Clear SQL 6.9 - Code Review Rules

	Group	Title	Description
1	**Program structure**	Use of the NULL statement	Raised when the NULL statement is used in code.
2	**Program structure**	Use of GOTO	The use of GOTO creates spaghetti code that is hard to analyze and debug. A GOTO can jump anywhere to the destination label. This type of design breaks the "one in - one out" ideal of a function creating code which can be impossible to analyze and maintain.
3	**Program structure**	Use of a backward GOTO	The backward GOTO should definitely be replaced by a LOOP statement.
4	**Program structure**	GOTO used in a loop	The use of GOTO creates spaghetti code and should generally be avoided. In many issues the use of GOTO can be replaced with a more structured constructs as conditional and loop logic.
5	**Program structure**	An EXIT statement is used in a FOR loop	A FOR loop assumes the fixed number of execution. The use of EXIT within a FOR loop creates multiple exit points and causes an unstructured termination from the loop. This type of design breaks the "one in - one out" ideal of a loop creating code which can be hard to debug and maintain.
6	**Program structure**	An EXIT statement is used in a WHILE loop	The test for termination of a WHILE loop takes place in the loop boundary. The use of EXIT within a WHILE loop creates multiple exit points and causes an unstructured termination from the loop. This type of design breaks the "one in - one out" ideal of a loop creating code which can be hard to debug and maintain.
7	**Program structure**	A RETURN statement is used in a FOR loop	The RETURN statement completes the execution of a subprogram immediately. The use of RETURN within a FOR loop creates multiple exit points and causes an unstructured termination of a subprogram. Though a subprogram can contain several RETURN statements, this is a poor programming practice.
8	**Program structure**	A RETURN statement is used in a WHILE loop	The RETURN statement completes the execution of a subprogram immediately. The use of RETURN within a WHILE loop creates multiple exit points and causes an unstructured termination of a subprogram. Though a subprogram can contain several RETURN statements, this is a poor programming practice.
9	**Program structure**	A RETURN statement is used in a PROCEDURE	The RETURN statement completes the execution of a subprogram immediately. A RETURN statement been used in a PROCEDURE simply returns control to the caller before the normal end of the procedure is reached. This type of design creates multiple exit points and causes an unstructured termination of a subprogram.
10	**Program structure**	FUNCTION does not contain a RETURN statement	A function must have at least one RETURN statement in its execution section of statements. If none of RETURN statements is executed ORACLE will raises an error 'ORA-06503: PL/SQL: Function returned without value' at run-time. The best way to avoid this error is to place the RETURN statement containing an expression as the last executable statement in the function's body. On the other hand, multiple RETURNs can make code knotty and unreadable so avoid multiple exit paths from the function as well.

Clear SQL 6.9 - Code Review Rules

	Group	Title	Description
11	**Program structure**	FUNCTION with more than one RETURN statement in the executable section	Though a function can contain several RETURN statements, this is a poor programming practice, because multiple RETURNs can make code knotty and unreadable.
12	**Program structure**	Presence of more than one exit point from a loop	Multiple exit points cause an unstructured termination from the loop. This type of design breaks the "one in - one out" ideal of a loop creating code which can be hard to debug and maintain.
13	**Program structure**	FUNCTION has no parameter	A function without any parameters is not as reusable as it could be. On the other hand, these functions refer to global variables for data and thus create hidden dependencies or side effects. It breaks the encapsulation rules and indicates poor modular design.
14	**Program structure**	PROCEDURE has no parameter	A procedure without any parameters is not as reusable as it could be. On the other hand, these procedures refer to global variables for data and thus create hidden dependencies or side effects. It breaks the encapsulation rules and indicates poor modular design.
15	**Program structure**	FUNCTION has OUT parameter	A function should return all of its data through its RETURN clause. If function refers to global variables for data exchange through its OUT or IN OUT parameters, then it creates hidden dependencies or side effects. A function with an OUT argument cannot be called from within a SQL statement as well.
16	**Program structure**	Mode of parameter is not specified with IN parameter	The parameter is considered an IN parameter if a parameter mode is not specified. It is recommended to explicitly declare the parameter mode to keep code more readable and self-documented.
17	**Program structure**	This definition hides another one	The program object has already been declared in a higher scope. This object becomes temporarily hidden by the appearance of a duplicate identifier, and any references to it in the current scope will reference the latest definition. The first object still exists but the original identifier cannot be used to access it until the scope of the duplicate identifier is ended.
18	**Program structure**	Unreferenced parameter	A parameter was declared in the parameter list, but it is not used inside the subprogram. A code that does not have such dead declarations is easier to maintain, but sometimes there is some reason to keep this dead code in place. In such cases the necessary documentation should be included in the code.
19	**Program structure**	Unreferenced local subprogram	A local subprogram was defined, but it is not used inside the package. A package that does not have such dead code is easier to maintain, but sometimes there is some reason to keep this dead code in place. In such cases the necessary documentation should be included in the code.
20	**Program structure**	Unreferenced loop index	A loop index variable is not used inside the loop. While this is not necessarily a problem, sometimes there is some reason to refer to loop index variable (in a cursor FOR loop, for example).
21	**Program structure**	LOOP index redeclared	The loop index is implicitly declared at run-time and should never be declared explicitly by the programmer. The scope of this loop index variable is restricted to the body of the loop. When a loop index variable is redeclared, a completely separate variable is declared with block (not loop!) scope. It can be used outside the loop and can be confused with the loop''s index variable.

