5

Using Oracle-Supplied Packages in Application Development

Objectives

After completing this lesson, you should be able to do the following:

- Describe how the DBMS_OUTPUT package works
- Use UTL_FILE to direct output to operating system files
- Use the HTP package to generate a simple Web page
- Describe the main features of UTL_MAIL
- Call the DBMS_SCHEDULER package to schedule PL/SQL code for execution

Using Oracle-Supplied Packages

The Oracle-supplied packages:

- Are provided with the Oracle server
- Extend the functionality of the database
- Enable access to certain SQL features that are normally restricted for PL/SQL

For example, the `DBMS_OUTPUT` package was originally designed to debug PL/SQL programs.

List of Some Oracle-Supplied Packages

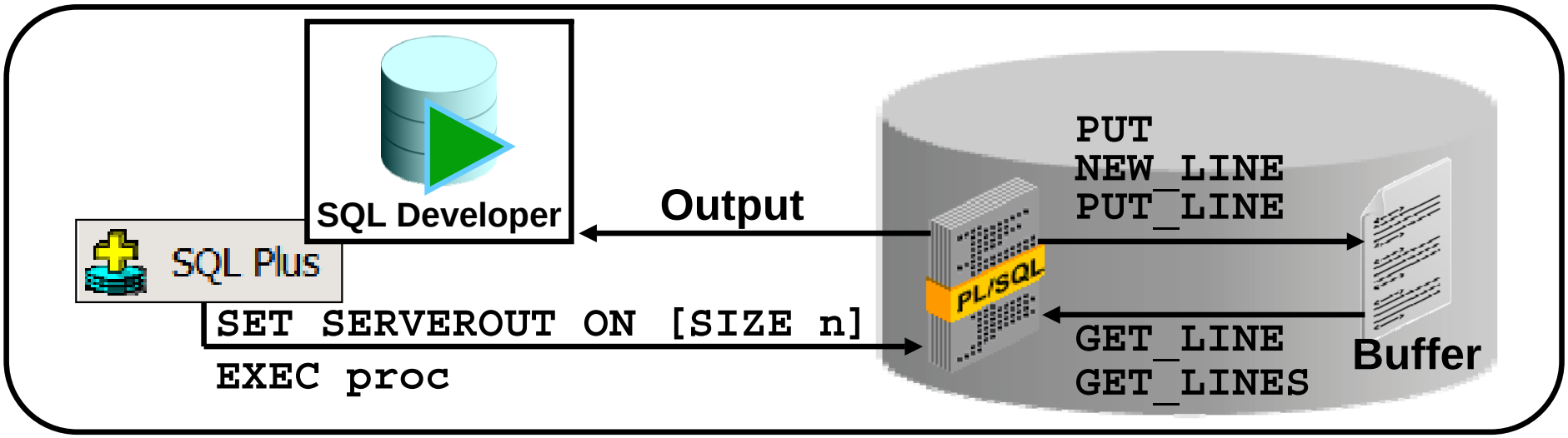
Here is an abbreviated list of some Oracle-supplied packages:

- DBMS_ALERT
- DBMS_LOCK
- DBMS_SESSION
- DBMS_OUTPUT
- HTP
- UTL_FILE
- UTL_MAIL
- DBMS_SCHEDULER

How the DBMS_OUTPUT Package Works

The DBMS_OUTPUT package enables you to send messages from stored subprograms and triggers.

