SQL*Plus
Oracle Web Server

Architettura di un'applicazione

RAD (Delphi, Forms…)
o HTML o Java

PL/SQL (embedded)

Trigger,
stored procedures

SGBD

in prima approssimazione, possiamo usare
sh ed **sqlplus** per gestire input ed output

SQLPLUS

- Per entrare
  - sqlplus
  - poi dare nome e password
- Permette di:
  - eseguire comandi DDL come **create table**, **create procedure**, **create package**, **create trigger**
  - eseguire interattivamente query e aggiornamenti SQL, formattando l'output
  - fare un po' di editing locale
  - chiamare procedure pl/sql passando loro variabili sqlplus
  - caricare ed eseguire file di comandi sqlplus, sql, plsql (script)

SQL*Plus

Script

- Possono contenere:
  - comandi sqlplus
  - comandi SQL
  - comandi PL/SQL
- Per creare uno script ed eseguirlo:
  - dare al file un nome <n>.sql
  - scrivere @<n> (oppure start <n>) in sqlplus
- Lo script può contenere variabili &1 ... &n, che gli vengono passate dal comando @:
  - @script val1 ... valn

Editing

APPEND text A text adds text at the end of a line

CHANGE /old/new

C /old/new changes old to new in a line

CHANGE /text C /text deletes text from a line

CLEAR BUFFER CL BUFF deletes all lines

DEL (none) deletes the current line

DEL n (none) deletes line n

DEL * (none) deletes the current line

DEL LAST (none) deletes the last line

DEL m n (none) deletes a range of lines (m to n)

Editing

INPUT	I	adds one or more lines
INPUT text	I text	adds a line consisting of text
LIST	L	lists all lines in the SQL buffer
LIST n	L n or n	lists line n
LIST *	L *	lists the current line
LIST LAST	L LAST	lists the last line
LIST m n	L m n	lists a range of lines (m to n)

SQLPLUS - informazioni utili

- SQL: il ";" termina ed esegue
- PL/SQL: il ";" compila ed il **run** (/) esegue
  - errori di compilazione: **show errors**
- Metadati: **describe** user_objects
- **select** object_name , object_type **from** user_objects;
- **help**
- **host** ls

Formattare l'output

- Una pausa ad ogni schermata:
  - **set pagesize** 23
  - **set pause on**
- Formattare le colonne:
  - **column** object_name **heading** 'nome|oggetto'
  - **column** object_name **format** a20
  - **column** object_name **format** a6
  - **set wrap on / off**
  - select object_name "IlNome", object_type from user_objects;
- Campi numerici: **format** 999

Chiamare procedure PL/SQL da SQLPLUS

- **execute** aa.bb('c')
- Variabili di comunicazione

```
variable x number    -- def variabile sqlplus
begin
  :x := 3;
end;
/                     -- sua inizializzazione
print x               -- visualizzarla
execute incr(:x)      -- passarla ad una procedura
print x
```

- Output da PL/SQL verso sqlplus:
  - in PL/SQL usare: dbms_output.put_line(stringa)
  - in sqlplus scrivere: **set serveroutput on**

SQLPLUS: Messaggi di errore

- Non confondere gli errori nel package con quelli nel package body
- Comandi utili:
  - mostra errori:
    - show err
    - show err package pippo
  - lista le righe 80-100:
    - l 80 100
- Messaggi tipici:
  - @pippo
    - 43
    - 44

  => avete scordato di scrivere "/"

Embedded PL/SQL

- Immergere il codice in un programma:
  - **EXEC SQL EXECUTE**
    begin

    end
    END-EXEC;

Embedded PL/SQL

Embedded PL/SQL

- Definire variabili di comunicazione:
  - **EXEC SQL BEGIN DECLARE SECTION;**
    VARCHAR empname[11];
    int salary;
    VARCHAR uid[20];
    VARCHAR pwd[20];
    EXEC SQL END DECLARE SECTION;

  ora empname e salary sono variabili C, ed :empname e :salary sono le stesse variabili dentro un blocco pl/sql

Embedded PL/SQL

- Connettersi:
  - **EXEC SQL CONNECT** :uid **IDENTIFIED BY** :passwd
- Altri comandi EXEC SQL:
  - **WHENEVER SQLERROR** { **DO** expr | **CONTINUE** }
  - **COMMIT WORK / ROLLBACK WORK**
  - **DECLARE TABLE**: descrive una tabella non ancora definita

Embedded PL/SQL - SQL dinamico

- SQL dinamico (per eseguire statement SQL non noti a tempo di compilazione):
  - **EXECUTE IMMEDIATE** stringa
  - **PREPARE FROM / EXECUTE USING**: compila / collega ed esegue una stringa

```
EXECUTE IMMEDIATE dynamic_string
[INTO {define_variable[, define_variable]... | record}]
[USING [IN | OUT | IN OUT] bind_argument
[, [IN | OUT | IN OUT] bind_argument]...]
[{RETURNING | RETURN} INTO bind_argument[,
  bind_argument]...];
```

Embedded PL/SQL - SQL dinamico

- Il comando EXECUTE IMMEDIATE è composto da tre parti principali:
  - stringa costruita dinamicamente (statement SQL o blocco PL/SQL)
  - INTO: variabili nelle quali inserire valori tupla risultato (se viene restituita una singola tupla)
  - USING: parametri da utilizzare nella stringa dinamica
  - RETURNING: variabili nelle quali inserire alcuni valori di risultato

Embedded PL/SQL - SQL dinamico Esempio

```
DECLARE
sql_stmt VARCHAR2(200);
plsql_block VARCHAR2(500);
emp_id NUMBER(4) := 7566;
salary NUMBER(7,2);
dept_id NUMBER(2) := 50;
dept_name VARCHAR2(14) := 'PERSONNEL';
location VARCHAR2(13) := 'DALLAS';
emp_rec emp%ROWTYPE;
```