Clear SQL 6.9 - Code Review Rules

	Group	Title	Description
22	**Program structure**	Stored program calls itself recursively	If the stored program's identifier is used within the stored program's block, the stored program is executed recursively. Recursion is supported by PL/SQL engine, but each recursive (self-calling) stored program should be carefully analyzed to make sure that recursion is valid and efficient.
23	**Program structure**	CASE without ELSE (default) section	A CASE statement must always have a default condition or this logic construct is non-deterministic. Generally the default condition should warn the user of an anomalous condition which was not anticipated by the programmer.
24	**Program structure**	Last statement in function must be a RETURN	Last statement in function must be a RETURN, otherwise you supply to production some amount of an unreachable, "dead" code in the function body.
25	**Program structure**	FUNCTION exception handler does not contain a RETURN statement	A function must return a value unless it propagates an exception unhandled. For a function, this means that an exception handler should also issue a RETURN statement unless it is re-raising an exception.
26	**Program structure**	Exception masked by a NULL statement	The NULL statement appears to be the only statement in the exception handler. This statement will simply handle the corresponded exception within a block without any recovery action.
27	**Program structure**	Global public variables defined in package specification	Avoid unnecessary visibility. Objects visible within package specifications can be modified by any program unit that has visibility to them. These object cannot be protected or represented abstractly with Get/Set subroutines to provide controlled access to them. Objects whose value depends on program units external to their enclosing package are probably either in the wrong package or are better accessed by a subprogram specified in the package specification.
28	**Program structure**	Exception is not handled inside unit	If you can predict that a certain error can occur in a subroutine, you should create a handled block of code with appropriated exception handler to protect against undesired propagation outside the abstraction. If this exception will propagate out, the subroutine would be unpredictable and hard to integrate in the application, because exception handlers must be coded in the caller's code.
29	**Program structure**	Raise of a predefined exception	It is possible, but not recommended to rise a predefined exception. In an exception handler, it is very difficult to determine exactly which statement and which operation within that statement raised an exception, particularly the predefined exceptions. The predefined exceptions are candidates for propagation to higher abstraction levels for handling there. User-defined exceptions, being more closely associated with the application, are better candidates for recovery within small blocks of code by associated handlers.
30	**Program structure**	An exception is raised and handled in the same scope	When only fault handling code is included in exception handlers, the separation makes the code easier to read. The reader can skip all the exception handlers and still understand the normal flow of control of the code. For this reason, exceptions should never be raised and handled within the same scope, as a form of a goto statement to exit from a loop, if, case, or block statement.