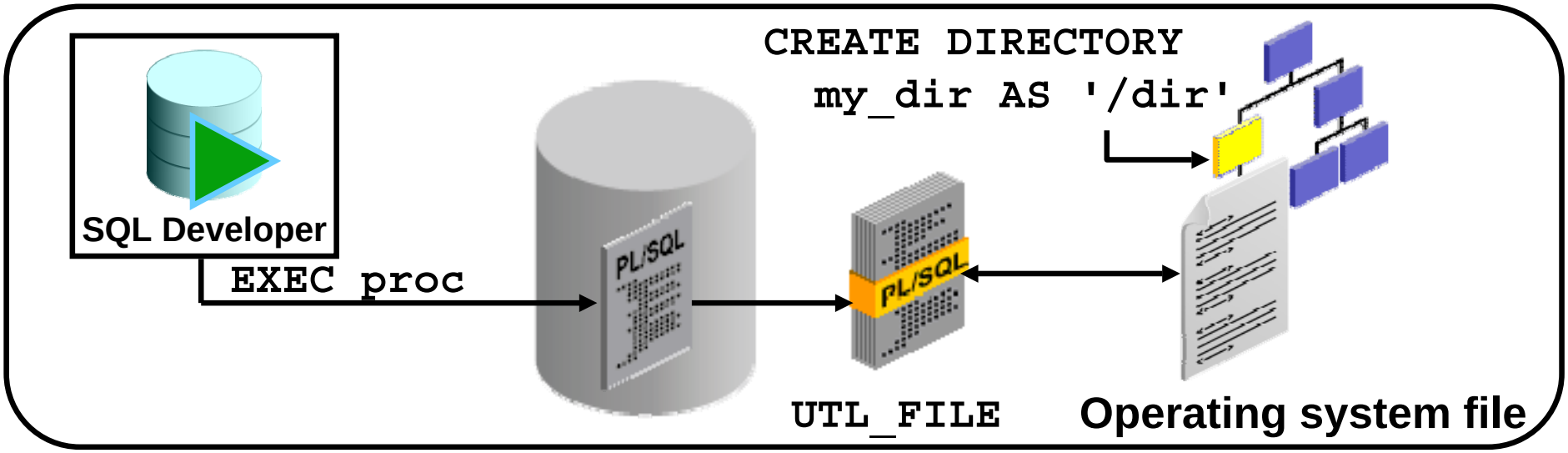
- PUT and PUT_LINE place text in the buffer.
- GET_LINE and GET_LINES read the buffer.
- Messages are not sent until the sender completes.
- Use SET SERVEROUTPUT ON to display messages in SQL*Plus and SQL Developer.



Interacting with Operating System Files

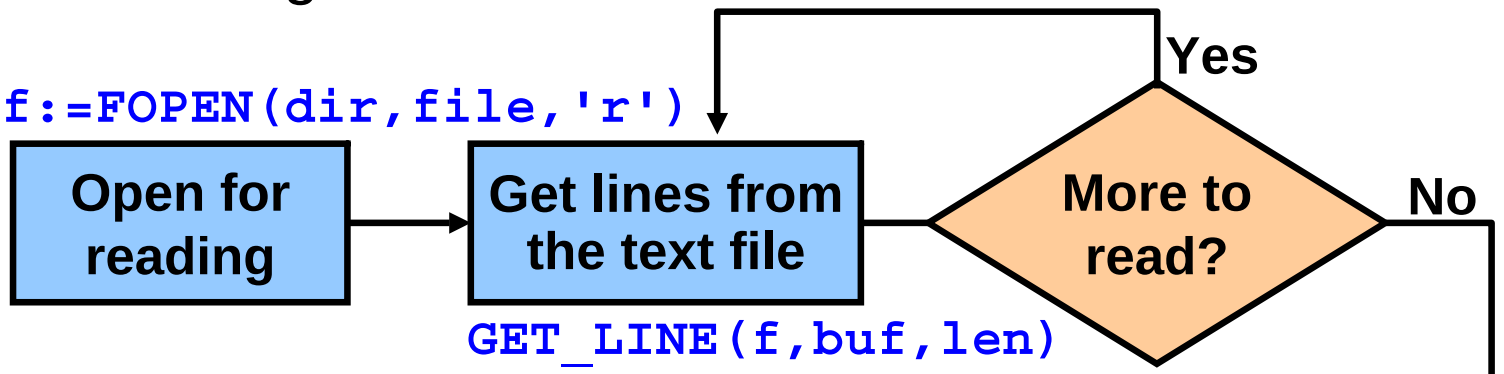
The `UTL_FILE` package extends PL/SQL programs to read and write operating system text files. `UTL_FILE`:

- Provides a restricted version of the operating system stream file I/O for text files
- Can access files in operating system directories defined by a `CREATE DIRECTORY` statement. You can also use the `utl_file_dir` database parameter.

