14

Oracle Supplied Packages

Schedule:	Timing	Topic
	50 minutes	Lecture
	50 minutes	Practice
	100 minutes	Total

Objectives

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

- **Write dynamic SQL statements using `DBMS_SQL` and `EXECUTE IMMEDIATE`**
- **Describe the use and application of some Oracle server-supplied packages:**
  - **`DBMS_DDL`**
  - **`DBMS_JOB`**
  - **`DBMS_OUTPUT`**
  - **`UTL_FILE`**
  - **`UTL_HTTP` and `UTL_TCP`**

ORACLE

Lesson Aim

In this lesson, you learn how to use some of the Oracle server supplied packages and to take advantage of their capabilities.

Instructor Note

In this lesson, only a few of the many Oracle server supplied packages and their procedures, functions, and parameters are explained.

Using Supplied Packages

Oracle-supplied packages:

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

ORACLE

Using Supplied Packages

Packages are provided with the Oracle server to allow either PL/SQL access to certain SQL features, or to extend the functionality of the database.

You can take advantage of the functionality provided by these packages when creating your application, or you may simply want to use these packages as ideas when you create your own stored procedures.

Most of the standard packages are created by running `catproc.sql`.

Instructor Note

The `catproc.sql` script is found in the `$ORACLE_HOME/rdbms/admin directory`. Other packages may have to be created in the `SYS` schema by running corresponding scripts located in the directory `$ORACLE_HOME/rdbms/admin`. The scripts to create supplied packages have prefix `DBMS_`.

Using Native Dynamic SQL

Dynamic SQL:

- **Is a SQL statement that contains variables that can change during runtime**
- **Is a SQL statement with placeholders and is stored as a character string**
- **Enables general-purpose code to be written**
- **Enables data-definition, data-control, or session-control statements to be written and executed from PL/SQL**
- **Is written using either `DBMS_SQL` or native dynamic SQL**

ORACLE

Using Native Dynamic SQL (Dynamic SQL)

You can write PL/SQL blocks that use dynamic SQL. Dynamic SQL statements are not embedded in your source program but rather are stored in character strings that are input to, or built by, the program. That is, the SQL statements can be created dynamically at run time by using variables. For example, you use dynamic SQL to create a procedure that operates on a table whose name is not known until run time, or to write and execute a data definition language (DDL) statement (such as `CREATE TABLE`), a data control statement (such as `GRANT`), or a session control statement (such as `ALTER SESSION`). In PL/SQL, such statements cannot be executed statically.

In Oracle8, and earlier, you have to use `DBMS_SQL` to write dynamic SQL.

In Oracle 8*i*, you can use `DBMS_SQL` or native dynamic SQL. The `EXECUTE IMMEDIATE` statement can perform dynamic single-row queries. Also, this is used for functionality such as objects and collections, which are not supported by `DBMS_SQL`. If the statement is a multirow `SELECT` statement, you use `OPEN-FOR`, `FETCH`, and `CLOSE` statements.

Execution Flow

SQL statements go through various stages:

- **Parse**
- **Bind**
- **Execute**
- **Fetch**

Note: Some stages may be skipped.

ORACLE

Steps to Process SQL Statements

All SQL statements have to go through various stages. Some stages may be skipped.

Parse

Every SQL statement must be parsed. Parsing the statement includes checking the statement's syntax and validating the statement, ensuring that all references to objects are correct, and ensuring that the relevant privileges to those objects exist.

Bind

After parsing, the Oracle server knows the meaning of the Oracle statement but still may not have enough information to execute the statement. The Oracle server may need values for any bind variable in the statement. The process of obtaining these values is called binding variables.

Execute

At this point, the Oracle server has all necessary information and resources, and the statement is executed.

Fetch

In the fetch stage, rows are selected and ordered (if requested by the query), and each successive fetch retrieves another row of the result, until the last row has been fetched. You can fetch queries, but not the DML statements.

Instructor Note

Do not go into too much detail when discussing this topic. Processing of SQL statements is covered in more detail in other courses.

Using the DBMS_SQL Package

The DBMS_SQL package is used to write dynamic SQL in stored procedures and to parse DDL statements. Some of the procedures and functions of the package include:

- **OPEN_CURSOR**
- **PARSE**
- **BIND_VARIABLE**
- **EXECUTE**
- **FETCH_ROWS**
- **CLOSE_CURSOR**

ORACLE

Using the DBMS_SQL Package

Using DBMS_SQL, you can write stored procedures and anonymous PL/SQL blocks that use dynamic SQL.

DBMS_SQL can issue data definition language statements in PL/SQL. For example, you can choose to issue a DROP TABLE statement from within a stored procedure.

The operations provided by this package are performed under the current user, not under the package owner SYS. Therefore, if the caller is an anonymous PL/SQL block, the operations are performed according to the privileges of the current user; if the caller is a stored procedure, the operations are performed according to the owner of the stored procedure.

Using this package to execute DDL statements can result in a deadlock. The most likely reason for this is that the package is being used to drop a procedure that you are still using.

Components of the DBMS_SQL Package

The DBMS_SQL package uses dynamic SQL to access the database.

Function or Procedure	Description
OPEN_CURSOR	Opens a new cursor and assigns a cursor ID number
PARSE	Parses the DDL or DML statement: that is, checks the statement's syntax and associates it with the opened cursor (DDL statements are immediately executed when parsed)
BIND_VARIABLE	Binds the given value to the variable identified by its name in the parsed statement in the given cursor
EXECUTE	Executes the SQL statement and returns the number of rows processed
FETCH_ROWS	Retrieves a row for the specified cursor (for multiple rows, call in a loop)
CLOSE_CURSOR	Closes the specified cursor

Instructor Note

DBMS_SQL has many more procedures and functions than are shown here. The procedures and functions in the table should show the main stages of dynamic SQL statements. If you want to cover DBMS_SQL in more detail, start by showing the data dictionary information on DBMS_SQL.

```
SELECT text FROM all_source WHERE name ='DBMS_SQL' ORDER BY LINE;
```

or

Use DESC DBMS_SQL to view the description of the procedures and functions in the package.

The above SELECT statement returns 1050 rows.

Using DBMS_SQL

```
CREATE OR REPLACE PROCEDURE delete_all_rows
   (p_tab_name IN VARCHAR2, p_rows_del OUT NUMBER)
 IS
   cursor_name   INTEGER;
 BEGIN
   cursor_name := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(cursor_name, 'DELETE FROM '||p_tab_name,
                  DBMS_SQL.NATIVE );
   p_rows_del := DBMS_SQL.EXECUTE (cursor_name);
   DBMS_SQL.CLOSE_CURSOR(cursor_name);
END;
/
```

Use dynamic SQL to delete rows

```
VARIABLE deleted NUMBER
EXECUTE  delete_all_rows('employees', :deleted)
PRINT deleted
```

```
PL/SQL procedure successfully completed.
```

DELETED
109

ORACLE

Example of a DBMS_SQL Package

In the slide, the table name is passed into the DELETE_ALL_ROWS procedure by using an IN parameter. The procedure uses dynamic SQL to delete rows from the specified table. The number of rows that are deleted as a result of the successful execution of the dynamic SQL are passed to the calling environment through an OUT parameter.

