと似た技術

・ JAVA アプレット

サーバ側にある JAVA アプレット自体が、ブラウザの存在するクライアント側にダウンロードして実行する。つまり、クライアントサイドプログラム。

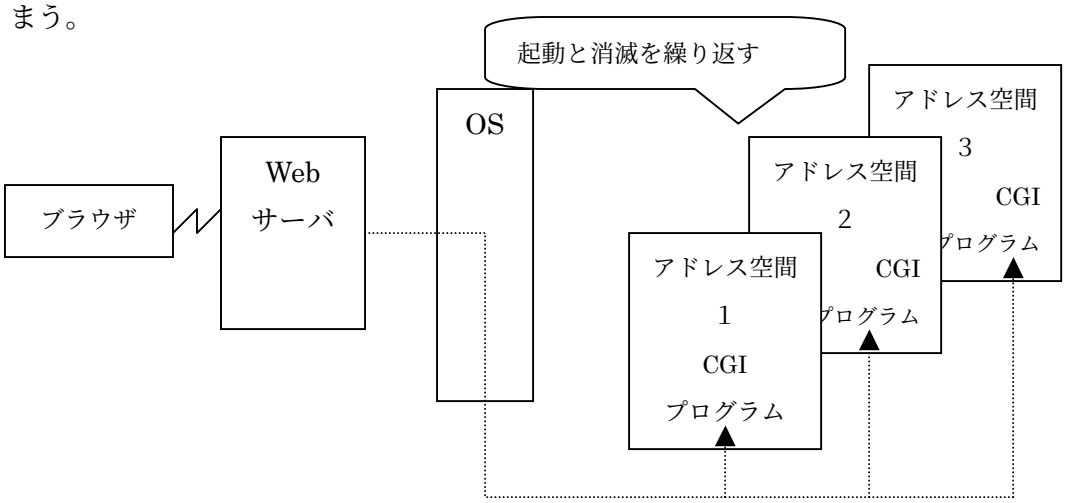
クライアント側の JVM によって、動作が制限されてしまう欠点がある。



・ CGI

リクエストを受けると、Web サーバと CGI の間に直接 OS が割り込んできてプロセスを作り出し、その中でプログラムを起動する。

処理が終了すると、プロセスをメモリ上から消滅させる。つまりリクエストの数だけプロセスの作成を繰り返す。これは OS に非常に負荷をかけてしまう。



[JDBC とは]

SQL 文を実行するための JAVA　API である。

データベースへの接続や切断、JAVA プログラムからの SQL 文の送信、SQL 文実行結果の処理など行うことができる。

ただし、JAVA プログラムが利用するリレーショナルデータベースは JDBC と接続するための JDBC ドライバが必要である。

JAVA プログラムはこの JDBC と JDBC ドライバを使って、リレーショナルデータベースを利用する。

[SQL とは]

リレーショナルデータベースを操作するための言語のこと。

Structured Query Language の略で、「構造化された問い合わせ言語」という意味。

SQL には insert、select、delete などの命令がある。実験で使用したのは select 文である。

＜実験で使用した命令＞

select * from {テーブル名}

where {フィールド名} like '%{検索するキーワード}%';

指定されたテーブルの指定されたフィールドから、指定されたキーワードが含まれている行を検索するという命令。

[動作環境・開発環境]

AP サーバ

- OS：RedHatLinux7.2(Kerne2.4.7)
- CPU：K6/300MHz
- Memory：128MB

DB サーバ

- OS：RedHatLinux7.2(Kerne2.4.7)
- DBMS：PostgreSQL7.2.1
- CPU：Celeron/333MHz
- Memory：64MB

[実験のおおまかな流れ]

1．データベース作成。

2．サーブレットプログラムの作成。

3．HTML で初期画面の作成。

[実験方法]

１．　データベース作成

（１） エクセルでデータ入力。

（２） データベースの作成。

１．DOS プロンプトから telnet を開く。

telnet 10.30.2.29

２．ログインする。

Postgres

（パスワード）

３．データベース作成。

Createdb ｛データベース名｝；

実験では、

createdb yamamichi;

