PART I

Advanced Concepts, Internals, and Debugging Concepts
CHAPTER 1

Introduction to Advanced Concepts, Patterns, and Techniques
Having, involving, or displaying special skill or knowledge derived from training or experience.

—Miriam-Webster Online Dictionary

F ive years of programming. This is the time it takes for most programmers to feel as if they’ve come of age as a professional. Traditional programming concepts are second nature, like driving a car or typing on a keyboard. Employers recognize this, often advertising jobs requiring five or more years’ experience. Something happens around this time though. Passion for the craft either accelerates, driving its possessor to new heights, or it withers, causing year five to be the high-water mark of many careers.

Those that push through this five-year breaking point often come through with a renewed desire to learn. If you’re reading this book, chances are you have a desire to push through and succeed, and are searching for knowledge that will help you do your job better, and make you more efficient.

Expert PL/SQL is for those who wish to accelerate their skills, and go beyond traditional PL/SQL programming. You may have just switched from another programming language, and want to delve more deeply into PL/SQL than other books do, or you may want to expand on the PL/SQL knowledge you already have. In this book, each chapter deals directly with tough questions, and introduces new concepts that you can apply to your current development projects.

This chapter foreshadows the rest of the book by covering the following topics:

- What it means to be an expert
- Who should read this book
- How examples are structured
- Oracle 10g Release 2 new features for PL/SQL

What Is an Expert?
An expert programmer is someone others turn to for answers. They use their abilities to do more than build to spec. They influence every aspect of the development process, helping others who are on their way to becoming successful themselves. Being recognized as an expert is part skill and part attitude—and attitude is contagious.

Do you want to be an expert? In addition to reading this book, here are a few suggestions to increase your knowledge:
Help others in your company. Offer short classes and target specific training needs you have observed.

Monitor user forums and answer questions. Oracle’s forum on OTN (http://otn.oracle.com/forums) is a good place to start. Metalink forums and those on www.OraFAQ.com are well attended, too. This is a great place to gain experience from other people’s issues, and at the same time help them resolve problems they’re having.

Take note of problems that impact your organization, and find the solution. One difficulty every company seems to have is information overload. Terabytes of data—and no way to use most of it—is a problem you can help resolve. Warehouses of paper documents that could be digitally archived are a problem you can help resolve. (See Chapter 9.) Use these opportunities to improve your skills, while simultaneously improving the business.

Did you notice a common theme about these suggestions, aside from building your own success? They all help others succeed, too.

10g Release 2 New Features

One way to move toward expert status is to stay up-to-date on new features. It’s tough to take advantage of enhancements if you do not know what they are! This section demonstrates many of the PL/SQL new features in Oracle 10g Release 2. Additions to existing functionality are also shown in various sections throughout the book so they won’t be discussed in this chapter.

In this section, we cover the following new features:

- WRAP dynamically generated PL/SQL
- Conditional Compilation
- Asynchronous Commit
- Predictive Analysis
- Using UTL_MATCH to diff code
- Modifications to DBMS_OUTPUT

These new features are discussed in detail in the sections that follow.
Obfuscation of PL/SQL Source Code

Pages 98–100 in the authors’ Oracle Database 10g PL/SQL Programming book covered the PL/SQL Wrapper utility, showing how to hide source code by converting it to hexadecimal digits. The utility, found in $ORACLE_HOME/bin, is called WRAP. The drawback to using a command-line utility is that dynamically generated PL/SQL cannot be hidden.

Oracle 10g Release 2 adds an overloaded function and procedure to the DBMS_DDL supplied package. They are listed in Table 1-1.