How to Process Dynamic DML

1. Use OPEN_CURSOR to establish an area in memory to process a SQL statement.
2. Use PARSE to establish the validity of the SQL statement.
3. Use the EXECUTE function to run the SQL statement. This function returns the number of row processed.
4. Use CLOSE_CURSOR to close the cursor.

Instructor Note

If you want to demonstrate this code, remember to do a ROLLBACK in order to undo the deletion. The sample code contains DELETE command on another test table that is created in the script.

Demonstration: 14_dmldynam.sql, 14_ddldynam.sql, and 14_seldynam.sql

Purpose: The first two scripts create a table to perform either a DROP TABLE or a DELETE command on it. The last script accepts three parameters and selects data from the table. Use EMPLOYEES, JOB, and MANAGER as the values when prompted. The values are not case-sensitive.

Using the EXECUTE IMMEDIATE Statement

Use the EXECUTE IMMEDIATE statement for native dynamic SQL with better performance.

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

- **INTO is used for single-row queries and specifies the variables or records into which column values are retrieved.**
- **USING is used to hold all bind arguments. The default parameter mode is IN.**

ORACLE

Using the EXECUTE IMMEDIATE Statement

Syntax Definition

Parameter	Description
dynamic_string	A string expression that represents a dynamic SQL statement (without terminator) or a PL/SQL block (with terminator)
define_variable	A variable that stores the selected column value
record	A user-defined or %ROWTYPE record that stores a selected row
bind_argument	An expression whose value is passed to the dynamic SQL statement or PL/SQL block

You can use the INTO clause for a single-row query, but you must use OPEN-FOR, FETCH, and CLOSE for a multirow query.

Note: The syntax shown in the slide is not complete. The other clauses of the statement are discussed in the *Advanced PL/SQL* course.

Instructor Note

To process most dynamic SQL statements, you use the EXECUTE IMMEDIATE statement. However, to process a multirow query (SELECT statement), you must use the OPEN-FOR, FETCH, and CLOSE statements.

OPEN-FOR, FETCH, and CLOSE are not covered in this course, because they use cursor variables.

Using the EXECUTE IMMEDIATE Statement (continued)

In the EXECUTE IMMEDIATE statement:

- The INTO clause specifies the variables or record into which column values are retrieved. It is used only for single-row queries. For each value retrieved by the query, there must be a corresponding, type-compatible variable or field in the INTO clause.
- The RETURNING INTO clause specifies the variables into which column values are returned. It is used only for DML statements that have a RETURNING clause (without a BULK COLLECT clause). For each value returned by the DML statement, there must be a corresponding, type-compatible variable in the RETURNING INTO clause.
- The USING clause holds all bind arguments. The default parameter mode is IN. For DML statements that have a RETURNING clause, you can place OUT arguments in the RETURNING INTO clause without specifying the parameter mode, which, by definition, is OUT. If you use both the USING clause and the RETURNING INTO clause, the USING clause can contain only IN arguments.

At run time, bind arguments replace corresponding placeholders in the dynamic string. Thus, every placeholder must be associated with a bind argument in the USING clause or RETURNING INTO clause. You can use numeric, character, and string literals as bind arguments, but you cannot use Boolean literals (TRUE, FALSE, and NULL).

Dynamic SQL supports all the SQL data types. For example, define variables and bind arguments can be collections, LOBs, instances of an object type, and REFs. As a rule, dynamic SQL does not support PL/SQL-specific types. For example, define variables and bind arguments cannot be Booleans or index-by tables. The only exception is that a PL/SQL record can appear in the INTO clause.

You can execute a dynamic SQL statement repeatedly, using new values for the bind arguments. However, you incur some overhead because EXECUTE IMMEDIATE reparses the dynamic string before every execution.

Dynamic SQL Using EXECUTE IMMEDIATE

```
CREATE PROCEDURE del_rows
  (p_table_name  IN  VARCHAR2,
   p_rows_deld   OUT NUMBER)
IS
BEGIN
  EXECUTE IMMEDIATE 'delete from '||p_table_name;
  p_rows_deld := SQL%ROWCOUNT;
END;
/
```

Procedure created.

```
VARIABLE deleted NUMBER
EXECUTE del_rows('test_employees',:deleted)
PRINT deleted
```

PL/SQL procedure successfully completed.

DELETED
109

ORACLE

Dynamic SQL Using EXECUTE IMMEDIATE

This is the same dynamic SQL as seen with DBMS_SQL, using the Oracle8*i* statement EXECUTE IMMEDIATE. The EXECUTE IMMEDIATE statement prepares (parses) and immediately executes the dynamic SQL statement.

Instructor Note

If you run this example, remember to use ROLLBACK to undo the deletion.

Using the DBMS_DDL Package

The DBMS_DDL Package:

- **Provides access to some SQL DDL statements from stored procedures**
- **Includes some procedures:**
  - **ALTER_COMPILE (object_type, owner, object_name)**

```
DBMS_DDL.ALTER_COMPILE('PROCEDURE','A_USER','QUERY_EMP')
```

  - **ANALYZE_OBJECT (object_type, owner, name, method)**

```
DBMS_DDL.ANALYZE_OBJECT('TABLE','A_USER','JOBS','COMPUTE')
```

Note: This package runs with the privileges of calling user, rather than the package owner SYS.

ORACLE

Using the DBMS_DDL package

This package provides access to some SQL DDL statements, which you can use in PL/SQL programs.

DBMS_DDL is not allowed in triggers, in procedures called from Forms Builder, or in remote sessions. This package runs with the privileges of calling user, rather than the package owner SYS.

Practical Uses

- You can recompile your modified PL/SQL program units by using DBMS_DDL.ALTER_COMPILE. The object type must be either procedure, function, package, package body, or trigger.
- You can analyze a single object, using DBMS_DDL.ANALYZE_OBJECT. (There is a way of analyzing more than one object at a time, using DBMS_UTILITY.) The object type should be TABLE, CLUSTER, or INDEX. The method must be COMPUTE, ESTIMATE, or DELETE.
- This package gives developers access to ALTER and ANALYZE SQL statements through PL/SQL environments.

Instructor Note

You can check the column LAST_DDL_TIME in USER_OBJECTS and LAST_ANALYZED in USER_TABLES to monitor the results of these operations.

Using DBMS_JOB for Scheduling

DBMS_JOB Enables the scheduling and execution of PL/SQL programs:

- **Submitting jobs**
- **Executing jobs**
- **Changing execution parameters of jobs**
- **Removing jobs**
- **Suspending Jobs**

ORACLE

Scheduling Jobs by Using DBMS_JOB

The package DBMS_JOB is used to schedule PL/SQL programs to run. Using DBMS_JOB, you can submit PL/SQL programs for execution, execute PL/SQL programs on a schedule, identify when PL/SQL programs should run, remove PL/SQL programs from the schedule, and suspend PL/SQL programs from running.

It can be used to schedule batch jobs during nonpeak hours or to run maintenance programs during times of low usage.

Instructor Note

In the `init.ora` file, the JOB_QUEUE_PROCESSES parameter must be set to greater than zero (at least 1), because this parameter enables job queue processing in the background.