（３） テーブルの作成。

１．自分のデータベースへログインする。

Psql –U kadai ｛データベース名｝

実験では、

Psql-U kadai gakuseimeibo

２．テーブルを作成する。

Create table ｛テーブル名｝（{主キー名} ｛型｝ primary key, ｛フィールド名｝ ｛型｝ ,･･･）；

実験では、

create table gakuseimeibo (no char(5) primary key,shimei varchar(20),juusho varchar(200));

（４） データをファイルからテーブルにコピーする。

１．エクセルのファイルをタブ区切りのテキストファイルとして保存しなおす。

２．データの入っているファイルをデータベースのあるコンピュータに ftp で送る。

３．文字コードを SJIS から EUC コードに変換する。

¥usr¥bin¥nkf -e ｛変換前のファイル名｝ > ｛変換後のファイル名｝

実験では、

¥usr¥bin¥nkf –e gakusei1>gakusei1

４．コピーする。

¥copy ｛テーブル名｝ from ｛ファイル名｝

実験では、

¥copy gkuseimeibo from gakusei1

（５） コピーされてるか確認。

１．Select * from （テーブル名）；

実験では、

select * from gakuseimeibo;

２．　サーブレットプログラムの作成

（１） メモ帳でソースプログラムを作成した。(拡張子を java、文字コードを EUC コードにして保存)

ソースプログラムは以下に記す。

（２） コンパイルして、クラスファイルの作成。

１．TP コマンドで AP サーバにソースプログラムを送る。

２．コンパイル。

Javac {ファイル名}.java

実験では、

javac gakusei1.java

（３） 作成したクラスファイルを classes ディレクトリにコピーする。

１．WEB-INF ディレクトリの下に移動する。

Cd webapp

Cd WEB-INF

２．Classes ディレクトリを作成する。

Mkdir classes

３．J99338 ディレクトリの下に戻る。

Cd ..

Cd ..

４．コピーする。

Cp {ファイル名}.class webapp/WEB-INF/classes

実験では、

cp gakusei1.class webapp/WEB-INF/classes

（４）WEB-INF ディレクトリの下の web.xml ファイルの内容を書き換える。

１．WEB-INF ディレクトリの下に移動する

２．ファイルを開く。

Vi web.xml

３．書き込む。

「i」を押して書き込みできるようにする。

```
<web-app>
 <servlet>
      <servlet-name>{ファイル名}</servlet-name>
      <servlet-class>{ファイル名}</servlet-class>
```

<servlet>
<servlet-mapping>
<servlet-name>{ファイル名}</servlet-name>
<url-pattern>/{ファイル名}</url-pattern>
<servlet-mapping>
</web-app>
実験では、
<web-app>
<servlet>
<servlet-name> gakusei1 </servlet-name>
<servlet-class> gakusei1 </servlet-class>
<servlet>
<servlet-mapping>
<servlet-name> gakusei1 </servlet-name>
<url-pattern>/ gakusei1 </url-pattern>
<servlet-mapping>
</web-app>

