1. **Introduction**

This document describes rules and recommendations for developing applications using the PL/SQL & SQL Language.

* 1. **Scope**

This document applies to the PL/SQL and SQL language as used within ORACLE databases and tools which access ORACLE databases.

1. **Naming Conventions**
   1. **General Guidelines**

* Do not use names with a leading numeric character.
* Always choose meaningful and specific names.
* Avoid using abbreviations unless the full name is excessively long.
* Avoid long abbreviations. Abbreviations should be shorter than 5 characters.
* Any abbreviations must be widely known and accepted. Create a glossary with all accepted abbreviations.
* Do not use ORACLE reserved words as names. A list of ORACLE’s reserved words may be found in the dictionary view V$RESERVED\_WORDS.
* Avoid adding redundant or meaningless prefixes and suffixes to identifiers. Example: CREATE TABLE EMP\_TABLE.
* Always use one spoken language (e.g. English, German, French) for all objects in your application.
* Always use the same names for elements with the same meaning.
  1. **Naming Conventions for variables**

In general ORACLE is not case sensitive with names. A variable named personname is equal to one named PersonName, as well as to one named PERSONNAME. Some products (e.g. TMDA by Trivadis, APEX, OWB) put each name within double quotes (“) so ORACLE will treat these names to be case sensitive. Using case sensitive variable names force developers to use double quotes for each reference to the variable. Our recommendation is to write all names in lowercase.

A widely used convention is to follow {prefix\_} variablecontent\_{\_suffix} pattern. The following table shows a possible set of naming conventions.

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Prefix / Suffix** | **Example** |
| Global Variable | P: g | g\_version |
| Local Variable | P: l | l\_version |
| Cursor | P: c | c\_employees |
| Record | P: r | r\_employee |
| Array / Table | P: t | t\_employees |
| Object | P: o | o\_employee |
| Type Definitions | S: type | r\_employee\_type |
| Exception | P: e | e\_employee\_exists |
| Constants | P: co | co\_empno |

* 1. **Database Object Naming Conventions**

Never enclose object names (table names, column names, etc.) in double quotes to enforce mixed case or lower case object names in the data dictionary.

|  |  |
| --- | --- |
| **Identifier** | **Naming Convention** |
| **Collection Type** | * A collection type should include the name of the collected objects in their name. Furthermore they should have the suffix “\_ct” to identify it as a collection.   **Examples:** - employees\_ct  - orders\_ct |
| **Column** | * Singular name of what is stored in the column   o unless the column data type is a collection, then you use a plural name   * Add a comment to the database dictionary for every column. |
| **DML / Instead of Trigger** | Choose a naming convention that includes:  Either   * the name of the object the trigger is added to, * any of the triggering events:   + \_br\_iud  Before Row on Insert, Update and Delete   + \_io\_id  Instead of Insert and Delete   Or   * the name of the object the trigger is added to, * the activity done by the trigger, * the suffix “\_trg”   Examples: - employees\_br\_iud   * orders\_audit\_trg * orders\_journal\_trg |
| **Foreign Key Constraint** | * Table abbreviation followed by referenced table abbreviation followed by a “\_fk” an and optional number suffix. * Optionally prefixed by a project abbreviation.   Examples: - empl\_dept\_fk  - sct\_icmd\_ic\_fk1 |
| **Function** | * Name is built from a verb followed by a noun. The name of the function should answer the question “What is the outcome of the function?”   Examples: get\_employee   * If more than one function provides the same outcome, you have to be more specific with the name.   Examples: get\_employee\_by\_name  get\_employee\_by\_email get\_employee\_by\_phone\_no |
| **Index** | * Indexes serving a constraint (primary, unique or foreign key) are named accordingly. * Other indexes should have the name of the table and columns (or their purpose) in their name and should also have \_idx as a suffix. |
| **Object Type** | * The name of an object type is built by its content (singular) followed by an “\_ot” suffix.   Examples: employee\_ot |
| **Package** | * Name is built from the content that is contained within the package.   Examples: - employees\_api (API package for the employee table).  - logging\_up (Utility package including logging support). |
| **Primary Key Constraint** | * Table name or table abbreviation followed by the suffix “\_pk”. * Optionally prefixed by a project abbreviation.   Examples: - employees\_pk   * + departments\_pk   + sct\_contracts\_pk |
| **Procedure** | * Name is built from a verb followed by a noun. The name of the procedure should answer the question “What is done?” Procedures and functions are often named with underscores between words because some editors write all letters in upper case in the object tree (e.g. Quest), so it is difficult to read them.   Examples: - calculate\_salary   * + set\_hiredate   + check\_order\_state |
| **Sequence** | * Name is built from the table name (or its abbreviation) the sequence serves as primary key generator and the suffix \_seq or the purpose of the sequence followed by a \_seq.   Examples: - employees\_seq  - order\_number\_seq |
| **Synonym** | * Synonyms should be used to address an object in a foreign schema rather than to rename an object. Therefore synonyms should share the name with the referenced object. |
| **System Trigger** | * Name of the event the trigger is based on. * Activity done by the trigger. * Suffix “\_trg”.   Examples: - ddl\_audit\_trg  logon\_trg |
| **Table** | * Plural name of what is contained in the table.   o unless the table is designed to always hold one row only – then you should use a singular name   * Optionally prefixed by a project abbreviation. * Add a comment to the database dictionary for every table.   Examples: - employees   * + departments   + sct\_contracts   + sct\_contract\_lines   + sct\_incentive\_modules |
| **Temporary Table**  **(Global Temporary Table)** | * Naming as described for tables. * Optionally suffixed by “\_tmp”   Examples: - employees\_tmp  - contracts\_tmp |
| **Unique Key Constraint** | * Table name or table abbreviation followed by the role of the unique constraint, an “\_uk” and an optional number suffix. * Optionally prefixed by a project abbreviation.   Examples: - employees\_name\_uk   * + departments\_deptno\_uk   + sct\_contracts\_uk   + sct\_coli\_uk   + sct\_icmd\_uk1 |
| **View** | * Plural name of what is contained in the view. * Optionally prefixed by a project abbreviation. * Optionally suffixed by an indicator identifying the object as a view (mostly used, when a 1:1 view layer lies above the table layer) * Add a comment to the database dictionary for every view and every column.   Examples: - active\_orders  - orders\_v (a view to the orders table) |