Clear SQL 6.9 - Code Review Rules

	Group	Title	Description
31	**Program structure**	DELETE or UPDATE without WHERE clause	The DELETE and UPDATE statements without a WHERE clause are perfectly valid but have global impact on the referenced tables (all rows will be affected). You should investigate such statements closely to make sure that they are written as required.
32	**Program structure**	Place default parameters at the end of the formal prameter list	Placing default parameters at the end of the formal parameter list allows the caller to use positional association on the call. Otherwise, defaults are available only when named association is used.
33	**Program structure**	The initialization section of the package body contains a RETURN statement	It is possible to use a RETURN statement in an initialization section of the package body and the initialization will be terminated immediately. You should not do this because it results in unstructured code that is hard to debug and maintain, moreover, some amount of section code remain unreachable.
34	**Program structure**	Dynamic SQL is used	Raised when DBMS_SQL.PARSE call, EXECUTE IMMEDIATE statement or OPEN FOR statement with dynamic expression is used.
35	**Program structure**	COMMIT and ROLLBACK are allowed if pragma is used	Raised when a COMMIT or ROLLBACK statement used in the stored program that doesn't contain PRAGMA AUTONOMOUS_TRANSACTION statement.
36	**Program structure**	DROP statement is used	Raised when the script contains a DROP statement.
37	**Program structure**	RULE optimizer hint is used	Raised when the SQL statement contains the RULE optimizer hint
38	**Program structure**	PARALLEL optimizer hint is used	Raised when the SQL statement contains the PARALLEL optimizer hint
39	**Program structure**	Exception is not handled in the BEGIN/END block	Raised when the BEGIN/END block doesn't contain the exception handler part
40	**Program structure**	An opened cursor must be closed	Raised when the subprogram doesn't contain CLOSE statement for the cursor that was previously opened.
41	**Program structure**	Avoid use of an explicit cursor	The implicit cursor is more concise to write than explicit and does all the appropriate checking, opening and closing for you. Implicit cursors will run FASTER than explicit since typically you are using less code to achieve the same task. Equivalent implicit cursors are faster and much easier to code. When the number of rows returned by query is small (around 100), then an explicit cursor can be avoided safely since in those cases using an explicit cursor is performance degrading.
42	**Program structure**	Unreferenced local variable	A local variable has been defined, but it is not used inside a subprogram. A subprogram that does not have such dead code is easier to maintain, but sometimes there may be a reason to keep this code in place. For example, if it is used in a string literal of a dynamic SQL statement. In such cases it cannot be tracked by the analyzer and the necessary documentation should be included in the code.
43	**Program structure**	Unreferenced package/type method or standalone subprogram	A package/type method or standalone subprogram was defined, but it is not used inside the project. A project that does not have such dead code is easier to maintain, but sometimes there may be a reason to keep this code in place. For example, if it is used in a string literal of a dynamic SQL statement. In such cases the necessary documentation should be included in the code.

Clear SQL 6.9 - Code Review Rules

	Group	Title	Description
44	**Program structure**	Identifier redeclared	All declared identifiers must be unique within the same scope. Variables, constants, and parameters cannot share the same name even if they have different datatypes.
45	**Program structure**	Trigger may potentially cause mutating table error	Mutating table exceptions occur when trying to query or modify a table from a row-level trigger that the triggering statement is modifying. You can avoid the mutating table error in either of these ways: use a compound trigger and a collection; use a temporary table or a collection in the package variable and a combination of row-level and statement-level triggers; etc.
46	**Program structure**	Raise of a user-defined exception with the RAISE statement	Avoid raising a programmer-defined exception with the RAISE statement because there is no way to provide an explanation in an error message. Use the RAISE_APPLICATION_ERROR procedure because it associates a user-defined exception with a user-defined message.