DBMS_JOB Subprograms

Available subprograms include:

- **SUBMIT**
- **REMOVE**
- **CHANGE**
- **WHAT**
- **NEXT_DATE**
- **INTERVAL**
- **BROKEN**
- **RUN**

ORACLE

DBMS_JOB Subprograms

Subprogram	Description
SUBMIT	Submits a job to the job queue
REMOVE	Removes a specified job from the job queue
CHANGE	Alters a specified job that has already been submitted to the job queue (you can alter the job description, the time at which the job will be run, or the interval between executions of the job)
WHAT	Alters the job description for a specified job
NEXT_DATE	Alters the next execution time for a specified job
INTERVAL	Alters the interval between executions for a specified job
BROKEN	Disables job execution (if a job is marked as broken, the Oracle server does not attempt to execute it)
RUN	Forces a specified job to run

Submitting Jobs

You can submit jobs by using `DBMS_JOB.SUBMIT`.

Available parameters include:

- **`JOB OUT BINARY_INTEGER`**
- **`WHAT IN VARCHAR2`**
- **`NEXT_DATE IN DATE DEFAULT SYSDATE`**
- **`INTERVAL IN VARCHAR2 DEFAULT 'NULL'`**
- **`NO_PARSE IN BOOLEAN DEFAULT FALSE`**

ORACLE

`DBMS_JOB.SUBMIT` Parameters

The `DBMS_JOB.SUBMIT` procedure adds a new job to the job queue. It accepts five parameters and returns the number of a job submitted through the `OUT` parameter `JOB`. The descriptions of the parameters are listed below.

Parameter	Mode	Description
JOB	OUT	Unique identifier of the job
WHAT	IN	PL/SQL code to execute as a job
NEXT_DATE	IN	Next execution date of the job
INTERVAL	IN	Date function to compute the next execution date of a job
NO_PARSE	IN	Boolean flag that indicates whether to parse the job at job submission (the default is false)

Note: An exception is raised if the interval does not evaluate to a time in the future.

Submitting Jobs

Use DBMS_JOB.SUBMIT to place a job to be executed in the job queue.

```
VARIABLE jobno NUMBER
BEGIN
  DBMS_JOB.SUBMIT (
   job  => :jobno,
   what => 'OVER_PACK.ADD_DEPT(''EDUCATION'',2710);',
   next_date =>  TRUNC(SYSDATE + 1),
   interval  =>  'TRUNC(SYSDATE + 1)'
  );
  COMMIT;
END;
/
PRINT jobno
```

PL/SQL procedure successfully completed.

JOBNO
1

ORACLE

Example

The block of code in the slide submits the `ADD_DEPT` procedure of the `OVER_PACK` package to the job queue. The job number is returned through the `JOB` parameter. The `WHAT` parameter must be enclosed in single quotation marks and must include a semicolon at the end of the text string. This job is submitted to run every day at midnight.

Note: In the example, the parameters are passed using named notation.

The transactions in the submitted job are not committed until either `COMMIT` is issued, or `DBMS_JOB.RUN` is executed to run the job. `COMMIT` in the slide commits the transaction.

Instructor Note

You can demonstrate this code with the `14_16s.sql` file.

Changing Job Characteristics

- **DBMS_JOB.CHANGE: Changes the WHAT, NEXT_DATE, and INTERVAL parameters**
- **DBMS_JOB.INTERVAL: Changes the INTERVAL parameter**
- **DBMS_JOB.NEXT_DATE: Changes the next execution date**
- **DBMS_JOB.WHAT: Changes the WHAT parameter**

ORACLE

Changing Jobs After Being Submitted

The CHANGE, INTERVAL, NEXT_DATE, and WHAT procedures enable you to modify job characteristics after a job is submitted to the queue. Each of these procedures takes the JOB parameter as an IN parameter indicating which job is to be changed.

Example

The following code changes job number 1 to execute on the following day at 6:00 a.m. and every four hours after that.

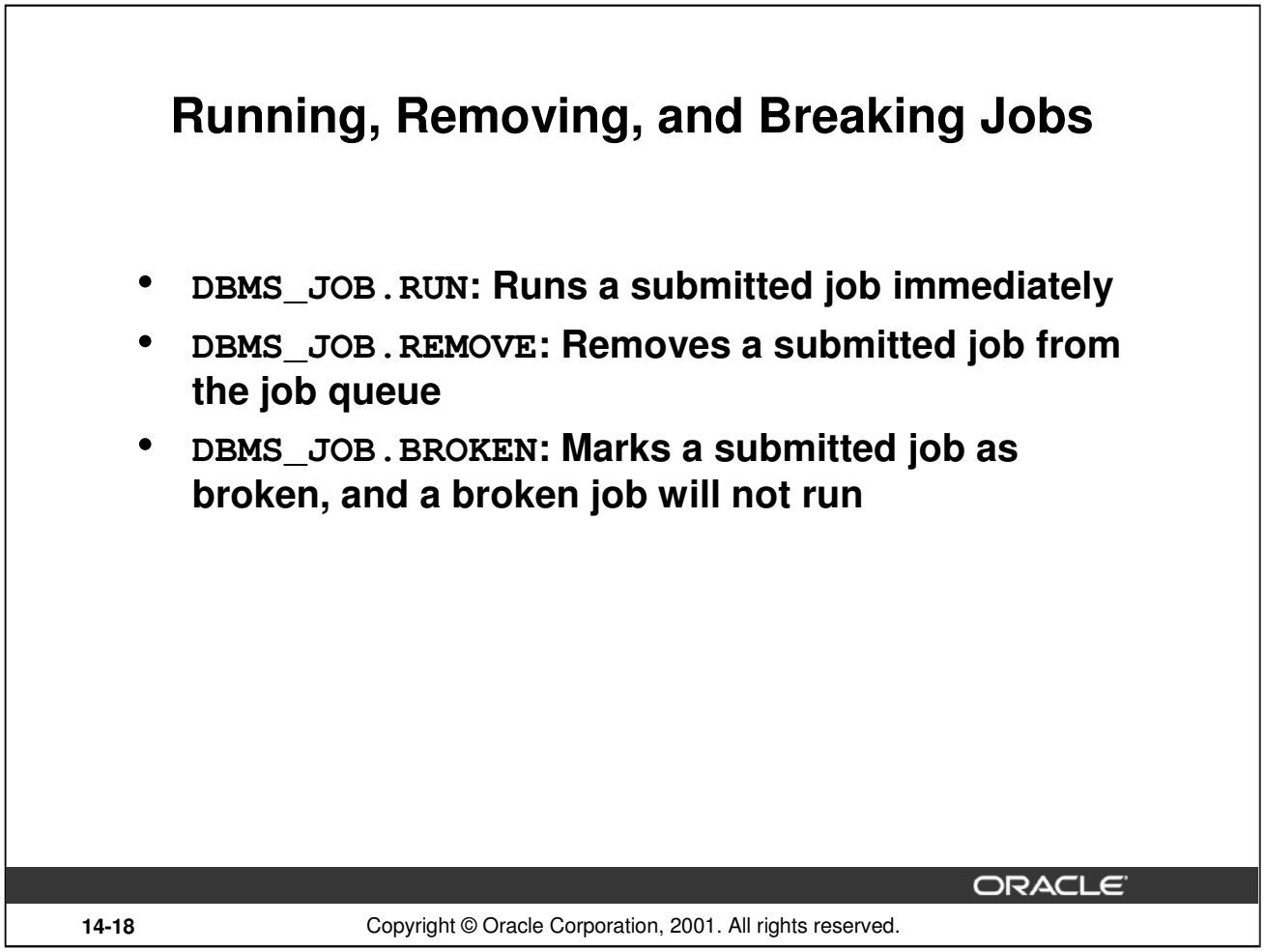
```
BEGIN
   DBMS_JOB.CHANGE(1, NULL, TRUNC(SYSDATE+1)+6/24, 'SYSDATE+4/24');
END;
/
```

```
PL/SQL procedure successfully completed.
```

Note: Each of these procedures can be executed on jobs owned by the username to which the session is connected. If the parameter what, next_date, or interval is NULL, then the last values assigned to those parameters are used.

