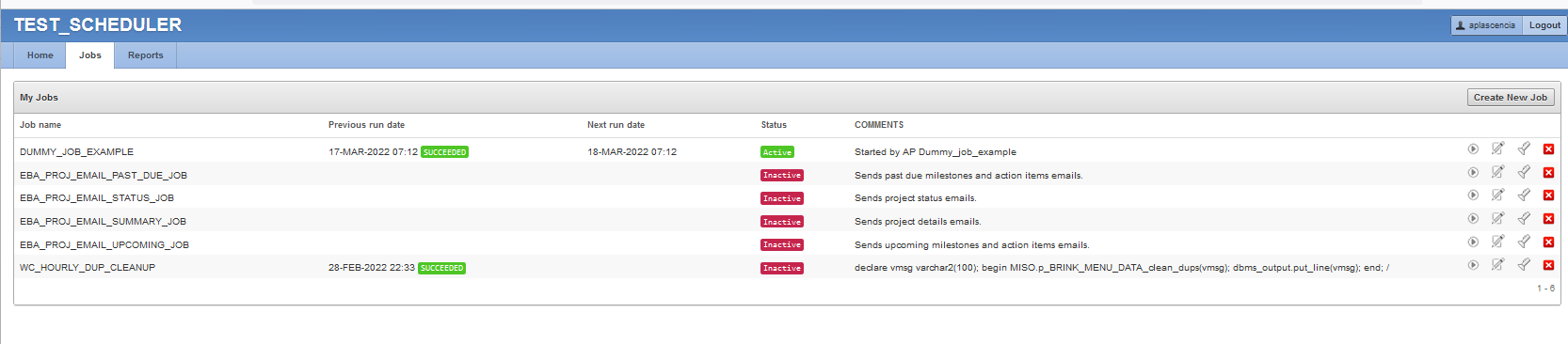
The app itself is named => ORACLE\_APEX\_SCHEDULER\_APP.sql

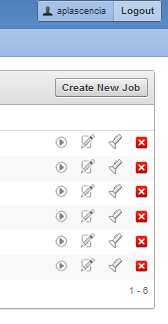
The purpose of this doc is to provide the db schema design and documentation for the support of scheduled jobs. The goal is to build a simple schedular and alerting utility, identifying via sql, some “event” based on data changes or specific sql returned data values , upon that event take on specific actions like run jobs, send emails, rerun failed processes, etc.

The basic idea is we start up a job using DBMS\_SCHEDULER that runs every N minutes. When it runs, it checks to see if any alerts need to be triggered. If so, it then takes one or more actions associated with that alert.

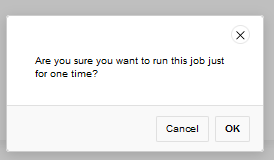


In the Miso Apex Schema the views that control batch scheduling are names AP\_SCHED\*

Below create new job is how we can control what to run next, and how to batch schedule that



The Play run that job while you wait.



The edit allows you to schedule a job on the fly

Graphical user interface, application, Word

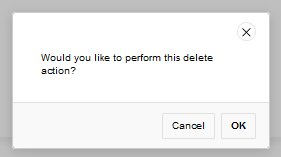
Description automatically generated

The inspect shows job history

Graphical user interface, table, Excel

Description automatically generated

Delete removes a job



Under reports are the intervals to control batch jobs actions

A screenshot of a computer

Description automatically generated with medium confidence

Let get a view of the design:

Diagram, table

Description automatically generated

Key info for the system to hold

**ALERT\_CONFIG**

CREATE TABLE alert\_config

(

job\_wakeup\_minutes INTEGER,

keep\_checking VARCHAR2 (1),

send\_email\_proc\_name VARCHAR2 (300),

apex\_workspace\_name VARCHAR2(100)

)

**Alerts**

We store all registered alerts here. The trigger code is executed to determine if the alert actions should be run. The trigger type describes the code and how to execute it. You can set the frequency with which the alert is checked, how often to check again once it was triggered, and more.

