EXAM OBJECTIVE: PROVIDE GREATER FLEXIBILITY BY ENABLING RESUMABLE TIMEOUT AT THE INSTANCE LEVEL

RESUMEABLE SPACE ALLOCATION FEATURE OVERVIEW

In Oracle, the Resumable space allocation feature provides a means for suspending and later resuming the execution of large database operations in the event of space allocation failures. A suspended statement can be forced to throw the exception using the DBMS_RESUMABLE.ABORT() procedure. This procedure can be called by a DBA, or a user who issued the statement. A suspension timeout interval may be associated with resumable statements. A resumable statement that is suspended for the timeout interval (default being two hours) wakes up and returns the exception to the user.

In Oracle 10g, you can enable resumable space allocation system wide and specify a timeout interval in seconds by setting the **RESUMABLE_TIMEOUT** initialization parameter.

In the example given below, the value of the RESUMABLE_TIMEOUT parameter causes all sessions to initially be enabled for resumable space allocation and sets the timeout period to 1 hour:

RESUMABLE_TIMEOUT=3600

If this parameter is set to 0, then resumable space allocation is disabled initially for all sessions. This is the default.

You can also use the ALTER SYSTEM SET command to change the value of this parameter dynamically.

ALTER SYSTEM SET RESUMABLE_TIMEOUT=0;

The above statement will disable resumable space allocation for all sessions.

EXAM OBJECTIVE: USE REGULAR EXPRESSION SUPPORT IN SQL AND PL/SQL FOR STRING SEARCHING, MATCHING AND REPLACING

Regular expressions specify patterns to search for in string data using standardized syntax conventions. A regular expression can specify complex patterns of character sequences.

For example, the following regular expressions:

a(b|c)d searches for the pattern 'a', followed by either 'b' or 'c', then followed by 'd'. This regular expression matches both 'abd' and 'acd'.

A regular expression is specified using two types of characters:

- Metacharacters – operators that specify algorithms for performing the search.
- Literals – the actual characters to search for.

Regular expression support is implemented with a set of Oracle Database SQL functions that allow you to search and manipulate string data. The metacharacters supported in regular expression is given below:

Metacharacter Syntax	Operator Name	Description
.	Any Character -- Dot	Matches any character
+	One or More -- Plus Quantifier	Matches one or more occurrences of the preceding subexpression
?	Zero or One -- Question Mark Quantifier	Matches zero or one occurrence of the preceding subexpression
*	Zero or More -- Star Quantifier	Matches zero or more occurrences of the preceding subexpression
{m}	Interval--Exact Count	Matches exactly m occurrences of the preceding subexpression
{m,}	Interval--At Least Count	Matches at least m occurrences of the preceding subexpression
{m,n}	Interval--Between Count	Matches at least m, but not more than n occurrences of the preceding subexpression
[...]	Matching Character List	Matches any character in list ...
[^ ...]	Non-Matching Character List	Matches any character not in list ...
|	Or	'a|b' matches character 'a' or 'b'.
(...)	Subexpression or Grouping	Treat expression ... as a unit. The subexpression can be a string of literals or a complex expression containing operators.
\\n	Backreference	Matches the nth preceding subexpression, where n is an integer from 1 to 9.
\\	Escape Character	Treat the subsequent metacharacter in the expression as a literal.

Metacharacter Syntax	Operator Name	Description
^	Beginning of Line Anchor	Match the subsequent expression only when it occurs at the beginning of a line.
$	End of Line Anchor	Match the preceding expression only when it occurs at the end of a line.
[:*class*:]	POSIX Character Class	Match any character belonging to the specified character *class*. Can be used inside any list expression.
[.*element*.]	POSIX Collating Sequence	Specifies a collating sequence to use in the regular expression. The *element* you use must be a defined collating sequence, in the current locale.
[=*character*=]	POSIX Character Equivalence Class	Match characters having the same base character as the *character* you specify.