４．保存する。

:wq

（５）tomcat をリセットする。

３．HTML で初期画面の作成

ソースは以下に記す。

[サーブレットのソースプログラム]

```
1:  import   java.io.*;
2:  import   javax.servlet.*;
3:  import   javax.servlet.http.*;
4:  import   java.net.*;
5:  import   java.sql.*;
6:  import   java.util.*;

7:  public class yama extends HttpServlet{
8:      public void doPost(HttpServletRequest request,HttpServletResponse res)
throws IOException,ServletException
        {
9:          String url="jdbc:postgresql://10.30.2.29:5432/yamamichi";
```

```
10:             String user="postgres";
11:             String pass="******";
12:             String msg="";
13:             Connection con=null;

14:     res.setContentType("text/html; charset=EUC-JP");

                //HTML
15:             PrintWriter out=res.getWriter();
16:             out.println("<html>¥n");
17:             out.println("<head><title>example1</title></head>¥n");
18:             out.println("<body>¥n");
19:             out.println("検索結果は・・・<br>");

                try{

                    //KENSAKU KEY
20:                 String inFild=request.getParameter("FILD");
21:                 String inKey=request.getParameter("KEY");
22:                 String tKey=request.getParameter("GAKUNEN");
23:                 String kKey=request.getParameter("GAKKA");

24:                 inKey=new String(inKey.getBytes("8859_1"),"EUC-JP");

25:                 if(inFild.equals("class")){
26:                 inFild="no";
27:                 inKey=tKey+kKey;
                    }

                    //JDBC driver
28:                 Class.forName("org.postgresql.Driver");
29:                 con=DriverManager.getConnection(url,user,pass);

30:                 Statement stmt=con.createStatement();
31:                 ResultSet rs=null;
                    //SQL
```

```
                    try{
32:                     rs=stmt.executeQuery("SELECT * from gakuseimeibo where
"+inFild+" like '%"+inKey+"%'");
33:                     out.println("<table border>");
34:                     int i=0;
35:                     out.println("<tr><td>学籍番号</td><td>氏名</td><td>住所
</td>");
36:                     while (rs.next()){
37:                         i=i+1;
38:                         String no=rs.getString("no");
39:                         String shimei=rs.getString("shimei");
40:                         String juusho=rs.getString("juusho");
41: out.println("<tr><td>"+no+"</td><td>"+shimei+"</td><td>"+juusho+"</td>");
                        }
42:                 out.println("</table>");
43:                 out.println("該当数は"+i+"件です。");

44:             }finally{
45:                 rs.close();
46:                 stmt.close();
                     }

47:             con.commit();

                }
48:         catch(Exception ex){
49:             msg="shippai"+ex;
            }

            try{
50:             con.close();
51:         }catch(Exception ex){}

52:         out.println("</body>");
53:         out.println("</html>");
            }
```

}

プログラムの解説

1～6行目： このサーブレットに必要なパッケージをインポートする。

java.io.*; サーブレットに必要な Writer オブジェクトを使うのに必要。

javax.servlet.*;
javax.servlet.http.*; } サーブレットの実行時に必要。

java.sql.*; DB アクセスのための SQL ステートメントを発行する時に必要。

7行目： このサーブレットのクラス名は yama とする。

8行目： この doPost メソッドを使う。DoPost メソッドを使うと、ＨＴＭＬフォームを使ってこのサーブレットを動かすことができる。

9行目： 接続する DB 名を変数 url に代入する。

10 行目： DB に接続するためのユーザ名を変数 user に代入する。

11 行目： DB に接続するためのパスワードを変数 pass に代入する。

12 行目： 変数 msg を宣言する。

13 行目： connection オブジェクトを定義しておく。

14 行目： ブラウザに EUC コードで返信することを示す。Text/html は通常の HTML タグをサーブレットからブラウザに返信する時に使う。

15 行目： HTML タグを出力するための Writer オブジェクトを準備する。

16～19 行目：HTML タグを出力する。

20～23 行目：ブラウザからサーブレットに送られてきたデータを取り込む。

inFild 学籍番号、氏名、住所のどのフィールドから、または、クラスから検索するかを取り込む。

inKey 検索するキーワードを取り込む。

tKey 検索する学年を取り込む。

kKey 検索する学科を取り込む。

24 行目： 取り込んだキーワードの文字コードを EUC コードに変換する。

25～27 行目：もし、 クラスから検索する場合は、inFild に no を代入し、inKey に tKey と kKey のデータを結合する。

28 行目： 使う JDBC ドライバを宣言する。

29 行目： DB に対して接続オブジェクトを生成、つまりインスタンス化する。

30 行目： 検索結果を格納する結果オブジェクトを生成する。

31 行目： 検索結果を格納するためのオブジェクトを宣言する。

32 行目： SQL を実行して、31 行目で宣言したオブジェクトに格納する。テーブル gakuseimeibo の inFild で指定されたフィールドの中から、

