GAMS/ORAQUEUE: A queueing batch system for GAMS jobs in an Oracle environment

ERWIN KALVELAGEN*

August 16, 2002

1 Introduction

This document describes a package for queueing GAMS jobs in an Oracle database environment. Jobs are executed when resources are available, and are otherwise queued for later execution. To demonstrate and test the concept in an actual application, we implemented a Web-based job submission tool. All the software is implemented in Oracle using PL/SQL and Java stored procedures.

The complete sources are available from the author at no charge. It is noted however that to install and customize the system to your particular needs will require a reasonable knowledge of Oracle, Oracle Application Server and PL/SQL.

2 Design

2.1 Polling

The conceptually simplest way to implement a queueing system is to use a simple queue table containing waiting jobs, and an external program that periodically checks this table to see if there are jobs to run. The external program is preferably running as a background process. On UNIX systems this means a *daemon* and on Windows NT systems such a background process can be implemented as an *NT Service*.

A polling loop design has a number of drawbacks. First, a polling system will waste resources (database connections, query processing) as it will query the database even if there are no jobs waiting. It is not always obvious what the polling interval should be. Choosing a resolution that is too fine may cause degradation in system performance while a resolution that is too coarse will cause longer than needed job turn around times due to extra latency in the system. This also causes the system to be idle (i.e. not running GAMS jobs) while in fact there is capacity to run jobs.

Using an external program (daemon or NT service) causes additional complications. A decent implementation of a daemon or service is not a trivial task [6, 7, 24]. In addition the task to manage both a database and a separate program gives extra complexity. Issues like starting and stopping the daemon, error logging, security, permissions and system documentation can make this solution less attractive in a production environment. An alternative is to implement the polling loop not outside the database in an external background process but inside the database. Modern databases such as Oracle provide excellent programming environments including PL/SQL [16], a language loosely based on Ada, and Java [13, 15]. Other database systems such as SQL Server from Microsoft and DB2 from IBM provide similar facilities: SQL Server uses Transact SQL and DB2 uses typically SQL and Java. One problem is that relational databases are by nature event driven: things happen when a stored procedure is called or a trigger is launched. Some databases have facilities to run periodically a task, such as an alert, but these tasks are to be run every hour, day or week, i.e. at a much coarser interval than our polling process would require.

2.2 Event driven approach

An alternative is not to poll, but to use an event driven state machine. In this case we only do management tasks when the state of the system is changed due to an event. An event can be a submission of a new job, or

*GAMS Development Corporation, Washington D.C.

the finishing of a running job. This method can be implemented inside the database using stored procedures and triggers. The complexity of such a system is somewhat larger, but it has the advantage that only processing power is needed when actually processing needs to be performed. In addition, there is no idling when jobs are available to be executed.

The state of the queue can be described with two integers:

State variables:

$nJobsWaiting$ { number of jobs in the queue to be executed }
$nJobsRunning$ { number of jobs currently running }

There are two parameters that indicate the capacity of the system:

Parameters:

$nMaxJobsWaiting$ { maximum number of jobs waiting to be executed }
$nMaxJobsRunning$ { maximum number of jobs that can be executing at the same time }

I.e. we have the following allowable ranges:

$nJobsWaiting = 0, .., nMaxJobsWaiting$
$nJobsRunning = 0, .., nMaxJobsRunning$

The following constraints can be applied to the system:

$nJobsRunning < nMaxJobsRunning \Rightarrow nJobsWaiting = 0$
$nJobsWaiting > 0 \Rightarrow nJobsRunning = nMaxJobsRunning$

I.e. if we have capacity available to run extra jobs, the waiting queue must be empty (otherwise we could have started executing a waiting job), and if there are jobs in the waiting queue, then the execution system is working at full capacity.

The following events can be defined:

Events:

$IncomingJob$ { a new job is submitted by a user }
$FinishingJob$ { a job finished executing }
$KillingJob$ { a job needs to be removed from the system }

The initial state of the system is:

Initial state:

$nJobsWaiting = 0$
$nJobsRunning = 0$

There is no final state in this system. The following state transitions in response to events can be defined

IncomingJob:

if $nJobsRunning < nMaxJobsRunning$ **then**
  $nJobsRunning := nJobsRunning + 1$ { execute job }
else if $nJobsWaiting < nMaxJobsWaiting$ **then**
  $nJobsWaiting := nJobsWaiting + 1$ { put job in queue }
else
  error(System full)
end if

FinishingJob:

if $nJobsWaiting > 0$ **then**
  $nJobsWaiting := nJobsWaiting - 1$ { execute waiting job }
else
  $nJobsRunning := nJobsRunning - 1$ { no waiting jobs }
end if

KillingJob:

if $JobID \in JobsWaiting$ **then**
  $nJobsWaiting := nJobsWaiting - 1$ { remove from queue }
else if $JobID \in JobsRunning$ **then**
  raise event $FinishingJob$
else
  error(Job not found)
end if

3 Implementation

3.1 Oracle

A prototype implementation has been built using Oracle 9i on Windows 2000. Most of the functionality is implemented using PL/SQL stored procedures, conveniently packages as a socalled *package*.

We have followed more or less the advice of [4]: if possible use SQL, then try to do it in PL/SQL, if this turns out impossible use Java and as a last resort use external DLL's written in C. Most of the system is coded in PL/SQL, with the GAMS execution part being written in Java. We did not have to use external DLL's.

Oracle 9i is a large software system and in the following discussion we will cover quite some ground. If you are very unfamiliar with relational database systems in general and Oracle in particular, you may want to skip reading this section.

3.2 Background processing

To keep the system responsive we needed a way to process GAMS models in the background. I.e. the calling procedure should return before the job is finished. Our initial thought was that Java could provide us with the needed multi-threading functionality to implement this. Unfortunately, the Java virtual machine (JVM) inside Oracle has a somewhat simpler threading model, causing this approach not to work. We therefore used the PL/SQL package DBMS_JOB [19] to implement background processing. DBMS_JOB is normally used to schedule jobs at a later time. However, in our implementation, we don't use the scheduling part, but only use the background processing feature of DBMS_JOB. Another possible way to implement this system would be the use of the Advanced Queueing mechanism in Oracle 9i [11, 19]. We have not investigated this possibility further.

The use of DBMS_JOB requires proper configuration of the Oracle Server: a number of background processes need to be specified.

3.3 Running external programs

As some stage in the process we need to run GAMS from withing Oracle. This means calling an external program. Before Oracle 8, the usual way to implement such a calling mechanism was to use the package DBMS_PIPE. This requires a reasonable amount of programming: an external listener program needs to be written that can exchange data through a pipe. With Oracle 8 two new ways of talking to the external world were introduced: the possibility to call external Dynamic Link Libraries (DLL's) or shared objects as they are called in the UNIX word, and using Java's *exec()* method from the *Runtime* class. We have used the Java approach, as this gives us more security and protection against programming mistakes (Java stored procedures live in a protected "sandbox", so that they cannot harm the database server), and gives us platform independence (sometimes somewhat optimistically described as write-once-run-anywhere).

The use of *exec()* requires additional privileges, as this is a potential harmful operation.

The current implementation does not allow to kill a running job. Of course it is possible to kill a job externally using the Windows NT Task Manager. In addition there is no functionality te set limits on the job, like maximum memory usage and maximum CPU time. It would be possible to add this by writing a wrapper for `gams.exe` that enforces these restrictions. On UNIX systems this is somwhat simpler: several limits can be specified for a shell (e.g. the c-shell uses `limit` and the Korn and bash shell has `ulimit`). Example:

```
[erwin@webster ~]$ limit
cputime         unlimited
filesize        unlimited
datasize        unlimited
stacksize       8192 kbytes
coredumpsize    unlimited
memoryuse       unlimited
descriptors     1024
memorylocked    unlimited
maxproc         1023
openfiles       1024
[erwin@webster ~]$
```

3.4 Platform independence

The whole system is virtually platform independent. All PL/SQL packages, SQL scripts and Java classes can be compiled and executed on any machine on which Oracle 9i is installed. One exception comes to mind: the path where GAMS is installed is installation dependent, and this string will need to be changed accordingly during installation time of the PL/SQL packages.

3.5 The QADMIN table

The queueing system is built around the table QADMIN. Here is the description:

```
SQL> describe qadmin
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 JOBID                                     NOT NULL VARCHAR2(28)
 JOBSTATUS                                          CHAR(1)
 USERNAME                                           VARCHAR2(80)
 PASSWORD                                           VARCHAR2(80)
 LONGDESCRIPTION                                    VARCHAR2(255)
 SHORTDESCRIPTION                                   VARCHAR2(80)
 JOBPRIORITY                                        NUMBER(38)
 JOBCREATIONDATE                                    TIMESTAMP(6)
 JOBCOMPLETIONDATE                                  TIMESTAMP(6)
 EMAIL                                              VARCHAR2(80)
 LOG                                                BLOB
 LISTING                                            BLOB
 FILE1                                              BLOB
 FILE2                                              BLOB
 FILE3                                              BLOB
 FILE4                                              BLOB
 FILE5                                              BLOB
 FILENAME1                                          VARCHAR2(80)
 FILENAME2                                          VARCHAR2(80)
 FILENAME3                                          VARCHAR2(80)
 FILENAME4                                          VARCHAR2(80)
 FILENAME5                                          VARCHAR2(80)
 EXECERROR                                          VARCHAR2(255)

SQL>
```

Here follows a short description of the columns:

jobid This is a unique identifier for each job. To make sure this really unique we use an Oracle sequence to generate this id, i.e. the user does not have to provide this, but instead it is generated in the application code. The jobid is passed on from one PL/SQL procedure to another to identify the job in question.

jobstatus A letter indicating the status of the job. Most commonly you will see 'W' for waiting jobs, 'R' for running jobs and 'F' for finished jobs. Occasionally one could see 'S' for a job that is being submitted for background processing (it should shortly change into 'R'), or 'E' for a job that could not be executed due to an error occurring in the Java class that executes GAMS jobs.

username Name of the user. Together with the password this is used to search for jobs submitted by a certain user.

password Password of the user. Note that this is not encrypted, so this is relatively unsecure. It is mainly used to find jobs that belong to a certain user: a user/password combination minimizes the chance that two users use the same indentification.

shortdescription Short description of the job. This can be used to identify the job from a list of jobs submitted.

longdescription This longer description is used to tell more about the job.

jobpriority The job priority is used in scheduling jobs. Jobs with higher priority will be executed before jobs with a lower priority. Otherwise the submission date is used: older jobs are executed before newer ones.

jobcreationdate The timestamp indicating when the job was inserted in the QDAMIN table.

jobcompletiondate The timestamp indication when the job was finished executing. If the job is not finished yet, this field is NULL.

email This is the email address that is used to notify the user when a job is completed. If this field is NULL or contains an empty string, sending the mail is skipped.

log BLOB containing the GAMS log file. This field is NULL is the job has not finished. The log file is what is normally sent to the screen.

listing BLOB containing the GAMS listing file. This field is NULL is the job has not finished.

file1,..,file5 Input files. The first file **file1** should be the main .gms file. Other files are optional, and can contain include files or solver option files.

filename1,..,filename5 The file names for the input files.

execerror If we could not execute the job, this will contain the exception message.