1	**Readability**	Initialization to NULL is superfluous	Variables are initialized to NULL by default. Source code may be more readable without the superfluous initialization.
2	**Readability**	END of program unit or package is not labeled	The names on the END of program unit or package ensures consistency throughout the code.
3	**Readability**	END of labeled block should also be labeled	Repeating label names on the END of the labeled blocks ensures consistency throughout the code. In addition, the named END provides a reference for the reader if the block contains the nested blocks.
4	**Readability**	END of labeled LOOP should also be labeled	Repeating label names on the END of the labeled LOOPs ensures consistency throughout the code. In addition, the named END provides a reference for the reader if the LOOP contains the nested loops. Without the END LOOP label, it can be very difficult to keep track of which LOOP goes with which END LOOP.
5	**Readability**	The label near the END of the block doesn't match the block label, or a block label is missing	Source code must be easily read. Though ORACLE allows any label to be associated with the END statement, it should match the start label.
6	**Readability**	The label near the END of the loop doesn't match the loop label, or a loop label is missing	Source code must be easily read. Though ORACLE allows any label to be associated with the END statement, it should match the start label.
7	**Readability**	An EXIT statement in a labeled loop should have the loop's label	An EXIT label helps readers find an associated loop and increases control over the execution of the loops. This is especially true if the loop contains the nested loops.
8	**Readability**	Nested loops should all be labeled	The labels in nested loops improve readability and increase control over the execution of the loops. In addition, labels allow to use dot notation to refer to loop-related variables.
9	**Readability**	IF..THEN..EXIT should be replaced by EXIT WHEN	The EXIT-WHEN statement is easier to read and understand in comparison with the simple IF statement.
10	**Readability**	END of CASE statement should also be labeled	Repeating label names on the END of the CASE statements ensures consistency throughout the code.

Clear SQL 6.9 - Code Review Rules

	Group	Title	Description
11	**Readability**	Elements in the SELECT list (columns/expressions) are not qualified by a table/view name	Elements in the SELECT list (columns/expressions) should be qualified by their scope (name of procedure or function, label name for an anonymous block) or by a table/view name. That way you can be sure that there is no ambiguity between SQL and PL/SQL identifiers. Note that SQL always takes precedence over PL/SQL when resolving identifiers.
12	**Readability**	The column alias name is not specified after the AS keyword	Raised when the column name is not specified in the column list of the SQL statement after the AS keyword.
13	**Readability**	Complex expression is not fully parenthesized	The common precedence rules in PL/SQL make many parentheses unnecessary. When a complex expression occurs, however, it may be helpful to add parentheses even when the precedence rules apply.
14	**Readability**	The alias is missing for a table reference in a multi-source query	Raised when the alias is not specified after a table reference clause in the FROM clause of a query, except for single-table queries and references to the DUAL table. A table alias clarifies which table you are referring to in a query and improves the readability of a SELECT statement.

1	**Maintainability**	Local program unit reference to an external variable	Local program unit tied to particular variable in the program. It breaks the encapsulation rules and indicates poor modular design. These units are difficult to maintain and are not as reusable as they could be.
2	**Maintainability**	Cursor reference to an external variable (use a parameter)	Using parameters makes a cursor more reusable.
3	**Maintainability**	Positional parameters used instead of named parameters	Calls of stored programs with many formal parameters can be difficult to understand without referring to the subprogram's interface. Named association allows stored programs to have new parameters inserted with minimal changes of existing calls.
4	**Maintainability**	Specify a full column list (as opposed to using '*') in each DML statement and cursor	Specify a full column list (as opposed to using '*') in each DML statement and cursor, otherwise your code will not adapt automatically (will fail to compile) to changes in the underlying tables, views, or materialized views listed in the FROM clause.
5	**Maintainability**	The comment percentage of a stored program is less than specified minimum	The degree of commenting within the source code measures the care taken by the programmer to make the source code understandable. Poorly commented code makes the maintainance phase of the software life cycle an extremely expensive one.
6	**Maintainability**	The eLOC within a stored program exceeds the specified maximum	An extremely large stored program is very difficult to maintain and understand. These types of stored programs are generally not well designed and could be broken down into several ones.
7	**Maintainability**	The Interface Complexity metric of a stored program exceeds the specified maximum	Stored programs with a large number of input parameters and return points are difficult to use on a routine basis and are problematic to parameter ordering and exceeding return points break the single entry - single exit design constraint. Stored programs of this type are difficult to debug and maintain.