Embedded PL/SQL - SQL dinamico
Esempio (continua)

```
BEGIN
EXECUTE IMMEDIATE 'CREATE TABLE bonus (id NUMBER,
    amt NUMBER)';

sql_stmt := 'INSERT INTO dept VALUES (:1, :2, :3)';
EXECUTE IMMEDIATE sql_stmt USING dept_id, dept_name,
    location;

sql_stmt := 'SELECT * FROM emp WHERE empno = :id';
EXECUTE IMMEDIATE sql_stmt INTO emp_rec USING
    emp_id;

plsql_block := 'BEGIN emp_pkg.raise_salary(:id, :amt); END;';
EXECUTE IMMEDIATE plsql_block USING 7788, 500;
```

Embedded PL/SQL - SQL dinamico
Esempio (continua)

```
sql_stmt := 'UPDATE emp SET sal = 2000 WHERE empno = :1
RETURNING sal INTO :2';
EXECUTE IMMEDIATE sql_stmt USING emp_id RETURNING
    INTO salary;

EXECUTE IMMEDIATE 'DELETE FROM dept WHERE deptno =
    :num'
USING dept_id;
END;
```

Embedded PL/SQL - SQL dinamico
Esempio

```
CREATE PROCEDURE delete_rows (
table_name IN VARCHAR2,
condition IN VARCHAR2 DEFAULT NULL) AS
where_clause VARCHAR2(100) := ' WHERE ' ||
   condition;
BEGIN
IF condition IS NULL THEN where_clause := NULL;
   END IF;
EXECUTE IMMEDIATE 'DELETE FROM ' ||
   table_name || where_clause;
EXCEPTION
...
END;
```

Embedded PL/SQL
SQL dinamico - Cursori

- unica differenza con i cursori per SQL statico: dichiarazione e apertura vengono unificate in un'unica operazione

  OPEN {cursor_variable | :host_cursor_variable}

  FOR dynamic_string

  [USING bind_argument[, bind_argument]...];

- esempio:

  OPEN emp_cv FOR -- open cursor variable

  'SELECT ename, sal FROM emp WHERE sal > :s'
  USING my_sal;

Variabili indicatore

- Per gestire valori ORACLE troppo grandi per stare in una variabile C, ed il valore nullo, una variabile di comunicazione può essere associata ad un indicatore, con la sintassi :variabile:indicatore:
  - **SELECT** a **INTO** :x:xind
- In questo caso, se ORACLE non riesce a memorizzare un valore nella variabile di comunicazione, memorizza -2 o un intero > 0 nell'indicatore
- Se il valore era NULL, memorizza -1 nell'indicatore; analogamente, se l'indicatore vale -1, ORACLE interpreta la variabile come NULL

OCI

- OCI è un API che permette di spedire stringhe SQL al server per farle eseguire, analogamente ad EXECUTE IMMEDIATE
- Permette anche di definire delle variabili di collegamento e di usarle dentro le stringhe
- Mette a disposizione funzioni per effettuare la connessione
- Più complesso da usare rispetto a embedded PL/SQL; equivale ad effettuare la precompilazione a mano

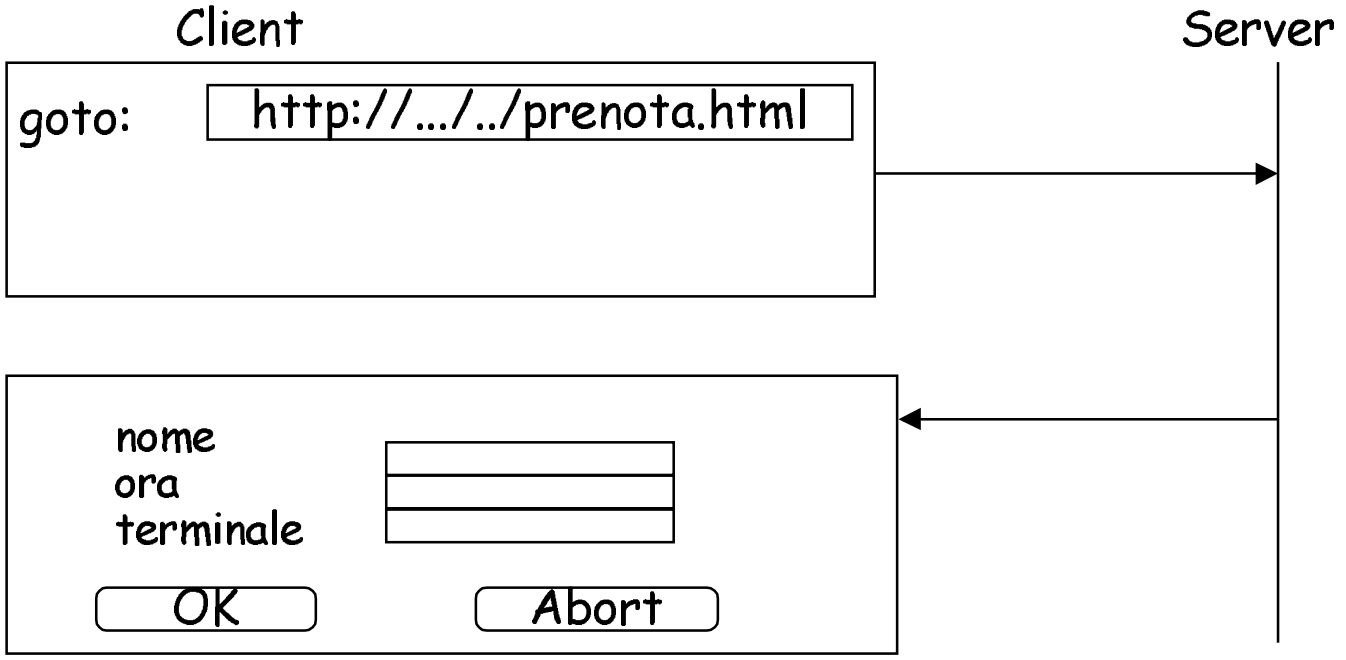
ORACLE Forms

- Per programmare l'interfaccia
- Si disegna una maschera e di definiscono bottoni e menu-item associandovi azioni
- Definisce automaticamente maschere con azioni per:
  - interrogare una o più tabelle con condizioni "**like**"
  - scorrere le registrazioni trovate
  - inserire / aggiornare / cancellare registrazioni
  - uscire dall'applicazione