inKey で指定された文字列を含んでいるデータを検索する。

33 行目：　HTML タグを出力する。検索結果を表にして表示するためのもの。

34 行目：　SQL を実行した結果、ヒットした件数を数えるための変数を宣言する。

35 行目：　検索結果の表の見出しを HTML で出力する。

36 行目：　ヒットした件数だけループ処理が行われ、検索結果を出力する。

37 行目：　ループした回数を数えている。その回数が、ヒットした件数になる。

38 行目：　ヒットした 1 件分の学籍番号のデータを取り出す。

39 行目：　ヒットした 1 件分の氏名のデータを取り出す。

40 行目：　ヒットした 1 件分の住所のデータを取り出す。

41 行目：　取り出した 1 件分のデータを表にして HTML で出力する。

42 行目：　表の終わりを HTML で出力する。

43 行目：　ヒットした件数を HTML で出力する。

45～46 行目：SQL ステートメントや検索結果を格納するのに使ったオブジェクトをクリーンアップして解放する。

47 行目：　どこかでエラーが起きたら、失敗したことを示すメッセージを変数 msg に代入する。

50 行目：　DB の接続に使ったオブジェクトをクリーンアップして解放する。

52～53 行目：HTML タグを終了させる。

[HTML のソース]

```
1:  <html>
2:  <head><title>入力画面</title></head>
3:  <body>
4:  <form method=post action="http://10.30.2.30/j99338/yama">
5:  どの項目で検索しますか？<br>
6:  <input type="radio" checked name="FILD" value="no">学籍番号<br>
7:  <input type="radio" name="FILD"value="shimei">氏名<br>
8:  <input type="radio" name="FILD"value="juusho">住所<br>
9:  検索したいキーワードを入力してください。<br>
10: <input type="text" name="KEY" size=30><br>
11: <dl><dt><input type="radio" name="FILD"      value="class">クラス<br>
12: <dd>学年
13: <dd><input type="radio" name="GAKUNEN" value="02">1 年<br>
14: <dd><input type="radio" name="GAKUNEN" value="01">2 年<br>
15: <dd><input type="radio" name="GAKUNEN" value="00">3 年<br>
```

```
16: <dd><input type="radio" name="GAKUNEN" value="99">4 年<br>
17: <dd><input type="radio" name="GAKUNEN" value="98">5 年<br>
18: <dd>学科
19: <dd><input type="radio" name="GAKKA" value="1">機械<br>
20: <dd><input type="radio" name="GAKKA" value="2">電気<br>
21: <dd><input type="radio" name="GAKKA" value="3">情報<br>
22: <dd><input type="radio" name="GAKKA" value="4">物質<br>
23: <dd><input type="radio" name="GAKKA" value="5">環境都市<br>
24: <dd>※留年した学生は、入学した年から数えた学年です。
25: </dl>
26: <input type="submit" value="送信">
27: <input type="reset" value="クリア">
28: </form>
29: </body>
30: </html>
```

解説

１～３行目：HTML 文が始まることを示す。

４行目：　doPost メソッドを起動する。

６～８行目：チェックボックスを表示する。

検索したい項目を選ぶ。

学籍番号をチェックするとサーブレットに渡す引数 FILD に“ no ”が代入される。

同様に、氏名にチェックすると“ shimei ”、住所だと“ juusho ”が代入される。

何もしていないと、学籍番号にチェックされている。

10 行目：　検索したいキーワードを入力するテキストボックスを表示する。ここで入力されたデータは渡す引数 KEY に代入される。

１１行目：　検索したい項目を選ぶチェックボックスと関連して、ここをチェックするとサーブレットに渡す引数 FILD に“ class ”が代入される。

12～17 行目：スペースをあけてから、チェックボックスを表示する。

検索したい学年を選ぶ。

１年をチェックするとサーブレットに渡す引数 GAKUNEN に“ １ ”が代入される。

同様に、２年にチェックすると“ ２ ”、３年だと“ ３ ”、４年だと“ ４ ”、５年だと“ ５ ”が代入される。

18～23 行目：スペースをあけてから、チェックボックスを表示する。

検索したい学科を選ぶ。
機械をチェックするとサーブレットに渡す引数 GAKKA に“ １ ”が代入される。
同様に、電気にチェックすると“ ２ ”、情報だと“ ３ ”、物質だと“ ４ ”、環境都市だと“ ５ ”が代入される。

26 行目：　サーブレットに引数を送信して起動するためのボタンを表示する。

27 行目：　入力したものを初期の状態に戻すためのボタンを表示する。

28～30 行目：HTML 文を終了することを示す。

[苦労した点]