Instructor Note

You can demonstrate this code with the 14_17n.sql file.



Running, Removing, and Breaking Jobs

The `DBMS_JOB.RUN` procedure executes a job immediately. Pass the job number that you want to run immediately to the procedure.

```
EXECUTE DBMS_JOB.RUN(1)
```

The `DBMS_JOB.REMOVE` procedure removes a submitted job from the job queue. Pass the job number that you want to remove from the queue to the procedure.

```
EXECUTE DBMS_JOB.REMOVE(1)
```

The `DBMS_JOB.BROKEN` marks a job as broken or not broken. Jobs are not broken by default. You can change a job to the broken status. A broken job will not run. There are three parameters for this procedure. The `JOB` parameter identifies the job to be marked as broken or not broken. The `BROKEN` parameter is a Boolean parameter. Set this parameter to `FALSE` to indicate that a job is not broken, and set it to `TRUE` to indicate that it is broken. The `NEXT_DATE` parameter identifies the next execution date of the job.

```
EXECUTE DBMS_JOB.BROKEN(1, TRUE)
```

Viewing Information on Submitted Jobs

- **Use the `DBA_JOBS` dictionary view to see the status of submitted jobs.**

```
SELECT job, log_user, next_date, next_sec,
       broken, what
FROM DBA_JOBS;
```

JOB	LOG_USER	NEXT_DATE	NEXT_SEC	B	WHAT
1	PLSQL	28-SEP-01	06:00:00	N	OVER_PACK.ADD_DEPT('EDUCATION',2710);

- **Use the `DBA_JOBS_RUNNING` dictionary view to display jobs that are currently running.**

ORACLE

Viewing Information on Submitted Jobs

The `DBA_JOBS` and `DBA_JOBS_RUNNING` dictionary views display information about jobs in the queue and jobs that have run. To be able to view the dictionary information, users should be granted the `SELECT` privilege on `SYS.DBA_JOBS`.

The query shown in the slide displays the job number, the user who submitted the job, the scheduled date for the job to run, the time for the job to run, and the PL/SQL block executed as a job.

Use the `USER_JOBS` data dictionary view to display information about jobs in the queue for you. This view has the same structure as the `DBA_JOBS` view.

Instructor Note

You may want to describe these views in *i*SQL*Plus. The columns are fairly descriptive of the information held in them. Refer students to the reference material for more in-depth descriptions.

You can demonstrate this code with the `14_19s.sql` file.

Using the DBMS_OUTPUT Package

The DBMS_OUTPUT package enables you to output messages from PL/SQL blocks. Available procedures include:

- **PUT**
- **NEW_LINE**
- **PUT_LINE**
- **GET_LINE**
- **GET_LINES**
- **ENABLE/DISABLE**

ORACLE

Using the DBMS_OUTPUT Package

The DBMS_OUTPUT package outputs values and messages from any PL/SQL block.

Function or Procedure	Description
PUT	Appends text from the procedure to the current line of the line output buffer
NEW_LINE	Places an end_of_line marker in the output buffer
PUT_LINE	Combines the action of PUT and NEW_LINE
GET_LINE	Retrieves the current line from the output buffer into the procedure
GET_LINES	Retrieves an array of lines from the output buffer into the procedure
ENABLE/DISABLE	Enables or disables calls to the DBMS_OUTPUT procedures

Practical Uses

- You can output intermediary results to the window for debugging purposes.
- This package enables developers to closely follow the execution of a function or procedure by sending messages and values to the output buffer.

Interacting with Operating System Files

- **UTL_FILE Oracle-supplied package:**
  - **Provides text file I/O capabilities**
  - **Is available with version 7.3 and later**
- **The DBMS_LOB Oracle-supplied package:**
  - **Provides read-only operations on external BFILES**
  - **Is available with version 8 and later**
  - **Enables read and write operations on internal LOBs**

ORACLE

Interacting with Operating System Files

Two Oracle-supplied packages are provided. You can use them to access operating system files.

With the Oracle-supplied `UTL_FILE` package, you can read from and write to operating system files. This package is available with database version 7.3 and later and the PL/SQL version 2.3 and later.

With the Oracle-supplied package `DBMS_LOB`, you can read from binary files on the operating system. This package is available from the database version 8.0 and later. This package is discussed later in the lesson "Manipulating Large Objects."

What Is the UTL_FILE Package?

- Extends I/O to text files within PL/SQL
- Provides security for directories on the server through the init.ora file
- Is similar to standard operating system I/O
  - Open files
  - Get text
  - Put text
  - Close files
  - Use the exceptions specific to the UTL_FILE package

ORACLE

The UTL_FILE Package

The UTL_FILE package provides text file I/O from within PL/SQL. Client-side security implementation uses normal operating system file permission checking. Server-side security is implemented through restrictions on the directories that can be accessed. In the init.ora file, the initialization parameter UTL_FILE_DIR is set to the accessible directories desired.

```
UTL_FILE_DIR = directory_name
```

For example, the following initialization setting indicates that the directory /usr/ngreenbe/my_app is accessible to the fopen function, assuming that the directory is accessible to the database server processes. This parameter setting is case-sensitive on case-sensitive operating systems.

```
UTL_FILE_DIR = /user/ngreenbe/my_app
```

The directory should be on the same machine as the database server. Using the following setting turns off database permissions and makes all directories that are accessible to the database server processes also accessible to the UTL_FILE package.