Interfaccia in HTML - Form

Interfaccia usando HTML

- Scenario:
  - Il terminalista apre un Web browser (e.g. Netscape) e inserisce l'URL dell'applicazione
  - Riceve dal server in cambio una pagina HTML che contiene una "form"



Form in HTML

```
<HTML>
<HEAD><TITLE>Servizio prenotazione terminali
</TITLE></HEAD>

<BODY>
<H1 ALIGN=CENTER>Immetti i dati della
  prenotazione</H1>
<BR>

<FORM METHOD = "GET" ACTION = "/cgi-
  bin/prenota.cgi">
```

Form in HTML

```
Nome di login: <INPUT TYPE = "text" NAME =
   "Nome">
<BR>
Ora: <INPUT TYPE = "text" NAME = "Ora" SIZE = 2
   MAXLENGTH = 2>
<BR>
Data (gg/mm/aa): <INPUT TYPE = "text" NAME =
   "Data" SIZE = 8 MAXLENGTH = 8>
<BR>

<INPUT TYPE="submit" VALUE="Spedisci">
</FORM></BODY></HTML>
```

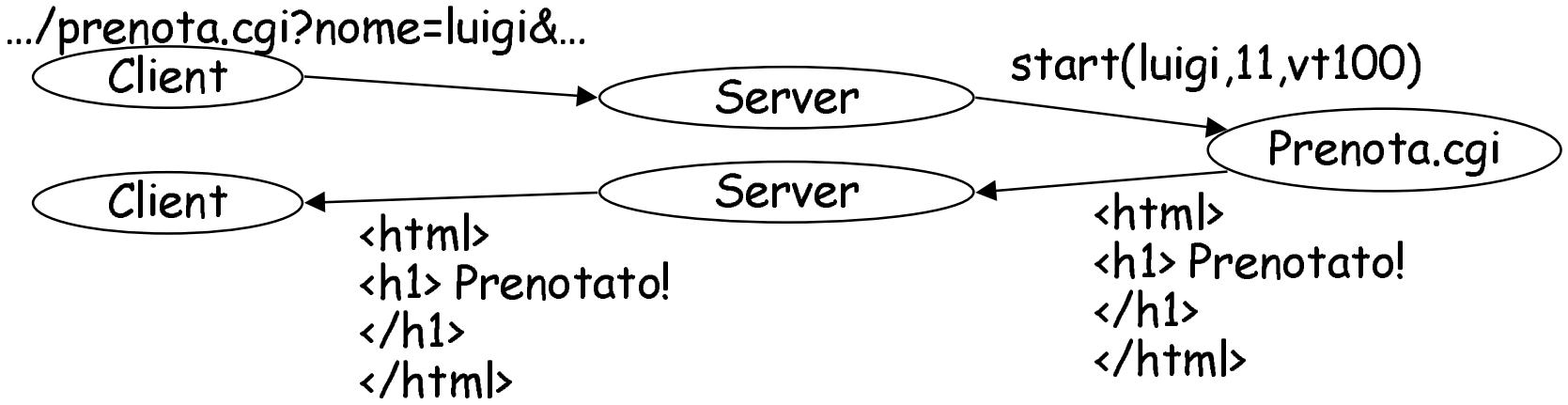
Risultato


Location:
http://www.di.unipi.it/~ghelli/prenota.html
Immetti i dati della prenotazione
Nome di login:
Ora:
Data (gg/mm/aa):
Spedisci

Scenario (continua)

- Il terminalista riempie i campi della form e preme OK; il browser manda al server un URL particolare, che identifica un'applicazione (prenota.cgi) ed i valori di alcuni parametri
  - http://...../.../cgi-bin/prenota.cgi?nome=luigi&ora=11&Data=11/12/97
- Il server trova l'applicazione (prenota.cgi) e la lancia passando i parametri
- L'applicazione ritorna al server una pagina html o un altro valore MIME
- Il server spedisce la pagina (che può contenere un'altra form) al client

Scenario (continua)



Vantaggi e svantaggi

- Vantaggi:
  - non c'è bisogno di installare software sul lato client
  - l'applicazione è totalmente portabile
- È facile?
  - senza tool sarebbe molto più facile che programmare interfacce a basso livello
  - attualmente ci sono più tool per l'approccio alternativo

Vantaggi e svantaggi

- Applets:
  - la pagina che contiene la forma può contenere anche del codice (applet Java, Orblet...) che può fare del lavoro sul client
- Active server pages:
  - un modo rapido per scrivere programmi che saranno eseguiti dal server per genere la pagina effettiva

Problemi

- Il protocollo è senza stato
  - è molto complicato scrivere applicazioni che si ricordano di quello che è successo nelle schermate precedenti
- La comunicazione dal client al server è limitata a stringhe
  - non è un grosso problema per questo tipo di applicazioni