To illustrate their use, we’ve created the following table to hold airport codes, regions, and countries:

```sql
CREATE TABLE airport_list (
    airport_id NUMBER(10) PRIMARY KEY,
    airport_code VARCHAR2(10 CHAR) NOT NULL,
    region VARCHAR2(30 CHAR) NOT NULL,
    country VARCHAR2(30 CHAR) NOT NULL);
```

The following is a redacted list of data:

```sql
-- Available online as part of wrap.sql
INSERT INTO airport_list (airport_id, airport_code, region, country)
VALUES (1, 'AKL', 'Auckland', 'NEW ZEALAND');
INSERT INTO airport_list (airport_id, airport_code, region, country)
VALUES (2, 'BHE', 'Blenheim', 'NEW ZEALAND');
INSERT INTO airport_list (airport_id, airport_code, region, country)
VALUES (3, 'CHC', 'Christchurch', 'NEW ZEALAND');
INSERT INTO airport_list (airport_id, airport_code, region, country)
VALUES (4, 'CHT', 'Chatham Islands', 'NEW ZEALAND');
INSERT INTO airport_list (airport_id, airport_code, region, country)
VALUES (5, 'DUD', 'Dunedin', 'NEW ZEALAND');
...
```

### Table 1-1. WRAP

<table>
<thead>
<tr>
<th>Procedure/Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRAP</td>
<td>Overloaded function that returns the wrapped PL/SQL source code when provided with the original source.</td>
</tr>
<tr>
<td>CREATE_WRAPPED</td>
<td>Procedure that wraps the source code provided as input. It’s faster than using WRAP.</td>
</tr>
</tbody>
</table>
WRAP
Use the overloaded \texttt{WRAP} function with \texttt{EXECUTE IMMEDIATE} to create the wrapped code, as the following example illustrates:

```sql
-- Available online as part of wrap.sql
DECLARE
    v_procedure VARCHAR2(32767);
BEGIN
    v_procedure := 'CREATE OR REPLACE PROCEDURE wrap_test ' || 'IS ' || '    v_airport_codes AIRPORT_LIST.AIRPORT_CODE%TYPE; ' || '    CURSOR airport_cur IS ' || '    SELECT airport_code ' || '    FROM airport_list ' || '    ORDER BY airport_code;' || '    BEGIN ' || '    FOR y IN airport_cur LOOP ' || '    DBMS_OUTPUT.PUT_LINE(''Airport Code: ''||y.airport_code);' || '    END LOOP;' || '    END;'
EXECUTE IMMEDIATE DBMS_DDL.WRAP(v_procedure);
END;
/
```

To see the wrapped procedure, select the text from the \texttt{USER\_SOURCE} view.

```sql
-- Available online as part of wrap.sql
SELECT text
FROM user_source
WHERE name = 'WRAP\_TEST';

This shows the wrapped source as displayed next:

```
PROCEDURE wrap_test wrapped
a000000
369
abcd
abcd
abcd
abcd
abcd
```

CREATE_WRAPPED

DBMS_DDL.CREATE_WRAPPED works in a similar way. The following example shows how it differs from the WRAP function.

```sql
-- Available online as part of wrap.sql
DECLARE
  v_procedure VARCHAR2(32767);
BEGIN
  v_procedure := 'CREATE OR REPLACE PROCEDURE create_wrapped_test ' ||
                  'IS ' ||
                  '  v_airport_codes AIRPORT_LIST.AIRPORT_CODE%TYPE; ' ||
                  '  ' ||
                  '  CURSOR airport_cur IS ' ||
                  '    SELECT airport_code' ||
                  '    FROM airport_list' ||
                  '    ORDER BY airport_code;' ||
                  '  ' ||
                  '  BEGIN ' ||
                  '    FOR y IN airport_cur LOOP ' ||
                  '      DBMS_OUTPUT.PUT_LINE(''Airport Code: ''||y.airport_code);' ||
                  '    END LOOP;' ||
                  '  END;' ||
  SYS.DBMS_DDL.CREATE_WRAPPED(v_procedure);
END;
/
Chapter 1: Introduction to Advanced Concepts, Patterns, and Techniques

The use of `EXECUTE IMMEDIATE` is not required. `USER_SOURCE` shows the wrapped source code once again.

```sql
--- Available online as part of wrap.sql
EXEC create_wrapped_test
```

This returns the following result:

```
Airport Code: AKL
Airport Code: BHE
Airport Code: CHC
Airport Code: CHT
Airport Code: DUD
Airport Code: GIS
...
```

Wrapping PL/SQL source is not so different using this method than it is using the `WRAP` binary. It is more flexible, however.