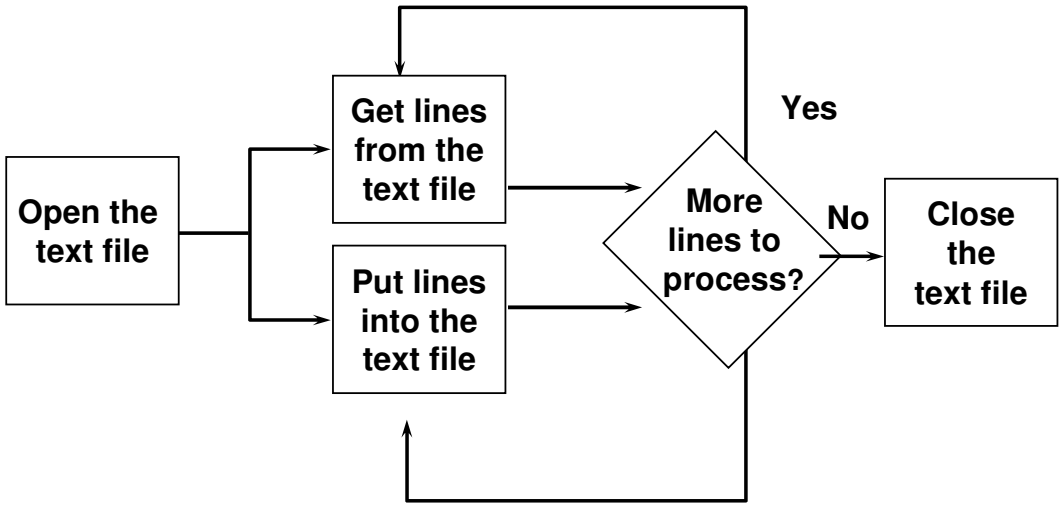
```
UTL_FILE_DIR = *
```

Using the procedures and functions in the package, you can open files, get text from files, put text into files, and close files. There are seven exceptions declared in the package to account for possible errors raised during execution.

Instructor Note

UTL_FILE_DIR = * is not recommended. Your init.ora file needs to include the parameter UTL_FILE_DIR. If it does not, edit the init.ora file and add the parameter. Then restart the database by using the SHUTDOWN and STARTUP commands.

File Processing Using the UTL_FILE Package



ORACLE

File Processing Using the UTL_FILE Package

Before using the UTL_FILE package to read from or write to a text file, you must first check whether the text file is open by using the IS_OPEN function. If the file is not open, you open the file with the FOPEN function. You then either read the file or write to the file until processing is done. At the end of file processing, use the FCLOSE procedure to close the file.

Note: A summary of the procedures and functions within the UTL_FILE package is listed on the next page.

Instructor Note

Reading or writing to a file in PL/SQL or SQL is the same as reading or writing to files in other third-generation languages.

UTL_FILE Procedures and Functions

- Function FOPEN
- Function IS_OPEN
- Procedure GET_LINE
- Procedure PUT, PUT_LINE, PUTF
- Procedure NEW_LINE
- Procedure FFLUSH
- Procedure FCLOSE, FCLOSE_ALL

ORACLE

The UTL_FILE Package: Procedures and Functions

Function or Procedure	Description
FOPEN	A function that opens a file for input or output and returns a file handle used in subsequent I/O operations
IS_OPEN	A function that returns a Boolean value whenever a file handle refers to an open file
GET_LINE	A procedure that reads a line of text from the opened file and places the text in the output buffer parameter (the maximum size of an input record is 1,023 bytes unless you specify a larger size in the overloaded version of FOPEN)
PUT, PUT_LINE	A procedure that writes a text string stored in the buffer parameter to the opened file (no line terminator is appended by put; use new_line to terminate the line, or use PUT_LINE to write a complete line with a terminator)
PUTF	A formatted put procedure with two format specifiers: %s and \n (use %s to substitute a value into the output string. \n is a new line character)
NEW_LINE	Procedure that terminates a line in an output file
FFLUSH	Procedure that writes all data buffered in memory to a file
FCLOSE	Procedure that closes an opened file
FCLOSE_ALL	Procedure that closes all opened file handles for the session

Note: The maximum size of an input record is 1,023 bytes unless you specify a larger size in the overloaded version of FOPEN.

Exceptions Specific to the UTL_FILE Package

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

ORACLE

Exceptions to the UTL_FILE Package

The UTL_FILE package declares seven exceptions that are raised to indicate an error condition in the operating system file processing.

Exception Name	Description
INVALID_PATH	The file location or filename was invalid.
INVALID_MODE	The OPEN_MODE parameter in FOPEN was invalid.
INVALID_FILEHANDLE	The file handle was invalid.
INVALID_OPERATION	The file could not be opened or operated on as requested.
READ_ERROR	An operating system error occurred during the read operation.
WRITE_ERROR	An operating system error occurred during the write operation.
INTERNAL_ERROR	An unspecified error occurred in PL/SQL.

Note: These exceptions must be prefaced with the package name.

UTL_FILE procedures can also raise predefined PL/SQL exceptions such as NO_DATA_FOUND or VALUE_ERROR.

The FOPEN and IS_OPEN Functions

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

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

ORACLE

FOPEN Function Parameters

Syntax Definitions

Where	location	Is the operating-system-specific string that specifies the directory or area in which to open the file
	filename	Is the name of the file, including the extension, without any pathing information
	open_mode	Is string that specifies how the file is to be opened; Supported values are: 'r' read text (use GET_LINE) 'w' write text (PUT, PUT_LINE, NEW_LINE, PUTF, FFLUSH) 'a' append text (PUT, PUT_LINE, NEW_LINE, PUTF, FFLUSH)

The return value is the file handle that is passed to all subsequent routines that operate on the file.

IS_OPEN Function

The function IS_OPEN tests a file handle to see if it identifies an opened file. It returns a Boolean value indicating whether the file has been opened but not yet closed.

Note: For the full syntax, refer to *Oracle9i Supplied PL/SQL Packages and Types Reference*.

Using UTL_FILE

sal_status.sql

```
CREATE OR REPLACE PROCEDURE sal_status
(p_filedir IN VARCHAR2,  p_filename IN VARCHAR2)
IS
 v_filehandle UTL_FILE.FILE_TYPE;
 CURSOR emp_info IS
   SELECT last_name, salary, department_id
    FROM employees
    ORDER BY department_id;
 v_newdeptno employees.department_id%TYPE;
 v_olddeptno employees.department_id%TYPE := 0;
BEGIN
 v_filehandle := UTL_FILE.FOPEN (p_filedir, p_filename,'w');
UTL_FILE.PUTF (v_filehandle,'SALARY REPORT: GENERATED ON
                            %s\n', SYSDATE);
UTL_FILE.NEW_LINE (v_filehandle);
 FOR v_emp_rec IN emp_info LOOP
  v_newdeptno := v_emp_rec.department_id;
...
```

ORACLE

Using UTL_FILE

Example

The SAL_STATUS procedure creates a report of employees for each department and their salaries. This information is sent to a text file by using the UTL_FILE procedures and functions.

The variable v_filehandle uses a type defined in the UTL_FILE package. This package defined type is a record with a field called ID of the BINARY_INTEGER datatype.

```
TYPE file_type IS RECORD (id BINARY_INTEGER);
```

The contents of file_type are private to the UTL_FILE package. Users of the package should not reference or change components of this record.

The names of the text file and the location for the text file are provided as parameters to the program.

```
EXECUTE sal_status('C:\UTL_FILE', 'SAL_RPT.TXT')
```

Note: The file location shown in the above example is defined as value of UTL_FILE_DIR in the init.ora file as follows: UTL_FILE_DIR = C:\UTL_FILE.

When reading a complete file in a loop, you need to exit the loop using the NO_DATA_FOUND exception. UTL_FILE output is sent synchronously. DBMS_OUTPUT procedures do not produce output until the procedure is completed.