Support for metacharacters

The dot (.) operator - matches any single character in the current character set.
Regular Expression :   a.c
Match :   abc , adc , a1c , a&c
Does not match: ann

The + operator - matches one or more occurrences of the preceding expression.
Regular expression : a+
Match : a, aa, aaa
Does not match: bbb

The question mark (?) - matches zero or one--and only one--occurrence of the preceding character or subexpression.
Regular Expression : ab?c
Match : abc , ac
Does not match: adc, abbc

The '*' operator - matches zero or more occurrences of the preceding character or subexpression.
Regular expression : ab*c
Match : ac ,abc, abbc, abbbbc
Does not match :adc

The {} operator - The exact-count interval operator is specified with a single digit enclosed in braces. You use this operator to search for an exact number of occurrences of the preceding character or subexpression.

Regular expression : a{5}
Matches: aaaaa
Does not match : aaaa

At-least-count interval operator - You use the at-least-count interval operator to search for a specified number of occurrences, or more, of the preceding character or subexpression.
Regular expression: a{3,}
Match : aaa, aaaaa
Does not match : aa

Interval--Between Count - You use the between-count interval operator to search for a number of occurrences within a specified range.
Regular expression : a{3,5}
Matches : aaa ,aaaa ,aaaaa
Does not match: aa

Matching Character List -You use the matching character list to search for an occurrence of any character in a list.
Regular expression : [abc]
Matches : at ,bet, cot
Does not match : def

Non-Matching Character List - Use the non-matching character list to specify characters that you do not want to match. Characters that are not in the non-matching character list are returned as a
Regular Expression : [^abc]
This expression matches characters 'd' and 'g' in the following strings:
abcdef
ghi
The expression does not match: abc

As with the matching character list, the following regular expression operators are allowed within the non-matching character list (any other metacharacters included in a character list are ignored):

- Range operator '-'
- POSIX character class [::]
- POSIX collating sequence [. .]
- POSIX character equivalence class [= =]

Regular expression : [^a-i]
This expression matches the characters 'j' and 'l' in the following strings:

Hijk ,lmn
Does not match : abcdefghi

Subexpression - You can use the subexpression operator to group characters that you want to find as a string or to create a complex expression. For example, to find the optional string 'abc', followed by 'def', use the following regular expression:
(abc)?def

This expression matches strings 'abcdef' and 'def' in the following strings:
abcdefghi ,defghi
Does not match the string: ghi

Backreference - The backreference lets you search for a repeated expression. You specify a backreference with '\n', where n is an integer from 1 to 9 indicating the nth preceding subexpression in your regular expression.
For example, to find a repeated occurrence of either string 'abc' or 'def'.
Regular Expression: (abc|def)\1
Matches: abcabc ,defdef
Does not match the following strings: abcdef ,abc

Escape Character - Use the escape character '\' to search for a character that is normally treated as a metacharacter. For example to search for the '+' character.
regular expression: \+
Matches : abc+def
Does not match: abcdef

Beginning of Line Anchor - Use the beginning of line anchor ^ to search for an expression that occurs only at the beginning of a line.
Regular Expression: ^def
Matches: defghi
Does not match: abcdef

End of Line Anchor- The end of line anchor metacharacter '$' lets you search for an expression that occurs only at the end of a line.
Regular expression: def$
Matches: abcdef
Does not match: defghi

POSIX Character Class
Specifies character classes (for example, [:alpha:]). It matches any character within the character class. The full set of POSIX character classes is supported. For example, to search for one or more consecutive uppercase characters, use
Regular expression: [[:upper:]]+

Matches the 'DEF': abcDEFghi
Does not match: abcdefghi

Collating Sequence - [..]: A collating element is a unit of collation and is equal to one character in most cases, but may comprise two or more characters in some languages. Historically, regular expression syntax does not support ranges containing multicharacter collation elements, such as the range 'a' through 'ch'.

The POSIX standard introduces the collation element delimiter '[..]', which lets you delimit multicharacter collection elements such like this one as follows: '[a-[.ch.]]'.
The collation elements supported by Oracle are determined by the setting of the NLS_SORT initialization parameter. The collation element is valid only inside the bracketed expression.