**Conditional Compilation**

Conditional compilation allows you to write code containing version-specific features so that the post-processed code includes only the sections relevant to the database version the PL/SQL is written for. It uses `DBMS_DB_VERSION` and the following preprocessor control tokens:

- `$ERROR`
- `$IF`
- `$THEN`
- `$ELSIF`
- `$ELSE`
- `$END`
This feature is best explained through an example. The `COMPILE_BY_VERSION` procedure shown next uses predefined control tokens to tell Oracle the portion of code to evaluate prior to compilation:

```
-- Available online as part of compile_by_version.sql
CREATE OR REPLACE PROCEDURE compile_by_version
IS
BEGIN
  $IF DBMS_DB_VERSION.VER_LE_10_2
  THEN
    DBMS_OUTPUT.PUT_LINE('10.2 and under');
  $ELSIF DBMS_DB_VERSION.VER_LE_10_1
  THEN
    DBMS_OUTPUT.PUT_LINE('10.1 and under');
  $ELSE
    DBMS_OUTPUT.PUT_LINE('Not 10g');
  $END
END;
/
```

Take note that the `$IF` does not have an `$END IF`. Rather, it uses `$END`. This is obviously a departure from traditional `IF-THEN-END IF` syntax. The reason? `END IF` has a space in it; something that a control token cannot have. Use `$END` instead.

Now that the procedure is created, let's take a look at `USER_SOURCE`. The following `SELECT` gets the procedure text:

```
-- Available online as part of compile_by_version.sql
SET PAGES 9999
SELECT TEXT
FROM USER_SOURCE
WHERE NAME = 'COMPILE_BY_VERSION';
```
Chapter 1: Introduction to Advanced Concepts, Patterns, and Techniques

This returns the following:

```sql
PROCEDURE compile_by_version
IS
BEGIN
$IF DBMS_DB_VERSION.VER_LE_10_2
$THEN
   DBMS_OUTPUT.PUT_LINE('10.2 and under');
$ELSIF DBMS_DB_VERSION.VER_LE_10_1
$THEN
   DBMS_OUTPUT.PUT_LINE('10.1 and under');
$ELSE
   DBMS_OUTPUT.PUT_LINE('Not 10g');
$END
END;

This is no different than the original! Instead of looking to USER|ALL|DBA_SOURCE, use the DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE procedure to see the impact.

-- Available online as part of compile_by_version.sql
SET SERVEROUTPUT ON
BEGIN
   DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE ('PROCEDURE', 'PLSQL', 'COMPILE_BY_VERSION');
END;
/

This returns the following in our instance:

```sql
PROCEDURE compile_by_version
IS
BEGIN
   DBMS_OUTPUT.PUT_LINE('10.2 and under');
END;
```

NOTE
The output may very well differ in your environment if you're not running Oracle 10g Release 2.

The $ is actually the preprocessor trigger character that tells Oracle what to process before completing compilation. ERROR, IF, THEN, ELSIF, ELSE, and END are already defined as control tokens.
Asynchronous Commit

When a transaction is committed, Oracle writes the redo entry from memory (the Redo Buffer) to the Redo Log Files. Control is not returned to the client until this completes. Asynchronous commit allows you to return to the client without waiting for the redo to be written to disk.

Commit can use the options shown in Table 1-2.

To change COMMIT using these options, set either the COMMIT_WRITE parameter or specify the options during COMMIT. The following example sets the COMMIT_WRITE parameter:

```
SQL> conn / as sysdba
SQL> ALTER SYSTEM SET COMMIT_WRITE = NOWAIT;
```

Alternatively, specify the option(s) when issuing the COMMIT.

