CANOE VENTURES

DATA ARCHITECTURE STANDARDS

Naming Conventions, Diagramming Guidelines and PL/SQL Standards

DRAFT

Author: Derek Reiner

Creation Date: 07-April-2011

Last Updated: N/A

Version Number: 0.01

**Change Record**

| Date | Author | Version | Change |
| --- | --- | --- | --- |
|  |  |  |  |
| 07-Apr-2011 | Derek Reiner | 0.01 | Document Creation |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Reviewers**

| Name | Position |
| --- | --- |
|  |  |
| Domenic Centoranti | VP, Chief Data Architect |
| Joe Moenich | Engineering Director, Enterprise Service Bus |
| Doug Tracy | Senior Oracle Developer |
| Donnie Evans | BI Reporting Analyst/Developer |
| David Parent | Senior Database Administrator |
| Derek Reiner | Data Architect |
|  |  |

**Distribution**

| Copy No | Name | Location |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. The copy numbers referenced above should be written into the **Copy Number** space on the cover of each distributed copy. If the document is not controlled, you can delete this table and the **Copy Number** label from the cover.

Table of Contents

General 1

Introduction 1

Objectives 1

Applicability 1

Authorization 2

Access 2

Document Organization 2

Naming Oracle Objects: General Rules 3

Length of Labels 3

Acronyms/Abbreviations 3

Use of Significant Words 4

Use of Oracle Reserved Words 4

Use of Special Characters 4

Use of Underscores 4

Use of Prefixes 4

Use of Suffixes 5

Analysis 6

Applications 6

Application Short Names 6

Entities 6

Entity Names 6

Table Names 6

Entity Short Name (Alias) 6

Intersection Entities 7

Association Entities 7

Validation Entities 7

Attributes 7

Primary Unique Identifier (UID) Attributes 8

Audit Attributes 8

Unique Identifiers (UIDs) 9

Relationships 9

Processing and Exception Log 10

Data Abstraction Layer (DAL) 10

Data Integrity Layer (DIL) 10

Server Model Objects 11

Tables 11

Table Names 11

Table Aliases 11

Table Comments 11

Data Archive Tables 11

Partitioning 12

Columns 12

Column Names 12

Column Comments 12

Constraints 12

Primary Key Constraints 12

Unique Key Constraints 13

Foreign Key Constraints 13

Check Constraints 13

NOT NULL Constraints 14

Indexes 14

Constraint Indexes 14

Unique Indexes 14

Non-unique Indexes 14

Function-Based Indexes 14

Bit-mapped Indexes 15

Partitioning 15

Sequences 15

Views 15

Materialized Views 16

Other Database Objects 17

Functions 17

Packaged Functions 17

Stand-alone Functions 17

Object Views 17

Packages 18

Procedures 18

Packaged Procedures 18

Stand-alone Procedures 19

SQL files 19

Synonyms 20

Triggers 20

Diagramming Guidelines 21

General 21

Introduction 21

Objectives 21

Applicability 21

Entity Relationship (ER) Diagrams 21

Multiple Modular Diagrams 21

Avoid Overcrowding 21

Display a Legend on the Diagram 21

ER Diagram Layout Convention #1 22

ER Diagram Layout Convention #2 22

ER Diagram Layout Convention #3 22

Consolidating Diagrams 24

Deleting Un-needed Diagrams 24

Drawing Relationships to Sub-types 24

Color Coding Diagrams 24

PL/SQL Standards 25

PL/SQL Program Data 25

Constants 25

Exceptions 25

Record types 25

Index-by table types 25

Nested table types 26

Varray types 26

Ref cursor types 26

Object types 26

Subtypes 26

Cursors 26

Variables 27

Parameters 27

FOR Loop Index Names 28

Bibliography 29

Appendix A – Object Format Chart 30

Appendix B - Approved Acronym List 32

Appendix C - Approved Abbreviation List 33

Appendix D- Application Abbreviation List 34

Appendix E– Oracle Reserved and Key Words 35

Appendix F - PL/SQL Element Format Chart 38

Appendix G – Domain Examples 39

Appendix H – Relationship Names Examples 40

Appendix I – Standard Processing Log Table Structure 42

1. To update the table of contents, select it and press F9.   
   To change the number of levels displayed, click on it, press Shift-F9, change "1-3" to "1-2" or "1-4", press Shift-F9 again, then F9 to update.

