12 Miscellaneous New Features

CERTIFICATION OBJECTIVES

The first part of this chapter deals with enhancements in Oracle's VPD (Virtual Private Database) feature. You'll learn about new VPD features like applying column-level security policies, static and nonstatic policies, and sharing VPD functions. You'll also review the concept of a unified audit trail and the use of fine-grained auditing for DML statements.

The latter part of this chapter deals with miscellaneous new features of Oracle Database 10*g*, including some new SQL enhancements and new PL/SQL packages. You'll look at the aggregation of meaningful statistics across a multitier environment. These new statistics help you perform end-to-end tracing of Oracle sessions. The exciting new topic of regular expressions provides POSIX-like regular expression capabilities that help you write more powerful SQL search statements. You'll also learn about the new linguistic and sorting methods in Oracle Database 10*g* SQL.

Let's start this final chapter by reviewing the enhancements in the Virtual Private Database and auditing areas.

CERTIFICATION OBJECTIVE 12.01

VPD and Auditing Enhancements

Oracle8*i* and Oracle9*i* introduced the concepts of fine-grained access control (FGAC) and Secure Application Contexts. Oracle calls the combination of these two features Virtual Private Database (VPD). Fine-grained access policies use the application attributes defined in an application context, helping you enforce security policies at the row level. VPD lets you implement security policies at a fine level of granularity by attaching your security policy to the data objects themselves (tables, views, and synonyms). These security policies contain directives to transparently change a user's statement whenever a user accesses a table, view, or synonym that's part of a VPD security policy.

exam watch

VPD policies apply to tables, views, and synonyms. You can apply VPD policies to `SELECT`, `INSERT`, `DELETE`, `UPDATE`, and any `INDEX` statements.

You can apply VPD policies to index maintenance operations that you may perform with any `CREATE INDEX` and `ALTER INDEX` statements. You may want to apply VPD policies to these operations because you need full table access to create an index. Thus, if you can create and rebuild an index, you can view all the table's data.

on the job ***You can use an application context independently of FGAC.***

Column-Level VPD

Oracle Database 10*g* introduces the capability to enforce security at the column level. VPD can apply fine-grained access control (which you define with the help of policy functions) only when you access certain security-relevant columns. For example, if you define salary as a security-relevant column in table emp, then employees can continue to freely access information in all the other columns. However, Oracle will dynamically modify any SQL statements that refer to the salary column in order to protect sensitive data. Since Oracle already provides row-level access control (FGAC), this new column-level access control extends access control to the individual data item level in a table. Note that Oracle will rewrite a SQL statement only if a SQL query accesses the security-relevant columns.

exam watch ***A column-level VPD policy applies only to tables and views and not to synonyms. You may apply a policy function to queries as well as DML statements.***

Types of Behavior

You apply the column-level VPD to tables and views with the help of the `DBMS_RLS.ADD_POLICY` procedure. You specify the security-relevant column names with the `sec_relevant_cols` parameter of this package. When you use column-level VPD, you have a choice of two types of behavior by the policy—default behavior and column-masking behavior. *Default behavior* will restrict the number of rows returned by any query that contains the security-relevant columns(s). *Column-masking behavior*, on the other hand, will return all the rows, but show null values for the security-relevant columns in those rows.

Creating a Column-Level Policy

In order to create a column-level security policy, you must have EXECUTE privileges on the `DBMS_RLS` package. You must first create a policy function that will implant your VPD policy. You then use the `DBMS_RLS` package to apply your policy to a table or view. Here's an example of how you create a column-level policy:

```
begin
   dbms_rls.add_policy (object_schema=>'scott',
                         object_name=>'emp',
                         policy_name=>'test_policy',
                                function_schema=>'test_schema',
                                policy_function=>'test_function',
                                statement_type='insert,update'
                                sec_relevant_cols=>'salary,commission');
end;
```

Example of a Column-Level VPD

In the preceding column-level policy, the salary and commission columns in table emp are the security-relevant columns. The VPD predicate will apply to all `INSERT` and `UPDATE` statements that include either or both the salary and commission columns. Normally, a VPD policy is applied to an entire row. A column-level VPD policy will restrict the rows retrieved by a query if the query references any of the security-relevant columns. The following example shows how to create VPD with a column-level security policy.

Creating the Policy Function First create a new policy function, test_function, which will restrict access to the salary and commission columns if an employee is not part of department 5.

```
create or replace function test_function (objowner IN      2, varchar2
  objname IN varchar2)
  return varchar2 as
  con varchar2 (200);
begin
  con := 'deptno = 5';
  return (con);
end test_function;
/
```

Create the VPD Once you create the `test_function` policy function, you can create the VPD by applying it to a table, as shown here:

```
begin
  dbms_rls.add_policy (object_schema     => 'scott',
                       object_name       => 'emp',
                       policy_name       => 'test_policy',
                       function_schema   => 'scott',
                       policy_function   => 'test_function',
                       sec_relevant_cols => 'salary,commission');
end;
/
```

If you reference either or both of the security-relevant columns, salary and commission, the rows in the output are restricted, as shown in the following example:

```
SQL> select  deptno, empno, ename, job, sal, comm from emp;
DEPTNO EMPNO ENAME      JOB           SAL       COMM
---------- ---------- --------- ---------- ----------
    5  7369 SMITH      CLERK         10000
    5  7566 JONES      MANAGER        2975
    5  7788 SCOTT      ANALYST        3000
    5  7876 ADAMS      CLERK          1100
    5  7902 FORD       ANALYST        3000
5 rows selected.
```

You can implement column-masking behavior by using the `sec_relevant_cols_opt => DBMS_RLS.ALL_ROWS` parameter. This allows you to display all rows but hide the values of the specified columns for the restricted rows. Here's an example:

```
begin
  dbms_rls.add_policy (object_schema         => 'scott',
                       object_name           => 'emp',
                       policy_name           => 'sp_job',
                       function_schema       => 'scott',
                       policy_function       => 'pf_job',
                       sec_relevant_cols     => 'sal,comm',
                       sec_relevant_cols_opt => DBMS_RLS.ALL_ROWS);
end;
/
```

Once you choose the column-masking behavior by using the `DBMS_RLS.ALL_ROWS` option, your query will return all rows, but it will show the salary and commission values for only those employees that belong to department 5.

```
SQL> select deptno,empno, ename, job, sal, comm from emp;
 DEPTNO      EMPNO    ENAME     JOB          SAL       COMM
------------------------------------------------------------
      5      7369    SMITH     CLERK        10000
     10      7499   ALLEN     SALESMAN
     10      7521    WARD      SALESMAN
      5      7566    JONES     MANAGER       2975
      2      7654    MARTIN    SALESMAN
      1      7698    BLAKE     MANAGER
      8      7782    CLARK     MANAGER
      5      7788    SCOTT     ANALYST       3000
      2      7839    KING      PRESIDENT
      1      7844    TURNER    SALESMAN
      5      7876    ADAMS     CLERK         1100
      4      7900    JAMES     CLERK
      5      7902    FORD      ANALYST       3000
      2      7934    MILLER    CLERK
```

on the job

You can grant the privilege *`GRANT EXEMPT ACCESS POLICY`* ***to a user so that he or she may bypass a security policy, as shown here:***

```
SQL>  grant exempt access policy to hr;
Grant succeeded.
```

New Policy Types

By default, all Oracle VPD policy functions are dynamic in nature. That is, Oracle will execute the security policy statement each time a DML statement refers to it. This leads to the expenditure of valuable system resources due to the constant need to reexecute and evaluate security policies. Each time Oracle reparses a SQL statement or executes it, it reexecutes any policy function that the statement might have. Oracle Database 10*g* introduces new policy types designed to improve server performance by avoiding the need for reexecution and by allowing the sharing of policies across multiple database objects. Let's look at these new policy types in the following sections.