```
SQL> COMMIT NOWAIT;
```

The default is WAIT and IMMEDIATE, so modifying the COMMIT option as just shown results in the transaction being written to disk immediately, instead of waiting for confirmation prior to returning to the client.

Data Mining—Using Predictive Analytics

Data mining is not new. Predictive analysis using the DBMS_PREDICTIVE_ANALYTICS package is. The package is very straightforward, with only two procedures to examine current trends in existing data, and to predict future results based on patterns identified during package execution.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATCH</td>
<td>Write the transaction from the Redo Buffer to disk when capable, rather than on commit.</td>
</tr>
<tr>
<td>IMMEDIATE</td>
<td>Write the transaction from the Redo Buffer on commit.</td>
</tr>
<tr>
<td>NOWAIT</td>
<td>Do not wait for the redo to be written. Return to the client right away.</td>
</tr>
<tr>
<td>WAIT</td>
<td>Do not return to the client until the redo entry is written to disk.</td>
</tr>
</tbody>
</table>

**TABLE 1-2. Commit Options**
Chapter 1: Introduction to Advanced Concepts, Patterns, and Techniques 13

Examining Current Data

The EXPLAIN procedure’s structure is shown in Table 1-3.

To execute, pass the source table name to DATA_TABLE_NAME, and the column to explain to the EXPLAIN_COLUMN_NAME parameter. The EXPLAIN procedure looks at all columns in the source table, and analyzes their contents to determine whether a pattern of influence exists over the column being explained. When analysis is complete, the results for each column are printed in the result table. This table should not exist prior to the execution of DBMS_PREDICTIVE_ANALYTICS.

NOTE

To run these examples, you must have the Data Mining option installed.

The example used in this section starts with a table to hold student assessment data. Its structure is shown next:

```sql
CREATE TABLE assessment (
    student_id NUMBER(10) PRIMARY KEY,
    performance NUMBER(10) NOT NULL,
    gender VARCHAR2(10) NOT NULL,
    ethnicity NUMBER(10) NOT NULL,
    age NUMBER(10) NOT NULL);
```

### Table 1-3. Explain Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_TABLE_NAME</td>
<td>The name of the table that contains EXPLAIN_COLUMN_NAME.</td>
</tr>
<tr>
<td>EXPLAIN_COLUMN_NAME</td>
<td>The name of the column to be analyzed.</td>
</tr>
<tr>
<td>RESULT_TABLE_NAME</td>
<td>Table name to store the results of the procedure. The table name must be unique.</td>
</tr>
<tr>
<td>DATA_SCHEMA_NAME</td>
<td>The default is the current schema. If the table and column being analyzed are in another schema, provide the schema name here.</td>
</tr>
</tbody>
</table>
The sample data (not real!) includes test performance levels with typical disaggregations of gender, age, and ethnicity for third-graders. Ethnicity has five different ethnicities marked 1–5. The age range is eight through ten.

-- Available online as part of predictive_analytics.sql
INSERT INTO assessment
VALUES (NULL, 3, 'M', 5, 10);
INSERT INTO assessment
VALUES (NULL, 3, 'F', 5, 9);
INSERT INTO assessment
VALUES (NULL, 2, 'F', 5, 9);
INSERT INTO assessment
VALUES (NULL, 4, 'M', 5, 9);
INSERT INTO assessment
VALUES (NULL, 4, 'M', 5, 9);
INSERT INTO assessment
VALUES (NULL, 3, 'F', 5, 8);
INSERT INTO assessment
VALUES (NULL, 3, 'M', 5, 9);
...
COMMIT;

A total of 1024 records are inserted into the assessment table when you run the predictive_analytics.sql script. As you would expect, the more data you have to analyze, the more precise the correlations will be. Outliers will have less impact when more records are present. For example, if only 64 records are in the assessment table, significant patterns will be difficult to determine, but improve the underlying data set, and relevant patterns begin to develop.

Execute the following to explain the ASSESSMENT.PERFORMANCE column:

-- Available online as part of predictive_analytics.sql
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.EXPLAIN (     
        'ASSESSMENT', 
        'PERFORMANCE', 
        'ASSESSMENT_ANALYSIS');
END;
/

DBMS_PREDICTIVE_ANALYTICS.EXPLAIN creates a table to store the output. We’ve named the table ASSESSMENT_ANALYSIS, but any valid table name will work. The tables, regardless of the name you provide, have the same structure. The table structure is shown next:

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE_NAME</td>
<td></td>
<td>VARCHAR2(40)</td>
</tr>
<tr>
<td>EXPLANATORY_VALUE</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>RANK</td>
<td></td>
<td>NUMBER</td>
</tr>
</tbody>
</table>
The `ATTRIBUTE_NAME` column includes one record for each column in the analyzed table, less the column that they are being compared to. `EXPLANATORY_VALUE` is a number from 0 to 1 that ranks the ability of that column to explain the value of the target column. A value of 0 indicates that there is no correlation found, while a value of 1 indicates that the correlation is perfect. The higher the value is, the more the column can be relied upon to explain the value in the target column.

To see the results, select all three columns from the `ASSESSMENT_ANALYSIS` table.