## General

### Introduction

This document establishes naming conventions, best practices and guidelines for the Canoe Ventures data teams. This document specifically addresses the logical and physical objects as well as the constructs used in the design, development, implementation and maintenance of Canoe’s data systems. This document has an Oracle focus as our OLTP, ODS and initial EDW databases utilize this RDBMS. As Canoe integrates SAP’s High Performance Analytical Appliance (HANA) this document will be updated to address any information gaps. Whenever possible, HANA development should follow the standards outlined in this document.

### Objectives

Standardization is critical to the successful achievement of quality in the system engineering process. The standards defined in this document are intended as a communication tool that allows Canoe-Ventures data teams to work together in a cohesive manner. Having established standards:

* Provides consistency across all projects
* Eliminates guesswork on the part of new team members as to how things are supposed to be done
* Improves the maintainability of models and systems over time (i.e. less guesswork)
* Reduces long-term costs (because systems are easier to maintain)
* Promotes reusability

It is expected that these standards will continue to evolve to support the changing needs of Canoe-Ventures.

Future standards will focus on expanding the contents to cover the following areas:

* Columnar Database Design Strategy
* SAP Business Objects Universe Design Strategy
* BI Best Practices
* ETL Best Practices with a focus on SAP Data Services
* Business Processes and Functions

**Legacy objects will not be changed**. However, objects created after the approval date for these standards will conform to the standards herein.

### Applicability

These standards apply to all database systems designed, developed, implemented or maintained by Canoe-Ventures.

### Authorization

The Data Architects (DA) are responsible for the establishment of these standards. The DAs will control all future additions and amendments to these standards.

Changes to these standards will go through the following process:

1. Requests for change(s) must be submitted, in writing, to the DAs.
2. The DAs will review the requested changes, and either accept or reject them. If the requested changes are rejected, the DAs will inform the requestor and tell them the reasons for rejection.
3. If the changes are accepted, the standards document will be updated, and all of the developers will be informed that the standards have changed and given a brief summary of the changes. This step can be done on a periodic basis if there are many changes in progress.

Deviations from the standards must be reviewed and approved by the DAs prior to implementation.

### Access

A copy of these standards can be located on the ESB SharePoint site located on Canoe-Ventures Intranet. Alternatively, a hard copy is available from the DAs.

### Document Organization

This standards document is organized into eight distinct sections:

* Naming Oracle Objects: General Rules
* Analysis
* Server Model Objects
* Other Database Objects
* Diagramming Guidelines
* PL/SQL Standards
* Bibliography
* Appendices

## Naming Oracle Objects: General Rules

All rules established in this section must be followed when naming objects, unless **specifically** accepted by individual object descriptions in later sections.

The individual sections contain any additional rules specific to an object and one or more examples to illustrate usage.

### Length of Labels

Oracle database object names are restricted to no more than 30 characters and include logical objects (e.g., entities), schema objects (e.g., tables) and object parts (e.g., attributes). However, some Oracle Designer utilities and Data Abstraction Layer Generators will add a suffix of 5 additional characters for example ‘\_FK01’. For this reason, the recommended maximum length is 25 characters. Names should be meaningful, and should accurately describe the object to which they are assigned. The consistent use of abbreviations and standard acronyms will assist in this endeavor.

Length restrictions (where appropriate) are detailed by object in **Appendix A - Object Format Chart**. These specifications are the basis of all object names and must be followed. These limits are established to promote consistency and to ensure that all objects are of appropriate length.

### Acronyms/Abbreviations

Full words should always be used, unless an approved acronym can be substituted or an abbreviation is required for length considerations. In other words, always substitute an approved acronym if an approved one exists, and only abbreviate when necessary because of name length considerations. (An acronym is a word formed from the initial letter or letters of each of the successive or major parts of a compound term.) . A list of approved acronyms is provided in **Appendix B – Approved Acronym List**.

If name length considerations mandate abbreviation of words, use an approved abbreviation. A list of approved abbreviations is provided in **Appendix C – Approved Abbreviation List**. If, after all approved acronyms and abbreviations have been substituted, it is still necessary to reduce the length of a label begin the abbreviation process from the right. Start with the last word in the label and continue to reduce one word at a time until the total length of the label is acceptable. Apply the following techniques in the order given to create a unique abbreviation of appropriate length.

* Remove internal vowels and final ‘e’

Example: country is abbreviated ‘cntry’, object is abbreviated ‘objct’.