Static Policies

Oracle Database 10*g* lets you configure *static* policies. The database executes a static policy function just once, and caches the predicate resulting from the policy evaluation in the SGA. It then applies this predicate to all queries accessing the protected objects. If you think that all queries, regardless of which user issues them, need the same policy predicate, you might want to configure a static policy.

A static policy is one that will always return the same `WHERE` clause, such as `deptno=something` so that even though the "something" may be different for each statement executed against the table, the actual SQL doesn't change (a bit like a bind variable). But a dynamic policy is one that could generate a different `WHERE` clause. For instance, the function could query usercontext and, depending on the result, generate different `WHERE` clauses. Thus, some users would get `deptno=`..., and others would get `mgr=`..., for example. So flagging a policy as static is simply telling Oracle that the function can only return one value (possibly containing a variable) no matter who executes it, whereas a dynamic policy is based on a function that has several different return values. If you tell Oracle that the policy will always get the same return value, then Oracle doesn't have to repeatedly generate it.

You can add a static policy using the `DBMS_RLS.ADD_POLICY` procedure, as shown here:

```
begin
   dbms_rls.add_policy (object_schema=>'scott',
                        object_name=>'emp',
                        policy_name=>'test_policy',
                        function_schema=>'test_schema',
                        policy_function=>'test_function',
                        statement_type='select,insert,update'
                        policy_type => dbms_rls.static
                        sec_relevant_cols=>'salary , commission');
end;
```

In the preceding example, the highlighted line shows how you specify a static policy type. The default policy type in Oracle Database 10*g* is dynamic.

If you want to allow the *sharing* of the same static policy function over different database objects, you can set the POLICY_TYPE parameter to the following value:

```
POLICY_TYPE  => DBMS_RLS.SHARED_STATIC
```

exam watch

The default Oracle policy type in Oracle Database 10g is dynamic, but you can configure static and context-sensitive policies. If you want to enforce the same business policy over several database objects, you can do so by sharing either a static or a context-sensitive policy.

Context-Sensitive Policies

You may sometimes want to configure a policy such that it will change based on any session context changes. That is, each time certain context attributes within a user's session change, you want the policy function to automatically change as well. In cases like this, you can use the new *context-sensitive* policies. When you use a context-sensitive function, Oracle will evaluate the policy function when it first parses the statement. Thereafter, Oracle will evaluate the policy function each time there is a local application context change. Here's our DBMS_RLS example, this time with a context-sensitive policy type.

on the job

Context-sensitive VPD policies are particularly useful in a web-based application with connection pooling, where a session needs to change its behavior depending on the CLIENT_IDENTIFIER of the user using the session at any given moment.

```
begin
   dbms_rls.add_policy (object_schema=>'scott',
                        object_name=>'emp',
                        policy_name=>'test_policy',
                        function_schema=>'test_schema',
                        policy_function=>'test_function',
                        statement_type='select,insert,update'
                        policy_type => dbms_rls.context_sensitive
                        sec_relevant_cols=>'sal,comm');
end;
```

If you want to allow the sharing of the same context-sensitive policy function over different database objects, you can set the POLICY_TYPE parameter to the following value when you use the DBMS_RLS.ADD_POLICY procedure:

```
POLICY_TYPE  => DBMS_RLS.SHARED_CONTEXT_SENSITIVE
```

exam watch

You now have five different policy types, with `DYNAMIC` being the default type. There are two static policy types—`STATIC` and `SHARED_STATIC`. There are two context-sensitive policies—`CONTEXT_SENSITIVE` and `SHARED_CONTEXT_SENSITIVE`. The purpose behind having several types of policies is aimed at reducing the number of executions of policy functions, which sometimes cripple performance.

Auditing Enhancements

Oracle databases provide several auditing mechanisms. You have the option to audit database activity based on data content by attaching audit policies directly to objects. Oracle calls this feature fine-grained auditing (FGA).When you use FGA, Oracle creates audit records based on the specific query and the data that the SQL statement accesses.

In Oracle Database 10*g*, enhancements to auditing features include uniform audit trails for standard auditing and FGA, enhanced auditing of enterprise users, and several FGA enhancements. Let's review Oracle Database 10*g* auditing enhancements by looking at uniform audit trails for standard and fine-grained auditing.

Uniform Auditing Trails

Oracle Database 10*g* helps you audit database activities in a uniform manner by using a new *uniform audit trail* for both standard and fine-grained audit log records. Both the standard audit trail and FGA audit trail track similar audit fields in Oracle Database 10*g*.

on the job

You can view the new uniform audit trail by using the `DBA_COMMON_AUDIT_TRAIL` ***view. In the following text, the variables in the parentheses are columns of this view.***

exam watch

You can view the new SCN and SQL text/bind variable information only if you use the new `AUDIT_TRAIL=DB_EXTENDED` ***specification in your initialization parameter file.***

Standard auditing now collects the following additional information:

- System change numbers (SCN)
- Bind variable information (SQL_BIND) and the exact SQL text (SQL_TEXT) in the audited statement

In Oracle Database 10*g*, FGA collects the following additional information:

- Serial number of audit records (ENTRYID)
- Statement numbers that help you identify which audit entries belong to a SQL statement (STATEMENTID)

In addition, the following attributes are common to both standard and FGA audit trails:

- Global timestamp (GLOBAL_UID)
- A unique instance number for Real Application Cluster instances (INSTANCE_NUMBER)
- A transaction identification number to trace individual records to a transaction (TRANSACTIONID)

Enterprise User Auditing

When you use an LDAP-compliant directory like the Oracle Internet Directory, your users are known as enterprise users. Oracle Database 10*g* lets you audit the activities of the enterprise users in the database. Enterprise users can map to either an exclusive or a shared schema in your database. If the enterprise user maps to an exclusive schema in your database, the database username and the enterprise user are identical. In shared schemas, the database username and the enterprise users are different. Let's see how Oracle enhances enterprise user auditing in both types of user mapping.

Exclusive Schemas If the enterprise user accesses the database by mapping to an *exclusive schema*, in both standard and fine-grained auditing, the GLOBAL_UID column shows the user's global identity. In standard auditing, the USERNAME column shows the user's identity in the database, and in fine-grained auditing, the DB_USER column shows the user's identity in the database.

Shared Schemas If the enterprise user accesses the database by mapping to a *shared schema*, in both standard and fine-grained auditing, the GLOBAL_UID column shows the user's global identity. In standard auditing, the USERNAME column shows the shared schema, and in fine-grained auditing, the DB_USER column shows the shared schema.

Fine-Grained Auditing Enhancements

In previous versions of Oracle, fine-grained auditing supported only `SELECT` statements. Oracle Database 10*g* provides the following fine-grained auditing enhancements:

- You can audit`SELECT`, `INSERT`, `UPDATE`, `DELETE`, and `MERGE` statements.
- You can provide more than one relevant column for fine-grained auditing.
- You can now use NULL fine-grained auditing policy predicates.
- Since fine-grained auditing imposes significant SQL information overhead, you can avoid the writing of SQL text and SQL bind information to LOBs.