Instructor Note

You can demonstrate this code with the 14_27s.sql and 14_27n.sql files. Run the script 14_27s.sql first. You can demonstrate the output generated in the file instructor.txt after you run 14_27n.sql.

Using UTL_FILE

sal_status.sql

```
    ...
    IF v_newdeptno <> v_olddeptno THEN
      UTL_FILE.PUTF (v_filehandle, 'DEPARTMENT: %s\n',
                     v_emp_rec.department_id);
    END IF;
    UTL_FILE.PUTF (v_filehandle,'  EMPLOYEE: %s earns: %s\n',
                   v_emp_rec.last_name, v_emp_rec.salary);
    v_olddeptno := v_newdeptno;
  END LOOP;
  UTL_FILE.PUT_LINE (v_filehandle, '*** END OF REPORT ***');
  UTL_FILE.FCLOSE (v_filehandle);
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;
/
```

ORACLE

Using UTL_FILE (continued)

The output for this report in the sal_rpt.txt file is as follows:

```
SALARY REPORT: GENERATED ON 08-MAR-01

DEPARTMENT: 10
    EMPLOYEE: Whalen earns: 4400
DEPARTMENT: 20
    EMPLOYEE: Hartstein earns: 13000
    EMPLOYEE: Fay earns: 6000
DEPARTMENT: 30
    EMPLOYEE: Raphaely earns: 11000
    EMPLOYEE: Khoo earns: 3100
    ...
DEPARTMENT: 100
    EMPLOYEE: Greenberg earns: 12000
    ...
DEPARTMENT: 110
    EMPLOYEE: Higgins earns: 12000
    EMPLOYEE: Gietz earns: 8300
    EMPLOYEE: Grant earns: 7000
*** END OF REPORT ***
```

The UTL_HTTP Package

The UTL_HTTP package:

- **Enables HTTP callouts from PL/SQL and SQL to access data on the Internet**
- **Contains the functions REQUEST and REQUEST_PIECES which take the URL of a site as a parameter, contact that site, and return the data obtained from that site**
- **Requires a proxy parameter to be specified in the above functions, if the client is behind a firewall**
- **Raises INIT_FAILED or REQUEST_FAILED exceptions if HTTP call fails**
- **Reports an HTML error message if specified URL is not accessible**

ORACLE

The UTL_HTTP Package

UTL_HTTP is a package that allows you to make HTTP requests directly from the database. The UTL_HTTP package makes hypertext transfer protocol (HTTP) callouts from PL/SQL and SQL. You can use it to access data on the Internet or to call Oracle Web Server Cartridges. By coupling UTL_HTTP with the DBMS_JOBS package, you can easily schedule reoccurring requests be made from your database server out to the Web.

This package contains two entry point functions: REQUEST and REQUEST_PIECES. Both functions take a string universal resource locator (URL) as a parameter, contact the site, and return the HTML data obtained from the site. The REQUEST function returns up to the first 2000 bytes of data retrieved from the given URL. The REQUEST_PIECES function returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL.

If the HTTP call fails, for a reason such as that the URL is not properly specified in the HTTP syntax then the REQUEST_FAILED exception is raised. If initialization of the HTTP-callout subsystem fails, for a reason such as a lack of available memory, then the INIT_FAILED exception is raised.

If there is no response from the specified URL, then a formatted HTML error message may be returned.

If REQUEST or REQUEST_PIECES fails by returning either an exception or an error message, then verify the URL with a browser, to verify network availability from your machine. If you are behind a firewall, then you need to specify proxy as a parameter, in addition to the URL.

This package is covered in more detail in the course *Administering Oracle9i Application Server*.

For more information, refer to *Oracle9i Supplied PL/SQL Packages Reference*.

Using the UTL_HTTP Package

```
SELECT UTL_HTTP.REQUEST('http://www.oracle.com',
                        'edu-proxy.us.oracle.com')
FROM DUAL;
```

UTL_HTTP.REQUEST('HTTP://WWW.ORACLE.COM','EDU-PROXY.US.ORACLE.COM')

<html> <head> <title>Oracle Corporation</title> <meta name="description" content="Oracle Corporation provides the software that powers the Internet. F or more information about Oracle, please call 650/506-7000."> <meta name="keywords" content="Oracle, Oracle Corporation, Oracle Corp,Oracle8i, Oracle 9i, 8i, 9i"> <script language="JavaScript" src="http://www.oracle.com/admin/jscripts/lib.js"> </script> </head> <body bgcolor="#FFFFFF" text="#000000 " link="#000000" vlink="#FF0000"> <!--Start Header--> <table border=0 cellspacing=0 cellpadding=3 width=850 align="center"> <tr> <td align= center valign="middle"> <img src="per sonalize.gif" width="50" height="50" border="0" alt="My Oracle"> <img src="products .gif" width="50" height="50" border="0" alt="Products"> <img src="store.gif" width=" 50" height="50" border="0" alt="Store"> </td> <td align=center valign="middle" width="34%"> <i mg src="logo.gif" width="175" height="28" border=0 alt="Oracle.com" align="middle"> </td> <td align=center valign="middle"> <img src="download.gif" width="50" height="50" border="0" alt="Download"> <img src="contact.gif" width="50" height="50" border="0" alt="Contact"> <img src="search.gif " width="50" height="50" border="0" alt="Search"> </td></tr></table> <!--End Header--> <table border=0 cellspacing=0 cellpadding=0 width=" 850"> <tr><td align="center" width="100%"> <table

ORACLE

Using the UTL_HTTP Package

The SELECT statement and the output in the slide show how to use the REQUEST function of the UTL_HTTP package to retrieve contents from the URL www.oracle.com. The second parameter to the function indicates the proxy because the client being tested is behind a firewall.

The retrieved output is in HTML format.

You can use the function in a PL/SQL block as shown below. The function retrieves up to 100 pieces of data, each of a maximum 2000 bytes from the URL. The number of pieces and the total length of the data retrieved are printed.

```
DECLARE
 x UTL_HTTP.HTML_PIECES;
BEGIN
 x := UTL_HTTP.REQUEST_PIECES('http://www.oracle.com/',100,
                        'edu-proxy.us.oracle.com');
 DBMS_OUTPUT.PUT_LINE(x.COUNT || ' pieces were retrieved.');
 DBMS_OUTPUT.PUT_LINE('with total length ');
 IF x.COUNT < 1 THEN DBMS_OUTPUT.PUT_LINE('0');
 ELSE DBMS_OUTPUT.PUT_LINE((2000*(x.COUNT - 1))+LENGTH(x(x.COUNT)));
 END IF;
END;
/
```

```
12 pieces were retrieved.
with total length
23553
PL/SQL procedure successfully completed.
```

Using the UTL_TCP Package

The UTL_TCP Package:

- Enables PL/SQL applications to communicate with external TCP/IP-based servers using TCP/IP
- Contains functions to open and close connections, to read or write binary or text data to or from a service on an open connection
- Requires remote host and port as well as local host and port as arguments to its functions
- Raises exceptions if the buffer size is too small, when no more data is available to read from a connection, when a generic network error occurs, or when bad arguments are passed to a function call

ORACLE

Using the UTL_TCP Package

The UTL_TCP package enables PL/SQL applications to communicate with external TCP/IP-based servers using TCP/IP. Because many Internet application protocols are based on TCP/IP, this package is useful to PL/SQL applications that use Internet protocols.