```sql
SET PAGES 9999
SELECT *
FROM ASSESSMENT_ANALYSIS;
```

This results in the following:

<table>
<thead>
<tr>
<th>ATTRIBUTE_NAME</th>
<th>EXPLANATORY_VALUE</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENDER</td>
<td>0.071302739</td>
<td>1</td>
</tr>
<tr>
<td>AGE</td>
<td>0.059764092</td>
<td>2</td>
</tr>
<tr>
<td>ETHNICITY</td>
<td>0.037652922</td>
<td>3</td>
</tr>
<tr>
<td>STUDENT_ID</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

So, with this sample set of data, gender is best able to explain performance, though it is by no means a perfect indicator. It is quite low in fact, turning in a dismal .07 explanatory value.

**NOTE**

*If you rerun this without dropping the ASSESSMENT_ANALYSIS table, it will fail with the following:*

```sql
BEGIN

ERROR at line 1:
ORA-00955: name is already used by an existing object
ORA-06512: at "DMSYS.DBMS_PREDICTIVE_ANALYTICS", line 1100
ORA-06512: at line 2

Drop the table prior to rerunning to avoid the error.
```

**Predicting Future Results**

It’s pretty easy to find correlations between different types of data to help understand the results, but it’s also possible to predict values with the same package. The `DBMS_PREDICTIVE_ANALYTICS.PREDICT` procedure has the parameters shown in Table 1-4.
The same table and source data is used for this example, but a few additional records are added where the students have yet to take the assessment. Notice that the second column is NULL.

```sql
-- Available online as part of predictive_analytics.sql
INSERT INTO assessment
VALUES (NULL, NULL, 'M', 5, 9);
INSERT INTO assessment
VALUES (NULL, NULL, 'F', 5, 9);
INSERT INTO assessment
VALUES (NULL, NULL, 'F', 5, 9);
INSERT INTO assessment
VALUES (NULL, NULL, 'M', 2, 9);
INSERT INTO assessment
VALUES (NULL, NULL, 'M', 5, 9);
INSERT INTO assessment
VALUES (NULL, NULL, 'M', 5, 9);
INSERT INTO assessment
VALUES (NULL, NULL, 'M', 5, 8);
INSERT INTO assessment
VALUES (NULL, NULL, 'F', 5, 9);
INSERT INTO assessment
VALUES (NULL, NULL, 'M', 2, 9);
INSERT INTO assessment
VALUES (NULL, NULL, 'M', 4, 10);
INSERT INTO assessment
VALUES (NULL, NULL, 'F', 5, 9);
INSERT INTO assessment
VALUES (NULL, NULL, 'M', 2, 9);
COMMIT;
```

ACCURACY

Accuracy is an OUT parameter that returns the accuracy with which the prediction can be made.

DATA_TABLE_NAME
The name of the table being analyzed.

CASE_ID_COLUMN_NAME
This is generally the primary key, but can be any unique column.

RESULT_TABLE_NAME
The name of the table to create that will hold the results.

DATA_SCHEMA_NAME
The default is the current schema. To override the default, specify the name of the schema.

### Table 1-4. Predict Procedure Parameters

The same table and source data is used for this example, but a few additional records are added where the students have yet to take the assessment. Notice that the second column is NULL.
Use `DBMS_PREDICTIVE_ANALYTICS.PREDICT` to predict the assessment values for these 11 students.