CREATE TABLE alerts (

id NUMBER NOT NULL

CONSTRAINT alerts\_id\_pk PRIMARY KEY,

alert\_name VARCHAR2 (100)

CONSTRAINT alerts\_alert\_name\_unq UNIQUE,

description VARCHAR2 (4000),

trigger\_code VARCHAR2 (4000),

trigger\_type VARCHAR2 (100),

check\_frequency VARCHAR2 (1000),

recurrence\_frequency VARCHAR2 (1000),

ignore\_failures VARCHAR2 (1),

is\_active VARCHAR2 (1),

last\_run\_id NUMBER,

created DATE NOT NULL,

created\_by VARCHAR2 (255) NOT NULL,

updated DATE NOT NULL,

updated\_by VARCHAR2 (255) NOT NULL,

CONSTRAINT trigger\_type\_choices CHECK

(trigger\_type IN ('ONEROW',

'NOROWS',

'MULTROWS',

'RETURNTRUE',

'RETURNFALSE')),

CONSTRAINT ignore\_failures\_yn

CHECK (ignore\_failures IN ('Y', 'N')),

CONSTRAINT active\_yn CHECK (is\_active IN ('Y', 'N'))

)

### Alert Actions

What to do when an action is needed.

CREATE TABLE alert\_actions (

id NUMBER NOT NULL

CONSTRAINT alert\_actions\_id\_pk PRIMARY KEY,

alert\_id NUMBER

CONSTRAINT alert\_actions\_alert\_id\_fk

REFERENCES alerts ON DELETE CASCADE,

action\_sequence INTEGER,

action\_type VARCHAR2 (100) NOT NULL,

action\_code VARCHAR2 (4000),

send\_email\_to VARCHAR2 (4000),

email\_subject VARCHAR2 (4000),

email\_body CLOB,

email\_query VARCHAR2 (4000),

created DATE NOT NULL,

created\_by VARCHAR2 (255) NOT NULL,

updated DATE NOT NULL,

updated\_by VARCHAR2 (255) NOT NULL,

CONSTRAINT action\_type\_choices CHECK

(action\_type IN ('EMAIL',

'SMS',

'PROC',

'REST'))

)

**Alert Runs**

Let's keep track of each run of an alert.

CREATE TABLE alert\_runs (

id NUMBER NOT NULL

CONSTRAINT alert\_runs\_id\_pk PRIMARY KEY,

alert\_id NUMBER

CONSTRAINT alert\_runs\_alert\_id\_fk

REFERENCES alerts ON DELETE CASCADE,

started\_at DATE,

completed\_at DATE,

completed\_successfully VARCHAR2 (1),

results VARCHAR2 (4000),

errors VARCHAR2 (4000),

created DATE NOT NULL,

created\_by VARCHAR2 (255) NOT NULL,

updated DATE NOT NULL,

updated\_by VARCHAR2 (255) NOT NULL,

CONSTRAINT success\_yn

CHECK (completed\_successfully IN ('Y', 'N'))

)

**The Alert Log**

Very simple table to store tracing and [error](https://github.com/OraOpenSource/Logger) logging.

CREATE TABLE alert\_log (

id NUMBER NOT NULL

CONSTRAINT alert\_log\_id\_pk PRIMARY KEY,

info VARCHAR2 (4000),

created DATE NOT NULL,

created\_by VARCHAR2 (255) NOT NULL,

updated DATE NOT NULL,

updated\_by VARCHAR2 (255) NOT NULL

)

**Start the alert checking job**

Well, that's easy. Let's have it wake up every minute.

BEGIN

alert\_mgr.start\_check\_alerts\_job (every\_n\_minutes\_in => 1);

END;

**Register alerts**

Now I will register my first - and very basic - alert.