The implementation of the state variables `nJobsRunning` and `nJobsWaiting` is done as follows. A small table `qcounter` is used to store these values:

```
SQL> describe qcounter
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 NAME                                      NOT NULL VARCHAR2(16)
 VALUE                                              NUMBER

SQL> select * from qcounter;

NAME                 VALUE
---------------- ----------
nRunningJobs              0
nWaitingJobs              0

SQL>
```

To make sure these values are kept in sync with the number or records in the table `qadmin` with job status 'W', 'R' or 'S' we use a set of triggers on the `qadmin` table. To interrogate the number of waiting jobs, we call a stored function `nJobsWaiting`, which is implemented as follows:

```
function nJobsWaiting return integer is
--
-- return the number of waiting jobs
--
   k integer;
begin
   select value into k from qcounter where name = 'nWaitingJobs';
   return k;
end;
```

The hope is that this approach is faster than counting each time the number of rows in the `qadmin` table with a certain job status.

The following script will install the tables:

gqtable.sql

```
-------------------------------------------------------------
--
-- GAMS/ORAQUEUE table creation script
--
-- Author:       Erwin Kalvelagen (erwin@gams.com)
-- Date:         Feb 2002
-- Requirements: Oracle 8i,9i
-- Called by:    all.sql
--
-------------------------------------------------------------
--
-- description of qadmin:
--  JobId              : Unique ID for each job (created from sequence)
--  JobStatus          : 'W' (waiting), 'R' (running), 'F' (finished)
--                       'S' (submitting)
--  UserName           : name of the user
--  JobPriority        : integer indicating priority used for scheduling purposes
--  email              : email address for notification
--  JobCreationDate    : date/time when submitted
```

```
--  JobCompletionDate  : date/time when finished
--  ShortDescription   : short identification of the job
--  LongDescription    : longer description
--  file1              : GMS source text
--  file2..5           : other files (include files, option files)
--  filename1..5       : filenames
--  Listing            : LST text
--  Log                : LOG text
--  ExecError          : exec() error message
--
-- note: the sequence is actually a number, but we store it
-- as a string, as this is a better data type for an Id (one
-- cannot add an id).
--


-- this is optional:
drop sequence IdSequence;
drop table qadmin;
drop table qcounter;

create sequence IdSequence;

create table qadmin (
        JobId             varchar2(28) primary key,
        JobStatus         char(1),
        UserName          varchar2(80),
        PassWord          varchar2(80),
        LongDescription   varchar2(255),
        ShortDescription  varchar2(80),
        JobPriority       integer,
        JobCreationDate   timestamp,
        JobCompletionDate timestamp,
        email             varchar2(80),
        log               blob,
        listing           blob,
        file1             blob,
        file2             blob,
        file3             blob,
        file4             blob,
        file5             blob,
        filename1         varchar2(80),
        filename2         varchar2(80),
        filename3         varchar2(80),
        filename4         varchar2(80),
        filename5         varchar2(80),
        ExecError         varchar2(255)
        );

commit;

--
-- table qcounter holds two numbers that correspond to
-- the number of waiting and running jobs
--

create table qcounter(
        name              varchar2(16) primary key,
        value             number
        );

commit;

insert into qcounter values ('nRunningJobs',0);
insert into qcounter values ('nWaitingJobs',0);

commit;
```

3.6 Scheduling

Jobs that arrive when there is capacity to execute them immediately are scheduling for direct execution. Otherwise, the job is stored in the `qadmin` table with a job status of 'W'. Once the system finishes a job, it will inspect the table to see if there are waiting jobs. If this is the case, two factors determine which job is chosen. First it looks at the priority of the job. It considers jobs that have the highest priority for execution. If there are multiple jobs waiting with the same maximum priority, the oldest job is selected. Age is measured by the time the job was inserted in the `qadmin` table.

The following select will implement this:

```
select JobId into id from
  (select * from qadmin where JobStatus='W'
              order by JobPriority desc nulls last, JobCreationDate asc nulls last)
  where RowNum < 2;
```

The condition `where RowNum < 2` ensures that only the first record is returned.

3.7 Large objects

The input files (the main .gms file and the additional include files and option files), the listing file and the log file are stored as a BLOB: Binary Large Object [12]. Although we could have used CLOB's (Character Large Objects), it turns out that the PL/SQL Gateway stores files that are uploaded as BLOB's. As we wanted to use that facility to use Web based job submission, we decided to use BLOB's instead.

When the log or listing file is to be passed on to a browser, we need to pass it on as character data. This can be done easily using the `CAST_TO_VARCHAR2` function in the supplied package `UTL_RAW`.

3.8 E-mail notification

When a job is finished, we send an e-mail to the user with the listing and log file as an attachment if the `email` field is not NULL or empty. We use for this the supplied PL/SQL package UTL_SMTP [19]. The MIME attachment handling is based on some existing code we found on the Web [25].

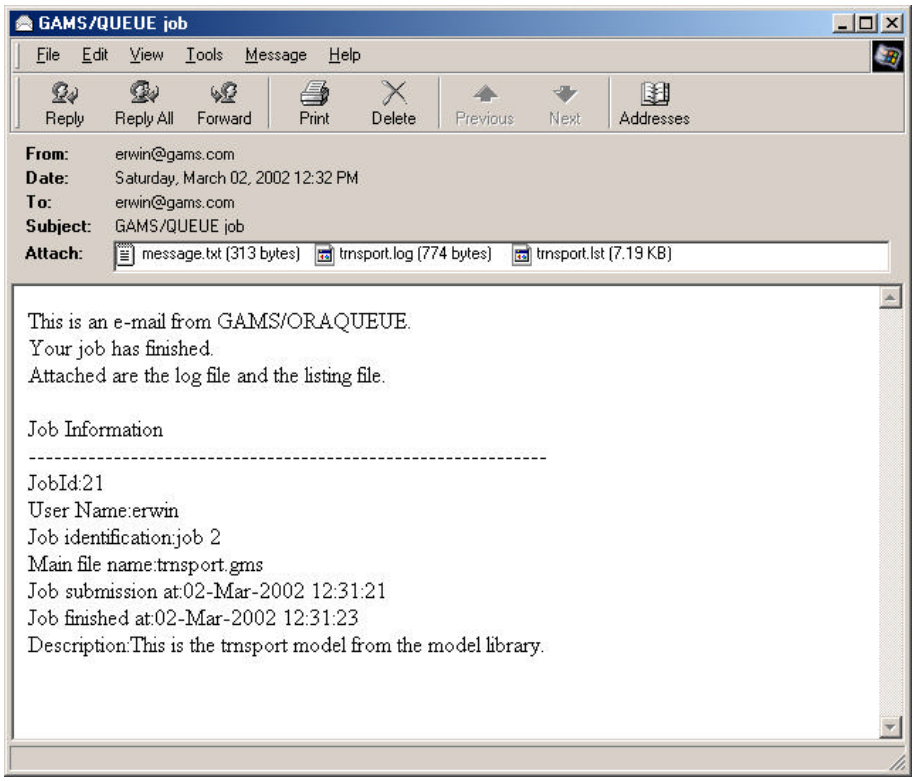

Figure 1: E-mail notification

The configuration of the mailer is done by two constants in the `gamsqueue` package interface located in `gqiface.sql`: `smtp_server` is the name of the SMTP server that will forward the e-mail message and `smtp_port` is the port number used on the SMTP server for incoming messages (this is usually port 25).

The sending of large messages can be slow. However, the system performs this task as part of a background process, so the user will not be aware of this.

3.9 Password protection

The system uses a username/password scheme to establish ownership of a model. Notice that the password is not encrypted. The actual additional protection it offers is therefore limited.

3.10 Put files

Currently there is no support for storing PUT files written by a GAMS job. It is not very difficult to add support to store PUT files with certain fixed names or fixed extensions. I.e. always store the file `RESULT.PUT` or all the files `*.PUT`. To detect if any PUT files are written and what name they have not directly available, although a hack would be to inspect the bottom of the listing file where such information is listed.

3.11 Submitting a job

To submit a new job to the queueing system, call the function

```
function NewJob(UserName in tUserName,
                Password in tPassWord,
                Priority in tJobPriority,
                Email in tEmail,
                ShortDescription in tShortDescription,
                LongDescription in tLongDescription,
                Blob1 in tBlob,
                Blob2 in tBlob,
                Blob3 in tBlob,
                Blob4 in tBlob,
                Blob5 in tBlob,
                Filename1 in tFileName,
                Filename2 in tFileName,
                Filename3 in tFileName,
                Filename4 in tFileName,
                Filename5 in tFileName) return tid; -- create a new job
```

The id is returned which is the primary key of the `qadmin` table, which allows easy access to any information one may need to know about the job. I.e. the status of a job can be interogated by querying the `qadmin` table. The full description of the types can be found in `gqiface.sql`.

The programmatic interface to the queueing system is simple enough to make it fairly straightforward to integrate into a larger software system.