FGA and DML Statements

Oracle Database 10*g*'s FGA features deal with DML statements of all types, but there are some important provisions that you must remember. Here's a summary of new DML statement considerations:

- Oracle Database 10*g* will audit a DML statement with an FGA policy defined on it if the data rows (old and new) qualify under the policy predicate.
- If you have a *relevant column(s)* in the security policy, the DML statement will be audited only if it references the column(s) and the data meets the FGA policy requirements.
- Oracle always audits a qualified `DELETE` statement, regardless of whether you have specified relevant columns or not.

exam watch

Oracle's FGA feature audits `MERGE` statements by viewing the `INSERT` and `DELETE` statements in the `MERGE` statement as individual statements. If there are applicable FGA policies for the `INSERT` or UPDATE statement, Oracle will audit the `MERGE` statement.

CERTIFICATION OBJECTIVE 12.02

Enhancements in Managing Multitier Environments

Debugging of performance problems in multitier environments is difficult, since you can't easily track a client across different database sessions. Oracle Database 10*g* introduces new statistical attributes that help you perform end-to-end tracing. Let's look at these new attributes in the following sections.

New Dimensions for Statistics Collection and Tracing

Previous versions of Oracle enabled statistics collection only at the individual SQL statement, session, or instance level. Most DBAs are very familiar with session-based tracing. These three levels are adequate in most cases for simple database configurations. However, when you are dealing with complex multitier architectures using connection pooling and shared server architectures, these simple statistical aggregation levels are simply inadequate in monitoring performance problems. Realizing this, Oracle Database 10*g* provides you with several new dimensions for collecting statistics, as follows:

- Client identifier
- Service name
- Combinations of service name, module name, and action name

These three new dimensions for collecting statistics offer you new and powerful means of monitoring problems in multitier architectures. The new statistics focus on the clients and services, rather than database sessions and SQL statements. With the help of metrics and statistics for these new dimensions, you can improve performance monitoring of clients and services, as well as manage your workload more efficiently.

One of the most difficult problems in a multitier environment is the inability to trace a client's work on an end-to-end basis. The middle tier would frequently transfer the same client to different sessions, making tracing of the client extremely difficult. In Oracle Database 10*g*, the new variable CLIENT_IDENTIFIER enables you to trace the same client from the time the user logs into the system until the user logs off, regardless of how many different sessions the client switches to through the application server.

You can see the value of the CLIENT_IDENTIFIER variable in two ways. You can use the V$SESSION view (CLIENT_IDENTIFIER column) or use the following query, which makes use of the system context.

```
SQL> select sys_context('USERENV', 'CLIENT_IDENTIFIER')
  2  from dual;
SYS_CONTEXT('USERENV','CLIENT_IDENTIFIER')
------------------------------------------
salapati
SQL>
```

exam watch

By default, waits and binds aren't traced.

Besides enabling the tracing of individual client sessions, end-to-end tracing also enables you to track a transaction using hierarchical combinations of the the service, module, and action names. You can find out what traces are currently enabled in your database by using the DBA_ENABLED_TRACES view.

Enabling the Collection of Client and Service Statistics

You use the DBMS_MONITOR package to enable client- and service-level statistics aggregation. The procedures you can use to enable and disable the tracing and collection of the new statistics are summarized next.

Client-Level Statistics

You use the CLIENT_ID_STAT_ENABLE procedure to enable the collection of statistics for a client identifier, as shown here:

```
SQL> execute dbms_monitor.client_id_stat_enable(<client_id>);
```

You can disable the collection of client-level statistics by using the `CLIENT_ID_STAT_DISABLE` procedure.

Service-Level Statistics

You use the `SERV_MOD_ACT_STAT_ENABLE` procedure to enable the collection of statistics for a client identifier, as shown here:

```
SQL> execute dbms_monitor.serv_mod_act_stat_enable(<service_name,
 <module_name>, <action_name>);
```

You can disable the collection of client-level statistics by using the `SERV_MOD_ACT_STAT_DISABLE` procedure.

Tracing Clients and Services

You can use the new client identifier (CLIENT_ID) and the combination of service, module, and action names to trace sessions on both a global and session level. Let's first see how you enable and disable tracing.

Enabling and Disabling Tracing

You can enable and disable tracing by using various procedures of the `DBMS_MONITOR` package. Here's a summary of how to enable and disable tracing.

Client Identifier Tracing You use the following procedures to enable and disable tracing for client identifiers:

- **`CLIENT_ID_TRACE_ENABLE`** Enable global (database-wide) tracing of a client identifier
- **`CLIENT_ID_TRACE_DISABLE`** Disable client identifier tracing

Service, Module, and Action Tracing You use the following procedures to enable and disable tracing for combinations of service names, module names, and action names on a global basis.

- **`SERVICE_MOD_ACT_TRACE_ENABLE`** Enable global (database-wide) tracing
- **`SERVICE_MOD_ACT_TRACE_DISABLE`** Disable global (database-wide) tracing (module and service names are optional)

on the job

You may want to refer to the `DBMS_APPLICATION_INFO` package to see how to set both a module name and action name so that you have a module and an action to trace after enabling their tracing with the `SERVICE_MOD_ACT_TRACE_ENABLE` procedure. The `DBMS_APPLICATION_INFO` has two procedures, `SET_MODULE` and `SET_ACTION`, which allow programmers to specify module and action names. The `SET_MODULE` procedure sets the name of the current application or module. The `SET_ACTION` procedure sets the name of the current action within the current module.

Using the TRCSESS Tool to Analyze Trace Files

When there are several trace files from a given session, it is hard to combine the output of all the traces into a unified format. You can use Oracle's `TRCSESS` command-line utility to consolidate the information from all your trace files into a single output file. Once you consolidate the trace files, you can use the familiar `TKPROF` utility to format the trace file output into a readable format. Here's the general syntax for using the `TRCSESS` tool:

```
$ trcsess  output=<user.trc> clientid=<user_name> *.trc
```

Viewing the New Statistics

Once you have enabled the collection of the new client identifier, service, module, and action names statistics, you can view them by using Database Control. There are also several new views that help you track these important statistics. Here are some of the important new views:

- **`DBA_ENABLED_AGGREGATIONS`** Displays information about enabled statistics aggregation
- **`DBA_ENABLED_TRACES`** Shows all enabled traces in the system
- **`V$CLIENT_STATS`** Displays statistics on a client level (CLIENT_IDENTIFIER based)
- **`V$SERVICE_STATS`** Displays basic performance statistics
- **`V$SERV_MOD_ACT_STATS`** Displays statistics for a combination of serve /module/action names.

CERTIFICATION OBJECTIVE 12.03

SQL and PL/SQL Enhancements

There are several SQL and PL/SQL enhancements in Oracle Database 10*g*, and you've already seen some of them, like the `MODEL` clause, in earlier chapters. In the following sections, I'll review some of the more important enhancements from the point of view of an Oracle DBA.

UTL_COMPRESS Package

Oracle Database 10*g* provides the new `UTL_COMPRESS` package to compress and uncompress data, with the compressed output compatible with the output of the familiar `GZIP` and `GUNZIP` compression utilities. You may want to compress data to save on storage space. You can choose betweeen compression speed and the quality of compression. Quality of compression refers to the reduction in size due to compressing data. There is an inverse relationship between speed and quality of the compression job.