```sql
-- Available online as part of predictive_analytics.sql
SET SERVEROUTPUT ON
DECLARE
  v_predict_accuracy NUMBER(10);
BEGIN
  DBMS_PREDICTIVE_ANALYTICS.PREDICT (
    v_predict_accuracy,
    'ASSESSMENT',
    'STUDENT_ID',
    'PERFORMANCE',
    'ASSESSMENT_PREDICTION');
  DBMS_OUTPUT.PUT_LINE('*** Accuracy ***');
  DBMS_OUTPUT.PUT_LINE(v_predict_accuracy);
END;
/
```

This adds the prediction to the `ASSESSMENT_PREDICTION` table. Selecting from this table for the rows added earlier shows us where, statistically, we can expect the performance of each person to end up.

```sql
-- Available online as part of predictive_analytics.sql
SELECT * FROM ASSESSMENT_PREDICTION WHERE STUDENT_ID > 1024;
```

This returns the following data set:

```
STUDENT_ID    PREDICTION   PROBABILITY
-------------  ----------   ---------------
1025         3           64883399
1026         3           890236437
1027         3           890236437
1028         3           728349805
1029         3           64883399
1030         3           64883399
1031         3           689166427
1032         3           890236437
1033         3           728349805
1034         2           999986529
1035         3           890236437
```

These students did not have any performance measure (the column is still `NULL`), but with a data set that is large enough, it’s possible to predict performance based on patterns. In this case, student 1034 is likely to get a performance level of 2,
whereas the other ten are likely to have a performance level of 3. We set no values to weight certain columns or values over others. This is determined by Oracle using DBMS_PREDICTIVE_ANALYTICS.

**TIP**
As with any statistic, look beyond the numbers.
Though it’s possible, with enough data, to define certain macro trends with people (in this case, third-grade students), be careful not to apply those macro trends to the individual level. It’s virtually impossible to determine, in advance, how a person will respond to events. To place artificial limitations on individuals based on predictive analysis allows this imprecise science to impact the outcome. It can limit potential, turning it into a self-fulfilling prophesy.

### String Comparisons in PL/SQL

Oracle added the UTL_MATCH package in Version 10g Release 2 to compare strings. The four functions included in the package use different methods to compare a source string and destination string, and return an assessment of what it would take to turn the source into the destination string.

The functions are broken down into two categories. The categories are actually the algorithms employed to analyze the strings.

#### Levenshtein Distance

The Levenshtein Distance (LD) algorithm, commonly called the Edit Distance (ED) algorithm, is the older of the two supported methods. It measures the distance between the source and destination strings. By distance, we’re referring to the number of changes required to turn the source string into the destination string.

For example, the following two strings differ by one character:

```sql
'expresso', 'espresso'
```

Using the LD/ED algorithm, the distance is 1 since one character must change to make the first string match the second. The following function works for all supported releases. Thanks to Barbara Boehmer (of OTN forums fame) for graciously allowing us to use it in this chapter.