* Sequentially remove consonants beginning at the end of the word

Example: address is abbreviated ‘addr’, abbreviation is abbreviated ‘abbr’.

An abbreviation should be no less than three and no more than five characters. In some special instances, an abbreviation of 2 characters is allowable if the meaning of the abbreviation is clear and/or it is in common usage such as ‘oz’, ‘ft’, or ‘lb’. If the qualifying word is four or fewer characters long, it should not be abbreviated.

If further assistance is required, contact the DAs to determine the acronym(s) or abbreviation(s) required. The DAs should be informed of any new acronyms or abbreviations for the lists of approved acronyms and abbreviations to be updated.

### Use of Significant Words

Use root words wherever possible. Dropping the suffix (-age, -ence, -ance, -ing, -ant, -ity, -any, -ive, -ary, -ony, -aty, -ory, -ation, -ment, -ed, -tion) will generally leave the root word. Do this only if the remaining root word is meaningful as is.

Words such as who, what, when, or where are not allowed.

The use of articles and prepositions (such as **the** or **of** ), adjoining words or conjunctions (such as **and** or **or**), qualifying words such as **new** or **old**, and numbers should be on an exception basis only.

### Use of Oracle Reserved Words

A name cannot be an Oracle reserved word. A list of Oracle Reserved words are provided in **Appendix D – Oracle Reserved Words.**

Oracle reserved words have a special syntactic meaning to Oracle or the PL/SQL language so should not be used for schema or program object names.

Additionally, the names of SQL built-in functions should not be used for schema or program object names.

### Use of Special Characters

Special characters, including brackets, quotation marks, question marks, dollar signs ($), hash signs (#) and slashes are not permitted. Dashes are allowed only in legitimately hyphenated words.

### Use of Underscores

When an Oracle object is viewed in the Oracle data dictionary it will be in all uppercase letters regardless of the case that was typed in the CREATE statement. The problem with all uppercase names is that when more than one word exists in a name the words tend to run together.

Underscores will only be used to separate words (or acronyms) or separate prefixes and suffixes from the base names of schema objects, object parts or PL/SQL elements.

### Use of Prefixes – Fact and Dimension Tables

Fact tables should begin with FACT\_.

Dimensional tables should begin with DIM\_.

### Use of Prefixes – PL/SQL

Prefixes should be used to identify PL/SQL elements.

Why use prefixes? There is a technical reason for the use of prefixes. In PL/SQL it is possible to compile code that is unpredictable and still within the syntax rules of the language. Consider the following program block;

FUNCTION title (isbn IN varchar2)

IS

title varchar2(200);

BEGIN

SELECT title

INTO title

FROM books

WHERE isbn = isbn;

RETURN title;

END title;

This code will compile without error but would not work as intended. The parameter name *isbn* matches exactly the *isbn* column in the cursor **where** clause. Instead of returning the title for the *isbn* that is passed in, the function will raise an error because the cursor will try to return all rows in the table. The *isbn* number will actually match itself so it will always be true.

Each of the identifiers in the above example is using the best name possible for its purpose. What better name is there for a variable to hold the title than 'title'? The problem is that PL/SQL has one big namespace for everything and then it uses a 'Name-Resolution Algorithm' to sort it all out. The *isbn* column name takes precedence over the *isbn* parameter in this case.

Look at the same function again but this time with prefixes added to break up the ambiguity:

FUNCTION title (pi\_isbn IN varchar2)

IS

v\_title varchar2(200);

BEGIN

SELECT title

INTO v\_title

FROM books

WHERE isbn = pi\_isbn;

RETURN v\_title;

END title;

By adding the p\_ prefix for parameters and a v\_ prefix for variables the same function now works fine. The prefixes, in effect, create different namespaces for each element so that the same base name can be used without ambiguity. This example shows that using a prefixing system in PL/SQL has some very real technical benefits.

In selecting prefixes for PL/SQL elements, these standards use the shortest possible prefix and use consistent logic throughout.

A list of PL/SQL element prefixes is provided in **Appendix E - PL/SQL Element Format Chart.**

### Use of Suffixes

The purpose of using an object type identifier suffix in an object name is to eliminate conflicts with other objects that are based on the same entity. Each object must have a unique name within the database regardless of its type.

In selecting suffixes for object types, these standards use the shortest possible suffix and use consistent logic throughout. The suffixes use the standard abbreviation of the object type.