UTL_MAIL Package

In previous versions, Oracle provided the `UTL_SMTP` package to send email. However, this isn't an intuitive package to understand and is quite unwieldy for sending simple email notes. In addition, you had to understand the underlying SMTP protocol features to use it. In Oracle Database 10g, you have access to a new utility for sending email, the `UTL_MAIL` package. You can now send programmatically composed emails using simple PL/SQL API. You use simple parameters to specify email components such as sender, receiver, copies, blind copies, subject, and so on.

In order to use the `UTL_MAIL` package to send email, you must first execute the utlmail.sql script located in your ORACLE_HOME/rdbms/admin directory.You must also execute the prvtmail.plb script located in the same directory. Here's a simple example that shows how to send an email using the `UTL_MAIL` package:

```
SQL> execute utl_mail.send  -
> (sender         => 'salapati@netbsa.org', -
>  recipients     =>    'mpotts@netbsa.org', -
>  subject        => 'testing 10g Packages',
>  message        => 'No attachements with this note');
```

Regular Expressions

Oracle provides several operators to help you with string manipulation and searching. Operators such as the SQL `LIKE`, `REPLACE`, `SUBSTRING` and IN`STRING` are old standbys in many a search expression. However, these older operators have serious limitations when it comes to searching and manipulating complex patterns in data. You would have to write numerous lines of SQL and PL/SQL code to search for complex expressions. Oracle Database 10*g* enhances your ability to perform convenient searches and string manipulation by adding regular expression support to both SQL and PL/SQL programming. Oracle's regular expression features follow the syntax and semantics of the operators defined by the POSIX regular expression standards. You use special regular expression functions that parallel the traditional search operators such as `LIKE`, `INSTRING`, `SUBSTRING`, and `REPLACE`.

A regular expression searches for patterns in a character string, which is the source for the search value. The source character string can be one of the CHAR, VARCHAR2, NCHAR, or NVARCHAR2 datatypes. The regular expression function can be one of the following:

```
REGEXP_LIKE
REGEXP_REPLACE
REGEXP_INSTRING
REGEXP_SUBSTRING
```

The `REGEXP_LIKE` function evaluates strings using characters as defined by the input character set. The regular expression function will search for a certain pattern in a regular expression, indicated by the `source_string` parameter in the function. The expression represented by the pattern variable is the *regular expression*. A regular expression is usually a text literal and can be any of the datatypes CHAR, VARCHAR2, NCHAR, or NVARCHAR2. Regular expressions can be long if you wish, with a maximum size of 512 bytes. In addition to the search pattern indicated by `source_ string`, you can use an optional matching condition (*match parameter*) to modify the default matching behavior. For example, a value of `'i'` specifies case-insensitive matching, and a value of `'c'` specifies case-sensitive matching.

Here's the generic syntax of the `REGEXP_LIKE` function:

The SQL operator `REGEXP_LIKE` is useful in searching for regular expressions. If you want to perform string manipulation tasks, you can use the built-in functions `REGEXP_INSTR`, `REGEXP_REPLACE`, and `REGEXP_SUBSTR`. You can use these regular expression functions as extensions of their normal SQL counterparts like `INSTR`, `REPLACE`, and `SUBSTR`. For example, the `REGEXP_LIKE` function is similar to the

LIKE condition, except that REGEXP_LIKE will perform regular expression matching instead of the simple pattern matching performed by the LIKE function.

To match characters and to indicate the beginning and end of lines, regular expression features use characters such as ., *, ^, and $, which are common in UNIX and Perl programming. The character ^, for example, tells Oracle that the characters following it should be at the beginning of the line. Similarly, the character $ indicates that a character or a set of characters must be at the very end of the line. Here's an example using the REGEXP_LIKE function. (The query picks up all names with consecutive vowels in their names.)

```
SQL> select last_name
     from employees
     where regexp_like (last_name, '([aeiou])\1', 'i');
LAST_NAME
-----------
De Haan
Greenberg
Khoo
Gee
Lee
......
```

EXERCISE 12-1

Using a regualr expression in a query

Use the appropriate regular expression built-in function to search for all employees that were hired in the years 1996 and 1997. (Use the table employees in the sample schema HR.)

```
SQL> SELECT last_name,first_name, salary,
  2  TO_CHAR(hire_date,'yyyy') year_of_hire
  3  FROM hr.employees
  4* WHERE REGEXP_LIKE (TO_CHAR (hire_date, 'yyyy'), '^199[6-7]$');
LAST_NAME                 FIRST_NAME           SALARY     YEAR
------------------------- -------------------- ---------- ----
Alapati                   Nina                  4800      1997
Chen                      John                  8200      1997
Sciarra                   Ismael                7700      1997
Baida                     Shelli                2900      1997
Tobias                    Sigal                 2800      1997
Wilson               Shannon                    8000      1996
```

Case-Insensitive and Accent-Insensitive Query and Sort

Oracle Database 10*g* provides support for case-insensitive queries and sorts, allowing users to search and sort their data whatever the case and accent of the characters might be. You use the `NLS_SORT` parameter to specify the lingusitic sort name and whether it is a case-insensitive or accent-insensitive sort.

When you use the `NLS_SORT` parameter, you can use the optional suffixes `AI` or `CI` to specify whether the sort is accent insensitive (`AI`) or case insensitive (`CI`). Here's the general syntax of the `NLS_SORT` parameter and an example of its usage:

```
NLS_SORT = <NLS_sort_name>[_AI| _CI]
NLS_SORT = FRENCH_M_AI.
```

In the previous example, the name of the `NLS_SORT` is `FRENCH_M`. The optional suffix `AI` indicates this is an accent-insensitive (and case-sensitive) sort.You can indicate a case-insenstitive sort by using the suffix CI.

CLOB and NCLOB Implicit Conversions

The maximum size of an LOB datatype (BLOB, CLOB, and NCLOB) in Oracle Database 10*g* is much larger than the old limit of 4GB. You can now have LOBs that can be sized in terabytes, with the maximum size being about 128TB.

Oracle Database 10*g* also provides implicit conversion between CLOB and NCLOB datatypes. Before this version, you had to explicitly convert CLOBs into NCLOBs and vice versa, using the `TO_CLOB` and `TO_NCLOB` functions. International applications supporting multiple language characters required the deployment of function calls to convert these kinds of data.

Oracle Database 10*g* introduces the implicit conversion between LOBs and NCLOBs. Oracle now supports implicit conversion in SQL IN/OUT bind variables, PL/SQL function and procedure parameter passing, and PL/SQL variable assignment.

User-Specified Quoting Characters

In prior versions of Oracle, programmers had to use quotation strings in character string literals. Oracle Database 10*g* lets you choose any convenient delimiter and define it dynamically as the quoting character. You can choose any single or multibyte character, or paired characters like [], {}, (), and <>. You may even choose a single quotation mark as your delimiter.

You use the new *quote operator* `q` to provide your own quotation mark delimiters. Here's a simple example that illustrates how you can use the new quote operator to avoid using escape characters for single quotation marks in text literals:

```
SQL> select cust_address FROM oe.customers
     where cust_last_name = q'X 'John's Bait Shop' X'
```

CERTIFICATION OBJECTIVE 12.04

Miscellaneous Enhancements

There are several enhancements in Oracle Database 10*g* that don't fit within the topics of any of the earlier chapters in this book. I summarize these enhancements in the following sections.