BEGIN

alert\_mgr.register\_alert (

alert\_name\_in => 'Always True',

description\_in => 'Trigger is TRUE, display string',

trigger\_code\_in => 'TRUE',

trigger\_type\_in => 'RETURNTRUE',

check\_frequency\_in => 'H',

recurrence\_frequency\_in => 'D',

ignore\_failures\_in => 'Y',

action\_type\_in => 'PROC',

action\_code\_in => 'alert\_mgr.log(''PROC EXECUTED'')',

send\_email\_to\_in => '',

is\_active\_in => 'Y');

END;

The type of trigger code is "RETURNTRUE" which means that when a PL/SQL expression returns TRUE, the alert is triggered. In this case, the trigger code is, well, a little bit trivial: TRUE.  
  
I tell the utility to check once an hour but to not re-trigger the alert more than once per day. And if (WHEN!) the trigger code evaluates to TRUE, execute a PL/SQL procedure, which in this case simply writes a row out to the alert\_log table.

About those frequencies: for my basic utility I implement only the following frequencies:

* H = Hourly
* D = Daily
* NNN = the number of minutes between checks.

So if you want the alert to be checked every two hours, pass 120 for check\_frequency\_in.

Here's a more elaborate alert, with two actions to be taken if triggered:

DECLARE

l\_actions alert\_mgr.actions\_t;

BEGIN

l\_actions (1).action\_type := 'PROC';

l\_actions (1).action\_code := 'alert\_mgr.log(''Action 1'')';

l\_actions (2).action\_type := 'EMAIL';

l\_actions (2).send\_email\_to := 'steven.feuerstein@oracle.com';

l\_actions (2).email\_subject := 'Subject is action 2';

l\_actions (2).email\_body := '<p>The body of the email</p>';

alert\_mgr.register\_alert (

alert\_name\_in => 'Two actions',

description\_in => 'Use the array approach',

trigger\_code\_in => 'select count(\*) from dual',

trigger\_type\_in => 'ONEROW',

check\_frequency\_in => 'H',

recurrence\_frequency\_in => 'H',

ignore\_failures\_in => 'Y',

actions\_in => l\_actions,

is\_active\_in => 'Y');

END;

In this case, the type of trigger code is "ONEROW" or "Execute alert actions if the query in trigger\_code returns just one row."

You can also specify a query to go with your email,

The query returns two items: subject and body. These will then be used to send an email with that associated subject and body to each email address provided in the send\_email\_to field.

DECLARE

l\_index integer := 1;

l\_actions alert\_mgr.actions\_t;

BEGIN

DELETE FROM earthquake\_alerts;

l\_index := 1;

l\_actions (l\_index).action\_type := 'EMAIL';

l\_actions (l\_index).send\_email\_to :=

'steven.feuerstein@oracle.com';

l\_actions (l\_index).email\_query := q'[

SELECT 'Magnitude ' || magnitude ||

' Earthquake detected in ' || location subject,

'Earthquake ID ' || id body

FROM earthquakes

WHERE magnitude >= 6.5

AND id NOT IN (SELECT earthquake\_id FROM earthquake\_alerts)]';

/\* Record as notified \*/

l\_index := 2;

l\_actions (l\_index).action\_type := 'PROC';

l\_actions (l\_index).action\_code := 'earthquake\_mgr.record\_notifications;';

/\* Record in log \*/

l\_index := 3;

l\_actions (l\_index).action\_type := 'PROC';

l\_actions (l\_index).action\_code :=

'alert\_mgr.log(

''Earthquakes checked at '' ||

TO\_CHAR (SYSDATE, ''YYYY-MM-DD HH24:MI:SS''))';

alert\_mgr.register\_alert (

alert\_name\_in => 'Notify at 6.5',

description\_in =>

'Send an email for earthquakes rated at 6.5 or higher. '

|| 'Don''t send a notification twice for the same earthquake.',

trigger\_code\_in =>

'SELECT 1 FROM earthquakes

WHERE magnitude >= 6.5

AND id NOT IN (

SELECT earthquake\_id FROM earthquake\_alerts)',

trigger\_type\_in =>

alert\_mgr.c\_trigger\_type\_select\_mult\_rows,

check\_frequency\_in => '1',

recurrence\_frequency\_in => '1',

ignore\_failures\_in => 'Y',

actions\_in => l\_actions,

is\_active\_in => 'Y');