Form HTML

- Il tag: <FORM METHOD = "GET" ACTION = "/prenota.cgi">
  - Method = "GET": parametri nella URL (max 255, URL compresa)
  - Method = "POST": parametri passati al server via standard input
- <INPUT TYPE = "text" NAME = "Nome">

  crea un campo editabile, specificando (opzionale):
- dimensioni sullo schermo: SIZE
- lunghezza massima: MAXLENGTH
- default: VALUE
- varianti: password, hidden

Form HTML

- <INPUT TYPE="submit" VALUE="Spedisci">: crea un bottone, con etichetta *Spedisci*, che, se premuto, invia al server la URL con l'informazione dentro
- se è presente un attributo NAME="nomeBott" spedisce anche la coppia nomeBott=Spedisci

Checkbox

- <INPUT TYPE = "checkbox" NAME = "Altri" VALUE = "Si" CHECKED>: un pulsante che, se selezionato, mette "Altri=Si" nella stringa risultato, altrimenti non mette nulla
  - nella form: <INPUT TYPE = "checkbox" NAME = "Condizione" VALUE = "T" CHECKED>
  - nella procedura: (...,Condizione char default 'F',...)

Radio Button e Reset

- <INPUT TYPE="radio" NAME="Term" VALUE="VT100" CHECKED>
  <INPUT TYPE="radio" NAME="Term" VALUE="X">: crea un insieme di pulsanti tali che, se l'utente ne sceglie uno, gli altri si deselezionano
- <INPUT TYPE="reset" VALUE="Cancella">: crea un bottone, con etichetta *Cancella*, che, se premuto, cancella i campi

Form HTML: Esempio

- Una form che accetta dei campi ed effettua o una ricerca o un inserimento:
  - Radio button:
    - <INPUT TYPE="text" ...>
    - <INPUT TYPE="radio" NAME="Scelta" VALUE="Cerca" CHECKED>Ricerca
 <INPUT TYPE="radio" NAME="Scelta" VALUE="Immetti">Immissione
    - <INPUT TYPE="submit" VALUE="Esegui">

Form HTML: Esempio

- Doppio submit:
  - <INPUT TYPE="text" ...>
  - <INPUT TYPE="submit" Name="Scelta" VALUE="Cerca">
  - <INPUT TYPE="submit" Name="Scelta" VALUE="Immetti">
- l'effetto, in entrambi i casi è:
  - Spedisce o la coppia Cosa=Cerca o la coppia Cosa=Immetti

Form HTML: Esempio completo

```
<HTML>
<HEAD><TITLE>Servizio prenotazione terminali
  </TITLE></HEAD>
<BODY>
<H1 ALIGN=CENTER>Immetti i dati della
  prenotazione</H1>
<BR>
<FORM METHOD = "GET" ACTION = "/cgi-
  bin/prenota.cgi">
Nome: <INPUT TYPE = "text" NAME = "Nome">
<BR>
Ora: <INPUT TYPE = "text" NAME = "Ora" SIZE = 2
  MAXLENGTH = 2>
<BR>
```

Form HTML: Esempio completo

Accetto prenotazioni su terminali diversi:
<INPUT TYPE = "checkbox" NAME = "Altri" VALUE =
"Si" CHECKED>

Tipo di terminale richiesto:
<INPUT TYPE="radio" NAME="term" VALUE="VT100"
CHECKED>VT100
<INPUT TYPE="radio" NAME="term"
VALUE="X">XTerm

<INPUT TYPE="reset" VALUE="Cancella">
<INPUT TYPE="submit" VALUE="Spedisci">
</FORM></BODY></HTML>

Form HTML: Esempio completo

- esempi di url generate:
  - http://www.di.unipi.it/cgi-bin/prenota.cgi?Nome=Luigi&Ora=11&Altri=Si&term=VT100
  - http://www.di.unipi.it/cgi-bin/prenota.cgi?
  - Nome=Luigi&Ora=11&term=X

Select

- Sintassi:

  <SELECT NOME="CodCompito">

  <OPTION VALUE = 1>1: Requisiti

  <OPTION VALUE = 2>2: Manutenzione Requisiti

  </SELECT>

- Crea un combo box con due scelte:
  - 1: Requisiti e
  - 2: Manutenzione Requisiti
- invia 1 nel primo caso, 2 nel secondo

Select

- <SELECT MULTIPLE NOME="CodCompito">: permette scelte multiple
- <SELECT ... SIZE=6>: visualizza una finestra scrollabile anziché un combo box

Oracle Web Server

Oracle Web Server

- ORACLE web server è un server http che supporta il protocollo cgi-bin
- ORACLE web agent è uno script cgi-bin che, una volta chiamato con il path ..../owa/p.q?p1=v1&p2=v2
  - Si connette ad un oracle db server usando un nome utente che dipende dalla configurazione; ad esempio, http://lina.cli.di.unipi:8000/prenota/owa si connette al server di lina.cli.di.unipi come utente 'prenota'
  - Esegue la procedura p.q(p1 => 'v1', p2 => 'v2')

Oracle Web Server

- Se la procedura p.q crea una pagina html, questa viene mostrata a chi aveva invocato il path ..../owa/p.q?p1=v1&p2=v2
- La procedura p.q crea la pagina html usando le procedure dei package htp e owa_util

Esempio

- Un file html statico (http://lina.cli.di.unipi:8000/bdl/guarda.html, ovvero ~ghelli/bdl/html/guarda.html):

```
<HTML>
<HEAD><TITLE>Esame tabella
</TITLE></HEAD>
<B
ODY>
<H1 ALIGN=CENTER>Inserisci il nome della tabella
  che vuoi guardare</H1>
<BR>
```

Esempio

```
<FORM METHOD = "GET"
  ACTION = "/prenota/owa/guarda.guarda">

<INPUT TYPE = "text" NAME = "rel">
<BR>

<INPUT TYPE="submit" VALUE="Spedisci">
<INPUT TYPE="reset" VALUE="Cancella">
</FORM></BODY></HTML>
```