Easy Connect Naming Method

Oracle DBAs can now simplify client configuration by using the new *easy connect naming method*. Your database clients can now connect to Oracle Database 10*g* database services without any configuration fields or name lookup services. All that your clients will need to connect to the database server is the host name and the optional port number and service name of the database.

The only condition for using the easy connect naming method is that you should have support for the TCP/IP protocol on both the client and the server. You don't have to configure a tnsnames.ora file. You can look at this new connecting method as an extension of the host naming method introduced in Oracle9*i*. Here's the general syntax of this new connecting method :

```
$ sqlplus /nolog

connect username/password@[//]host[:port][/service_name]
```

In the new easy connect method, there are four different things you need to focus on: host, the // notation, port number (default port = 1521), and service name. Of these four things, only the host name is mandatory. For example, you can connect to the dev1 database located on the server hp50 with the following connect identifier (note that I'm connecting directly from the operating system prompt, so I replace the keword `CONNECT` with `SQLPLUS`):

```
C:\>sqlplus appowner/password@hp50:1521/dev1
```

When you install an Oracle database, Oracle will configure your sqlnet.ora file to use the new configuration-less client connect method. The NAMES.DIRECTORY_PATH parameter in the sqlnet.ora file shows all connect methods. The new easy connect method is referred to as EZCONNECT in a sqlnet.ora file. Look at the following sqlnet.ora file:

```
# sqlnet.ora Network Configuration File:
C:\oracle\product\10.1.0\Db_1\network\admin\sqlnet.ora
# Generated by Oracle configuration tools.
NAMES.DEFAULT_DOMAIN = netbsa.org
SQLNET.AUTHENTICATION_SERVICES = (NTS)
NAMES.DIRECTORY_PATH = (TNSNAMES,EZCONNECT)
```

The last line shows you the connect methods that Oracle Net will use to resolve connect identifiers to connect descriptors. In this example, TNSNAMES is the first option, so Oracle NET will use the tnsnames.ora file first by default. If it can't connect using the tnsnames.ora file, it will then use the EZCONNECT connecting method. If you want Oracle Net to use the EZCONNECT method first, you can use Oracle Net Manager to reconfigure the sqlnet.ora file so your NAMES.DIRECTORY_PATH will be as follows. (You may manually change the sqlnet.ora file as well.)

```
NAMES.DIRECTORY_PATH = (EZCONNECT, TNSNAMES)
```

Simplified Shared Server Configuration

Oracle Database 10*g* is shared server aware, meaning that you don't have to configure any shared server parameters. A dispatcher will start automatically when you start a database instance, but no shared server process will start. If you want to start a shared server while your instance is running, you can do so by setting a non-zero value for the SHARED_SERVER initialization parameter, as shown here:

```
SQL> alter system set shared_servers=4
System altered.
SQL>
```

Enabling Resumable Space Allocation

Oracle Database 10*g* introduces a new initialization parameter, RESUMABLE_TIMEOUT to enable resumable statements at the system or the session level. To set the resumable time-out to three hours, for example, you'd specify RESUMABLE_TIMEOUT=10800 (3x60x60) in your initialization parameter file. The default for this initialization parameter is zero, meaning that resumable space allocation is disabled for all database sessions. You can dynamically use the parameter in the following way to enable resumable space allocation at the instance level:

```
SQL> alter system
  2  set resumable_timeout=7200;
System altered.
SQL>
```

You can dynamically change the RESUMABLE_TIMEOUT parameter at an individual session level as well, by using the ALTER SESSION command. The old method of enabling resumable statements at the session level by using the command ALTER SESSION ENABLE RESUMABLE still works in Oracle Database 10*g*.

Flushing the Buffer Cache

When you are testing SQL statements on a development system, there are times when you wish that you could wipe out the data in your buffer cache in order to accurately compare the output from various iterations of your queries. Previously, you could only flush the shared pool to remove any cached SQL statements from the SGA. You can flush the database buffer cache in Oracle Database 10*g* by using the following command:

```
SQL> alter system flush buffer cache;
System altered.
SQL>
```

LogMiner Enhancements

There are three important changes in the LogMiner tool. Let me summarize the changes briefly here.

Automatic Adding of Redo Log Files

You can now simply specify a time or SCN, and LogMiner will automatically add the necessary redo log files by scanning the control files for the log information. You must use the DBMS_LOGMNR.CONTINUOUS_MINE procedure to facilitate this automatic gathering of redo log files for mining purposes.

Disabling Generation of ROWIDs

You can disable the generation of physical ROWIDs by using the NO_ROWID_IN_STMT option when you use the DBMS_LOGMNR package.

Easier Removal of Redo Log Files

To remove redo log files, you can now use the new REMOVE_LOGFILE procedure with the DBMS_LOGMNR package.

Real-Time Transaction Monitoring

Oracle Database 10*g* introduces enhancements to the `V$FAST_START_SERVERS` and the `V$FAST_START_TRANSACTIONS` views to enable real-time monitoring of normal transaction rollback and transaction recovery by the SMON background process. You can also make better estimations of your transaction recovery time by calculating more accurate rollback times during transaction recoveries. Let's review the changes in real-time transaction monitoring in Oracle Database 10*g*.

The V$FAST_START_SERVERS View

This view provides information about all the recovery servers performing (or that have performed) parallel transaction recovery. In Oracle Database 10*g*, you have a new column, XID, that gives you the transaction ID of the transaction that a recovery server is working on.

The V$FAST_START_TRANSACTIONS View

This view shows information about the progress of the transactions that Oracle is currently recovering. The STATE column shows the value RECOVERING for all transactions that are being recovered right now. In addition, the view stores historical recovery information until the next shutdown of the instance. The transactions already recovered by Oracle have the value RECOVERED under the STATE column. If any transaction has yet to be recovered, the STATE column will show the value TO BE RECOVERED.

exam watch

The enhancements in transaction rollback monitoring help you set the `FAST_START PARALLEL_ ROLLBACK` initialization parameter at an appropriate level.

Oracle Database 10*g* has added the following new columns to the `V$FAST_START_TRANSACTIONS` view to enhance your ability to keep track of the progress of transaction recovery:

- **XID** Stands for the transaction ID.
- **PXID** Parent transaction ID.
- **RCVSERVERS** Number of servers used in the last recovery. If SMON is doing the recovery, the value of this parameter is always 1.

Automatic Checkpoint Tuning

DBAs have to consider the trade-off between a quick recovery time and I/O overhead when trying to figure out the appropriate checkpoint interval. In Oracle9*i*, you had access to the FAST_START_MTTR_TARGET initialization parameter, which enabled you to specify the mean time for a crash recovery.

In Oracle Database 10*g*, there is no need for you to set the FAST_START_MTTR_TARGET parameter because Oracle itself will automatically tune the checkpointing process. Automatic checkpoint tuning means that Oracle will write out the dirty buffers while incurring the least amount of impact in the way of physical I/Os.

You can enable automatic checkpoint tuning by simply setting the FAST_START_MTTR_TARGET parameter to any non-zero value. Oracle automatically tunes checkpointing even if you completely leave out the FAST_START_MTTR_TARGET parameter. However, by explicitly setting the parameter to zero, you can disable Oracle's automatic checkpoint tuning.

INSIDE THE EXAM

You can expect some type of question on the new column-level VPD policies. The exam tests your expertise in customized VPD policy types. What are the five VPD policy types, and when do you use them? Be sure to understand how to use shared policy functions. The exam will contain questions on the new auditing features. Understand what the new AUDIT_TRAIL=DB_EXTENDED specification will help you do. You must be familiar with the DBA_COMMON_AUDIT_TRAIL view and its important columns.