3.12 Source files

Here follow the source files that implement the GAMS queueing mechanism.

gqiface.sql

```
-------------------------------------------------------------
-- Package Interface GAMSQUEUE
-- Author: Erwin Kalvelagen (erwin@gams.com)
-- Date: feb 2002
-- Requirements: Oracle 8i,9i
-------------------------------------------------------------


create or replace package gamsqueue as

    nMaxJobsWaiting constant int := 1000;  -- max number of jobs in waiting queue
    nMaxJobsRunning constant int := 2;     -- max number of jobs running at the same time

    GamsExecuteDirectory constant varchar2(80) := 'C:\tmp\';
                                       -- this is the place where subdirectories for gams
                                       -- files are created. I.e. it needs enough free space
                                       -- to accomodate listing and log files and GAMS scratch
                                       -- files. Should include trailing '\'.

    GamsExeName constant varchar2(80) := 'g:\win9x program files\gams20.5\gams.exe';
                                       -- this is a fully qualified path for gams.exe


    smtp_server constant varchar2(80) := '192.168.1.101'; -- IP address or host name of the SMTP server
    smtp_port constant number := 25;      -- smtp port number
```

```
    email_from_address varchar2(80) := 'erwin@gams.com'; -- from address in email

    subtype tBlob is blob;                -- our blob type
    subtype tFileName is varchar2(80);    -- type to hold a file name
    type tBlobArray is varray(5) of tBlob; -- holds the .gms file and include, option files
    type tFileNames is varray(5) of tFileName; -- holds the filenames of the BLOB's
    subtype tStatus is char(1);           -- 'W','R','F': waiting, running or finished
    subtype tId is varchar2(32);          -- job identifier, should be unique (generated by sequence)
    subtype tUserName is varchar2(80);    -- user name, can be anything
    subtype tPassWord is varchar2(80);    -- password
    subtype tLongDescription is varchar2(255); -- longer description of the purpose of this job
    subtype tShortDescription is varchar2(80); -- short description of the purpose of this job
    subtype tJobPriority is integer;      --  higher number is higher priority
    subtype tEmail is varchar2(80);       --  e-mail address to send results

    type tjob is record (
            Id                  tId,
            Status              tStatus,
            UserName            tUserName,
            PassWord            tPassWord,
            Priority            tJobPriority,
            Email               tEmail,
            ShortDescription    tShortDescription,
            LongDescription     tLongDescription,
            Blobs               tBlobArray,
            FileNames           tFileNames
    ); -- job type


    function nJobsWaiting return integer;   -- actual number of jobs in waiting queue
    function nJobsRunning return integer;   -- actual number of jobs running


    function NewJob(UserName in tUserName,
                    Password in tPassWord,
                    Priority in tJobPriority,
                    Email in tEmail,
                    ShortDescription in tShortDescription,
                    LongDescription in tLongDescription,
                    Blob1 in tBlob,
                    Blob2 in tBlob,
                    Blob3 in tBlob,
                    Blob4 in tBlob,
                    Blob5 in tBlob,
                    Filename1 in tFileName,
                    Filename2 in tFileName,
                    Filename3 in tFileName,
                    Filename4 in tFileName,
                    Filename5 in tFileName) return tid ; -- this is called by external programs


    --
    -- these are mainly for internal use
    --
    function GetGamsExecutionDirectory return varchar2; -- called by Java Stored Procedure ExecuteJobJ
    function GetGamsExe return varchar2; -- called by Java Stored Procedure ExecuteJobJ
    procedure RunJob(id in tid); -- called by DBMS_JOB
    procedure IncrementWaiting; -- called by trigger
    procedure IncrementRunning; -- called by trigger
    procedure DecrementWaiting; -- called by trigger
    procedure DecrementRunning; -- called by trigger


end gamsqueue;

/

show errors;
```

gqbody.sql

```
-------------------------------------------------------------
--
-- Package Body GAMS/ORAQUEUE
--
-- Author:       Erwin Kalvelagen (erwin@gams.com)
-- Date:         Feb/Mar 2002
-- Requirements: Oracle 8i,9i
--               job_queue_processes in ora.init is > 0
-- Called by:    all.sql
-- To do:        1. row locking (also in java part)
```

```
--               2. check changestatus: what happens if no data found
--
------------------------------------------------------------

create or replace package body gamsqueue as


function qcountQuery(nam in varchar2) return integer is
--
-- query the qcounter table
--
   k integer;
begin
   select value into k from qcounter where name = nam;
   return k;
end qcountQuery;

procedure qcountIncrement(nam in varchar2) is
--
-- increment value
--
begin
   update qcounter set value = value + 1 where name = nam;
end qcountIncrement;

procedure qcountDecrement(nam in varchar2) is
--
-- decrement value
-- make sure we don't go below 0
--
begin
   update qcounter
   set value = case when value-1 < 0 then 0 else value - 1 end
   where name = nam;
end qcountDecrement;


function nJobsWaiting return integer is
--
-- return the number of waiting jobs
--
begin
   return qcountQuery('nWaitingJobs');
end nJobsWaiting;

function nJobsRunning return integer is
--
-- return the number of running jobs
--
begin
   return qcountQuery('nRunningJobs');
end nJobsRunning;


procedure IncrementWaiting is
--
-- called by trigger
--
   too_many_waiting_jobs exception;
begin
   if nJobsWaiting >= nMaxJobsWaiting then
      raise too_many_waiting_jobs;
   end if;

   qcountIncrement('nWaitingJobs');
end IncrementWaiting;

procedure IncrementRunning is
--
-- called by trigger
--
   too_many_running_jobs exception;
begin
   if nJobsRunning >= nMaxJobsRunning then
      raise too_many_running_jobs;
   end if;

   qcountIncrement('nRunningJobs');
end IncrementRunning;

procedure DecrementWaiting is
--
-- called by trigger
--
begin
```

```
   qcountDecrement('nWaitingJobs');
end DecrementWaiting;


procedure DecrementRunning is
--
-- called by trigger
--
begin
   qcountDecrement('nRunningJobs');
end DecrementRunning;


procedure InsertJob(r in tjob) is
--
-- insert SQL statement
--
begin

   insert into qadmin
   values (r.Id,
           r.Status,
           r.UserName,
           r.PassWord,
           r.LongDescription,
           r.ShortDescription,
           r.Priority,
           SYSTIMESTAMP,  -- jobcreationdate
           NULL,          -- jobcompletiondate
           r.Email,
           NULL,          -- log
           NULL,          -- listing
           r.blobs(1),
           r.blobs(2),
           r.blobs(3),
           r.blobs(4),
           r.blobs(5),
           r.filenames(1),
           r.filenames(2),
           r.filenames(3),
           r.filenames(4),
           r.filenames(5),
           NULL
      );
   commit;

end InsertJob;

procedure ChangeStatus(id in tid, oldstatus in char, newstatus in char) is
--
-- update SQL statement: change status of job identified by id
--
-- in rare occasions this can give a no_data_found exception: another
-- session already changed the status.
--
begin

   update qadmin
   set JobStatus = newstatus where JobId = id and JobStatus = oldstatus;

   commit;

end ChangeStatus;


procedure SubmitJob(id in tid) is
--
-- submit the job to oracle background queue
--
   JobNo binary_integer;  -- job number for DBMS_JOB package
   Job varchar2(100) := 'gamsqueue.runjob(''''||id||''''); -- actual job (i.e. procedure call)
begin
   ChangeStatus(id,'W','S');
   DBMS_JOB.SUBMIT(JobNo,Job);
   commit;
end;


function NewJob(UserName in tUserName,
                Password in tPassWord,
                Priority in tJobPriority,
                Email in tEmail,
                ShortDescription in tShortDescription,
                LongDescription in tLongDescription,
                Blob1 in tBlob,
```

```
                Blob2 in tBlob,
                Blob3 in tBlob,
                Blob4 in tBlob,
                Blob5 in tBlob,
                Filename1 in tFileName,
                Filename2 in tFileName,
                Filename3 in tFileName,
                Filename4 in tFileName,
                Filename5 in tFileName) return tid is

--
-- incoming job, this is called from the outside world
-- to indicate a new job is to be added to the queue.
-- If we can execute it immediately, do so.
--

   r tjob;
   id tid;
   too_many_waiting_jobs exception;
begin
   --
   -- return immediately if it looks we don't have space
   --
   if (nJobsWaiting >= nMaxJobsWaiting) then
      raise too_many_waiting_jobs;
   end if;

   --
   -- create unique id
   --
   select IdSequence.nextval into id from dual;

   --
   -- fill job record and insert it.
   -- as id (primary key) is taken from a sequence this will
   -- always work.
   --
   r.Id := id;
   r.UserName := UserName;
   r.PassWord := PassWord;
   r.ShortDescription := ShortDescription;
   r.LongDescription := LongDescription;
   r.Email := Email;
   r.Priority := Priority;
   r.Blobs := tBlobArray(Blob1,Blob2,Blob3,Blob4,Blob5);
   r.FileNames := tFileNames(FileName1,FileName2,FileName3,FileName4,FileName5);
   r.status := 'W'; -- mark as waiting
   InsertJob(r);

   if (nJobsRunning < nMaxJobsRunning) then
       SubmitJob(id);
   end if;

   return id;


exception

    when NO_DATA_FOUND then
        -- submit job concluded that job 'id' is already no longer waiting
        -- i.e. another process picked it up already. No problem.
        return id;


end NewJob;


procedure SelectWaitingJob is
--
-- select a waiting job to be executed
--

   id tid;
begin

    select JobId into id from
      (select * from qadmin where JobStatus='W'
                    order by JobPriority desc nulls last, JobCreationDate asc nulls last)
        where RowNum < 2;



    SubmitJob(id);

exception
```

```
    when NO_DATA_FOUND then
        -- no remaining jobs to be executed
        null;

end SelectWaitingJob;



procedure send_blob(conn in out nocopy utl_smtp.connection,
                    boundary in varchar2,
                    filename in varchar2,
                    file in out nocopy blob) as
--
-- send blob
--
   crlf varchar2(2):= chr(13) || chr(10);
   pos integer;
   cnt integer;
   buffer raw(32767);
begin

   if (file is NULL) then
      return;
   end if;

   utl_smtp.write_data(conn, crlf || boundary || crlf ||
                             'Content-Type: application/octet-stream; name="' || filename || '"' || crlf ||
                             'Content-Disposition: attachment; filename="' || filename || '"' || crlf ||
                             'Content-Transfer-Encoding: 8bit' || crlf || crlf) ;

   pos := 1;
   cnt := 32767;
   dbms_lob.open(file,dbms_lob.lob_readonly);
   loop
       dbms_lob.read(file,cnt,pos,buffer);
       utl_smtp.write_raw_data(conn,buffer);
       pos := pos + cnt;
   end loop;

--
-- This will always try read too much. We exploit this by
-- putting the close in the exception handler.
--

exception
   when no_data_found then
      dbms_lob.close(file);

end send_blob;

procedure SendEmail(id in tid) is
--
-- send email message
--
   conn utl_smtp.connection;
   email_to_address tEmail;
   crlf varchar2(2):= chr(13) || chr(10);
   subject varchar2(80);
   message varchar2(255);
   user_name varchar2(80);
   main_file_name varchar2(80);
   short_desc varchar2(80);
   long_desc varchar2(255);
   creation_date timestamp;
   finishing_date timestamp;
   logfile blob;
   logfilename varchar2(80);
   listingfile blob;
   listingfilename varchar2(80);
   k integer;
begin
   --
   -- get email address etc.
   --
   select email,username,filename1,shortdescription,longdescription,jobcreationdate,jobcompletiondate,log,listing
   into email_to_address,user_name,main_file_name,short_desc,long_desc,creation_date,finishing_date,logfile,listingfile
   from qadmin where JobId=id;
   if (email_to_address is NULL) then
      return;
   end if;

   --
   -- open connection
   --
```

```
   conn := utl_smtp.open_connection(smtp_server,smtp_port);

   --
   -- initial handshaking
   --
   utl_smtp.helo(conn, smtp_server);
   utl_smtp.mail(conn, email_from_address);
   utl_smtp.rcpt(conn, email_to_address);

   --
   -- message
   --
   subject := 'GAMS/QUEUE job ';
   message := 'This is an e-mail from GAMS/ORAQUEUE.'|| crlf ||
              'Your job has finished.' || crlf ||
              'Attached are the log file and the listing file.'|| crlf || crlf;


   utl_smtp.open_data(conn);
   utl_smtp.write_data(conn, 'Date: ' || TO_CHAR(SYSDATE,'dd Mon yy hh24:mi:ss') || crlf);
   utl_smtp.write_data(conn, 'From: ' || email_from_address || crlf);
   utl_smtp.write_data(conn, 'Subject: ' || subject || crlf);
   utl_smtp.write_data(conn, 'To: ' || email_to_address || crlf);
   utl_smtp.write_data(conn, 'Mime-Version: 1.0' || crlf);
   utl_smtp.write_data(conn, 'Content-Type: multipart/mixed; boundary="GamsQueue.Boundary.918273645"' || crlf);
   utl_smtp.write_data(conn, '' || crlf);
   utl_smtp.write_data(conn, 'This is a Mime message sent by GAMS/ORAQUEUE.' || crlf);
   utl_smtp.write_data(conn, '' || crlf);

   utl_smtp.write_data(conn, '--GamsQueue.Boundary.918273645' || crlf);
   utl_smtp.write_data(conn, 'Content-Type: text/plain; name="message.txt"; charset=US-ASCII' || crlf);
   utl_smtp.write_data(conn, 'Content-Disposition: inline; filename="message.txt"' || crlf);
   utl_smtp.write_data(conn, 'Content-Transfer-Encoding: 7bit' || crlf);
   utl_smtp.write_data(conn, '' || crlf);
   utl_smtp.write_data(conn, message);
   utl_smtp.write_data(conn, 'Job Information' || crlf ||
                             '------------------------------------------------------------' || crlf );
   utl_smtp.write_data(conn, 'JobId:'||id||crlf);
   utl_smtp.write_data(conn, 'User Name:' || user_name || crlf);
   utl_smtp.write_data(conn, 'Job identification:' || short_desc || crlf);
   utl_smtp.write_data(conn, 'Main file name:' || main_file_name || crlf);
   utl_smtp.write_data(conn, 'Job submission at:' || to_char(creation_date,'DD-Mon-YYYY HH24:MI:SS')||crlf);
   utl_smtp.write_data(conn, 'Job finished at:' || to_char(finishing_date,'DD-Mon-YYYY HH24:MI:SS')||crlf);
   utl_smtp.write_data(conn, 'Description:'||long_desc || crlf);

   --
   -- create names for log file/listing file
   --
   k := instr(main_file_name,'.gms');
   if (k=0) then
      k := instr(main_file_name,'.GMS');
   end if;
   if (k>0) then
      main_file_name := substr(main_file_name,1,k-1);
   end if;
   logfilename := main_file_name || '.log';
   listingfilename := main_file_name || '.lst';

   --
   -- send blobs
   --
   send_blob(conn, '--GamsQueue.Boundary.918273645', logfilename, logfile);
   send_blob(conn, '--GamsQueue.Boundary.918273645', listingfilename, listingfile);

   --
   -- end boundary
   --
   utl_smtp.write_data(conn, crlf || '--GamsQueue.Boundary.918273645--' || crlf);

   utl_smtp.close_data(conn);

   utl_smtp.quit(conn);

exception

   when others then
      dbms_output.put_line('Error:'||SQLCODE||':'||SQLERRM);

   -- ignore errors

end SendEmail;



function ExecuteJobJ(id in tid, errmess out varchar2) return number
```

```
as language java name 'GamsQueueJ.ExecuteJobJ(java.lang.String, java.lang.String[]) return int';

procedure RunJob(id in tid) is
--
-- this procedure is called by oracle job queue
--
   errno integer;
   errmess varchar2(255);
begin
   ChangeStatus(id,'S','R');    -- mark as running

   --
   -- call java method to execute GAMS
   --
   errno := ExecuteJobJ(id,errmess);

   --
   -- set job completion date/time
   --
   update qadmin set JobCompletionDate = SYSTIMESTAMP where JobId = id;


   --
   -- completion info
   --
   if (errno < 0) then
      update qadmin set ExecError = errmess where JobId = id;
      commit;
      ChangeStatus(id,'R','E'); -- mark as error
   else
      ChangeStatus(id,'R','F'); -- mark as finished
   end if;


   --
   -- send email to user
   --
   SendEmail(id);

   --
   -- select next job
   --
   SelectWaitingJob;


exception

   when others then
      SelectWaitingJob;

end RunJob;


function GetGamsExecutionDirectory return varchar2 is
--
-- return the directory
--
begin
   return GamsExecuteDirectory;
end;


function GetGamsExe return varchar2 is
--
-- return fully qualified gams.exe
--
begin
   return GamsExeName;
end;


end gamsqueue;

/

show errors
```