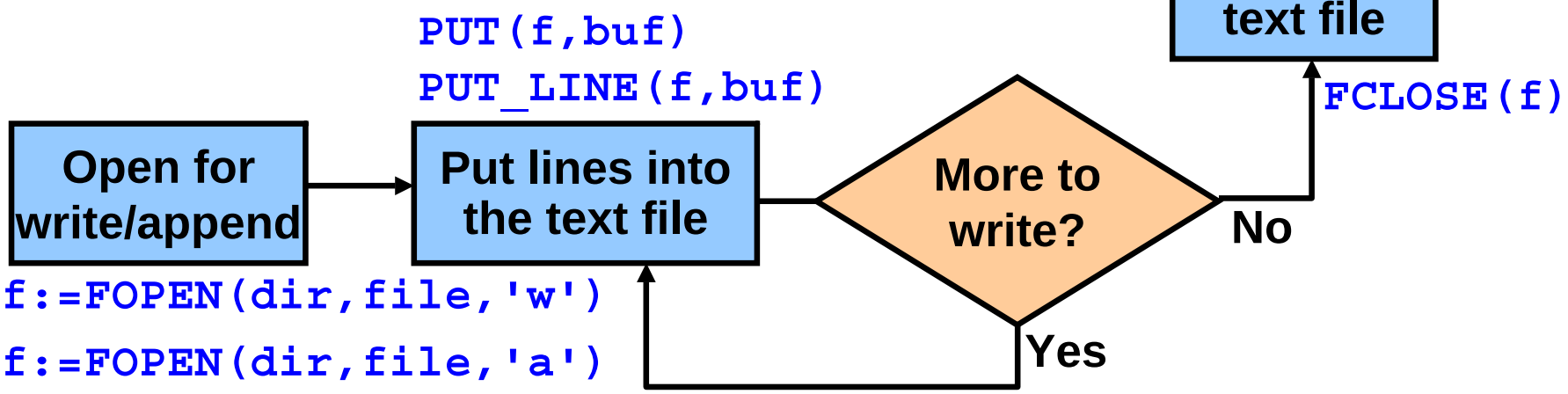


File Processing Using the UTL_FILE Package

- Reading a file:



- Writing or appending to a file:



Exceptions in the UTL_FILE Package

You may have to handle one of these exceptions when using UTL_FILE subprograms:

- INVALID_PATH
- INVALID_MODE
- INVALID_FILEHANDLE
- INVALID_OPERATION
- READ_ERROR
- WRITE_ERROR
- INTERNAL_ERROR

Other exceptions not in the UTL_FILE package are:

- NO_DATA_FOUND and VALUE_ERROR

FOPEN and IS_OPEN Function Parameters

```
FUNCTION FOPEN (location IN VARCHAR2,
                filename IN VARCHAR2,
                open_mode IN VARCHAR2)
RETURN UTL_FILE.FILE_TYPE;
```

```
FUNCTION IS_OPEN (file IN FILE_TYPE)
RETURN BOOLEAN;
```

Example:

```
CREATE PROCEDURE read_file(dir VARCHAR2, filename
VARCHAR2) IS file UTL_FILE.FILE_TYPE;
...
BEGIN
  ...
  IF NOT UTL_FILE.IS_OPEN(file) THEN
   file := UTL_FILE.FOPEN (dir, filename, 'R');
   ...
  END IF;
END read_file;
```