- Quando è selezionato, attiva la url:
  /prenota/owa/guarda.guarda?rel=....
  sulla macchina http://lina.cli.di.unipi:8000

Esempio

- La url: http://lina.cli.di.unipi:8000/prenota/owa/guarda.guarda?rel=studenti attiva l'agente cgi-bin http://lina.cli.di.unipi:8000/prenota/owa passandogli il parametro guarda.guarda?rel=studenti
- l'agente a sua volta chiama la procedura guarda.guarda(rel => 'studenti')

Esempio - Il package guarda

```
create or replace package guarda as
  procedure guarda(rel varchar2);
end guarda;

create or replace package body guarda as
procedure guarda(rel varchar2) is
  res boolean;
    begin
    htp.htmlOpen;
    htp.headOpen;
    htp.Title('guarda relazione');
    htp.headClose;
```

Esempio - Il package guarda

```
    htp.bodyOpen;
    res := owa_util.tablePrint(ctable => rel,
              cattributes => 'BORDER');
    htp.bodyClose;
    htp.htmlClose;
  end guarda;

end guarda;
```

- La procedura guarda
  - apre il documento html:
    - htp.htmlOpen;

Esempio - Il package guarda

- crea l'header con il titolo:
  - htp.headOpen;
  - htp.Title('guarda relazione');
  - htp.headClose;
- apre il corpo:
  - htp.bodyOpen;
- inserisce una tabella nel corpo:
  - res := owa_util.tablePrint(ctable => rel,
    cattributes => 'BORDER');
  - owa_util.tablePrint permette di generare automaticamente una query SQL e di inserire il risultato in una pagina html
- chiude il corpo e chiude il documento:
  - htp.bodyClose;
  - htp.htmlClose;

Altro esempio

- una pagina dinamica per invocarne un'altra generazione dei valori di una combo box mediante una query

```
procedure immetti is
  begin
    stampaTesta('Cerca studenti');
    htp.bodyOpen;
    htp.print('Scegli lo studente da cercare');
    htp.formOpen ('/prenota/owa/cercas.cerca','GET');
    htp.formSelectOpen ('cognomeScelto','Scegli il cognome');
    for c in (select distinct cognome from studenti )
    loop
      htp.formSelectOption(c.cognome);
    end loop;
```

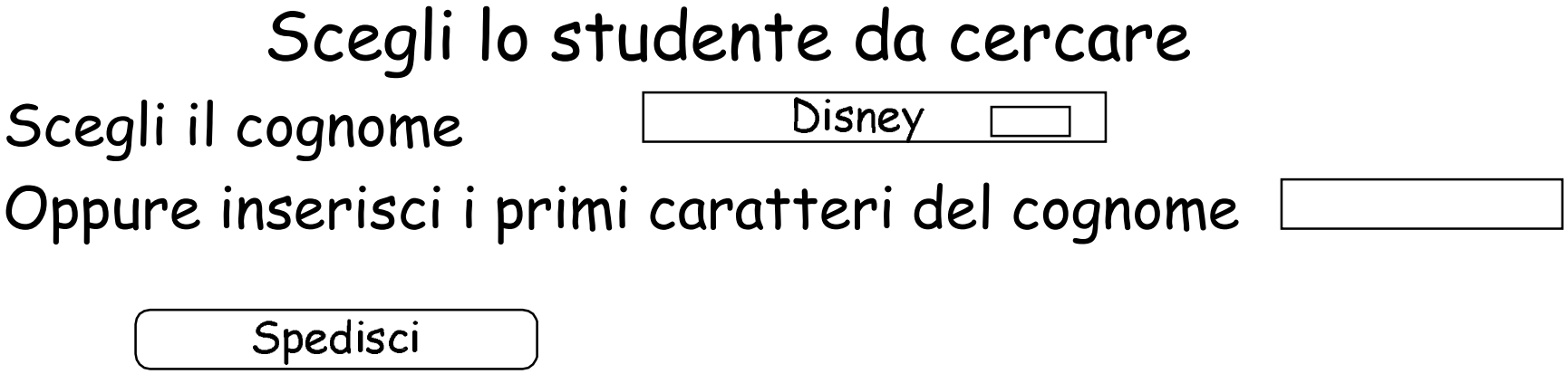
Altro esempio (segue)

```
  htp.formSelectClose;
  htp.br;
  htp.prn('Oppure inserisci i primi caratteri del cognome' );
  htp.formText ('cognomeBattuto');
  htp.formSubmit (cvalue => 'Spedisci');
  htp.formClose;
  htp.bodyClose;
  htp.htmlClose;
end immetti;
```

- generazione dinamica di una form che genera un'altra pagina dinamicamente

Altro esempio – La procedura immetti

- La procedura *immetti* si può invocare da una pagina html e ne produce un'altra che contiene la seguente form:



- Quando si preme submit, emette l'URL:
  - .../prenota/owa/cercas.cerca?cognomeScelto=...&cognomeBattuto=...
  - cognomeBattuto può mancare
- L'agente invoca prenota.cercas.cerca con i parametri opportuni

Altro esempio – La procedura immetti

- Il corpo della procedura immetti
  - htp.formOpen('/prenota/owa/cercas.cerca','GET') apre la form
  - htp.formSelectOpen /SelectOption(option) /SelectClose: ogni chiamata di SelectOption aggiunge una nuova opzione alla lista "cognome Scelto":

```
htp.formSelectOpen ('cognomeScelto','Scegli il
cognome');
        for c in ( select distinct cognome from studenti)
        loop
        htp.formSelectOption(c.cognome);
        end loop;
htp.formSelectClose;
```