gqtriggers.sql

```
-------------------------------------------------------------
--
-- Triggers for GAMS/ORAQUEUE
```

```
--
-- Author:       Erwin Kalvelagen (erwin@gams.com)
-- Date:         Feb 2002
-- Requirements: Oracle 8i,9i
--               job_queue_processes in ora.init is > 0
-- Called by:    all.sql
--
-- These triggers are used to keep track of the number
-- of waiting and running jobs.
-------------------------------------------------------------



create or replace trigger inserttrigger
before insert on qadmin
for each row
begin
  case :new.jobstatus
     when 'W' then gamsqueue.IncrementWaiting;
     when 'S' then gamsqueue.IncrementRunning;
     when 'R' then gamsqueue.IncrementRunning;
  else
     -- should we give error message?
     null;
  end case;
end inserttrigger;

/


create or replace trigger deletetrigger
before delete on qadmin
for each row
begin
  case :new.jobstatus
     when 'W' then gamsqueue.DecrementWaiting;
     when 'S' then gamsqueue.DecrementRunning;
     when 'R' then gamsqueue.DecrementRunning;
  else
     -- should we give error message?
     null;
  end case;
end deletetrigger;

/


create or replace trigger updatetrigger
before update of jobstatus on qadmin
for each row
begin

  --
  -- optimization: ignore if no change
  -- or if S -> R
  --
  if (:old.jobstatus = :new.jobstatus) then
     return;
  end if;
  if (:old.jobstatus = 'S' and :new.jobstatus = 'R') then
     return;
  end if;

  case :old.jobstatus
     when 'W' then gamsqueue.DecrementWaiting;
     when 'S' then gamsqueue.DecrementRunning;
     when 'R' then gamsqueue.DecrementRunning;
  else
     -- should we give error message?
     null;
  end case;
  case :new.jobstatus
     when 'W' then gamsqueue.IncrementWaiting;
     when 'S' then gamsqueue.IncrementRunning;
     when 'R' then gamsqueue.IncrementRunning;
  else
     -- should we give error message?
     null;
  end case;
end updatetrigger;

/
```