Using UTL_FILE: Example

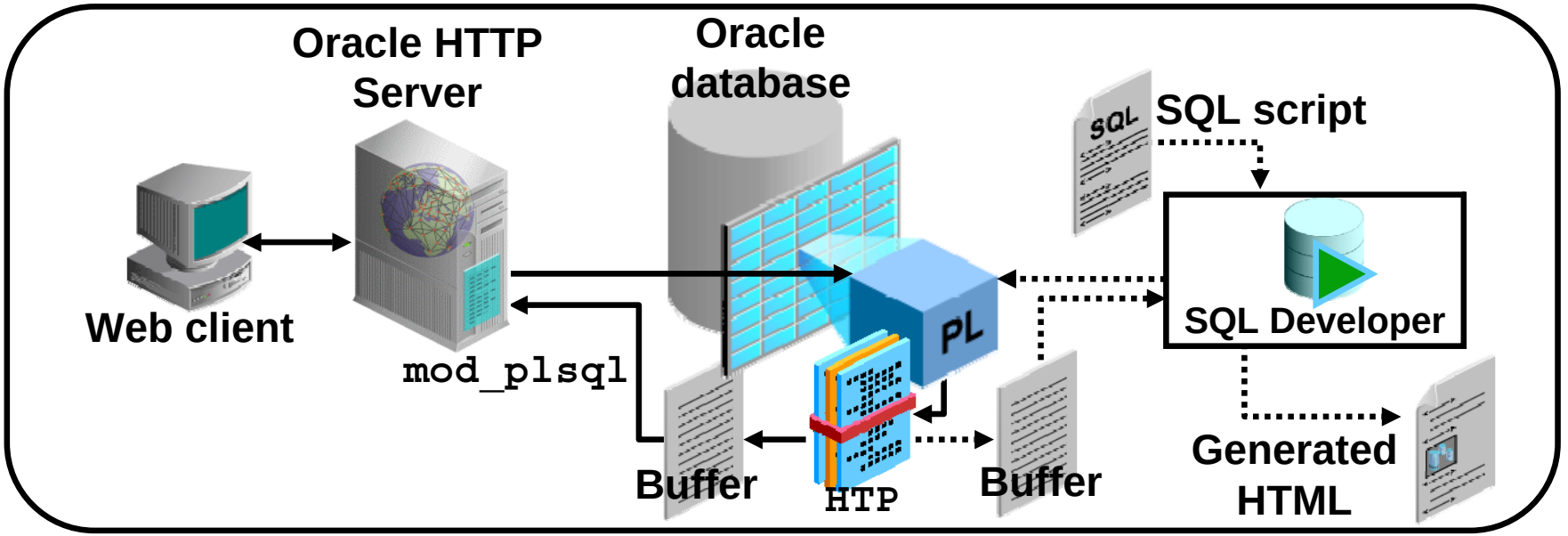
```
CREATE OR REPLACE PROCEDURE sal_status(
  dir IN VARCHAR2, filename IN VARCHAR2) IS
  file UTL_FILE.FILE_TYPE;
 CURSOR empc IS
  SELECT last_name, salary, department_id
  FROM employees ORDER BY department_id;
 newdeptno employees.department_id%TYPE;
 olddeptno employees.department_id%TYPE := 0;
BEGIN
  file:= UTL_FILE.FOPEN (dir, filename, 'w');
  UTL_FILE.PUT_LINE(file,
    'REPORT: GENERATED ON ' || SYSDATE);
  UTL_FILE.NEW_LINE (file); ...
```

Using UTL_FILE: Example

```
  FOR emp_rec IN empc LOOP
    IF emp_rec.department_id <> olddeptno THEN
      UTL_FILE.PUT_LINE (file,
       'DEPARTMENT: ' || emp_rec.department_id);
      UTL_FILE.NEW_LINE (file);
    END IF;
    UTL_FILE.PUT_LINE (file,
       '  EMPLOYEE: ' || emp_rec.last_name ||
       ' earns: ' || emp_rec.salary);
    olddeptno := emp_rec.department_id;
    UTL_FILE.NEW_LINE (file);
  END LOOP;
  UTL_FILE.PUT_LINE(file,'*** END OF REPORT ***');
  UTL_FILE.FCLOSE (file);
EXCEPTION
 WHEN UTL_FILE.INVALID_FILEHANDLE THEN
  RAISE_APPLICATION_ERROR(-20001,'Invalid File.');
 WHEN UTL_FILE.WRITE_ERROR THEN
  RAISE_APPLICATION_ERROR (-20002, 'Unable to write
to file');
END sal_status;
/
```