A list of object type suffixes is provided in **Appendix A – Object Format Chart.**

## Analysis

### Applications

#### Application Short Names

* Application short names should have a minimum of 2 characters and a maximum of 5.

**Example**: LOC is an application short name for the LOCATOR application.

### Entities

#### Entity Names

Entity names should be singular nouns or nominative phrases. Single or multiple full-word descriptive names may be used, but allow exceptions for approved acronyms and/or abbreviations (e.g., CSAT or ADDR) when the name exceeds 25 characters. Multiple-word entity names should contain one blank space between each word.

Entity names should not contain the names of *physical* constructs such as ‘file’ or ‘table’ as a qualifier.

**Example:** Use *EMAIL* not *EMAIL FILE or EMAIL TABLE*.

1. Entity names should have a minimum of 3 characters and a maximum of 25. The actual maximum length of entity names is governed by the entity plural.

**Example:** VESSEL NAME

#### Table Names

Table names are the singular form of the entity name.

The singular for each entity should be defined at the time an entity is named since this will be used to create the table name. Some utilities impose a restriction on the length of the entity singular name.

* Entity singular names should have a minimum of 4 characters and a maximum of 25 and should be prefixed with the short schema name.

**Example:** SHP\_VESSEL

#### Entity Short Name (Alias)

An entity short name is composed of a distinct word or words (10 characters or less) or a concatenation of entity word fragments. Many tools use the entity short name in the generation of names for constraints, foreign keys and sequences. A user should be able to look at an entity short name and know to which entity it refers. Since the entity short name may be used to create any migrated foreign-key column name, it is important that the short name indicates the entity from which it came. Use an approved abbreviation if one exists.

* Entity short names should have a minimum of 3 characters and a maximum of 10.

If an entity name consists of one word, the short name should be the first 3 to 10 characters, or it can be an approved acronym or abbreviation. If the entity name consists of two or more words, the short name should be the first letter of each of the words in the name not to exceed 6 characters, or each word can be an approved acronym or abbreviation with no space between them.

**Examples:** CUST is the short name for the entity CUSTOMER.

PHONE\_NUM is a possible short name for the entity PHONE NUMBER. PN will be the short name automatically created.

#### Intersection Entities

An intersection entity is used to resolve a many-to-many relationship between two different entities. It is named from the names of the entities with which it is associated.

**Example:** If entity one is named ORDER and entity two is named ITEM, an intersection entity may be created called ORDER ITEM.

#### Association Entities

An association entity is used to resolve a many-to-many **recursive** relationship between an entity and itself. It is named for either the entity it represents or it’s named after the parent entity with the word ASSOCIATION appended.

**Example 1:** CUSTOMER\_UPLINE

**Example 2:** CUSTOMER\_ASSOCIATION

#### Validation Entities

A validation entity is used for ‘lookup’ or code information. Validation entities should be suffixed with a blank space and the word ‘TYPE’ to distinguish them from other types of entities.

**Example:** SHP\_VESSEL TYPE

### Attributes

An attribute is any detail, which serves to identify, qualify, quantify, classify, express the state of, or otherwise describe properties of an entity. Each occurrence of an attribute within an entity has one and only one value.

Attributes can be classified by functionality and depicted on Entity Relationship Diagrams with the following symbols:

|  |  |
| --- | --- |
| **Symbol** | **Description** |
| # | Primary Unique Identifier (UID) |
| \* | Mandatory Attribute (may not be null) |
| O | Optional Attribute (may be null) |

Since attributes are always shown in the context of an owning entity, do not repeat the name of the entity as part of the attribute name, unless it participates in the unique identifier (UID). Attributes can have multi-part names and should have underscores between the words. Attribute names should be as descriptive as possible without being overly convoluted. They should be named using terms an end-user will recognize as much as possible. If necessary, abbreviations and acronyms may be used to meet length requirements.

* Attribute names should have a minimum of 3 characters.

**Examples:** Start Date

End Date

Description

#### Primary Unique Identifier (UID) Attributes

When naming an attribute that participates in the UID, use the entity name (if it is relatively short) or the entity short name plus one of the following words as a suffix:

**SID** is used for a UID that contains a system-generated number and is used internally during processing to access or link data. The values are meaningful only as pointers or keys, and usually do not show on end user reports or screens.