Equivalence Classes – [==] : Specifies equivalence classes. For example, [=a=] matches all characters having base letter 'a'. Oracle supports the equivalence classes through the POSIX '[==]' syntax. A base letter and all of its accented versions constitute an equivalence class. For example, the equivalence class '[=a=]' matches ä and â. The equivalence classes are valid only inside the bracketed expression.
For example, the following regular expression could be used to search for characters equivalent to 'n' in a Spanish locale:
Regular Expression : [[=n=]]
This expression matches both 'N' and 'ñ' in the following string: El Niño

NEW PL/SQL FUNCTIONS

Several new functions have been introduced in Oracle 10g, for regular expression support. The implementation complies with the Portable Operating system for UNIX (POSIX) standard. These functions are all prefixed with REGEXP_:

Function Name	Description
REGEXP_LIKE	Similar to the LIKE operator, but performs a regular expression matching instead of simple pattern matching. (srcstr,pattern[,match_option])
REGEXP_REPLACE	Searches for a regular expression pattern and replaces it with a replacement string. (srcstr, pattern [, replacestr [, position [

	, occurrence [, match_option]]]])
REGEXP_INSTR	Searches for a given string for a regular expression pattern and returns the position where the match is found. (srcstr,pattern[,position [,occurrence [, return option [, match option]]]])
REGEXP_SUBSTR	Searches for a regular expression pattern within a given string and returns the matched substring. (srcstr , pattern [, position [, occurrence [, match_position]]])

The REGEXP function syntax:

Srcstr	Search value
Pattern	Regular expression
Occurrence	Occurrence to search for
Position	Search starting position
Return_option	The start or end position of occurrence
Replacestr	Character string replacing pattern
Match_option	Option to change default matching, it can include one or more of the following values: 'c' – use character sensitive matching (default). 'i' – use character insensitive matching 'n' – allows match-any-character support 'm' – treats source string as multiple line

Examples : REGEXP_LIKE

The following query returns the first and last names for those employees with a first name of Steven or Stephen (where first_name begins with Ste and ends with en and in between is either v or ph):

SELECT first_name, last_name
FROM employees
WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$');

FIRST_NAME   LAST_NAME
-------------------- -------------------------
Steven   King
Steven   Markle
Stephen   Stiles

The following query returns the last name for those employees with a double vowel in their last name (where last_name contains two adjacent occurrences of either a, e, i, o, or u, regardless of case):

```
SELECT last_name
FROM employees
WHERE REGEXP_LIKE (last_name, '([aeiou])\1', 'i');
```

LAST_ NAME

De Haan
Greenberg
Khoo
Gee
Greene

Examples : REGEXP_REPLACE

The following example examines phone_number, looking for the pattern xxx.xxx.xxxx. Oracle reformats this pattern with (xxx) xxx-xxxx.

```
SELECT
  REGEXP_REPLACE(phone_number,
         '([[:digit:]]{3})\.([[:digit:]]{3})\.([[:digit:]]{4})',
         '(\1) \2-\3') "REGEXP_REPLACE"
  FROM employees;
```

REGEXP_REPLACE
----
(515) 123-4567
(515) 123-4568
(515) 123-4569
(590) 423-4567
. . .

The following example examines country_name. Oracle puts a space after each non-null character in the string.

```
SELECT
REGEXP_REPLACE(country_name, '(.)', '\1 ') "REGEXP_REPLACE"
FROM countries;
```

REGEXP_REPLACE

A r g e n t i n a
A u s t r a l i a
B e l g i u m
B r a z i l
C a n a d a
. . .