GamsQueueJ.java

```
//
// Run a GAMS job
//
// Author: Erwin Kalvelagen (erwin@gams.com)
// Date: feb 2002
// Environment: Oracle 9i, Java 2 SDK standard edition 1.4.0
// Usage:
//     > javac GamsQueueJ.java
//     > loadjava -u gams/queue GamsQueueJ.class
//
// The following Java Stored Procedures are callable from PL/SQL:
//     int ExecuteJobJ(String jobid, String[] errmess)
//

import java.sql.*;
import oracle.jdbc.*;
import oracle.sql.*;
import java.io.*;

public class GamsQueueJ {

    Connection conn;  // database JDBC connection
    String id;        // job id
    String FileNames[]; // filenames
    String BaseDir;   // directory
    String Dir;       // working directory
    String GamsExe;   // name+dir of gams.exe
    String ModelName; // fully qualified name of the model

    static final int MaxFiles = 5;

    public static int ExecuteJobJ(String jobid, String[] errmess) {

        //
        // This method is called by the Oracle JVM
        // from the PL/SQL package GamsQueue.
        //
        // parameters:
        //     jobid     : JOBID in table qadmin
        //     errmess[0]: error message can be returned here
        //     returns   : error code, 0 : success, -1:failure
        //

        GamsQueueJ gq;

        try {
            gq = new GamsQueueJ();
            gq.id = jobid;
            gq.connect();
            gq.run();
            errmess[0] = "";
            return 0;
        }
        catch(Exception e) {
            errmess[0] = e.toString();
            e.printStackTrace();
            return -1;
        }
    }

    private void connect() throws SQLException {

        //
        // connect to the host database
        //

        conn = DriverManager.getConnection("jdbc:default:connection:");
    }


    private void run() throws SQLException, IOException, InterruptedException {

        //
        // get name of base directory
        //
        BaseDir = getBaseDirectory();

        //
        // get name of GAMS.EXE
        //
        GamsExe = getGamsExe();

        //
```

```
        // get info from qadmin table
        //
        Statement stmt = conn.createStatement();
        FileNames = new String[MaxFiles];
        String q1 = "select filename1,filename2,filename3,filename4,filename5 from qadmin where JobId='"+id+"'";
        ResultSet rs = stmt.executeQuery(q1);
        if (rs.next()) {
            for (int i=0; i<MaxFiles; ++i)
               FileNames[i] = rs.getString(i+1);
        }
        else
            throw new SQLException("No rows in table qadmin with JobId="+id);
        rs.close();



        //
        // create temp directory
        //
        Dir = BaseDir + id;
        File fdir = new File(Dir);
        if (!fdir.mkdir()) {
            //
            // does it already exist?
            //
            if (!fdir.isDirectory())
                throw new SQLException("Could not create directory " + Dir);
        }

        //
        // stream the BLOBs to files in directory Dir
        //
        StreamFiles();

        //
        // call gams
        //
        String[] cmdArray = new String[5];
        cmdArray[0] = GamsExe;
        cmdArray[1] = ModelName;
        cmdArray[2] = "WDIR="+Dir;
        cmdArray[3] = "SCRDIR="+Dir;
        cmdArray[4] = "LO=2";
        Process p = Runtime.getRuntime().exec(cmdArray);
        p.waitFor();

        //
        // store listing and log file in database
        //
        String ListingFile = ForceExtension(ModelName,".lst");
        String LogFile = ForceExtension(ModelName,".log");
        StoreFile("listing",ListingFile);
        StoreFile("log",LogFile);

        //
        // clean up
        //
        RemoveFiles();


    }

    private boolean isNullOrEmpty(String s) {

        //
        // returns true if s is null or empty
        //

        if (s == null)
            return true;
        if (s.equals(""))
            return true;
        return false;
    }

    private void StreamFiles() throws SQLException, FileNotFoundException, IOException {

        //
        // stream BLOBs to files
        //

        Statement stmt = conn.createStatement();

        for (int i=0; i<MaxFiles; ++i) {
            if (isNullOrEmpty(FileNames[i]))
```

```
            continue;

        //
        // create full name
        //
        String FullName = Dir + File.separator + FileNames[i];
        if (i==0) {
            FullName = ForceExtension(FullName, ".gms");
            ModelName = FullName;
        }

        FileOutputStream f = new FileOutputStream(FullName);

        //
        // get stream for  file
        //
        int ip1 = i+1;
        String q = "select file"+ip1+" from qadmin where JobId='"+id+"'";
        ResultSet rs = stmt.executeQuery(q);
        InputStream is;
        if (rs.next())
            is = rs.getBinaryStream(1);
        else
            throw new SQLException("No rows in table qadmin with JobId="+id);
        rs.close();

        //
        // stream and copy data
        //
        int chunklen;
        byte[] chunk = new byte[1024];
        while(( chunklen = is.read(chunk)) != -1)
            f.write(chunk,0,chunklen);
        f.close();
        is.close();

    }
}

private void RemoveFiles() {

    //
    // remove all files in Dir and then remove the directory itself
    //

    File dir = new File(Dir);

    File[] files = dir.listFiles();

    for (int i=0; i<files.length; ++i)
        files[i].delete();

    dir.delete();
}

private String getStringFromDB(String ProcCall) throws SQLException {

    //
    // call PL/SQL function and retrieve string result
    //

    CallableStatement cs = conn.prepareCall(ProcCall);
    cs.registerOutParameter(1,Types.CHAR);
    cs.executeUpdate();
    String result = cs.getString(1);
    cs.close();
    return result;
}
private String getBaseDirectory() throws SQLException {

    //
    // get the name of the base directory by calling a PL/SQL function
    // gamsqueue.getGamsExecutionDirectory
    //

    String ProcCall = "{? = call gamsqueue.getGamsExecutionDirectory}";
    return getStringFromDB(ProcCall);
}

private String getGamsExe() throws SQLException {

    //
    // get the fully qualified name gams.exe
    //
```

```
        String ProcCall = "{? = call gamsqueue.getGamsExe}";
        return getStringFromDB(ProcCall);
    }

    private void StoreFile(String ColumnName, String FileName) throws SQLException,
          FileNotFoundException, IOException {

        //
        // store file in BLOB
        // to do : think about row locks.
        //

        String selectstmt = "select "+ColumnName+" from qadmin where JobId='"+id+"'";
        String updatestmt = "update qadmin set "+ColumnName+"=empty_blob() where JobId = '"+id+"'";

        BLOB blob;

        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(selectstmt);
        if (rs.next())
            blob = ((OracleResultSet)rs).getBLOB(1);
        else
            throw new SQLException("No rows in table qadmin with JobId="+id);
        rs.close();

        if (blob == null) {
            //
            // create empty blob first
            //
            stmt.executeQuery(updatestmt);
            StoreFile(ColumnName,FileName);
            return;
        }

        OutputStream os = blob.getBinaryOutputStream();
        int size = blob.getBufferSize();
        byte[] buffer = new byte[size];
        int length = -1;

        FileInputStream is = new FileInputStream(FileName);
        while (( length = is.read(buffer) ) != -1)
            os.write(buffer,0,length);

        os.close();
        is.close();

    }

    private String ForceExtension(String fln, String newExtension) {

        //
        // replace extension .gms
        //

        String s = fln;
        if (s.endsWith(".gms") || s.endsWith(".GMS"))
            s = s.substring(0,s.length()-4);
        s = s + newExtension;

        return s;

    }

}
```

3.13 Installation notes

First you need to make sure that Oracle allows background processes. This can be set through the parameter `job_queue_processes` in `ora.init`. Second, we need a Java compiler, which can be downloaded from Sun as part of the Java 2 SDK. We used standard edition 1.4.0. Then a database user need to be added. We used as username and password `gams/queue`. Make sure this user has enough privileges such as `JAVASYSPRIV` to use `Runtime.exec()`.

Some of settings in the beginning of `gqiface.sql` need to be tailored. It may be beneficial to use an Oracle PL/SQL Integrated Development Environment such as TOAD [23]. Such a tool will make it much easier to navigate around in your database and to quickly edit and compile stored procedures.

To install the complete system do:

all.cmd

```
:
: run all installation scripts
:
: author: Erwin Kalvelagen (erwin@gams.com)
:
call setclasspath.cmd
javac GamsQueueJ.java
call loadjava -u gams/queue GamsQueueJ.class
sqlplus gams/queue @all
call loadpsp -replace -user gams/queue gamsupload.psp
call loadpsp -replace -user gams/queue gamsjobs.psp
```

all.sql

```
-----------------------------------------------
--
-- run all install scripts
--
-- Author:    Erwin Kalvelagen (erwin@gams.com)
-- Date:      Feb 2002
-- Usage:     sqplus gams/queue @all
-- Called by: all.cmd
--
-----------------------------------------------
set serveroutput on
@@gqtable
@@htmltable
@@gqiface
@@gqbody
@@gqtriggers
show errors
@@htmlproc
show errors
quit
```

The above installation routines also include installation of the sample application descibed below.

4 Application: a Web submission tool

4.1 Introduction

As an application of the queueing system, we have implemented a Web based job submission tool. The user submits jobs through his Web browser. At the server end we have used the Oracle HTTP server, a version of the Apache HTTP Server which is included in Oracle's Application Server [22].

As server scripting language we have used PL/SQL, using the PL/SQL Gateway [21].

For the dynamic content we have used PL/SQL Server Pages (PSP), a concept similar to Java Server Pages and Active Server Pages where HTML is mixed with executable code in the form of PL/SQL statements.

A submission page is written that looks as depicted in figure 2.

After specifying the model and the other information, the response as shown in figure 3 will be generated.

The buttons in the log and listing columns allow for downloading the log file and listing file.

4.2 Implementation

The web submission tool pages are written in a combination of HTML, JavaScript, and PL/SQL Server Pages. The server side processing is performed using PL/SQL stored procedures. The whole system comprises of just four files, which are reproduced below.

It is noted that we not paid much attention to security. It is therefore not advised to run the system as is on a server that can be accessed unrestricted from the Internet. The system can be hardened in different ways, including restricting access to certain users, or certain domains. It is noted that we run GAMS in unrestricted, unsecure mode, which causes a big security hole. GAMS has a parameter `EXECMODE` to disable some of the very dangerous features, such as `$call` and `execute` but we believe that there remain serious security issues. A possibly better protection mechanism is to put the server behind a properly configured firewall or router that ensables access to just a number of trusted users.

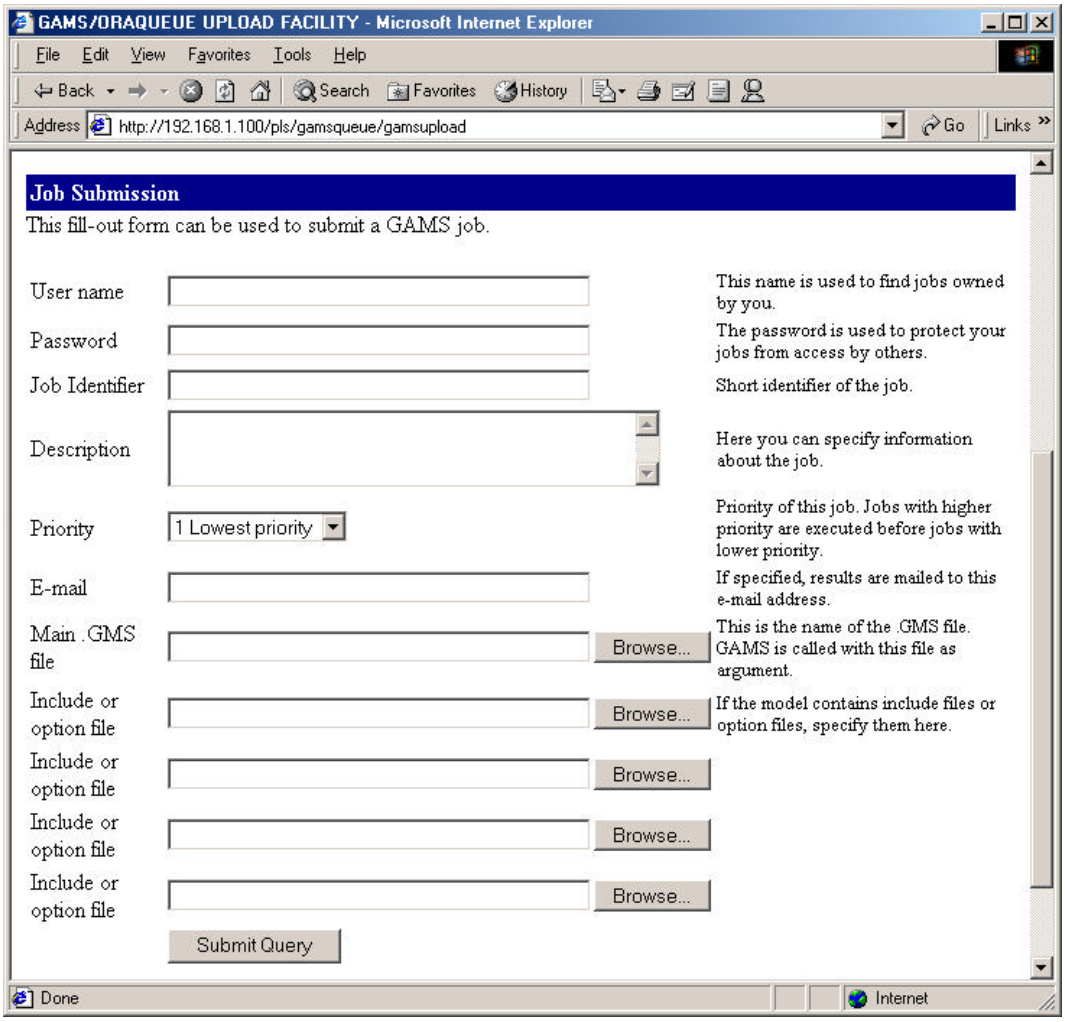

Figure 2: GAMS Model Submission Form

A big advantages of using a Web based front-end: user's do not need to have GAMS installed on their machine. Just a browser is enough to submit jobs and retrieve results. A powerful server may provide better throughput than an underpowered client laptop, and just a single license of GAMS and the solver components is needed.