Altro esempio – La procedura immetti

- Il corpo della procedura immetti (segue)
  - htp.br: va a capo
  - htp.prn('testo'): scrivi il testo
  - htp.formText ('cognomeBattuto'): inserisci un campo di tipo testo per immettere il parametro 'cognomeBattuto'
  - htp.formSubmit (cvalue => 'Spedisci'): aggiungi un bottone submit
  - htp.formClose: chiudi la form

Altro esempio – La procedura cerca

- La procedura cercas.cerca potrebbe ricevere oppure non ricevere il parametro cognomeBattuto: deve usare un valore di default:

```
procedure cerca(cognomeBattuto varchar2 default null,
                cognomeScelto varchar2 default null)
is
  cognomeCercato varchar2(160);
  conta number;
  res boolean;
  clausola varchar2(160);
```

Altro esempio – La procedura cerca

- Per prima cosa decide in base a quale dei due parametri effettuare la ricerca; il cognomeBattuto è considerato solo una sottostringa iniziale:

```
begin
    if cognomeBattuto is not null
    then cognomeCercato := cognomeBattuto || '%';
    else cognomeCercato := cognomeScelto;
    end if;
```

Altro esempio - La procedura cerca

- Poi apre il documento e scrive il valore del parametro che ha scelto (htp.line: una linea)

```
stampaTesta('studenti trovati');
htp.bodyOpen;
htp.print('Studente cercato: ' || cognomeCercato);
htp.line;
```

Altro esempio - La procedura cerca

- Poi conta quanti studenti ha trovato e si comporta di conseguenza:

```
select count(*) into conta
from studenti
where upper(cognome) like upper(cognomeCercato);
if conta = 0
  then  stampaNonTrovato(cognomeCercato);
elsif conta = 1
  then  stampaTrovato(cognomeCercato);
  else  stampaTrovati(cognomeCercato);
end if;
```

- Le funzioni stampaNonTrovato / Trovato / Trovati visualizzano i risultati

Altro esempio - La procedura cerca

- Poi visualizza alcune informazioni sugli studenti trovati, generando dinamicamente una query (in SQL due singoli apici dentro una stringa diventano un singolo apice, per cui valutando '''' si ottiene una stringa di un carattere <'>)

```
clausola := 'WHERE upper(cognome) like upper('''
                    || cognomeCercato || ''')';
 res := owa_util.tablePrint(ctable => 'STUDENTI',
          cattributes => 'BORDER',
          ccolumns => 'Nome, Cognome, Matricola',
          cclauses => clausola);
```

Altro esempio - La procedura cerca

- Dà la possibilità di ritornare alla procedura immetti; htp.anchor(url, testo) fa sì che 'testo' diventi un ancora per passare alla url specificata:

```
htp.prn('Se vuoi fare un''altra ricerca ');

htp.anchor('http://lina.cli.di.unipi.it:8000/prenota/
owa/cercas.immetti','premi qui.');
```

- Infine chiude corpo e html

Passare un elenco di parametri

- Per ricevere un elenco di parametri da una procedura: il tipo table
- Per ricevere un elenco di parametri da una URL:
  - Dichiarare il parametro di tipo **owa_util.ident_arr**:
    - elenco prenota.owa_util.ident_arr
  - Passare il parametro scrivendo
    - ...?elenco=luigi&elenco=luca
  - Da una form:
    - <INPUT Type="checkbox" Name="elenco" Value ="luigi">
    - <INPUT Type="checkbox" Name="elenco" Value ="luca">
  - Oppure:
    - <SELECT Name="elenco" Size="10" Multiple>

Passare un elenco di parametri

- L'elenco è una tabella di stringhe:

```
procedure RiceviElenco(
      Elenco      prenota.owa_util.ident_arr,
 ) is
 i := Elenco.FIRST;
while i is not null
loop
  opera su (Elenco(i));
  i := Elenco.NEXT(i);
end loop;
```

Il package HTP

- Creare una pagina HTML:
  - htp.htmlOpen
  - htp.headOpen
  - .....
  - htp.headClose
  - htp.bodyOpen
  - .....
  - htp.bodyClose
  - htp.htmlClose
- Equivalgono a: htp.print('<HTML>'), htp.print('<HEAD>'), htp.print('</HEAD>'), ...

Package HTP - Generare una tabella

- La tabella inizia e termina con tableOpen/tableClose
- Ogni riga inizia e termina con tableRowOpen/tableRowClose
- Ogni casella si crea in due modi:
  - tableData(stringa)
  - print('<TD>');...contenuto...;print('</TD>')
- Esempio:

```
prenota.htp.tableOpen;
prenota.htp.tableRowOpen;
prenota.htp.tableData('password');
prenota.htp.print('<TD>');
prenota.htp.formPassword('PassEl');
prenota.htp.print('</TD>');
prenota.htp.tableRowClose;
...
prenota.htp.tableClose;
```

Simulare una sessione

- Problema: voglio permettere di immettere più prenotazioni, e di decidere alla fine se confermare l'immissione
- Soluzione:
  - La schermata iniziale passa all'utente un identificatore unico
  - Tutte le richieste sono memorizzate in una tabella temporanea assieme all'identificatore
  - Le richieste sono eseguite, o cancellate, alla conferma

Simulare una sessione

```
creamenu() is
  apriForma('/prenota/owa/rossi.raccogli','GET');
  passaParametro('ide',newide());
  chiudiForma;
end creamenu;

raccogli(ide) is
  apriForma('/prenota/owa/rossi.immOConf','GET');
  passaParametro('ide',newide());
  input('login');
  submit('cosaFare','ancora');
  submit('cosaFare','ultimo');
 chiudiForma();
end raccogli;
```