Generating Web Pages with the HTP Package

- HTP package procedures generate HTML tags.
- The HTP package is used to generate HTML documents dynamically and can be invoked from:
  - A browser using Oracle HTTP Server and PL/SQL Gateway (mod_plsql) services
  - A SQL Developer script to display HTML output



Using the HTP Package Procedures

- Generate one or more HTML tags. For example:

```
htp.bold('Hello');                  -- <B>Hello</B>
htp.print('Hi <B>World</B>');       -- Hi <B>World</B>
```

- Are used to create a well-formed HTML document:

```
BEGIN                                   -- Generates:
 htp.htmlOpen;    ---------->             <HTML>
 htp.headOpen;    ---------->             <HEAD>
 htp.title('Welcome');  -->               <TITLE>Welcome</TITLE>
 htp.headClose;   ---------->             </HEAD>
 htp.bodyOpen;    ---------->             <BODY>
 htp.print('My home page');              My home page
 htp.bodyClose;   ---------->             </BODY>
 htp.htmlClose;   ---------->             </HTML>
END;
```

Creating an HTML File

To create an HTML file, perform the following steps:

1. Create a SQL script with the following commands:

```
SET SERVEROUTPUT ON
ACCEPT procname PROMPT "Procedure: "
EXECUTE &procname
EXECUTE owa_util.showpage
UNDEFINE proc
```

2. Load and execute the script in SQL Developer, supplying values for substitution variables.
3. Select, copy, and paste the HTML text that is generated to an HTML file.
4. Open the HTML file in a browser.

Using UTL_MAIL

The UTL_MAIL package:

- Is a utility for managing email that includes such commonly used email features as attachments, CC, BCC, and return receipt
- Requires the SMTP_OUT_SERVER database initialization parameter to be set
- Provides the following procedures:
  - SEND for messages without attachments
  - SEND_ATTACH_RAW for messages with binary attachments
  - SEND_ATTACH_VARCHAR2 for messages with text attachments

Installing and Using UTL_MAIL

- As SYSDBA:
  - Set the SMTP_OUT_SERVER (requires DBMS restart).

```
ALTER SYSTEM SET SMTP_OUT_SERVER='smtp.server.com'
SCOPE=SPFILE
```

  - Install the UTL_MAIL package.

```
@?/rdbms/admin/utlmail.sql
@?/rdbms/admin/prvtmail.plb
```

- As a developer, invoke a UTL_MAIL procedure:

```
BEGIN
 UTL_MAIL.SEND('otn@oracle.com','user@oracle.com',
   message => 'For latest downloads visit OTN',
   subject => 'OTN Newsletter');
END;
```

Sending Email with a Binary Attachment

Use the `UTL_MAIL.SEND_ATTACH_RAW` procedure:

```
CREATE OR REPLACE PROCEDURE send_mail_logo IS
BEGIN
  UTL_MAIL.SEND_ATTACH_RAW(
    sender => 'me@oracle.com',
    recipients => 'you@somewhere.net',
    message =>
      '<HTML><BODY>See attachment</BODY></HTML>',
    subject => 'Oracle Logo',
    mime_type => 'text/html'
    attachment => get_image('oracle.gif'),
    att_inline => true,
    att_mime_type => 'image/gif',
    att_filename => 'oralogo.gif');
END;
/
```