**NUMBER** is used for a UID that contains a number by which the end user recognizes the entity. NUMBERs **are** displayed on reports or screens to provide information to an end user. The number may be system generated but is usually determined by the user.

**CODE** is used for a UID that contains an alphanumeric value by which the end user recognizes the entity. CODEs are displayed on reports or screens to provide information to an end user.

**TYPE** is used for an attribute that contains an alphanumeric value to represent a categorization of an entity (i.e., a sub-type).

**Examples: CUSTOMER SID:** If the entity CUSTOMER has a system generated internal use UID, the attribute will be designated **CUSTOMER SID.**

**CUSTOMER NUMBER:** If the entity CUSTOMER has a system generated or user generated UID which is significant to the end user, the attribute will be designated **CUSTOMER NUMBER**.

**CUSTOMER CODE:** If the entity CUSTOMER were identified by an alphanumeric UID (e.g., GRAZ1), it would be called **CUSTOMER CODE.**

**CUSTOMER TYPE:** There may be several categories for Customers, such as commercial or private. Each grouping is identified by an alphanumeric value called **CUSTOMER TYPE**.

Except for these specific rules, attribute names should be determined by the same rules used for entities.

#### Audit Attributes

Audit capabilities are strongly recommended in all systems developed by Canoe-Ventures. Every entity should contain the following six audit attributes:

APP\_CREATED VARCHAR2(30) NOT NULL

DATE\_CREATED DATE NOT NULL DEFAULT SYSDATE

USER\_CREATED VARCHAR2(30) NOT NULL DEFAULT USER

APP\_MODIFIED VARCHAR2(30)

DATE\_MODIFIED DATE

USER\_MODIFIED VARCHAR2(30)

These audit attributes are to be detailed in the logical models for all entities. When tables and columns are later generated the appropriate columns are created.

### Unique Identifiers (UIDs)

Every entity must have at least one unique identifier that will eventually serve as the primary key. This will ensure the uniqueness of the data to be entered. It is also possible for an entity to have additional unique identifiers to help define the meaning of the data in a table.

<e*ntity name*>\_uk<*n*>

* Unique identifier names should have a minimum of 5 characters and a maximum of 12.

Where <*n*> is the distinguishing key, starting from 01.

**Example:** For the entity PERSON the primary UID will be named PERSON\_UK01.

A secondary UID on the attribute NAME should then be named PERSON\_UK02.

### Relationships

Relationships should be named so that the entity-relationship diagram can be easily read. To read any relationship simply and definitively, the following syntax is used:

*Each (and every) ENTITY-A {must be| may be} relationship-name {one and only one ENTITY-B (singular) | one or more ENTITY-B (plural)}*

Where *ENTITY-A* is the source entity of the relationship, *ENTITY-B* is the destination end of the relationship, and *relationship-name* is the name applied to the relationship in the direction the relationship is being read. Note the following rules:

* The choice between *must be* and *may be* is determined by the modality (optionality) of the relationship emanating from the source entity. A solid line represents *must be* (mandatory) and a dashed line represents *may be* (optional)*.*
* The choice between one and only one *ENTITY-B (singular)* and one or more *ENTITY-B plural*is determined, respectively, by the absence or presence of the ‘crows feet’ at the ENTITY-B end. This is also known as the cardinality of the relationship.

Since relationships are always bi-directional, naming a relationship requires two relationship names be supplied. Hence, the relationship must be readable using the above syntactic structure in both directions.

**Examples:** Each (and every) PERSON *may be* **located at** *one or more* ADDRESSes

Each (and every) ADDRESS *must be* **the location for** *one and only one* PERSON

Each (and every) DEPARTMENT *may be* **responsible for** *one or more* EMPLOYEEs

Each (and every) EMPLOYEE *must be* **assigned to** *one and only one* DEPARTMENT

Whenever multiple relationships are created into a single entity, from the same parent entity, the relationship labels **must be** unique to prevent duplicate index names from being created.

Do not use weak relationship names, such as ‘associated with’, or ‘related to’. Remember that relationships are documenting business rules and will be used to explain the model to end-users. Whenever possible, use business terminology. For example, the correct way to document a person and their address would be:

A PERSON *may be* **located at** one or more ADDRESSes, ***not*** a PERSON *may be* **associated with** one or more ADDRESSes.

* Relationship names should have a maximum of 60 characters.