A possible issue with platform independence is the file format of solver option files. GAMS can read UNIX and DOS .gms files and include files of both UNIX and DOS. The solvers however may need option files in native format with LF or CR/LF terminated lines.

There are a number of possible extensions possible such as: only registered users can use the system, cookies can be used to make login easier, forms can remember their last values, the HTML pages can be prettier, etc.

4.3 Password handling

Although the system provides a way to specify a password using the standard HTML input password control, we would like to emphasize that the password is not at all protected. It travels plain text over the line and it can be viewed by inspecting some of the HTML files served by the system. I.e. it provides limited functionality in making the system more secure.

4.4 Configuration

For the web submission tool to work, a properly configured HTTP server and PL/SQL gateway is needed. Both are part of Oracle's Application Server offering. For our example we configured a Database Access Descriptor

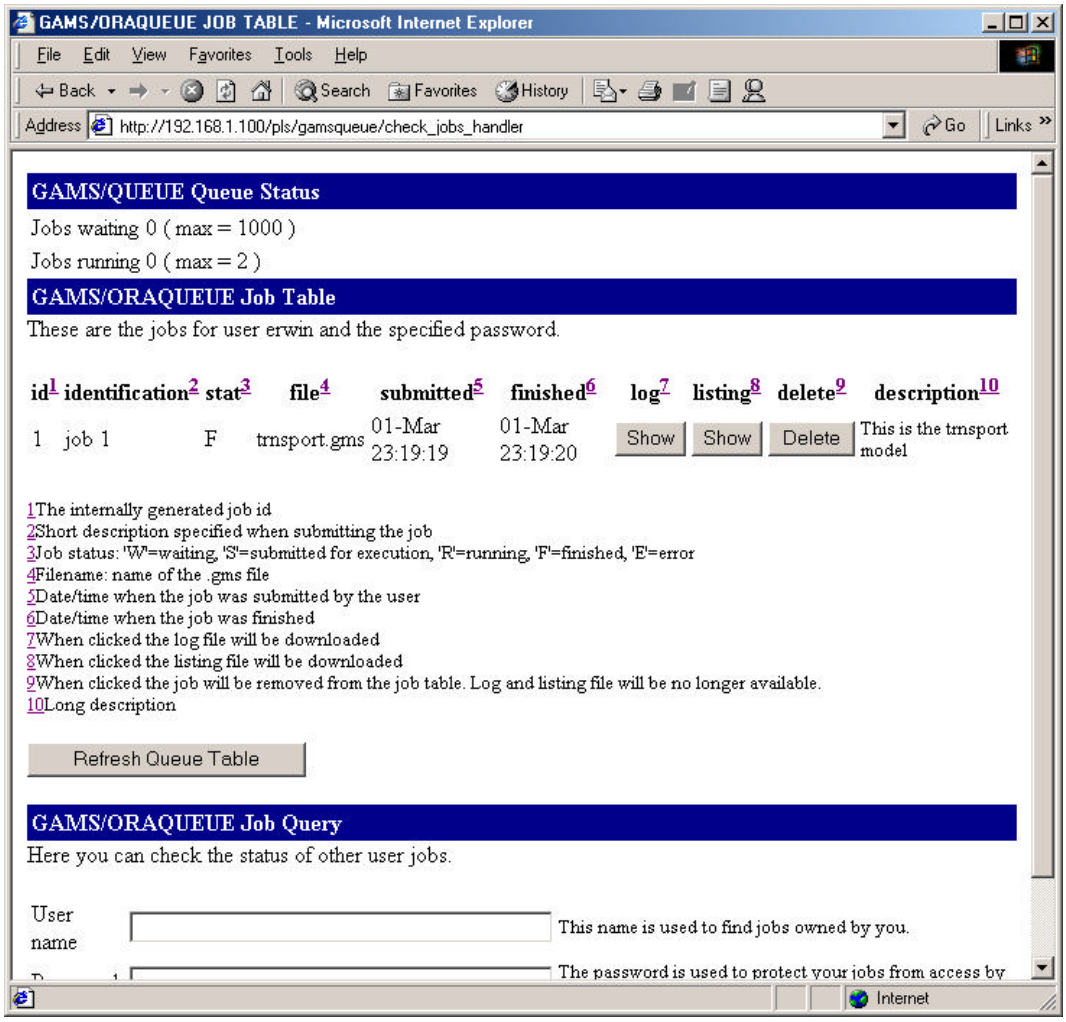

Figure 3: GAMS Job table

which uses the document table `doctable` which is created by the script `htmltable.sql` (see the next section). We used Basic HTTP Authentication in our example.

4.5 Sources

gamsupload.psp

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

<html>

<head>
<title>GAMS/ORAQUEUE UPLOAD FACILITY</title>
</head>

<body>

<% sectionheader('GAMS/ORAQUEUE Job Upload Facility'); %>

This tool allows you to submit GAMS jobs to the GAMS/QUEUE batch queue system.
After submission your job will be run once resources are available to do so.
You can periodically check the status of your job.
If the job has finished you can download the results or if you specified an
e-mail address, both the listing file and the log file will be e-mailed to
you.

<% sectionheader('GAMS/QUEUE Queue Status'); %>
```

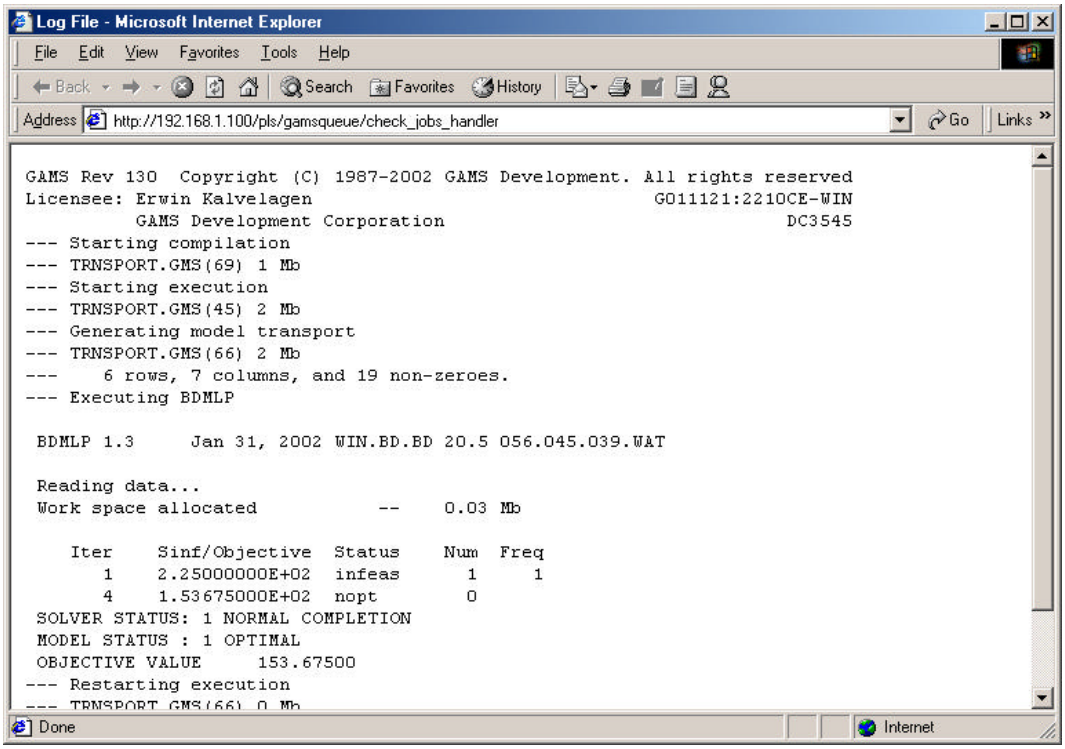

Figure 4: Log file downloaded in a browser

```
<table>
<tr>
     <td> Jobs waiting </td>
     <td> <% htp.print(gamsqueue.njobswaiting); %> </td>
     <td> ( max = <% htp.print(gamsqueue.nmaxjobswaiting); %> ) </td>
</tr>
<tr>
     <td> Jobs running </td>
     <td> <% htp.print(gamsqueue.njobsrunning); %> </td>
     <td> ( max = <% htp.print(gamsqueue.nmaxjobsrunning); %> ) </td>
</tr>
</table>

<% sectionheader('Job Query'); %>

Here you can check the status of your jobs.

<form enctype="multipart/form-data" action="check_jobs_handler" method="POST">

<table>
<tr>
  <td>User name</td>
  <td><input type="text" size=40 name="html_username"></td>
  <td><font size=-1">This name is used to find jobs owned by you.</font></td>
</tr>
<tr>
  <td>Password</td>
  <td><input type="password" size=40 name="html_password"></td>
  <td><font size=-1">The password is used to protect your jobs from access by others.</font></td>
</tr>
<tr>
  <td></td>
  <td><input type="submit"></td>
</tr>
</table>

<input type="hidden" name="html_id" value="">
<input type="hidden" name="html_action" value="">

</form>

<% sectionheader('Job Submission'); %>
```

```
This fill-out form can be used to submit a GAMS job.