Simulare una sessione

```
immOConf(ide, cosaFare, login, giorno,ora) is
    insert into prenotatemp
    values(ide, timestamp, login, giorno, ora)
    if cosaFare = 'ancora'
    then raccogli(ide);
    else conferma(ide);
    end if;
end immOConf;
```

Simulare una sessione

- conferma:

```
insert into prenota
(select login, giorno, ora
  from prenotatemp
  where sessionide = ide)
```

- annulla:

```
delete prenota
where sessionide = ide
```

- Ogni giorno eseguiamo:

```
delete prenotatemp
where timestamp < ieri
```

PSP
PL/SQL Server Pages

Codice PL/SQL Embedded in pagine Web (PL/SQL Server Pages)

- Per includere in una pagina Web contenuto dinamico, in particolare il risultato di una query, si può usare scripting server-side attraverso le PL/SQL Server Pages (PSP)
- questa alternativa può essere più semplice che usare i package HTP e HTF
- lo stesso risultato può quindi essere prodotto in due modi:
  - scrivendo una pagina html con embedded del codice PL/SQL e compilandola come una PL/SQL server page
  - scrivendo una stored procedure completa che produce HTML chiamando i package HTP e OWA

Esempio - Una pagina semplice

```
create or replace PROCEDURE provapl AS
begin
  htp.prn(' <HTML> <HEAD><TITLE>Mostra studenti
              </TITLE></HEAD> <BODY>
              <H1>Elenco studenti </H1>
              <TABLE> <TR> <TD> <I>Cognome</I></TD>
                 <TD> <I>Nome </I> </TD> </TR> ');
  for studente in
          (select nome, cognome from studenti)
  loop
     htp.prn(' <TR> <TD> ` || studente.cognome );
     htp.prn(' </TD> <TD> ` || studente.nome );
     htp.prn(' </TD> </TR> ');
  end loop;
  htp.prn(' </TABLE> </BODY> </HTML> ');
end;
```

Esempio - Passare un parametro

```
create or replace PROCEDURE provaparpl (
  ilCognome IN VARCHAR2)
AS
begin
  htp.prn(' <HTML> <HEAD../HEAD> <BODY>
              <H1>Elenco studenti </H1>
              <TABLE> <TR> <TD> <I>Cognome</I></TD>
                 <TD> <I>Nome </I> </TD> </TR> ');
  for studente in
          (select nome, cognome from studenti s
           where s.Cognome = ilCognome)
  loop
     htp.prn(' <TR> <TD> ` || studente.cognome );
     htp.prn(' </TD> <TD> ` || studente.nome );
     htp.prn(' </TD> </TR> ');
  end loop;
  htp.prn(' </TABLE> </BODY> </HTML> ');
end;
```

Esempio semplice - PSP

```
<%@ page language="PL/SQL"   %>
<HTML><HEAD><TITLE>Mostra studenti</TITLE></HEAD>
<BODY> <H1>Elenco studenti </H1>
<TABLE>
   <TR> <TD> <I> Cognome  </I>  </TD>
        <TD> <I> Nome     </I>  </TD>
   </TR>
<% for studente in
        (select nome, cognome from studenti)
  loop %>
   <TR>
      <TD> <%= studente.cognome %> </TD>
      <TD> <%= studente.nome %> </TD>
   </TR>
<% end loop; %>
</TABLE> </BODY> </HTML>
```

PSP - Il file generato

```
create or replace PROCEDURE provapsp AS
  BEGIN NULL;
  htp.prn(' <HTML> <HEAD><TITLE>Mostra studenti
    </TITLE></HEAD> <BODY> <H1>Elenco studenti </H1>
    <TABLE> <TR>
    <TD> <I><font size=+1> Cognome </font></I> </TD>
    <TD> <I><font size=+1> Nome </font></I> </TD> </TR>
    ');
  for studente in
              (select nome, cognome from studenti)
  loop
    htp.prn(' <TR> <TD> '); htp.prn(studente.cognome);
    htp.prn(' </TD> <TD> '); htp.prn(studente.nome);
    htp.prn(' </TD> </TR> ');
  end loop;
  htp.prn(' </TABLE> </BODY> </HTML> ');
  END;
```

Esempio parametro - PSP

```
<%@ page language="PL/SQL"   %>
<%@ plsql parameter="ilCognome" %>
<HTML><HEAD><TITLE>Mostra studenti</TITLE></HEAD>
<BODY> <H1>Elenco studenti </H1>
<TABLE>
   <TR> <TD> <I> …     </I>  </TD>
   </TR>
<% for studente in
        (select nome, cognome from studenti
         where studenti.cognome = ilCognome)
  loop %>
   <TR>
      <TD> … </TD>
   </TR>
<% end loop; %>
</TABLE> </BODY> </HTML>
```

Formato di file PSP

- il file per una PL/SQL Server Page deve avere estensione .psp
- se è una semplice pagina HTML compilarla come PL/SQL server page produce una stored procedure che fa l'output di tale file HTML
- in genere è un misto di HTML (per le parti statiche della pagina) e PL/SQL (per il contenuto dinamico)
- per identificare un file come PL/SQL Server Page bisogna includere

```
<%@ page language="PL/SQL"    %>
```

- per passare un parametro, usare

```
<%@ plsql parameter="…" %>
```

- il tipo di default è VARCHAR2, si può specificare un tipo diverso con `type ="…"` e il valore di default con `default="…"`

File PSP

- il file per una PL/SQL Server Page deve avere estensione .psp
- se è una semplice pagina HTML compilarla come PL/SQL server page produce una stored procedure che fa l'output di tale file HTML
- in genere è un misto di HTML (per le parti statiche della pagina) e PL/SQL (per il contenuto dinamico)
- per identificare un file come PL/SQL Server Page bisogna includere

```
<%@ page language="PL/SQL"    %>
```