The package contains functions such as:

OPEN_CONNECTION: This function opens a TCP/IP connection with the specified remote and local host and port details. The remote host is the host providing the service. The remote port is the port number on which the service is listening for connections. The local host and port numbers represent those of the host providing the service. The function returns a connection of PL/SQL record type.

CLOSE_CONNECTION: This procedure closes an open TCP/IP connection. It takes the connection details of a previously opened connection as parameter. The procedure CLOSE_ALL_CONNECTIONS closes all open connections.

READ_BINARY()/TEXT()/LINE(): This function receives binary, text, or text line data from a service on an open connection.

WRITE_BINARY()/TEXT()/LINE(): This function transmits binary, text, or text line message to a service on an open connection.

Exceptions are raised when buffer size for the input is too small, when generic network error occurs, when no more data is available to read from the connection, or when bad arguments are passed in a function call.

This package is discussed in detail in the course *Administering Oracle9i Application Server*. For more information, refer to *Oracle 9i Supplied PL/SQL Packages Reference*.

Oracle-Supplied Packages

Other Oracle-supplied packages include:

- **DBMS_ALERT**
- **DBMS_APPLICATION_INFO**
- **DBMS_DESCRIBE**
- **DBMS_LOCK**
- **DBMS_SESSION**
- **DBMS_SHARED_POOL**
- **DBMS_TRANSACTION**
- **DBMS_UTILITY**

ORACLE

Using Oracle-Supplied Packages

Package	Description
DBMS_ALERT	Provides notification of database events
DBMS_APPLICATION_INFO	Allows application tools and application developers to inform the database of the high level of actions they are currently performing
DBMS_DESCRIBE	Returns a description of the arguments for a stored procedure
DBMS_LOCK	Requests, converts, and releases userlocks, which are managed by the RDBMS lock management services
DBMS_SESSION	Provides access to SQL session information
DBMS_SHARED_POOL	Keeps objects in shared memory
DBMS_TRANSACTION	Controls logical transactions and improves the performance of short, nondistributed transactions
DBMS_UTILITY	Analyzes objects in a particular schema, checks whether the server is running in parallel mode, and returns the time

Oracle-Supplied Packages

The following list summarizes and provides a brief description of the packages supplied with Oracle9*i*.

Built-in Name	Description
CALENDAR	Provides calendar maintenance functions
DBMS_ALERT	Supports asynchronous notification of database events. Messages or alerts are sent on a COMMIT command. Message transmittal is one way, but one sender can alert several receivers.
DBMS_APPLICATION_INFO	Is used to register an application name with the database for auditing or performance tracking purposes
DBMS_AQ	Provides message queuing as part of the Oracle server; is used to add a message (of a predefined object type) onto a queue or dequeue a message
DBMS_AQADM	Is used to perform administrative functions on a queue or queue table for messages of a predefined object type
DBMS_DDL	Is used to embed the equivalent of the SQL commands ALTER, COMPILE, and ANALYZE within your PL/SQL programs
DBMS_DEBUG	A PL/SQL API to the PL/SQL debugger layer, Probe, in the Oracle server
DBSM_DEFER DBMS_DEFER_QUERY DBMS_DEFER_SYS	Is used to build and administer deferred remote procedure calls (use of this feature requires the Replication Option)
DBMS_DESCRIBE	Is used to describe the arguments of a stored procedure
DBMS_DISTRIBRUTED_ TRUST_ADMIN	Is used to maintain the Trusted Servers list, which is used in conjunction with the list at the central authority to determine whether a privileged database link from a particular server can be accepted
DBMS_HS	Is used to administer heterogeneous services by registering or dropping distributed external procedures, remote libraries, and non-Oracle systems (you use dbms_hs to create or drop some initialization variables for non-Oracle systems)
DBMS_HS_EXTPROC	Enables heterogeneous services to establish security for distributed external procedures
DBMS_HS_PASSTHROUGH	Enables heterogeneous services to send pass-through SQL statements to non-Oracle systems
DBMS_IOT	Is used to schedule administrative procedures that you want performed at periodic intervals; is also the interface for the job queue
DBMS_JOB	Is used to schedule administrative procedures that you want performed at periodic intervals
DBMS_LOB	Provides general purpose routines for operations on Oracle large objects (LOBs) data types: BLOB, CLOB (read only) and BFILES (read-only)

Oracle Supplied Packages (continued)

Built-in Name	Description
DBMS_LOCK	Is used to request, convert, and release locks through Oracle Lock Management services
DBMS_LOGMNR	Provides functions to initialize and run the log reader
DBMS_LOGMNR_D	Queries the dictionary tables of the current database, and creates a text based file containing their contents
DBMS_OFFLINE_OG	Provides public APIs for offline instantiation of master groups
DBMS_OFFLINE_SNAPSHOT	Provides public APIs for offline instantiation of snapshots
DBMS_OLAP	Provides procedures for summaries, dimensions, and query rewrites
DBMS_ORACLE_TRACE_AGENT	Provides client callable interfaces to the Oracle TRACE instrumentation within the Oracle7 server
DBMS_ORACLE_TRACE_USER	Provides public access to the Oracle7 release server Oracle TRACE instrumentation for the calling user
DBMS_OUTPUT	Accumulates information in a buffer so that it can be retrieved out later
DBMS_PCLXUTIL	Provides intrapartition parallelism for creating partition-wise local indexes
DBMS_PIPE	Provides a DBMS pipe service that enables messages to be sent between sessions
DBMS_PROFILER	Provides a Probe Profiler API to profile existing PL/SQL applications and identify performance bottlenecks
DBMS_RANDOM	Provides a built-in random number generator
DBMS_RECTIFIER_DIFF	Provides APIs used to detect and resolve data inconsistencies between two replicated sites
DBMS_REFRESH	Is used to create groups of snapshots that can be refreshed together to a transactionally consistent point in time; requires the Distributed option
DBMS_REPAIR	Provides data corruption repair procedures
DBMS_REPCAT	Provides routines to administer and update the replication catalog and environment; requires the Replication option
DBMS_REPCAT_ADMIN	Is used to create users with the privileges needed by the symmetric replication facility; requires the Replication option
DBMS_REPCAT_INSTATIATE	Instantiates deployment templates; requires the Replication option
DBMS_REPCAT_RGT	Controls the maintenance and definition of refresh group templates; requires the Replication option
DBMS_REPUTIL	Provides routines to generate shadow tables, triggers, and packages for table replication
DBMS_RESOURCE_MANAGER	Maintains plans, consumer groups, and plan directives; it also provides semantics so that you may group together changes to the plan schema

Oracle Supplied Packages (continued)