END;

### Sending Mail

The main alerting package does not contain any email-related code. The way you send emails may vary greatly, so we made all of that code dynamic. If you want to send emails, you need to register the email procedure with the utility. I do this below, using the email package that comes with the utility (and relies on APEX\_MAIL):

BEGIN

/\* Set up email using APEX and the default API from the utility \*/

alert\_mgr.register\_email\_proc (

proc\_name\_in => 'alert\_email\_mgr.send\_email',

workspace\_name\_in => 'MISO');

END;

In other words, if you plan to use APEX\_MAIL to send emails with the alerting utility, then execute the above block with the name of your workspace, and you will be all set.

Note: I do not push emails directly onto the queue in the package body, so you may want to change that after installation.

CREATE OR REPLACE FORCE VIEW "AP\_SCHED\_JOBS\_HISTORY" ("LOG\_ID", "JOB\_NAME", "SUCCEEDED", "FAILED", "ERROR\_DESC", "FIRST\_RUN", "LAST\_RUN", "RUN\_DURATION", "CPU\_USED") AS

  SELECT

    MAX(d.log\_id)       AS log\_id,

    d.job\_name,

    --

    NULLIF(SUM(CASE WHEN d.status = 'SUCCEEDED'  THEN 1 ELSE 0 END), 0)     AS succeeded,

    NULLIF(SUM(CASE WHEN d.status != 'SUCCEEDED' THEN 1 ELSE 0 END), 0)     AS failed,

    --

    MAX(d.errors)       AS error\_desc,

    --

    MIN(CAST(d.actual\_start\_date AS DATE))          AS first\_run,

    NULLIF(MAX(CAST(d.actual\_start\_date AS DATE)),  MIN(CAST(d.actual\_start\_date AS DATE)))     AS last\_run,

    d.run\_duration    AS run\_duration,

    d.cpu\_used        AS cpu\_used

FROM user\_scheduler\_job\_run\_details d

where

   d.actual\_start\_date between sysdate-30 and sysdate

GROUP BY d.job\_name, run\_duration, cpu\_used

/

CREATE OR REPLACE FORCE VIEW "AP\_SCHED\_RUNNING\_JOBS" ("LOG\_ID", "JOB\_NAME", "JOB\_STYLE", "ELAPSED\_TIME", "CPU\_USED", "DESTINATION", "SESSION\_ID", "RESOURCE\_CONSUMER\_GROUP", "CREDENTIAL\_NAME") AS

  SELECT

    j.log\_id,

    j.job\_name,

    j.job\_style,

    j.elapsed\_time,

    j.cpu\_used,

    j.destination,

    j.session\_id,

    j.resource\_consumer\_group,

    j.credential\_name

FROM user\_scheduler\_running\_jobs j

/

CREATE OR REPLACE FORCE VIEW "AP\_SCHED\_SCHEDULED\_JOBS" ("JOB\_NAME", "JOB\_TYPE", "JOB\_PRIORITY", "JOB\_ACTION", "JOB\_ARGS", "REPEAT\_INTERVAL", "RUN\_COUNT", "FAILURE\_COUNT", "RETRY\_COUNT", "NEXT\_RUN\_DATE", "LAST\_START\_DATE", "LAST\_RUN\_DURATION", "AUTODROP\_", "ENABLED\_", "COMMENTS") AS

  SELECT

    j.job\_name,

    j.job\_type,

    j.job\_priority,

    j.job\_action,

    j.number\_of\_arguments AS job\_args,

    j.repeat\_interval,

    --

    NULLIF(j.run\_count, 0)      AS run\_count,

    NULLIF(j.failure\_count, 0)  AS failure\_count,

    NULLIF(j.retry\_count, 0)    AS retry\_count,

    --

    j.next\_run\_date,

    j.last\_start\_date,

    --

    --app.get\_duration(j.last\_run\_duration) AS

    last\_run\_duration,

    --

    CASE

        WHEN j.auto\_drop = 'TRUE'