- per passare un parametro, usare

```
<%@ plsql parameter="…" %>
```

- il tipo di default è VARCHAR2, si può specificare un tipo diverso con `type ="…"` e il valore di default con `default="…"`

File PSP

- Per default, la stored procedure generata ha il nome del file, per specificare un altro nome usare

  ```
  <%@ page procedure="..."    %>
  ```

- per dichiarare variabili globali si può includere un blocco di dichiarazioni usando

  ```
  <%! %>
  ```

- tutti i comandi PL/SQL (e blocchi) possono essere inclusi tra

  ```
  <% %>
  ```

- per includere un valore che è il risultato di un'espressione PL/SQL, si include l'espressione tra

  ```
  <%= %>
  ```

- nelle pagine PSP si possono usare comunque le funzioni dei package HTP e OWA (nel codice PL/SQL)

PSP

- Per caricare uno o più file PSP nel database come stored procedure usare

  loadpsp –replace –user *username/password[@connect_string] psp_file_name*

- es.
  - **loadpsp –replace –user *user/pwd*@oracledb.datatop provaparpsp.psp**

- la risposta che si riceve è:
  - **"....psp": procedure "..." created.**

JavaScript

JavaScript

- Linguaggio definito da Netscape
- JScript: la versione MicroSoft (basata su ECMAScript)
- Serve ad arricchire una pagina HTML con codice da eseguirsi sul client
  - es.
  - semplici controlli sull'input
  - parte di form che appare solo quando una checkbox viene selezionata

Variabili, Operatori, Commenti

- Variabili con tipo, ma non dichiarato (e conversione implicita a string)
  - var x = 5
  - var s = "luigi"
  - s + x -> "luigi5"
- Tipi: numerici, stringhe, bool, funzioni, oggetti, null
- Identificatori: lettere+_, case sensitive, anche interi
- Terminazione dei comandi: newline, ;, o entrambi
- Operatori del C: +,-,*,/,%, &&,||, ==, !=, <, >
- && ed || sono valutati in modo ordinato
- Commenti: da // a fine linea (consigliato) e tra /* e */

Costanti

- 3.9 // numeric literal
- "Hello!" // string literal
- "Perche•" // string literal
- •Value = "aa"• // string literal
- false // boolean literal
- null // literal null value
- {x:1, y:2} // Object literal
- [1,2,3] // Array literal
- function(x){return x*x;} // function literal

Coercion

- Stringhe, booleani, e interi sono convertiti mutuamente se necessario
- Ad esempio:
  - "a" + 1 -> "a1"
- parseInt("123") si usa per convertire una stringa in un intero (accetta anche: 123abc)

Esempio di codice

```
var parsedF = parseInt(document.Forma1.Anno.value);
if (isNan(parsedF)) {
  alert(document.Forma1.Anno.value +
                        "is not an integer!");
} else {
  document.Forma1.Anno.value = parsedF;
}
```

Comandi: if e while

- If (attenti alle linee):

```
if ( cond ) {
comandi
}
```

- Oppure:

```
if ( cond ) {
comandi
} else {
comandi
}
```

- While (attenti alle linee):

```
while ( cond ) {
comandi
}
```

For e Funzioni

- for:

```
for (init; cond; incr) {
comandi
}
```

- Funzioni:

```
function NOME (listaParams) {
body
}
```

- parametri separati da virgole, valore ritornato con **return** (valore)
- i parametri sono passati per valore

Stringhe

- Concatenazione: +
- Alcuni metodi:
  - stringa.length
  - stringa.substring(start,end)
  - stringa.substr(start,length)
  - stringa.charAt(index)
- JScript supporta le espressioni regolari

Eventi gestiti dal browser

- Di pagina:
  - loading, unloading
- Associati ai bottoni:
  - click, submit
- Associati ai campi di tipo text e select:
  - change
  - select: selezionare una porzione di testo (non nei componenti select)
  - focus/blur: rendere il campo pronto ad accettare input

Associare codice ed eventi

- Attributo *onEvento*, per i componenti di una form:
  - **<INPUT TYPE="button" NAME="mycheck" VALUE="HA!" onClick="alert('I told you not to click me');">**
  - Il valore dell'attributo è un pezzo di codice che gestisce l'evento
- Attributo *onSubmit*, per l'intera form; se la funzione ritorna false, la sottomissione non avviene:
  - **<FORM NAME="formname" ... onSubmit="submithandler()">**
- Per il documento, *onLoad*, *onUnload*:
  - **<BODY onLoad="loadfunc()" onUnload="unloadfunc()">**

Il Tag SCRIPT

- Meglio metterlo nello head
- Carica da file:

```
<SCRIPT LANGUAGE="JavaScript"
   SRC="jscode/click.js">
</SCRIPT>
```

- Codice immediato: tra <SCRIPT> e </SCRIPT>, meglio se commentato con <!-- -->:

```
<!--
function dontclickme()      {
      alert("I told you not to click me");
      return( false );
}
<!-- end script -->
```

Leggere e scrivere i campi di una form

- Se la form si chiama myForm, con un campo text chiamato myText, posso scrivere, in una funzione:
  - document.myForm.myText.value += 1;
- Oppure, nel tag del campo:
  - onChange = "this.value += 1"
- Per un campo select, posso accedere alla prima opzione scrivendo:
  - document.myForm.mySelect.options[0].value += 1;
- Posso accedere all'opzione corrente scrivendo:
  - document.myForm.mySelect.options[document.myForm.mySelect.selectedIndex].value += 1;

Una funzione che fa un controllo

```
function checkit() {
    var strval = document.myform.mytext.value;
    var intval = parseInt(strval);
    if ( 0 < intval && intval < 10 ) {
        return( true );
    } else {
        alert("Value " + strval + " out of range");
        return( false );
    }
}
```