Clear SQL 6.9 - Code Review Rules

	Group	Title	Description
8	**Maintainability**	The Parameter Complexity metric of a stored program exceeds the specified maximum	Stored programs with a large number of input parameters are difficult to use and are subject to parameter order errors. These types of stored programs are generally not well designed and should be examined for the design constraint for a subprogram to perform a single purpose.
9	**Maintainability**	The Halstead Volume metric of a stored program exceeds the specified maximum	Halstead Volume complexity metrics were developed by Maurice Halstead as a means of determining a quantitative measure of complexity directly from the operators and operands in the module to measure a program module's complexity directly from source code. A value greater than 1000 indicates that the routine probably does too many things.
10	**Maintainability**	The Maintainability Index metric of a stored program is lower than the specified minimum	Maintainability Index is a software metric that measures how maintainable (easy to support and change) the source code is. The Coleman-Oman model is the most used model for determining the Maintainability Index (MI) of a source code. Maintainability Index is based on Halstead Volume, Cyclomatic Complexity, the average number of lines of code per module, and the percentage/ratio (depends on the setting) of comment lines per module. The higher the MI, the more maintainable a system is deemed to be. A value lower than 64 indicates that the routine is probably difficult to maintain.
11	**Maintainability**	The Functional Complexity metric of a stored program exceeds the specified maximum	The stored program's functional complexity is comprised of Interface Complexity and Cyclomatic Complexity. An extremely complex stored program is very difficult to understand and maintain. Stored programs of this type could be broken down into a more modular design of smaller subroutines.
12	**Maintainability**	The Cyclomatic Complexity [v(G) McCabe] metric of a stored program exceeds the specified maximum	Cyclomatic complexity [McCabe] is the degree of logical branching within a stored program. A high degree of V(G) indicates that the stored program could be broken down into a set of smaller stored programs whereby supporting the design concept that a stored program should be specific to one purpose.
13	**Maintainability**	"Magic" hard-coded literal numeric value is used in the WHERE clause of the SQL statement	Raised when a hard-coded literal numeric value is used in the WHERE clause of the SQL statement
14	**Maintainability**	"Magic" hard-coded literal string value is used in the WHERE clause of the SQL statement	Raised when a hard-coded literal string value is used in the WHERE clause of the SQL statement
15	**Maintainability**	SQL*Plus command "SHOW ERRORS" is missing in the script	Raised when there are no "SHOW ERROR" command in the script.
16	**Maintainability**	Mandatory comment header is missing or incorrect in the script	Raised when the script does not contain the comment header defined in Code Analyzer Options or it's incorrect. Header template definition may contain {$ANY_LINE_ENDING} and {$ANY_LINES} wildcards to make the template definition flexible.
17	**Maintainability**	More than one CREATE statement in the script	Raised when the script contains more than one CREATE statement

Clear SQL 6.9 - Code Review Rules

	Group	Title	Description
18	**Maintainability**	The object type doesn't correspond to the file extension	It is convenient to store the code of each single object in a separate file to ensure you can maintain version control independently. This also makes it easier to install a new version of an object while having minimal impact on other schema objects. A distinct file extension helps to identify the type of the object and allows storing the package specification and body in files with different extensions, but same names.
19	**Maintainability**	"Magic" hard-coded literal numeric value is used in PL/SQL code	Do not use a literal value because it has no obvious meaning. Declare a constant to hold a literal value and use it in your code. This helps in maintenance as the human-readable name of the constant explains the meaning and may suggest what it stands for. It also helps to ensure consistency in case the values change. EXCEPTIONS: Because the meaning/purpose of the literal value is obvious in such code, this rule is not regarded as violated in the following cases: - initialization of the standalone subprogram parameter CREATE PROCEDURE DO_IT(PARAM1 NUMBER DEFAULT 1) - simple loop range value when it's 0 or 1 FOR I IN 0..COUNT-1 LOOP FOR I IN 0..1 LOOP - compare results of COUNT and LENGTH functions with 0 COUNT(SOME_TABLE) > 0, LENGTH(STR_VAR) = 0... - increment/decrement by 1 in the assignment statement I := I + 1; I := I - 1;
20	**Maintainability**	"Magic" hard-coded literal string value is used in PL/SQL code	Do not use a literal value because it has no obvious meaning. Declare a constant to hold a literal value and use it in your code. This helps in maintenance as the human-readable name of the constant explains the meaning and may suggest what it stands for. It also helps to ensure consistency in case the values change. EXCEPTION: Because the meaning/purpose of the literal value is obvious in such code, this rule is not regarded as violated in the following case: initialization of the standalone subprogram parameter CREATE PROCEDURE DO_IT(PARAM1 VARCHAR(10) DEFAULT 'DUMMY')

46	Program structure
14	Readability
20	Maintainability
80	**Total**