            THEN

                  --apex.get\_icon('fa-check-square', 'Autodrop enabled')

                  'Autodrop enabled'

            END AS autodrop\_,

    --

    CASE

        WHEN j.state = 'SCHEDULED' AND j.enabled = 'TRUE'

            THEN --apex.get\_icon('fa-check-square', 'Job is enabled')

            'Job is enabled'

            END AS enabled\_,

    --

    j.comments

FROM user\_scheduler\_jobs j

/

CREATE OR REPLACE FORCE VIEW "AP\_SCHED\_SCHED\_JOB\_DUP1" ("SCHEMA\_NAME", "JOB\_NAME", "JOB\_STYLE", "JOB\_TYPE", "JOB\_ACTION", "START\_DATE", "SCHEDULE", "LAST\_START\_DATE", "NEXT\_RUN\_DATE", "STATE") AS

  select owner as schema\_name,

       job\_name,

       job\_style,

       case when job\_type is null

                 then 'PROGRAM'

            else job\_type end as job\_type,

       case when job\_type is null

                 then program\_name

                 else job\_action end as job\_action,

       start\_date,

       case when repeat\_interval is null

            then schedule\_name

            else repeat\_interval end as schedule,

       last\_start\_date,

       next\_run\_date,

       state

from sys.all\_scheduler\_jobs

order by owner,

         job\_name

/

CREATE OR REPLACE FORCE VIEW "AP\_SCHED\_SCHED\_JOB\_RUN\_DET" ("JOB\_NAME", "RUN\_DUR") AS

  select

   job\_name,

   avg( extract( day from run\_duration )\*24\*60\*60 + extract( hour from run\_duration )\*60\*60 + extract( minute from run\_duration )\*60 + extract( second from run\_duration )) run\_dur

from

   user\_scheduler\_job\_run\_details

group by

   job\_name

--having avg(extract( day from run\_duration )\*24\*60\*60 + extract( hour from run\_duration )\*60\*60 + extract( minute from run\_duration )\*60 + extract( second from run\_duration )) > 0

order by 1

/

create or replace PACKAGE      "AT\_SCHEDULER" as

type t\_job\_action\_arg is record(

    argument\_name varchar2(100),

    argument\_type varchar2(100),

    default\_value varchar2(100),

    current\_value varchar2(4000)

);

/\*\*

 \* Check if job exists

 \* @param p\_job\_name name of the scheduler job

 \*/

function exists\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type)

return boolean;

/\*\*

 \* add / save job to scheduler jobs

 \* @param p\_job\_name name of the scheduler job

 \* @param p\_job\_name action of the scheduler job

 \*/

function add\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type

                  , p\_job\_action in user\_scheduler\_jobs.job\_action%type

                  , p\_start\_date in user\_scheduler\_jobs.start\_date%type

                  , p\_repeat\_interval in user\_scheduler\_jobs.repeat\_interval%type

                  , p\_end\_date in user\_scheduler\_jobs.end\_date%type

                  , p\_enabled in boolean := false

                  , p\_argument\_list in t\_job\_action\_arg := null

                  , p\_comments in user\_scheduler\_jobs.comments%type

                  )

return boolean;

/\*\*

 \* To force immediate job execution

 \* @param p\_job\_name name of the scheduler job

 \*/

procedure execute\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type);

/\*\*

 \* Remove job from scheduler by name

 \* @param p\_job\_name name of the scheduler job

 \*/

procedure remove\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type) ;

/\*\*

 \* Indicates whether the job is enabled (TRUE) or not (FALSE)

 \* @param p\_job\_name name of the scheduler job

 \*/

function is\_actief\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type)

return boolean;

/\*\*

 \* Enable job from scheduler by name

 \* @param p\_job\_name name of the scheduler job

 \*/

procedure enable\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type);