Sending Email with a Text Attachment

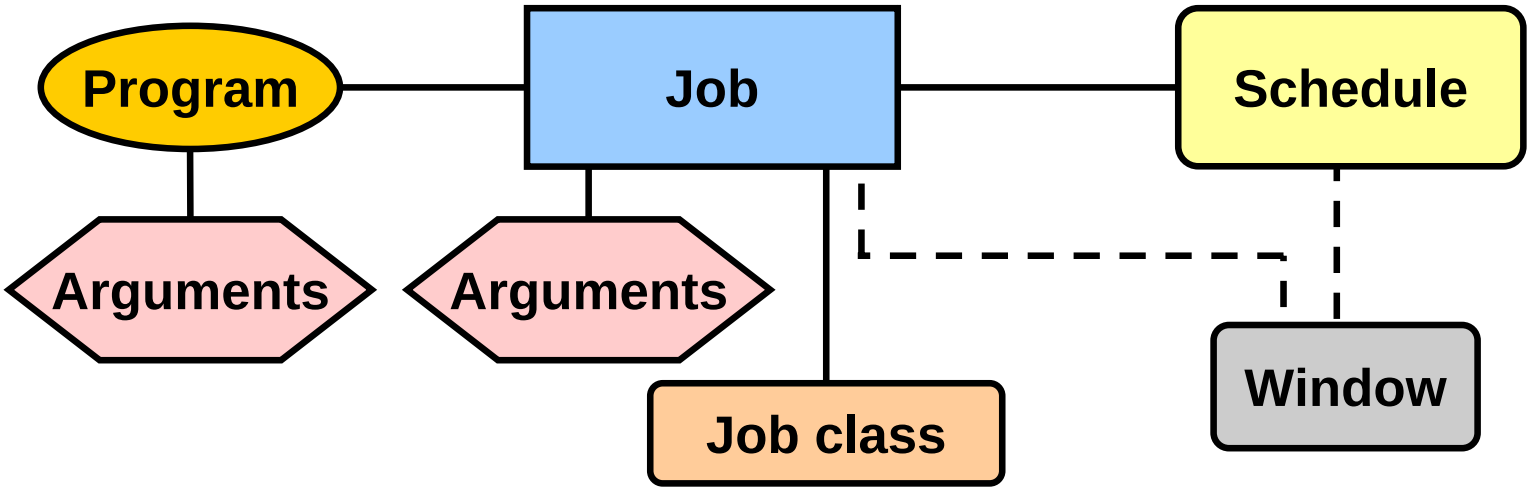
Use the `UTL_MAIL.SEND_ATTACH_VARCHAR2` procedure:

```
CREATE OR REPLACE PROCEDURE send_mail_file IS
BEGIN
  UTL_MAIL.SEND_ATTACH_VARCHAR2(
    sender => 'me@oracle.com',
    recipients => 'you@somewhere.net',
    message =>
      '<HTML><BODY>See attachment</BODY></HTML>',
    subject => 'Oracle Notes',
    mime_type => 'text/html'
    attachment => get_file('notes.txt'),
    att_inline => false,
    att_mime_type => 'text/plain',
    att_filename => 'notes.txt');
END;
/
```

DBMS_SCHEDULER Package

The database scheduler comprises several components to enable jobs to be run. Use the DBMS_SCHEDULER package to create each job with a:

- Unique job name
- Program ("what" should be executed)
- Schedule ("when" it should run)



Creating a Job

A job can be created in several ways by using a combination of in-line parameters, named `Programs`, and named `Schedules`. You can create a job with the `CREATE_JOB` procedure by:

- Using in-line information with the “what” and the schedule specified as parameters
- Using a named (saved) program and specifying the schedule in-line
- Specifying what should be done in-line and using a named Schedule
- Using named Program and Schedule components

Note: Creating a job requires the `CREATE JOB` system privilege.

Creating a Job with In-Line Parameters

Specify the type of code, code, start time, and frequency of the job to be run in the arguments of the `CREATE_JOB` procedure.

Here is an example that schedules a PL/SQL block every hour:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB(
   job_name => 'JOB_NAME',
   job_type => 'PLSQL_BLOCK',
   job_action => 'BEGIN ...; END;',
   start_date => SYSTIMESTAMP,
   repeat_interval=>'FREQUENCY=HOURLY;INTERVAL=1',
   enabled => TRUE);
END;
/
```

Creating a Job Using a Program

- Use `CREATE_PROGRAM` to create a program:

```
BEGIN
  DBMS_SCHEDULER.CREATE_PROGRAM(
   program_name => 'PROG_NAME',
   program_type => 'PLSQL_BLOCK',
   program_action => 'BEGIN ...; END;');
END;
```

- Use the overloaded `CREATE_JOB` procedure with its `program_name` parameter:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB('JOB_NAME',
   program_name => 'PROG_NAME',
   start_date => SYSTIMESTAMP,
   repeat_interval => 'FREQ=DAILY',
   enabled => TRUE);
END;
```