Built-in Name	Description
DBMS_RESOURCE_ MANAGER_PRIVS	Maintains privileges associated with resource consumer groups
DBMS_RLS	Provides row-level security administrative interface
DBMS_ROWID	Is used to get information about ROWIDs, including the data block number, the object number, and other components
DBMS_SESSION	Enables programmatic use of the SQL ALTER SESSION statement as well as other session-level commands
DBMS_SHARED_POOL	Is used to keep objects in shared memory, so that they are not aged out with the normal LRU mechanism
DBMS_SNAPSHOT	Is used to refresh one or more snapshots that are not part of the same refresh group and purge logs; use of this feature requires the Distributed option
DBMS_SPACE	Provides segment space information not available through standard views
DBMS_SPACE_ADMIN	Provides tablespace and segment space administration not available through standard SQL
DSMS_SQL	Is used to write stored procedure and anonymous PL/SQL blocks using dynamic SQL; also used to parse any DML or DDL statement
DBMS_STANDARD	Provides language facilities that help your application interact with the Oracle server
DBMS_STATS	Provides a mechanism for users to view and modify optimizer statistics gathered for database objects
DBMS_TRACE	Provides routines to start and stop PL/SQL tracing
DBMS_TRANSACTION	Provides procedures for a programmatic interface to transaction management
DBMS_TTS	Checks whether if the transportable set is self-contained
DBMS_UTILITY	Provides functionality for managing procedures, reporting errors, and other information
DEBUG_EXTPROC	Is used to debug external procedures on platforms with debuggers that can attach to a running process
OUTLN_PKG	Provides the interface for procedures and functions associated with management of stored outlines
PLITBLM	Handles index-table operations
SDO_ADMIN	Provides functions implementing spatial index creation and maintenance for spatial objects
SDO_GEOM	Provides functions implementing geometric operations on spatial objects
SDO_MIGRATE	Provides functions for migrating spatial data from release 7.3.3 and 7.3.4 to 8.1.*x*
SDO_TUNE	Provides functions for selecting parameters that determine the behavior of the spatial indexing scheme used in the Spatial Cartridge

Oracle Supplied Packages (continued)

Built-in Name	Description
STANDARD	Declares types, exceptions, and subprograms that are available automatically to every PL/SQL program
TIMESERIES	Provides functions that perform operations, such as extraction, retrieval, arithmetic, and aggregation, on time series data
TIMESCALE	Provides scale-up and scale-down functions
TSTOOLS	Provides administrative tools procedures
UTL_COLL	Enables PL/SQL programs to use collection locators to query and update
UTL_FILE	Enables your PL/SQL programs to read and write operating system (OS) text files and provides a restricted version of standard OS stream file I/O
UTL_HTTP	Enables HTTP callouts from PL/SQL and SQL to access data on the Internet or to call Oracle Web Server Cartridges
UTL_PG	Provides functions for converting COBOL numeric data into Oracle numbers and Oracle numbers into COBOL numeric data
UTL_RAW	Provides SQL functions for RAW data types that concatenate, obtain substring, and so on, to and from RAW data types
UTL_REF	Enables a PL/SQL program to access an object by providing a reference to the object
VIR_PKG	Provides analytical and conversion functions for visual information retrieval

Summary

In this lesson, you should have learned how to:

- **Take advantage of the preconfigured packages that are provided by Oracle**
- **Create packages by using the `catproc.sql` script**
- **Create packages individually.**

ORACLE

DBMS Packages and the Scripts to Execute Them

DBMS_ALERT	dbmsalrt.sql
DBMS_APPLICATION_INFO	dbmsutil.sql
DBMS_DDL	dbmsutil.sql
DBMS_LOCK	dbmslock.sql
DBMS_OUTPUT	dbmsotpt.sql
DBMS_PIPE	dbmspipe.sql
DBMS_SESSION	dbmsutil.sql
DBMS_SHARED_POOL	dbmsspool.sql
DBMS_SQL	dbmssql.sql
DBMS_TRANSACTION	dbmsutil.sql
DBMS_UTILITY	dbmsutil.sql

Note: For more information about these packages and scripts, refer to *Oracle9i Supplied PL/SQL Packages and Types Reference*.

Instructor Note

Point out to the students that these script files often have useful comments embedded within them that supplement the documentation. See the `dbmspipe.sql` script for an example.

Practice 14 Overview

This practice covers using:

- **DBMS_SQL for dynamic SQL**
- **DBMS_DDL to analyze a table**
- **DBMS_JOB to schedule a task**
- **UTL_FILE to generate text reports**

ORACLE

Practice 14 Overview

In this practice, you use DBMS_SQL to implement a procedure to drop a table. You also use the EXECUTE IMMEDIATE command to drop a table. You use DBMS_DDL to analyze objects in your schema, and you can schedule the analyze procedure through DBMS_JOB.

In this practice, you also write a PL/SQL program that generates customer statuses into a text file.

Practice 14

1. a. Create a DROP_TABLE procedure that drops the table specified in the input parameter. Use the procedures and functions from the supplied DBMS_SQL package.

   b. To test the DROP_TABLE procedure, first create a new table called EMP_DUP as a copy of the EMPLOYEES table.

   c. Execute the DROP_TABLE procedure to drop the EMP_DUP table.

2. a. Create another procedure called DROP_TABLE2 that drops the table specified in the input parameter. Use the EXECUTE IMMEDIATE statement.

   b. Repeat the test outlined in steps 1-b and 1-c.

3. a. Create a procedure called ANALYZE_OBJECT that analyzes the given object that you specified in the input parameters. Use the DBMS_DDL package, and use the COMPUTE method.

   b. Test the procedure using the EMPLOYEES table. Confirm that the ANALYZE_OBJECT procedure has run by querying the LAST_ANALYZED column in the USER_TABLES data dictionary view.

LAST_ANAL
27-SEP-01

If you have time:

4. a. Schedule ANALYZE_OBJECT by using DBMS_JOB. Analyze the DEPARTMENTS table, and schedule the job to run in five minutes time from now. (To start the job in five minutes from now, set the parameter NEXT_DATE = 5/(24*60) = 1/288.)

   b. Confirm that the job has been scheduled by using USER_JOBS.

5. Create a procedure called CROSS_AVGSAL that generates a text file report of employees who have exceeded the average salary of their department. The partial code is provided for you in the file lab14_5.sql.

   a. Your program should accept two parameters. The first parameter identifies the output directory. The second parameter identifies the text file name to which your procedure writes.

   b. Your instructor will inform you of the directory location. When you invoke the program, name the second parameter sal_rpt*xx*.txt where *xx* stands for your user number, such as 01, 15, and so on.

   c. Add an exception handling section to handle errors that may be encountered from using the UTL_FILE package.

   Sample output from this file follows:

```
EMPLOYEES OVER THE AVERAGE SALARY OF THEIR DEPARTMENT:
REPORT GENERATED ON  26-FEB-01

Hartstein                      20      $13,000.00
Raphaely                       30      $11,000.00
Marvis                         40      $6,500.00
...
*** END OF REPORT ***
```

Instructor Note

You can locate the directory name `UTL_FILE`, where the students write their files as follows:

Use Telnet to connect to the server provided by the ESS group. Log in to the server operating system using the username and password supplied by ESS.

Type the command `ls` to list the contents and observe that the directory `UTL_FILE` is listed.

Type the command `pwd` to display the present working directory. Give this value to the students to use as a parameter in their practice.