/\*\*

 \* Disable job from scheduler by name

 \* @param p\_job\_name name of the scheduler job

 \*/

procedure disable\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type);

 /\*\*

 \* Dummy procedure for testing

 \*/

procedure dummy\_proc;

end;

create or replace PACKAGE BODY      "AT\_SCHEDULER" as

/\*\*

 \* Check if job exists

 \* @param p\_job\_name name of the scheduler job

 \*/

function exists\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type)

return boolean is

    l\_found integer;

begin

    select decode(count(1), 0, 0, 1) into l\_found

    from   user\_scheduler\_jobs

    where  (job\_name = p\_job\_name);

    return (l\_found = 1);

end exists\_job;

/\*\*

 \* add to scheduler jobs

 \* @param p\_job\_name name of the scheduler job

 \* @param p\_job\_name action of the scheduler job

 \*/

function add\_job (p\_job\_name          in user\_scheduler\_jobs.job\_name%type

                  , p\_job\_action      in user\_scheduler\_jobs.job\_action%type

                  , p\_start\_date      in user\_scheduler\_jobs.start\_date%type

                  , p\_repeat\_interval in user\_scheduler\_jobs.repeat\_interval%type

                  , p\_end\_date        in user\_scheduler\_jobs.end\_date%type

                  , p\_enabled         in boolean := false

                  , p\_argument\_list   in t\_job\_action\_arg := null

                  , p\_comments        in user\_scheduler\_jobs.comments%type

                  )

return boolean is

begin

    if p\_job\_name is not null then

        -- remove existing job by name

        -- remove\_job(p\_job\_name);

        if exists\_job(p\_job\_name) then

           raise\_application\_error (-20000, 'Job already exists');

        else

           dbms\_scheduler.create\_job (job\_name                => p\_job\_name,

                                      job\_type             => 'STORED\_PROCEDURE',

                                      job\_action           => p\_job\_action,

                                      start\_date           => p\_start\_date,

                                      repeat\_interval      => p\_repeat\_interval,

                                      end\_date             => p\_end\_date,

                                      enabled              => p\_enabled,

                                      auto\_drop            => false,

                                      comments             => p\_comments);

           dbms\_scheduler.set\_attribute (name => p\_job\_name, attribute => 'logging\_level', value => dbms\_scheduler.logging\_full);

        end if;

    end if;

    return true;

end add\_job;

/\*\*

 \* To force immediate job execution

 \* @param p\_job\_name name of the scheduler job

 \*/

procedure execute\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type) is

begin

    if exists\_job(p\_job\_name) then

        dbms\_scheduler.run\_job(p\_job\_name);

    end if;

end execute\_job;

/\*\*

 \* Remove job from scheduler by name

 \* @param p\_job\_name name of the scheduler job

 \*/

procedure remove\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type) is

begin

    if exists\_job(p\_job\_name) then

        dbms\_scheduler.drop\_job(p\_job\_name, true);

    end if;

end remove\_job;

/\*\*

 \* Indicates whether the job is enabled (TRUE) or not (FALSE)

 \* @param p\_job\_name name of the scheduler job

 \*/

function is\_actief\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type)

return boolean is

    l\_found integer;

begin

    select decode(count(1), 0, 0, 1) into l\_found

    from   user\_scheduler\_jobs job

    where  (job.job\_name = p\_job\_name)

    and    job.enabled = 'TRUE';

    return (l\_found = 1);

end is\_actief\_job;

/\*\*

 \* Enable job from scheduler by name

 \* @param p\_job\_name name of the scheduler job

 \*/

procedure enable\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type) is

begin

    if exists\_job(p\_job\_name) then

        dbms\_scheduler.enable (p\_job\_name);

    end if;

end enable\_job;

 /\*\*

 \* Disable job from scheduler by name

 \* @param p\_job\_name name of the scheduler job

 \*/

procedure disable\_job (p\_job\_name in user\_scheduler\_jobs.job\_name%type) as

begin

    if exists\_job(p\_job\_name) then

        dbms\_scheduler.disable (p\_job\_name);

    end if;

end disable\_job;