<form enctype="multipart/form-data" action="submit_job_handler" method="POST">
<table>
<tr>
  <td>User name</td>
  <td><input type="text" size=40 name="html_username"></td>
  <td><font size=-1">This name is used to find jobs owned by you.</font></td>
</tr>
<tr>
  <td>Password</td>
  <td><input type="password" size=40 name="html_password"></td>
  <td><font size=-1">The password is used to protect your jobs from access by others.</font></td>
</tr>
<tr>
  <td>Job Identifier</td>
  <td><input type="text" size=40 name="html_identifier"></td>
  <td><font size=-1">Short identifier of the job.</font></td>
</tr>
  <tr>
  <td>Description</td>
  <td><textarea rows=3 cols=40 name="html_description"></textarea></td>
  <td><font size=-1>Here you can specify information about the job.</font></td>
</tr>
<tr>
  <td>Priority</td>
  <td><select name="html_priority">
      <option value="1">1 Lowest priority
      <option value="2">2
      <option value="3">3
      <option value="4">4 Highest priority
  </td>
  <td><font size=-1>Priority of this job. Jobs with higher priority are executed before jobs with lower priority.</font></td>
</tr>
<tr>
  <td>E-mail</td>
  <td><input type="text" size=40 name="html_email"></td>
  <td><font size=-1>If specified, results are mailed to this e-mail address.</font></td>
</tr>
<tr>
  <td>Main .GMS file</td>
  <td><input type="file" size=40 name="html_file1"></td>
  <td><font size=-1>This is the name of the .GMS file. GAMS is called with this file as argument.</font></td>
</tr>
<tr>
  <td>Include or option file</td>
  <td><input type="file" size=40 name="html_file2"></td>
  <td><font size=-1>If the model contains include files or option files, specify them here.</font></td>
</tr>
<tr>
  <td>Include or option file</td>
  <td><input type="file" size=40 name="html_file3"></td>
</tr>
<tr>
  <td>Include or option file</td>
  <td><input type="file" size=40 name="html_file4"></td>
</tr>
<tr>
  <td>Include or option file</td>
  <td><input type="file" size=40 name="html_file5"></td>
</tr>
<tr>
  <td></td>
  <td><input type="submit"></td>
</tr>
</table>

</form>

<% sectionheader('Powered by'); %>

<% htp.print(owa_util.get_cgi_env('SERVER_SOFTWARE')); %>

</body>

</html>
```

gamsjobs.psp

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

```
<%@ plsql parameter="user_name" type="varchar2" %>
<%@ plsql parameter="pass_word" type="varchar2" %>


<html>

<head>
<title>GAMS/ORAQUEUE JOB TABLE</title>
</head>

<body>

<script language="javascript"> function ShowLog(id,user,pw) {
   f = document.frm;
   f.html_id.value=id;
   f.html_action.value="log";
   f.html_username.value=user;
   f.html_password.value=pw;
   f.submit();
}
</script>

<script language="javascript"> function ShowListing(id,user,pw) {
   f = document.frm;
   f.html_id.value=id;
   f.html_action.value="listing";
   f.html_username.value=user;
   f.html_password.value=pw;
   f.submit();
}
</script>

<script language="javascript"> function DeleteRecord(id,user,pw) {
   f = document.frm;
   f.html_id.value=id;
   f.html_action.value="delete";
   f.html_username.value=user;
   f.html_password.value=pw;
   f.target="";
   f.submit();
}
</script>

<script language="javascript"> function refresh(user,pw) {
   f = document.frm;
   f.html_username.value=user;
   f.html_password.value=pw;
   f.target="";
   f.submit();
}
</script>


<% sectionheader('GAMS/QUEUE Queue Status'); %>

<table>
<tr>
     <td> Jobs waiting </td>
     <td> <% htp.print(gamsqueue.njobswaiting); %> </td>
     <td> ( max = <% htp.print(gamsqueue.nmaxjobswaiting); %> ) </td>
</tr>
<tr>
     <td> Jobs running </td>
     <td> <% htp.print(gamsqueue.njobsrunning); %> </td>
     <td> ( max = <% htp.print(gamsqueue.nmaxjobsrunning); %> ) </td>
</tr>
</table>

<% sectionheader('GAMS/ORAQUEUE Job Table'); %>

<a name="table">These are the jobs for user <%=user_name%> and the specified password.</a><p>

<table>
  <tr>
     <th>id<a href="#id"><sup>1</sup></a></th>
     <th>identification<a href="#shortdesc"><sup>2</sup></a></th>
     <th>stat<a href="#status"><sup>3</sup></a></th>
     <th>file<a href="#file"><sup>4</sup></a></th>
     <th>submitted<a href="#submitdate"><sup>5</sup></a></th>
     <th>finished<a href="#finishdate"><sup>6</sup></a></th>
     <th>log<a href="#log"><sup>7</sup></a></th>
     <th>listing<a href="#listing"><sup>8</sup></a></th>
     <th>delete<a href="#delete"><sup>9</sup></a></th>
     <th>description<a href="#longdesc"><sup>10</sup></a></th>
  </tr>
```

```
<% for item in (select jobid,shortdescription,longdescription,jobstatus,jobcreationdate,jobcompletiondate,filename1
       from qadmin where username=user_name and password=pass_word) loop %>
  <tr>
     <td><%= item.jobid %></td>
     <td><%= item.shortdescription %></td>
     <td><%= item.jobstatus %></td>
     <td><%= item.filename1 %></td>
     <td><%= to_char(item.jobcreationdate,'DD-Mon HH24:MI:SS') %></td>
     <td><%= to_char(item.jobcompletiondate,'DD-Mon HH24:MI:SS') %></td>
     <td><button onClick='ShowLog("<%=item.jobid%>","<%=user_name%>","<%=pass_word%>")'>Show</button></td>
     <td><button onClick='ShowListing("<%=item.jobid%>","<%=user_name%>","<%=pass_word%>")'>Show</button></td>
     <td><% if (item.jobstatus = 'F' or item.jobstatus = 'E' or item.jobstatus = 'W') then %>
             <button onClick='DeleteRecord("<%=item.jobid%>","<%=user_name%>","<%=pass_word%>")'>Delete</button>
         <%  end if;  %> </td>
     <td><font size=-1><%= item.longdescription %></font></td>
  </tr>
<% end loop; %>
</table>

<p>

<font size=-1>
<a name="id"><a href="#table">1</a>The internally generated job id</a><br>
<a name="shortdesc"><a href="#table">2</a>Short description specified when submitting the job</a><br>
<a name="status"><a href="#table">3</a>Job status: 'W'=waiting, 'S'=submitted for execution,
'R'=running, 'F'=finished, 'E'=error</a><br>
<a name="file"><a href="#table">4</a>Filename: name of the .gms file</a><br>
<a name="submitdate"><a href="#table">5</a>Date/time when the job was submitted by the user</a><br>
<a name="finishdate"><a href="#table">6</a>Date/time when the job was finished</a><br>
<a name="log"><a href="#table">7</a>When clicked the log file will be downloaded</a><br>
<a name="listing"><a href="#table">8</a>When clicked the listing file will be downloaded</a><br>
<a name="delete"><a href="#table">9</a>When clicked the job will be removed from the job table.
Log and listing file will be no longer available.</a><br>
<a name="longdesc"><a href="#table">10</a>Long description</a><br>
</font>

<p>

<button onClick='refresh("<%=user_name%>","<%=pass_word%>")'>Refresh Queue Table</button>

<p>

<% sectionheader('GAMS/ORAQUEUE Job Query'); %>

Here you can check the status of other user jobs.

<form name="frm" enctype="multipart/form-data" action="check_jobs_handler" method="POST" target="new">

<table>
<tr>
  <td>User name</td>
  <td><input type="text" size=40 name="html_username"></td>
  <td><font size=-1">This name is used to find jobs owned by you.</font></td>
</tr>
<tr>
  <td>Password</td>
  <td><input type="password" size=40 name="html_password"></td>
  <td><font size=-1">The password is used to protect your jobs from access by others.</font></td>
</tr>
<tr>
  <td></td>
  <td><input type="submit"></td>
</tr>
</table>

<input type="hidden" name="html_id" value="">
<input type="hidden" name="html_action" value="">

</form>


</body>
</html>
```

htmltable.sql

```
-------------------------------------------------------
--
-- Create table for HTML interface
--
-- Author:    Erwin Kalvelagen (erwin@gams.com)
-- Date:      Feb 2002
```

```
-- Called by: all.sql
-- See also:  Oracle Corp., Using the PL/SQL gateway
--
-------------------------------------------------------

drop table doctable;

--
-- document table definition
--
create table doctable (
   name           varchar2(256) unique not null,
   mime_type      varchar2(128),
   doc_size       number,
   dad_charset    varchar2(128),
   last_updated   date,
   content_type   varchar2(128),
   blob_content   blob
);


commit;
```

htmlproc.sql

```
-------------------------------------------------------
--
-- Create stored procedures for HTML interface
--
--
-- Author: Erwin Kalvelagen (erwin@gams.com)
-- See also: Oracle Corp., Using the PL/SQL gateway
--
-------------------------------------------------------



create or replace procedure sectionheader(header in varchar2) is
--
-- a section header is printed
--
begin
   htp.print('<TABLE WIDTH=100% STYLE="color:white; background-color:darkblue" ><TR><TH ALIGN=left>'
             || header
             || '</TH></TR></TABLE>');
end sectionheader;

/

create or replace procedure welcome as
--
-- for testing
--
begin
   htp.htmlopen;
   htp.headopen;
   htp.title('Welcome');
   htp.headclose;
   htp.bodyopen;
   htp.header(1,'welcome');
   htp.print('Welcome to GAMSQUEUE');
   htp.header(2,'OWA_UTIL info');
   htp.header(3,'version');
   owa_util.print_version;
   htp.header(3,'cgi environment');
   owa_util.print_cgi_env;
   htp.bodyclose;
   htp.htmlclose;
end welcome;

/

create or replace procedure report_error(mess in varchar2) as
--
-- very basic error reporting
--
begin
   htp.htmlopen;
   htp.headopen;
   htp.title('Error');
   htp.headclose;
```

```
   htp.bodyopen;
   htp.print(mess);
   htp.bodyclose;
   htp.htmlclose;
end report_error;