1. **Coding Style**
   1. **Formatting**

Use IDE Code Formatting

* 1. **Code Commenting**

Inside a program unit only use the line commenting technique “--“.To comment the source code for later document generation, comments like /\*\* … \*/ are used.

1. **Language Usage**
   1. **Variables & Types**
   * Try to use anchored declarations for variables, constants and types.
   * Try to have a single location to define your types. This single type could either be a type specification package or the database (database defined types).
   * Try to use subtypes for constructs used often in your application.
   * Never initialize variables with NULL.
   * Avoid comparisons with null value, consider using IS [NOT] NULL.
   * Avoid initializing variables using functions in the declaration section.
   * Never overload data structure usages.
   * Never use quoted identifiers.
   * Avoid using overly short names for declared or implicitly declared identifiers.
   * Avoid the use of ROWID or UROWID.
   * Avoid declaring NUMBER variables or subtypes with no precision.
   * Try to use PLS\_INTEGER instead of NUMBER for arithmetic operations with integer values (no decimal point).
   * Avoid using CHAR data type.
   * Avoid using VARCHAR data type.
   * Never use zero-length strings to substitute NULL.
   * Always define your VARCHAR2 variables using CHAR SEMANTIC.
   * Try to use boolean data type for values with dual meaning.
   1. **DML and SQL**
   * Always specify the target columns when executing an insert command.



* + Always use table aliases when your SQL statement involves more than one source.



* + Try to use ANSI-join syntax, if supported by your ORACLE version.
  + Try to use anchored records as targets for your cursors.
  + Use BULK OPERATIONS (BULK COLLECT, FORALL) whenever you have to repeatedly execute a DML or SELECT command.
  1. **CURSOR**
  + Always use %NOTFOUND instead of NOT %FOUND to check whether a cursor was successful.



* + Avoid using %NOTFOUND directly after the FETCH when working with BULK OPERATIONS and LIMIT clause. Use [array\_name].COUNT() instead to check whether further FETCHs are needed.
  + Always close locally opened cursors.
  + Avoid procedure or function calls between a SQL operation and an implicit cursor test.
  1. **Flow Control**
  + Never use GOTO statements in your code.
  + Always use a CURSOR FOR loop to process the complete cursor results unless you are using bulk operations.



* + Always use a NUMERIC FOR loop to process a dense array.
  + Use 1 as lower boundary and COUNT() as upper boundary when looping over a dense array
  + Always use a WHILE loop to process a loose array.
  + Avoid using EXIT to stop loop processing unless you are in a basic loop.
  + Always use EXIT WHEN instead of an IF statement to exit from a loop.



* + Try to label your EXIT WHEN statements.
  + Do not use a cursor for loop to check whether a cursor returns data.
  + Avoid use of unreferenced FOR loop indexes.
  + Avoid hard-coded upper or lower bound values with FOR loops.
  1. **Exception Handling**
  + Never handle unnamed exceptions using the error number.



* + Never assign predefined exception names to user defined exceptions.
  + Avoid use of WHEN OTHERS clause in an exception section without any other specific handlers.
  + Avoid use of EXCEPTION\_INIT pragma for a -20,NNN error.
  + Avoid use of the RAISE\_APPLICATION\_ERROR built-in procedure with a hard-coded - 20,NNN error number or hard-coded message.
  + Avoid unhandled exceptions.
  + Avoid using Oracle’s predefined exceptions.
  1. **Dynamic SQL**
  + Always use a string variable to execute dynamic SQL.



* + Try to use output bind arguments in the RETURNING INTO clause of dynamic INSERT, UPDATE, or DELETE statements rather than the USING clause.
  1. **Packages**
  + Try to keep your packages small. Include only few procedures and functions that are used in the same context.
  + Always use forward declaration for private functions and procedures.
  + Avoid declaring global variables public.
  + Avoid using a IN OUT parameters as IN / OUT only.
  1. **Procedures**
  + Avoid standalone procedures – put your procedures in packages.
  + Avoid using RETURN statements in a PROCEDURE.
  1. **Functions**
  + Avoid standalone functions – put your functions in packages.
  + Try to use no more than one RETURN statement within a function.
  + Always make the RETURN statement the last statement of your function.
  + Never use OUT parameters to return values from a function.
  + Never return a NULL value from a BOOLEAN function.