You are going to see questions on fine-grained auditing enhancements, including topics like support for DML statements (and the MERGE statement), multicolumn FGA policies, and NULL FGA policy predicates. How does Oracle audit MERGE statements? Look up the DBMS_FGA package and study the ADD_POLICY procedure carefully, with particular emphasis on the AUDIT_COLUMN, AUDIT_COLUMN_OPTS, and the AUDIT_TRAIL columns.

The exam will test your knowledge of the new CLIENT_IDENTIFIER, SERVICE_NAME, MODULE_NAME, and ACTION_NAME attributes. What are these attributes useful for? Please look up the DBMS_MONITOR package and study the CLIENT_ID_TRACE_ENABLE and the SERV_MOD_ACT_TRACE_ENABLE procedures. Also look up the procedures that enable and disable statistics collection for the new attributes. You must know how to enable and disable end-to-end tracing using the new attributes.

Among the miscellaneous SQL and PL/SQL topics, focus on regular expressions and the new packages to send email and compress data. Know the Oracle Database 10*g* enhancements in CLOB to NCLOB conversions, the new quote operator, and how to perform case- and accent-insensitive querying and sorting. The exam also may have questions on the new resumable statement parameter, the easy connect naming method, and the easy configuration of shared servers.

CHAPTER SUMMARY

You reviewed the new column-level VPD policies in this chapter. You learned how to create a VPD with security-relevant columns.You also learned about the new static and context-sensitive policies and how you can share them across objects.The chapter covered the Oracle Database 10*g* enhancements in the auditing area, including the new uniform audit trail for standard and fine-grained auditing .You learned about the new enterprise user auditing enhancements as well as other FGA enhancements.

The new attributes—client identifier, service name, module name, and action name—enhance your ability to perform end-to-end tracing as well as monitor your database workload. You learned how to enable and disable client- and service-level tracing and statistics collection.

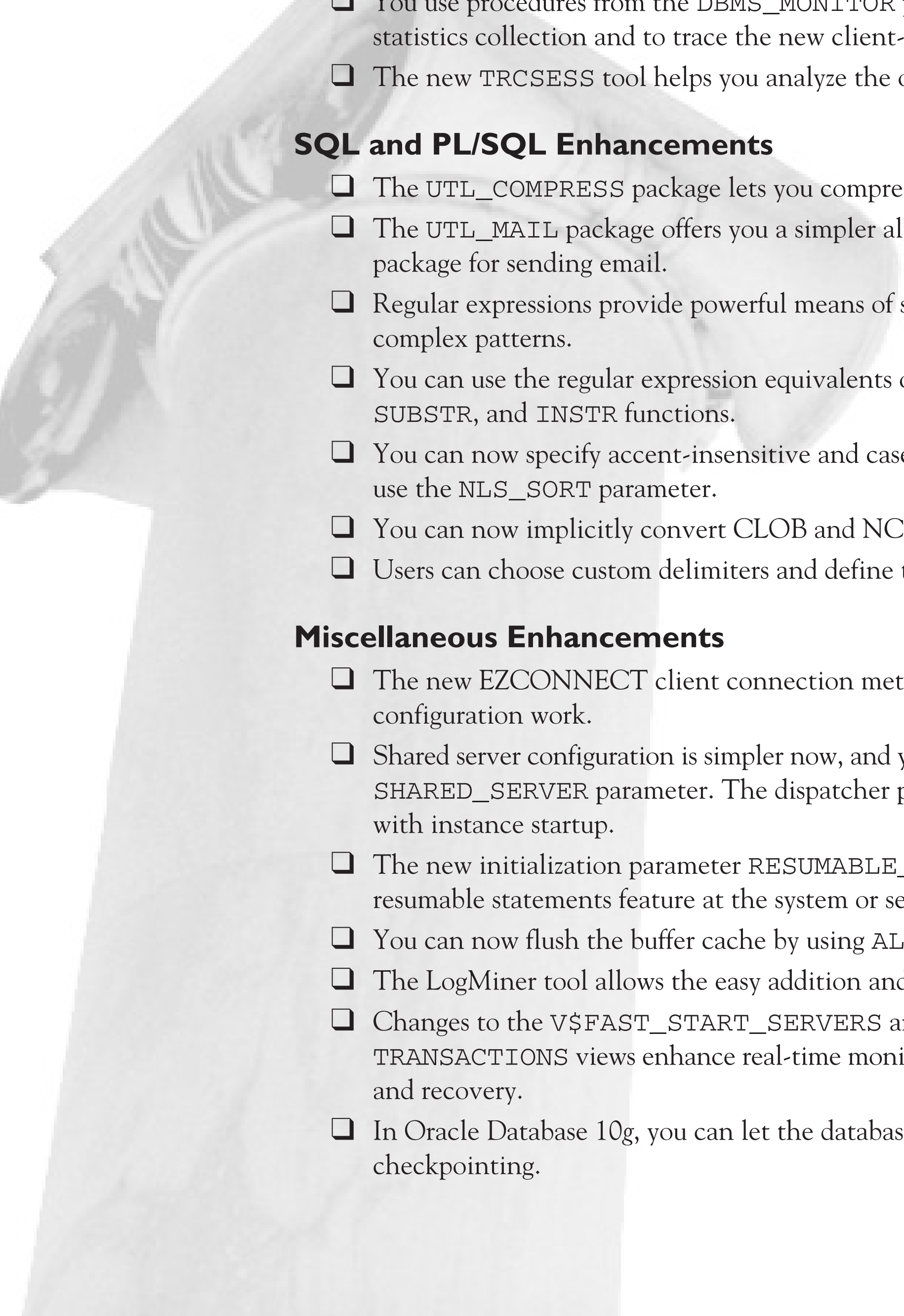
The chapter introduced you to the new `UTL_MAIL` and the `UTL_COMPRESS` PL/SQL packages. You had an introduction to the powerful regular expressions feature. The chapter also reviewed the new case- and accent-insensitive sorting techniques as well as user-specified delimiters.

Under miscellaneous enhancements, you briefly reviewed the new easy connect naming method, simpler shared server configuration, the new resumable space allocation initialization parameter, and enhancements to the LogMiner tool and real-time transaction monitoring.

✓ TWO-MINUTE DRILL

VPD and Auditing Enhancements

- ❑ Oracle Database 10g introduces the new column-level security mechanisms (VPD).
- ❑ You can now assign certain columns as security-relevant columns.
- ❑ You can apply a column-level VPD policy to tables and views, but not to synonyms.
- ❑ You can apply policy functions to queries as well as DML statements.
- ❑ You implement the column-level VPD policies with the help of the `DBMS_RLS.ADD_POLICY` procedure.
- ❑ Default behavior will restrict the number of rows returned.
- ❑ Column-masking behavior will return all rows, but show NULL for the security-relevant columns.
- ❑ You implement column-masking behavior by using the `sec_relevant_cols_opt => DBMS_RLS.ALL_ROWS` parameter when using the `ADD_POLICY procedure.`
- ❑ By default, all Oracle VPD policy functions are dynamic.
- ❑ A static policy is only executed once and reused for all subsequent queries.
- ❑ A context-sensitive policy changes with session context changes.
- ❑ Both static and context-sensitive policies can be shared or nonshared policies.
- ❑ You have a new uniform audit trail for both standard and FGA audit log records.
- ❑ You have to set the `AUDIT_TRAIL=DB_EXTENDED` parameter value in order to see bind variables and the complete SQL text of the audited statements.
- ❑ There are several enhancements in the enterprise user auditing in Oracle Database 10g.
- ❑ You can audit queries and DML statements (including the `MERGE` statement) in the course of fine-grained auditing.
- ❑ You can provide multiple columns for FGA as well as use NULL FGA auditing policy predicates.
- ❑ Oracle will always audit a qualified `DELETE` statement under FGA.
- ❑ Oracle's FGA feature audits `MERGE` statements by viewing them as consisting of individual `UPDATE` and `INSERT` statements.