Creating a Job for a Program with Arguments

- Create a program:

```
DBMS_SCHEDULER.CREATE_PROGRAM(
  program_name => 'PROG_NAME',
  program_type => 'STORED_PROCEDURE',
  program_action => 'EMP_REPORT');
```

- Define an argument:

```
DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT(
  program_name => 'PROG_NAME',
  argument_name => 'DEPT_ID',
  argument_position=> 1, argument_type=> 'NUMBER',
  default_value => '50');
```

- Create a job specifying the number of arguments:

```
DBMS_SCHEDULER.CREATE_JOB('JOB_NAME', program_name
  => 'PROG_NAME', start_date => SYSTIMESTAMP,
  repeat_interval => 'FREQ=DAILY',
  number_of_arguments => 1, enabled => TRUE);
```

Creating a Job Using a Schedule

- Use `CREATE_SCHEDULE` to create a schedule:

```
BEGIN
  DBMS_SCHEDULER.CREATE_SCHEDULE('SCHED_NAME',
   start_date => SYSTIMESTAMP,
   repeat_interval => 'FREQ=DAILY',
   end_date => SYSTIMESTAMP +15);
END;
```

- Use `CREATE_JOB` by referencing the schedule in the `schedule_name` parameter:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB('JOB_NAME',
   schedule_name => 'SCHED_NAME',
   job_type => 'PLSQL_BLOCK',
   job_action => 'BEGIN ...; END;',
   enabled => TRUE);
END;
```

Setting the Repeat Interval for a Job

- Using a calendaring expression:

```
repeat_interval=> 'FREQ=HOURLY; INTERVAL=4'
repeat_interval=> 'FREQ=DAILY'
repeat_interval=> 'FREQ=MINUTELY;INTERVAL=15'
repeat_interval=> 'FREQ=YEARLY;
                   BYMONTH=MAR,JUN,SEP,DEC;
                   BYMONTHDAY=15'
```

- Using a PL/SQL expression:

```
repeat_interval=> 'SYSDATE + 36/24'
repeat_interval=> 'SYSDATE + 1'
repeat_interval=> 'SYSDATE + 15/(24*60)'
```

Creating a Job Using a Named Program and Schedule

- Create a named program called PROG_NAME by using the CREATE_PROGRAM procedure.
- Create a named schedule called SCHED_NAME by using the CREATE_SCHEDULE procedure.
- Create a job referencing the named program and schedule:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB('JOB_NAME',
   program_name => 'PROG_NAME',
   schedule_name => 'SCHED_NAME',
   enabled => TRUE);
END;
/
```

Managing Jobs

- Run a job:

```
DBMS_SCHEDULER.RUN_JOB('SCHEMA.JOB_NAME');
```

- Stop a job:

```
DBMS_SCHEDULER.STOP_JOB('SCHEMA.JOB_NAME');
```

- Drop a job even if it is currently running:

```
DBMS_SCHEDULER.DROP_JOB('JOB_NAME', TRUE);
```

Data Dictionary Views

- [DBA | ALL | USER]_SCHEDULER_JOBS
- [DBA | ALL | USER]_SCHEDULER_RUNNING_JOBS
- [DBA | ALL]_SCHEDULER_JOB_CLASSES
- [DBA | ALL | USER]_SCHEDULER_JOB_LOG
- [DBA | ALL | USER]_SCHEDULER_JOB_RUN_DETAILS
- [DBA | ALL | USER]_SCHEDULER_PROGRAMS

Summary

In this lesson, you should have learned how to:

- Use various preinstalled packages that are provided by the Oracle server
- Use the following packages:
  - `DBMS_OUTPUT` to buffer and display text
  - `UTL_FILE` to write operating system text files
  - `HTP` to generate HTML documents
  - `UTL_MAIL` to send messages with attachments
  - `DBMS_SCHEDULER` to automate processing
- Create packages individually or by using the `catproc.sql` script

Practice 5: Overview

This practice covers the following topics:

- Using `UTL_FILE` to generate a text report
- Using `HTP` to generate a Web page report
- Using `DBMS_SCHEDULER` to automate report processing