The following example examines the string, looking for two or more spaces.
Oracle replaces each occurrence of two or more spaces with a single space.

```
SELECT
  REGEXP_REPLACE('500   Oracle    Parkway,   Redwood   Shores, CA',
         '( ){2,}', ' ') "REGEXP_REPLACE"
  FROM DUAL;
```

REGEXP_REPLACE

----

500 Oracle Parkway, Redwood Shores, CA

Examples : REGEXP_INSTR

The following example examines the string, looking for occurrences of one or
more non-blank characters. Oracle begins searching at the first character in the
string and returns the starting position (default) of the sixth occurrence of one or
more non-blank characters.

```
SELECT
  REGEXP_INSTR('500 Oracle Parkway, Redwood Shores, CA',
         '[^ ]+', 1, 6) "REGEXP_INSTR"
  FROM DUAL;
```

REGEXP_INSTR

     37

The following example examines the string, looking for occurrences of words
beginning with s, r, or p, regardless of case, followed by any six alphabetic
characters. Oracle begins searching at the third character in the string and returns

the position in the string of the character following the second occurrence of a seven letter word beginning with s, r, or p, regardless of case.

```
SELECT
  REGEXP_INSTR('500 Oracle Parkway, Redwood Shores, CA',
        '[s|r|p][[:alpha:]]{6}', 3, 2, 1, 'i') "REGEXP_INSTR"
  FROM DUAL;

REGEXP_INSTR

------------
      28
```

Examples : REGEXP_SUBSTR

The following example examines the string, looking for the first substring bounded by commas. Oracle Database searches for a comma followed by one or more occurrences of non-comma characters followed by a comma. Oracle returns the substring, including the leading and trailing commas.

```
SELECT
  REGEXP_SUBSTR('500 Oracle Parkway, Redwood Shores, CA',
        ',[^,]+,') "REGEXPR_SUBSTR"
  FROM DUAL;

REGEXPR_SUBSTR

-----------------
, Redwood Shores,
```

The following example examines the string, looking for http:// followed by a substring of one or more alphanumeric characters and optionally, a period (.). Oracle searches for a minimum of three and a maximum of four occurrences of this substring between http:// and either a slash (/) or the end of the string.

```
SELECT
  REGEXP_SUBSTR('http://www.oracle.com/products',
        'http://([[:alnum:]]+\.?){3,4}/?') "REGEXP_SUBSTR"
  FROM DUAL;

REGEXP_SUBSTR

----------------------
http://www.oracle.com/
```

EXAM OBJECTIVE: USE ADDITIONAL LINGUISTIC COMPARISON AND SORTING METHODS IN SQL

Case insensitive comparison and sorting are frequently used features in a database. Two new sorting options introduced in Oracle 10g are:

- Case-insensitive sort
- Accent-insensitive sort

You can use the NLS_SORT session parameter to specify a case-insensitive or an accent-insensitive sort. This can be achieved by either appending an _CI or _AI to an Oracle sort name.

_CI : results in a case_insensitive sort
_AI : results in an accent-insensitive and case_insensitive sort.

Examples of setting the parameter are displayed below:
1) Accent-insensitive and case-insensitive 'Spanish' Sort
NLS_SORT = SPANISH_AI
2) Accent-sensitive and case-insensitive 'French' sort
NLS_SORT = FRENCH_CI
3) Accent-sensitive and case-sensitive 'Japanese' sort
NLS_SORT=JAPANESE
4) Accent-sensitive and case-insensitive 'Binary' sort
NLS_SORT = BINARY_CI
5) Accent-insensitive and case-insensitive 'Binary' sort
NLS_SORT=BINARY_AI

Consider the example given below: For example, with the nls_lang environment variable set to american_america.we8iso8859p1 you can create a table called test1 and populate it as follows:

```
SQL> CREATE TABLE test1 (letter VARCHAR2(10));
SQL> INSERT INTO test1 VALUES('ä');
SQL> INSERT INTO test1 VALUES('a');
SQL> INSERT INTO test1 VALUES('A');
SQL> INSERT INTO test1 VALUES('Z');
SQL> SELECT * FROM test1;
LETTER
------
ä
a
A
Z
```

Since the default value of nls_sort is BINARY you don't need to specify anything extra to use a binary sort. Use the following statement to do a binary sort of the characters in table test1:

SELECT * FROM test1 ORDER BY letter;
To change the value of nls_sort, you would enter a statement like (only using your sort specifier):
ALTER SESSION SET NLS_SORT=BINARY_AI;

The following table shows the results from setting nls_sort to binary, binary_ci, and binary_ai.