```sql
-- Available online as part of edit_distance.sql
CREATE OR REPLACE FUNCTION edit_distance
  (i_source_string IN VARCHAR2,
   i_target_string IN VARCHAR2)
```

```sql
```
RETURN NUMBER
DETERMINISTIC
AS
v_length_of_source NUMBER := NVL (LENGTH (i_source_string), 0);
v_length_of_target NUMBER := NVL (LENGTH (i_target_string), 0);

TYPE mytabtype IS TABLE OF NUMBER
   INDEX BY BINARY_INTEGER;

v_column_to_left mytabtype;
v_current_column mytabtype;
v_cost NUMBER := 0;
BEGIN
IF v_length_of_source = 0 THEN
   RETURN v_length_of_target;
ELSIF v_length_of_target = 0 THEN
   RETURN v_length_of_source;
ELSE
   FOR j IN 0 .. v_length_of_target LOOP
      v_column_to_left(j) := j;
   END LOOP;
   FOR i IN 1.. v_length_of_source LOOP
      v_current_column(0) := i;
      FOR j IN 1 .. v_length_of_target LOOP
         IF SUBSTR (i_source_string, i, 1) =
            SUBSTR (i_target_string, j, 1)
            THEN v_cost := 0;
            ELSE v_cost := 1;
            END IF;
            v_current_column(j) := LEAST (v_current_column(j - 1) + 1,
                v_column_to_left(j) + 1,
                v_column_to_left(j - 1) + v_cost);
      END LOOP;
      FOR j IN 0 .. v_length_of_target LOOP
         v_column_to_left(j) := v_current_column(j);
      END LOOP;
      END LOOP;
      END IF;
   RETURN v_current_column(v_length_of_target);
END edit_distance;
/

Test this out by using the expresso/espresso example mentioned earlier.

-- Available online as part of edit_distance.sql
SELECT EDIT_DISTANCE('espresso', 'expresso') AS DISTANCE
FROM dual;
20 Expert Oracle PL/SQL

This returns a value of 1 as expected. This function performs essentially the same thing as the built-in UTL_MATCH.EDIT_DISTANCE function now available in Oracle 10g Release 2. To test this out, run the following:

```
-- Available online as part of edit_distance.sql
SELECT UTL_MATCH.EDIT_DISTANCE('espresso', 'expresso') AS DISTANCE FROM dual;
```

This returns a value of 1 as well.

UTL.Match includes a second Edit Distance function that measures the similarity between strings. The return value is an integer between 0 and 100, where 0 indicates no similarity at all and 100 indicates a perfect match. This example tests out the similarity between expresso and espresso:

```
-- Available online as part of edit_distance.sql
SELECT UTL_MATCH.EDIT_DISTANCE_SIMILARITY('expresso', 'espresso') AS DISTANCE FROM dual;
```

This returns a value of 88 indicating that they are very much alike.

**Jaro-Winkler**

The Jaro-Winkler algorithm is the second category of algorithms used in UTL_MATCH. These functions take the same two arguments, but instead of simply calculating the number of steps required to change the source string to the destination string, it determines how closely the two strings agree with each other. The algorithm also tries to take into account the possibility of a data entry error when determining similarity.

The following example uses the first Jaro-Winkler function, simply called JARO_WINKLER:

```
-- Available online as part of jaro_winkler.sql
SELECT UTL_MATCH.JARO_WINKLER('expresso', 'espresso') AS AGREEMENT FROM dual;
```

This returns the following value:

```
AGREEMENT
----------
 9.25E-001
```

This is the measure of agreement between the strings.
How Does That Work?
The Jaro-Winkler algorithm tries to take into account some level of human error in the analysis of the data. The following calculation, discussed in a presentation by Dr. Adrian Esterman and John Bass on various name-matching techniques, shows how the Jaro-Winkler algorithm works.

\[
\text{JW} = \left( \frac{1}{3} \cdot \frac{c}{s_1} \right) + \left( \frac{1}{3} \cdot \frac{c}{s_2} \right) + \left( \frac{1}{3} \cdot \frac{t}{c} \right)
\]

Where:

- \( c \) = The number of characters that match
- \( s_1 \) = The length of the first string
- \( s_2 \) = The length of the second string
- \( t \) = The number of transpositions required

Just like Edit Distance, there is a similarity function. To run it, do the following:

```sql
-- Available online as part of jaro_winkler.sql
SELECT UTL_MATCH.JARO_WINKLER_SIMILARITY(
    'expresso', 'espresso') AS AGREEMENT
FROM dual;
```

The result is as follows:

```
AGREEMENT
----------
   92
```

Like `EDIT_DISTANCE_SIMILARITY`, the closer the return value is to 100, the closer the two strings are to each other.

**DBMS_OUTPUT.PUT_LINE**

Of course, `DBMS_OUTPUT.PUT_LINE` is not new. It has been available for years to print lines to the screen as follows:

```sql
-- Available online as part of dbms_output.sql
SET SERVEROUTPUT ON
```

Of course, `DBMS_OUTPUT.PUT_LINE` has been available for years to print lines to the screen as follows:
Expert Oracle PL/SQL

BEGIN
    DBMS_OUTPUT.PUT_LINE('PRINT ME');
END;
/

If you have used this procedure for anything but a ‘Hello World’ example though, you know about the 255-byte limitation with the procedure. The following example tries to print more than 255 bytes of text in Oracle 9i:

--Available online as part of dbms_output.sql
SET SERVEROUTPUT ON
DECLARE
    v_string VARCHAR2(500 CHAR);
BEGIN
    v_string := 'Five years of programming. This is the time it'
    || 'takes for most programmers to feel as if they''ve'
    || 'come of age as a professional. Traditional'
    || 'programming concepts are second nature, like'
    || 'driving a car or typing on a keyboard. Employers'
    || 'recognize this, often ...';
    DBMS_OUTPUT.PUT_LINE(v_string);
END;
/

It returns the following:

ERROR at line 1:
ORA-20000: ORU-10028: line length overflow, limit of 255 chars per line

The solution in prior releases was to use the SUBSTR function as follows:

--Available online as part of dbms_output.sql
SET SERVEROUTPUT ON
DECLARE
    v_string VARCHAR2(500 CHAR);
BEGIN
    v_string := 'Five years of programming. This is the time it'
    || 'takes for most programmers to feel as if they''ve'
    || 'come of age as a professional. Traditional'
    || 'programming concepts are second nature, like'
    || 'driving a car or typing on a keyboard. Employers'
    || 'recognize this, often ...';
    DBMS_OUTPUT.PUT_LINE(SUBSTR(v_string, 1, 255));
    DBMS_OUTPUT.PUT_LINE(SUBSTR(v_string, 256));
END;
/
Chapter 1: Introduction to Advanced Concepts, Patterns, and Techniques

This is not the most user-friendly way of handling this. Oracle 10g Release 2 lifts the 255-byte limitation. Run the same example in 10g Release 2.

```sql
SET SERVEROUTPUT ON
DECLARE
  v_string VARCHAR2(500 CHAR);
BEGIN
  v_string := 'Five years of programming. This is the time it'
              || 'takes for most programmers to feel as if they''ve'
              || 'come of age as a professional. Traditional'
              || 'programming concepts are second nature, like'
              || 'driving a car or typing on a keyboard. Employers'
              || 'recognize this, often ...';
  DBMS_OUTPUT.PUT_LINE(v_string);
END;
/
```

This prints correctly and is a welcome sight for developers.

Summary

In this chapter, we defined what an expert is, both from the standpoint of the level of knowledge a person has, and from the range of impact on both your organization and other workers that a person might have. Finally, we demonstrated some new features introduced in Oracle 10g Release 2 and showed how they can be leveraged immediately.