 /\*\*

 \* Dummy procedure for testing

 \*/

procedure dummy\_proc as

begin

     --dbms\_lock.sleep(60);

     null;

end dummy\_proc ;

end;

create or replace PROCEDURE MISO\_AP\_SEND\_MAIL (

  msg\_to varchar2,

  msg\_subject varchar2,

  msg\_text varchar2 )

IS

  mail\_conn utl\_smtp.connection;

  username varchar2(1000) := 'ocid1.user.oc1..aaaaaaaahr7oxhekztltzlswuium6k6heb6dqsyoag2agf5cyztqtt2a7pea@ocid1.tenancy.oc1..aaaaaaaauzqaexjmy5i7nrrbttk4czh42lzvehrr4us3oayfa45e6vcnofga.hc.com';

  passwd   varchar2(500)  := 'ze0LkWvBU1)u;zqJ!xaJ';

  msg\_from varchar2(500)   := 'aplascencia@misorobotics.com';

  mailhost VARCHAR2(500)   := 'smtp.email.us-phoenix-1.oci.oraclecloud.com';

BEGIN

  mail\_conn := UTL\_smtp.open\_connection(mailhost, 587);

  utl\_smtp.starttls(mail\_conn);

  UTL\_SMTP.AUTH(mail\_conn, username, passwd, schemes => 'PLAIN');

  utl\_smtp.mail(mail\_conn, msg\_from);

  utl\_smtp.rcpt(mail\_conn, msg\_to);

  UTL\_smtp.open\_data(mail\_conn);

  UTL\_SMTP.write\_data(mail\_conn, 'Date: ' || TO\_CHAR(SYSDATE, 'DD-MON-YYYY HH24:MI:SS') || UTL\_TCP.crlf);

  UTL\_SMTP.write\_data(mail\_conn, 'To: ' || msg\_to || UTL\_TCP.crlf);

  UTL\_SMTP.write\_data(mail\_conn, 'From: ' || msg\_from || UTL\_TCP.crlf);

  UTL\_SMTP.write\_data(mail\_conn, 'Subject: ' || msg\_subject || UTL\_TCP.crlf);

  UTL\_SMTP.write\_data(mail\_conn, 'Reply-To: ' || msg\_to || UTL\_TCP.crlf || UTL\_TCP.crlf);

  UTL\_SMTP.write\_data(mail\_conn, msg\_text || UTL\_TCP.crlf || UTL\_TCP.crlf);

  UTL\_smtp.close\_data(mail\_conn);

  UTL\_smtp.quit(mail\_conn);

EXCEPTION

  WHEN UTL\_smtp.transient\_error OR UTL\_smtp.permanent\_error THEN

    UTL\_smtp.quit(mail\_conn);

    dbms\_output.put\_line(sqlerrm);

  WHEN OTHERS THEN

    UTL\_smtp.quit(mail\_conn);

    dbms\_output.put\_line(sqlerrm);

END;

declare

l\_return boolean := true;

begin

l\_return := at\_scheduler.add\_job (p\_job\_name => :P\_SCHEDULER\_JOB\_NAME

, p\_job\_action => :P\_SCHEDULER\_JOB\_ACTION

, p\_start\_date => to\_date(:P\_SCHEDULER\_JOB\_STARTDATE, 'DD-MON-YYYY HH24:MI')

, p\_repeat\_interval => :P\_SCHEDULER\_JOB\_REPEAT

, p\_end\_date => to\_date(:P\_SCHEDULER\_JOB\_ENDDATE, 'DD-MON-YYYY HH24:MI')

, p\_comments => :P\_SCHEDULER\_JOB\_COMMENTS

);

if l\_return then

:P\_SCHEDULER\_JOB\_ID := :P\_SCHEDULER\_JOB\_NAME;

end if;

end;

Graphical user interface, application

Description automatically generated