Enhancements in Managing Multitier Environments

- ❑ New dimensions for collect statistics include client identifier, service name, and combinations of service name, module name, and action name.
- ❑ The CLIENT_IDENTIFIER attribute facilitates end-to-end tracing.
- ❑ You use procedures from the `DBMS_MONITOR` package to enable and disable statistics collection and to trace the new client- and server-related attributes.
- ❑ The new `TRCSESS` tool helps you analyze the output of multiple trace files.

SQL and PL/SQL Enhancements

- ❑ The `UTL_COMPRESS` package lets you compress and uncompress data.
- ❑ The `UTL_MAIL` package offers you a simpler alternative to the `UTL_SMTP` package for sending email.
- ❑ Regular expressions provide powerful means of searching and manipulating complex patterns.
- ❑ You can use the regular expression equivalents of the `LIKE`, `REPLACE`, `SUBSTR`, and `INSTR` functions.
- ❑ You can now specify accent-insensitive and case-insensitive sorting when you use the `NLS_SORT` parameter.
- ❑ You can now implicitly convert CLOB and NCLOB datatypes.
- ❑ Users can choose custom delimiters and define them as a quoting character.

Miscellaneous Enhancements

- ❑ The new EZCONNECT client connection method cuts down on client configuration work.
- ❑ Shared server configuration is simpler now, and you only have to configure the `SHARED_SERVER` parameter. The dispatcher process starts automatically with instance startup.
- ❑ The new initialization parameter `RESUMABLE_TIMEOUT` enables the resumable statements feature at the system or session level.
- ❑ You can now flush the buffer cache by using `ALTER SYSTEM` command.
- ❑ The LogMiner tool allows the easy addition and removal of redo log files.
- ❑ Changes to the `V$FAST_START_SERVERS` and the `V$FAST_START_TRANSACTIONS` views enhance real-time monitoring of transaction rollback and recovery.
- ❑ In Oracle Database 10g, you can let the database automatically tune database checkpointing.

SELF TEST

VPD and Auditing Enhancements

1. Which of the following does a column-level VPD policy apply to?

A. Tables
B. Synonyms
C. Views
D. Materialized views
E. Indexes

2. Which of the following type of SQL statements may you apply a policy function to?

A. DDL statements
B. `SELECT` statements only
C. Queries and DML statements
D. `INSERT` and `DELETE` statements only
E. `INSERT`, `DELETE`, and `UPDATE` statements

3. Which of the following statements regarding column-level VPD are true?

A. Default behavior will return all columns.
B. Default behavior will restrict the number of rows returned.
C. Column-masking behavior will return all rows.
D. Column-masking behavior will restrict the number of rows returned.

4. Which of the following is the default policy type in Oracle Database 10*g*?

A. POLICY_TYPE=DBMS_RLS.STATIC
B. POLICY_TYPE=DBMS_RLS.DYNAMIC
C. POLICY_TYPE=DBMS_RLS.SHARED
D. POLICY_TYPE=DBMS_RLS.CONTEXT_SENSITIVE

5. How does Oracle Database 10*g* audit a `MERGE` statement?

A. It audits only the `UPDATE` part of the `MERGE` statement.
B. It audits only the `DELETE` part of the `MERGE` statement.
C. It audits the `MERGE` statement with the underlying `INSERT` or `UPDATE` statements.
D. It doesn't audit `MERGE` statements.

Enhancements in Managing Multitier Environments

6. What are the two ways in which you can find out the value of the new attribute, CLIENT_IDENTIFIER?

A. `V$INSTANCE`

B. `V$SESSION`

C. `DBA_USERS`

D. `SYS_CONTEXT`

7. How do you enable tracing for all calls for a given application client?

A. `EXEC DBMS_MONITOR.client_id_trace_enable`

B. `EXEC DBMS_MONITOR.client_id_stat_enable`

C. `EXEC DBMS_MONITOR.client_identifier_trace_enable`

D. `EXEC DBMS_MONITOR.serv_mod_act_trace_enable`

8. What do you use the `SERV_MOD_ACT_TRACE_ENABLE` procedure for?

A. To trace a client session throughout its life

B. To enable SQL tracing for a combination of client ID, service name, and module name

C. To enable SQL tracing for a combination of username, service name, and module name

D. To enable SQL tracing for a combination of service name, action name, and module name

9. What does the `TRCSESS` utility do?

A. Help you enable a sessionwide trace

B. Help you interpret end-to-end tracing output

C. Consolidate information from several trace files into a single output

D. Stress the system to see how much load it can handle when you introduce new SQL statements in the database.

10. What are the three benefits of using the new statistics aggregation dimensions in Oracle Database 10*g* (client identifier, service name, module name, and action name)?

A. Monitoring performance of individual clients

B. Managing the workload

C. Setting threshold-based alerts

D. Providing correct input to ADDM

SQL and PL/SQL Enhancements

11. Which of the following are new Oracle Database 10*g* enhancements regarding conversion between CLOB and NCLOB datatypes?

A. Implicit conversion of SQL IN and OUT bind variables in queries and DML statements
B. Explicit conversion of SQL IN and OUT bind variables in queries and DML statements
C. Implicit conversion of SQL IN and OUT bind variables in queries only
D. Implicit conversions of SQL IN and OUT bind variables in DML statements only, but not in queries

12. Which of the following is *not* an Oracle regular expression function?

A. `REGEXP_LIKE`
B. `REGEXP_BETWEEN`
C. `REGEXP_REPLACE`
D. `REGEXP_SUBSTR`

13. Which of the following shows a correct specification of a regular expression?

A. `REGEXP_LIKE (source_string, pattern, match_option)`
B. `REGEXP_LIKE (source_string, match_option, pattern)`
C. `REGEXP_LIKE (pattern, source_string, match_option)`
D. `REGEXP_LIKE (match, pattern, match_option)`

14. Which two of the following are enhancements to the `NLS_SORT` function?

A. Case-insensitive functionality
B. Case-sensitive functionality
C. Accent-sensitive functionality
D. Accent-insensitive functionality

15. The output of the `UTL_COMPRESS` compressed data is compatible with which of the following?

A. `GZIP` only
B. `GZIP` and `GUNZIP`
C. `GUNZIP` only
D. Neither `GZIP` nor `GUNZIP`

Miscellaneous Enhancements

16. Which two of the following statements will work if your goal is to enable resumable statements in Oracle Database 10*g*?