/

create or replace procedure print_submission_data(
   html_username in varchar2,
   html_password in varchar2,
   html_identifier in varchar2,
   html_description in varchar2,
   html_priority in varchar2,
   html_email in varchar2,
   html_file1 in varchar2,
   html_file2 in varchar2,
   html_file3 in varchar2,
   html_file4 in varchar2,
   html_file5 in varchar2
   ) as
--
-- for debugging only
--
begin
   htp.htmlopen;
   htp.headopen;
   htp.title('Files Uploaded');
   htp.headclose;
   htp.bodyopen;
   htp.header(1,'Info');
   htp.preOpen();
   htp.print('html_username:    '||html_username);
   htp.print('html_password:    '||html_password);
   htp.print('html_identifier:  '||html_identifier);
   htp.print('html_description: '||html_description);
   htp.print('html_priority:    '||html_priority);
   htp.print('html_email:       '||html_email);
   htp.print('html_file1:       '||html_file1);
   htp.print('html_file2:       '||html_file2);
   htp.print('html_file3:       '||html_file3);
   htp.print('html_file4:       '||html_file4);
   htp.print('html_file5:       '||html_file5);
   htp.preClose();
   htp.bodyclose;
   htp.htmlclose;
end print_submission_data;

/

create or replace procedure get_blob(
   filename in varchar2,
   file out blob
   ) as
--
-- get blob for filename
--
begin
   if (filename is NULL) then
      return;
   end if;

   select blob_content into file from doctable
   where name=filename;

end get_blob;

/

create or replace procedure get_filename(
   filename in varchar2,
   newfilename out varchar2
   ) as
--
-- remove xxxx/ from filename
--
   k integer;
   not_found exception;
begin
   if (filename is NULL) then
      return;
   end if;

   k := instr(filename,'/');
   if (k=0) then
```

```
      raise not_found;
   end if;
   newfilename := substr(filename,k+1);


end get_filename;

/


create or replace procedure htmlheader(title in varchar2) as
--
-- utility procedure
--
begin
   htp.htmlopen;
   htp.headopen;
   htp.title(title);
   htp.headclose;
   htp.bodyopen;
   htp.Preopen();
end htmlheader;

/

create or replace procedure htmlfooter as
--
-- utility procedure
--
begin
   htp.preClose();
   htp.bodyclose;
   htp.htmlclose;
end htmlfooter;

/

create or replace procedure streamrawdata(file in out nocopy blob) as
--
-- stream blob to web browser
--
   pos integer;
   cnt integer;
   buffer raw(32767);
begin
   pos := 1;
   cnt := 32767;
   dbms_lob.open(file,dbms_lob.lob_readonly);
   loop
       dbms_lob.read(file,cnt,pos,buffer);
       htp.prn(utl_raw.cast_to_varchar2(buffer));
       pos := pos + cnt;
   end loop;

exception
   when no_data_found then
      dbms_lob.close(file);
   when others then
      null;
end streamrawdata;

/

create or replace procedure show_log(user in varchar2, pw in varchar2, id in varchar2) as
--
-- show log file
--
   file blob;
begin
   htmlheader('Log File');

   select log into file from qadmin where jobid = id and username = user and password = pw;

   if (file is NULL) then
      htp.print('Log file not available');
   end if;

   streamrawdata(file);

   htmlfooter;

exception
   when no_data_found then
      htp.print('No job found with jobid='||id||',username='||user||',password=***');
      htmlfooter;
```

```
   when others then
      htp.print('error in show_log, code='||SQLCODE||' errmess='||SQLERRM);
      htmlfooter;
end show_log;

/


create or replace procedure show_listing(user in varchar2, pw in varchar2, id in varchar2) as
--
-- show listing file
--
   file blob;
begin
   htmlheader('Listing File');

   select listing into file from qadmin where jobid = id and username = user and password = pw;

   if (file is NULL) then
      htp.print('Listing file not available');
   end if;

   streamrawdata(file);

   htmlfooter;

exception
   when no_data_found then
      htp.print('No job found with jobid='||id||',username='||user||',password=***');
      htmlfooter;
   when others then
      htp.print('error in show_listing, code='||SQLCODE||' errmess='||SQLERRM);
      htmlfooter;
end show_listing;

/

create or replace procedure delete_record(user in varchar2, pw in varchar2, id in varchar2) as
begin


   delete from qadmin where jobid = id and username = user and password = pw;

   --
   -- show status of jobs
   --
   gamsjobs(user, pw);


exception
   when no_data_found then
      htp.print('No job found with jobid='||id||',username='||user||',password=***');
      htmlfooter;
   when others then
      htp.print('error in delete_record, code='||SQLCODE||' errmess='||SQLERRM);
      htmlfooter;
end delete_record;

/

create procedure gamsjobs(user_name varchar2, pass_word varchar2) as
--
-- This is just a placeholder.
-- Will be replaced by loadpsp call in all.cmd.
-- If already exist, don't replace.
--
begin
  null;
end gamsjobs;

/


create or replace procedure check_jobs_handler(
   html_username in varchar2,
   html_password in varchar2,
   html_action in varchar2,
   html_id in varchar2
   ) as
--
-- check gams jobs
--
begin
   case html_action
     when '' then gamsjobs(html_username, html_password);
```

```
     when 'log' then show_log(html_username, html_password, html_id);
     when 'listing' then show_listing(html_username, html_password, html_id);
     when 'delete' then delete_record(html_username, html_password, html_id);
   else
      gamsjobs(html_username, html_password);
   end case;
end check_jobs_handler;

/

create or replace procedure submit_job_handler(
   html_username in varchar2,
   html_password in varchar2,
   html_identifier in varchar2,
   html_description in varchar2,
   html_priority in varchar2,
   html_email in varchar2,
   html_file1 in varchar2,
   html_file2 in varchar2,
   html_file3 in varchar2,
   html_file4 in varchar2,
   html_file5 in varchar2
   ) as
--
-- gams upload script (multiple files)
--
   debug boolean := false;
   file1  blob;
   file2  blob;
   file3  blob;
   file4  blob;
   file5  blob;
   filename1  varchar2(80);
   filename2  varchar2(80);
   filename3  varchar2(80);
   filename4  varchar2(80);
   filename5  varchar2(80);
   priority integer;
   id gamsqueue.tid;

begin

   if (debug) then
      print_submission_data(html_username,html_password,html_identifier,html_description,
      html_priority,html_email,html_file1,html_file2,html_file3,html_file4,html_file5);
   --    return;
   end if;

   --
   -- handle data. First get blobs.
   --
   get_blob(html_file1,file1);
   get_blob(html_file2,file2);
   get_blob(html_file3,file3);
   get_blob(html_file4,file4);
   get_blob(html_file5,file5);

   --
   -- now remove leading xxxxx/ from filename
   --
   get_filename(html_file1,filename1);
   get_filename(html_file2,filename2);
   get_filename(html_file3,filename3);
   get_filename(html_file4,filename4);
   get_filename(html_file5,filename5);

   priority := to_number(html_priority);

   id := gamsqueue.newjob(html_username, html_password, priority, html_email,
           html_identifier, html_description, file1, file2, file3,
           file4, file5, filename1, filename2, filename3, filename4,
           filename5);


   --
   -- show status of jobs
   --
   gamsjobs(html_username, html_password);

exception
   when others then
      report_error('error in submit_job_handler, code='||SQLCODE||' errmess='||SQLERRM);

end submit_job_handler;
```

```
/
```

5 Conclusion

We have shown how Oracle can be used to implement a multi user batch queueing system as well as a Web based job submission tool. The whole system is platform independent: it should port easily to a UNIX architecture. The system can be used as a framework for customized applications.

References

[1] Ken Arnold, James Gosling and David Holmes, *The Java Programming Language*, Third Edition, Addison-Wesley, 2000.

[2] James Gosling, Bill Joy, Guy Steele and Gilad Bracha, *The Java Language Specification*, Second Edition, Addison-Wesley, 2000.

[3] Eric Granger, *Web Publishing using PL/SQL and Java*, CERN (European Organization for Nuclear Research), 2000.

[4] Thomas Kyte, *Expert one on one: Oracle*, Wrox Press, 2001.

[5] Bryn Llewellyn and Robert Pang, *Oracle 9iAS Best Practices in PSP Development*, paper, August, 2001.

[6] Kevin Miller, *Professional NT Services*, Wrox Press, 1998.

[7] Randy C. Morin, *Programming NT Services: Implementing Windows Application Servers*, Wiley, 2000.

[8] Oracle Corp., *Oracle 9i, Database Administrator's Guide*, June, 2001.

[9] Oracle Corp., *Oracle 9i, Database Administrator's Guide for Windows*, June, 2001.

[10] Oracle Corp., *Oracle 9i, Application Developer's Guide – Fundamentals*, June, 2001.

[11] Oracle Corp., *Oracle 9i, Application Developer's Guide – Advanced Queueing (AQ)*, July, 2001.

[12] Oracle Corp., *Oracle 9i, Application Developer's Guide – Large Objects (LOBs)*, July, 2001.

[13] Oracle Corp., *Oracle 9i, Java Developer's Guide*, June, 2001.

[14] Oracle Corp., *Oracle 9i, JDBC Developer's Guide and Reference*, June, 2001.

[15] Oracle Corp., *Oracle 9i, Java Stored Procedures Developer's Guide*, June, 2001.

[16] Oracle Corp., *Oracle 9i, PL/SQL User's Guide and Reference*, June, 2001.

[17] Oracle Corp., *Oracle 9i, SQL Reference*, June, 2001.

[18] Oracle Corp., *Oracle 9i, Supplied Java Packages Reference*, June, 2001.

[19] Oracle Corp., *Oracle 9i, Supplied PL/SQL Packages and Types Reference*, July, 2001.

[20] Oracle Corp., *Oracle 9i Application Server, PL/SQL Web Toolkit Reference*, April, 2001.

[21] Oracle Corp., *Oracle 9i Application Server, Using the PL/SQL Gateway*, May, 2001.

[22] Oracle Corp., *Oracle 9i Application Server, Overview Guide*, November 2000.

[23] Quest Software, `http://www.quest.com/toad/`.

[24] W. Richard Stevens, *Advanced Programming in the UNIX Environment*, Addison-Wesley, 1992.

[25] Dave Wotton, *Mail_files: an Oracle PL/SQL procedure to send an email with file attachments*, `http://home.clara.net/dwotton/dba/oracle_smtp.htm`.