・ 学籍番号が０から始まる場合、最初の０が表示されない。

例　02101　のときは　2101　になってしまう。

＜解決策＞

最初は学籍番号のフィールドは整数型にしていたのを、文字型にした。

・ 文字化けしてしてしまう。

＜解決策＞

DB へは EUC コードで入力しなくてはいけなかったのに、はじめは SJIS コードで入力していたため、はじめの文字化けが起こった。

それを解決するために、エクセルで入力したファイルをサーバへ送ってから、

¥usr¥bin¥nkf –e gakusei1>gakusei1

を実行して SJIS コードのデータを EUC コードに変換した。

次に文字化けが起こったのは、サーブレットプログラムと HTML 文書を EUC コードで作ったため、ブラウザで見たとき全部が文字化けしてしまった。

これを解決するためにサーブレットに

res.setContentType("text/html; charset=EUC-JP");

を入れて、HTML 文書を EUC コードで表示することを指定した。

次に文字化けが起こったのは、サーブレットが完成してからテストを行ったら、キーワードが SJIS コードで入力されてしまうため、SQL で検索をしても、どれにもヒットしない。

これを解決するために、サーブレットプログラムに

inKey=new String(inKey.getBytes("8859_1"),"EUC-JP");

を入れて、取り込んだ文字列を EUC コードに変換した。

[今後の課題]

・ クラスで検索をした場合、留年した学生についても現在のクラスで表示できるようにする。

今回作成したシステムは、クラスで検索する場合は、学年を入学した年と、学科を各学科に対応した数字をキーワードにして、学籍番号のフィールドから検索しているため、留年していたり、転科していたり、編入学している学生については、現在の正しいクラスで検索することができない。

これを克服するためには、DB のテーブルを、クラスを表すフィールドをつけて作り直さなければいけないと思う。

・ 新入生の分など新しくデータベースに書き込める機能も追加する。

現段階では、ブラウザからはデータを検索するしかできない。しかしそれでは新入生が入学してきたり、上記のようなことが実現できた場合、留年した学生などについてデータを変更するにはサーバのほうから変更しなくてはいけない。

これをブラウザから追加や変更をできるようにすれば、便利だと思う。そのためには、データを書き込むページを作って、そこで書き込まれたデータをサーブレットに送り、サーブレットは SQL の insert 文や、update 文で追加や変更を行うようにすればいいと思う。

しかし、個人データを扱っているので、追加や変更を行える人を限定するために、パスワードを設定して、それを通さないと書き込みのページへ飛べないようにしなければいけないと思う。

[参考文献]

「やさしくわかる SQL」　株式会社秀和システム
著者　溝渕寛和　　ISBN4-7908-0085-X C3055
「PostgreSPL 完全攻略ガイド」　株式会社技術評論社
著者　石井達夫　　ISBN4-7741-0890-1 C3055
「一番やさしい JSP&サーブレット入門塾」 翔泳社
著者　樋口研究室　　ISBN4-7981-0347-0 C3055
「誰にでもできる HTML 入門」　日本実業出版社
著者　小林憲夫　　ISBN4-534-02442-8 C0055
「逆引きサーブレット＆JSP リファレンス」株式会社技術評論社
著者　日立ソフトウェアエンジニアリング株式会社

インターネットビジネス推進部
ISBN4-7741-1379-4 C3055
「UNIX コマンド例文辞典　sh/bash 編」　株式会社技術評論社
著者　OfficeMASA　ISBN4-7741-1032-9 C3055

[感想]

今回の実験は、サーブレット、SQL、JDBC などほとんど全て初めて聞く言葉ばかりで、すべてがチンプンカンプンでした。やりたいことはかなり早く決まったんですが、それを実現するための、サーブレットなどの勉強が実験の半分以上を占めてしまいました。最初、一番苦労するだろうと思っていた DB に入れる約 900 人分のデータ作りは、思っていたより早く終わったのが嬉しかったです。

実験を始める前に先生と話していた、20 万円分の知識が得られたかどうかは疑問ですが（20 万円の講習がどこまでやるのかわからないので）、自分の中では、何もないところから作り上げることができたという達成感があるのでよかったです。