A. `ALTER SESSION SET RESUMABLE STATEMENTS=TRUE`
B. `ALTER SESSION SET RESUMABLE =TRUE`
C. `ALTER SESSION SET RESUMABLE_TIMEOUT=3600`
D. `RESUMABLE TIMEOUT=3600`

17. In Oracle Database 10g, which of the following do you have to do in order to start a shared server?

A. `ALTER SYSTEM SET SHARED_SERVERS=8`

B. `ALTER SYSTEM SET DISPATCHERS=2`

C. `ALTER SYSTEM ENABLE SHARED SERVER`

D. `ALTER SYSTEM START DISPATCHER`

18. Which two of the following will ensure automatic checkpoint tuning by Oracle?

A. Set the `FAST_START_MTTR_TARGET` parameter to a positive value.

B. Set the `FAST_START_MTTR_TARGET` parameter to zero.

C. Don't set the `FAST_START_MTTR_TARGET` parameter.

D. Set the `FAST_START_MTTR_TARGET` to a value of TRUE.

19. Which of the following is mandatory when you are using the easy connect naming method?

A. Host name

B. Database name

C. Service name

D. Port number

20. Which `DBMS_LOGMNR` feature facilitates the automatic adding of redo log files for mining?

A. `REMOVE_LOGFILE`

B. `CONTINUOUS_MINE`

C. `NO_ROWID_IN_STMT`

D. `AUTOMATIC_LOGFILE`

LAB QUESTION

Given the following statement, use regular expressions to search for a comma followed by a space. You must then have zero or more characters that are not commas, and include another comma in the end.

```
'first child, second child , third child',
```

SELF TEST ANSWERS

VPD and Auditing Enhancements

1. ☑ **A** and **C.** Column-level VPD policies apply only to tables and views.
☒ **B, D,** and **E** are wrong since column-level VPD policies don't apply to these objects.

2. ☑ **C.** You may apply a policy function to any`SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement.
☒ **A, B,** and D are wrong since they either point out the wrong type (DDL statements) or leave out some DDL statements that are allowed.

3. ☑ **B** and **C.** B is correct because the default behavior under a column-level VPD will restrict their rows returned to only those that meet the VPD column-level restrictions. C is correct since the column-masking behavior will always return all rows, but will have NULL values for the columns to which the column-level VPD policy applies.
☒ **A** and **D** are wrong since the actions they represent are exactly opposite to the actual behavior of Oracle, as explained above for the correct answers.

4. ☑ **B.** The default policy type in Oracle Database 10*g* is dynamic.
☒ **A, C,** and **D** are wrong since dynamic policy type is the default.

5. ☑ **C.** Oracle will treat a `MERGE` statement as individual `UPDATE` and `INSERT` statements, and will audit those statements on an individual basis.
☒ **A** and **B** are wrong since Oracle audits both `UPDATE` and `INSERT` statements. **D** is wrong since Oracle does audit `MERGE` statements.

Enhancements in Managing Multitier Environments

6. ☑ **B** and **D** are correct since you find out the value of the CLIENT_IDENTIFIER attribute by querying the `V$SESSION` view, or by using `SYS_CONTEXT`.
☒ **A** and **C** are wrong since neither the `V$INSTANCE` nor the `DBA_USERS` view has any information on the CLIENT_IDENTIFIER attribute.

7. ☑ **A.** You use the `client_id_trace_enable` procedure to enable tracing of the CLIENT_IDENTIFIER attribute, which enables end-to-end tracing of all calls for a database client.
☒ **B** is wrong because the procedure enables you to collect statistics, not trace sessions. **C** is wrong because it refers to CLIENT_IDENTIFIER instead of CLIENT_ID. **D** is wrong because this procedure is for tracing a combination of service name, module name, and action names.

8. ☑ **D.** The SERV_MOD_ACT_TRACE procedure enables SQL tracing for a combination of service name, action name, and module name.
☒ **A, B,** and **C** are wrong because the SERV_MOD_ACT_TRACE procedure doesn't help with tracing client activities directly.

9. ☑ **C.** The TRCSESS utility consolidates output from several tracing sessions into a single output, so you can trace a session more effectively.
☒ **A** is incorrect because the TRCSESS utility doesn't help you with enabling a trace. **B** is wrong because it's the TKPROF utility that helps interpret the output of a trace. **D** is wrong because the TRCSESS session isn't a stress testing tool.

10. ☑ **A, B,** and **C.** The new statistics aggregation dimensions help you monitor individual client performance, manage the workload, and set threshold-based alerts.
☒ **D** is wrong because the ADDM tool doesn't get any data based on the new statistics aggregation dimensions.

SQL and PL/SQL Enhancements

11. ☑ **A.** Oracle now implicitly converts SQL IN and OUT bind variables in both queries and DML statements when they involve CLOB and NCLOB datatypes.
☒ **B** is wrong since you had do the conversion explicitly in Oracle9*i*, not Oracle Database 10*g*. **C** and **D** are wrong since the implicit conversion happens in both queries and DML statements.

12. ☑ **B.** The REGEXP_BETWEEN is not a valid Oracle regular expression function.
☒ **A, C,** and **D** are wrong since all of these are valid Oracle regular expression functions.

13. ☑ **A.** All regular expression specification must first provide the source string, after which comes the pattern (regular expression) and finally, any optional match options.
☒ **B** and **C** are wrong since they present the right components in the wrong order. **D** is wrong since it includes a nonexistent component (match).

14. ☑ **A** and **D.** Oracle Database 10*g* provides both case-insensitive and accent-insensitive functionality through the NLS_SORT function.
☒ **B** and **C** are wrong since you already have these features in Oracle9*i*.

15. ☑ **B.** The output of the UTL_COMPRESS utility is compatible with the output of the GZIP and GUNZIP utilities.
☒ **A, C,** and **D** are wrong alternatives since the UTL_COMPRESS utility's output is compatible with output from both the GZIP and GUNZIP utilities.

Miscellaneous Enhancements

16. ☑ **C** and **D. C** is correct since this statement was valid in Oracle9*i* and continues to work the same way in Oracle Database 10*g*. **D** shows how you can enable resumable statements with the new `RESUMABLE TIMEOUT` initialization parameter.
☒ **A** and **B** are wrong because they present statements with the wrong syntax.

17. ☑ **A.** In Oracle Database 10*g*, you only need to set the number of shared servers by using the `ALTER SYSTEM SET SHARED SERVERS` command to start the shared server.
☒ **B, C,** and **D** are wrong since they are all nonexistent commands.

18. ☑ **A** and **C.** You let Oracle automatically tune checkpointing by either setting the FAST_START_MTTR_TARGET to a non-zero value or leaving it out completely.
☒ **B** is wrong since setting the FAST_START_MTTR_TARGET to zero will disable automatic checkpoint tuning. **D** refers to a nonexistent option.

19. ☑ **A.** Host name is the only item that's mandatory in the new easy connect naming method.
☒ **B, C,** and **D** are wrong since none of these is mandatory under the new naming method.

20. ☑ **B.** If you specify the `CONTINUOUS_MINE` option, and specify a `STARTSCN` or `STARTTIME` parameter value, Oracle will automatically add the redo logs by default.
☒ **A** is wrong because this option helps you remove redo log files, not add them automatically. **C** is wrong since this option is used for generating reconstructed statements without a ROWID. **D** refers to a spurious procedure name.

LAB ANSWER

The following regular expression-based query will get you the result you have been asked to produce. Notice the use of the pattern `', [^,]*,'` in the query.

```
SQL> select regexp_substr('first child, second child , third child',
  2    ', [^,]*,')
  3*   from dual;
REGEXP_SUBSTR('F
-------------------
, second child ,
SQL>
```