BINARY	BINARY_CI	BINARY_AI
A	a	ä
Z	A	a
a	Z	A
ä	ä	Z

EXAM OBJECTIVE: AGGREGATE MORE MEANINGFUL STATISTICS OVER A MULTITIER ENVIRONMENT

In multi-tier environments where statements are passed to different sessions by the application server it can become very difficult to race an individual process from start to finish. This is because a request from an end-client is routed to different database sessions by the middle tier. This means that the association between the end-tier client and the database session is non-static.

To meet this need Oracle has introduced **'End to End Application tracing** '. End-to-end tracing is made possible by the introduction of a new attribute the **CLIENT_IDENTIFIER**. This attribute uniquely identifies a given end-tier client and the value of the attribute is carried across all tiers to the database server. Enabling tracing based on the CLIENT_IDENTIFIER solves the problem of debugging performance problems in multi-tier environments. This allows a client process to be identified via the client identifier instead of the session ID, which is available in the V$SESSION view. It is also visible through the system context.

SQL> SELECT SYS_CONTEXT
     ('USERENV','CLIENT_IDENTIFIER')
     FROM DUAL;

In Oracle 10g, statistics aggregation has been implemented using certain additional dimensions.
- **Client identifier**

- **Service Name**
- **Hierarchical combination of service name, module name and action name.**

These additional dimensions make statistics accumulation for multi-tier architectures that use connection pooling or shared-server configuration more meaningful. These statistics are used for the following purposes:

- Performance monitoring of individual clients and services
- Workload management
- Setting threshold-based alerts such as elapsed time and CPU services.

You can use the newly introduced **DBMS_MONITOR** package to control additional tracing and statistics aggregation. After the application is executed, you can use either Enterprise Manager or certain views to obtain the statistical information. The DBMS_MONITOR package contains various procedures that can be used to enable and disable additional statistics aggregation. It gives you the ability to extract performance statistics from dynamic performance views or to activate the SQL trace based on client identifier, service name and module/action.

To manage performance statistics the following procedures are available:

- At a Client level:

CLIENT_ID_STAT_ENABLE and **CLIENT_ID_STAT_ DISABLE**
For example:
DBMS_MONITOR.CLIENT_ID_STAT_ ENABLE (client_id=>'web')

- At a component level

SERV_MOD_ACT_STAT_ENABLE and
SERV_MOD_ACT_STAT_DISABLE
For example:
DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE(service_name => 'SYS$USERS', module_name => 'SQL*Plus')

RETREIVING OUTPUT OF STATISTICS AGGREGATION FROM VIEWS

Performance statistics at a session and service level are always available from the **V$SESSTAT** and **V$SERVICE_STATS** view.

- **V$CLIENT_STATS -** it displays the collected performance statistics at client level. It displays measures for all sessions that are active for the client identifier per instance.
- **V$SERVICE_STATS** – minimal set of performance statistics.
- **V$SERV_MOD_ACT_STATS** - displays collected performance statistics at a component level.
- **V$SVCMETRIC** – continuous metrics for elapsed time and CPU use.

- **DBA_ENABLED_AGGREGATIONS** – displays information about enabled client or components.

EXAM OBJECTIVE: USING SQL TO FLUSH THE BUFFER

In Oracle 10g, there is a new command that can be used to clear the contents of the database buffer cache in the system global area (SGA). However the use of this clause is intended for use only on a test database, since subsequent queries will result in misses if done on a production databases.

The clause is useful if you need to measure the performance of rewritten queries or a suite of queries from identical starting points.
Command:
ALTER SYSTEM FLUSH BUFFER_CACHE;