A list of relationship name examples is provided in **Appendix G – Relationship Names Examples**.

### Processing and Exception Log

For all new application development, a standard area should be defined for logging process activity and unexpected exceptions. This table should be named as follows:

<schema\_prefix>\_PROCESSING\_LOG

See Appendix I for the definition for this table, and a sample format.

### Data Abstraction Layer (DAL)

For all new application development, an area should be defined that abstracts the end applications from the physical structures of the table for all selects, inserts, updates and deletes. This should be created at the lowest level of application interaction, meaning that if there are PL/SQL procedures, they should call the exact same abstraction layer that Java programs execute. This abstraction layer should be able handle calls from many different types of applications (through wrappers), and should NOT include any business logic specific code. Modifications to the generated code are not allowed.  If modifications are required, this code belongs in the Data Integrity Layer or an additional PL/SQL wrapper.  This allows the DAL to be easily re-generated for later table changes.

### Data Integrity Layer (DIL)

For all new application development, an area should be defined that performs logical, atomic actions. For example, the creation of a new piece of equipment would have the following steps:

1. Verify that the equipment being created needs to be created.
2. Insert into the equipment table.
3. Insert the appropriate values into the equipment status table.

None of this makes sense to perform without all of it being done at the same time, inside the same logical transaction. This could be defined as entity creation or manipulation logic. This could be defined in either Java or PL/SQL, but should still be defined in the lowest layer.

## Server Model Objects

This section describes naming conventions for objects that will normally be encountered or defined in the Design phase of development.

### Tables

#### Table Names

The naming convention for table names is:

<e*ntity singular*>

Table names should be the singular of entity names, with the spaces translated to underscores. Since tables are the primary objects in an Oracle database, table names will not be given an object type identifier suffix like other objects.

* Table names should have a minimum of 4 characters and a maximum of 25.

**Example:** INVOICE

#### Table Aliases

The naming convention for table aliases is:

<e*ntity short name*>

All tables must have an alias. The alias must conform to the same standard set forth for entity short names.

* Table aliases should have a minimum of 3 characters and a maximum of 10.

#### Table Comments

To ensure that appropriate meta-data is included in the Oracle data dictionary, all tables must have a comment. Comments can be entered with a DDL SQL COMMENT statement or via a third-party Oracle tool (e.g. Toad). This comment should describe the basic information stored in the table.

#### Data Archive Tables

Data archiving, or staging, is the process of copying (and purging) of data from a production database to another database or media. Tables used during the data archiving process should be identified as distinct from their production counterparts.

The naming convention for data archive tables is:

<*table name*>\_AUD

-or-

<*table name*>\_HIST

-or-

<*table name*>\_TXN

-or-

<*table name*>\_STG

Where <*table name*> is the production table being (data) archived.

* Data archive tables should have a minimum of 8 characters.

**Example:** CUSTOMER\_HIST

#### Partitioning

Partitioning can be used when a special need arises or, in the interest of table maintenance, it is determined by the Data Architect and the Database Administrator that partitioning would be beneficial.

The naming convention for partitioned tables is:

<*table name*>\_HPARTXX or \_HPXX for hash partitions

-or-

<*table name*>\_RPARTXX or \_RPXX for range partitions

-or-

<*table name*>\_LPARTXX or \_LPXX for list partitions

-or-

<*table name*>\_PYYYYMM for automated partition maintenance scripts

**Example:** CUSTOMER\_HPART01 or CUSTOMER\_HP01

### Columns

#### Column Names

A column name must be the same as the name of the attribute from which the column was mapped, with the spaces translated to underscores. The column name must not be a plural. If the resulting name has more than 30 characters, use the approved acronyms and abbreviations in the appendices to shorten it.

If the column is not based on an attribute, then it should be named using the naming standard set forth for attributes with the exception that an underscore is used instead of a space between segments.

Column names will not be prefixed with the table short name.

Columns in the same table or view cannot have the same name. However, columns in different tables or views can have the same name.

#### Column Comments

To ensure that appropriate meta-data is included in the Oracle data dictionary, all columns must have a comment. Comments can be entered with a DDL SQL COMMENT statement or via a third-party Oracle tool (e.g. Toad). This comment should describe the basic information stored in the column.

### Constraints

#### Primary Key Constraints

The naming convention for primary key constraints is:

<*Table name*>\_PK

* Primary key constraint names should have a minimum of 6 characters and a maximum of 30.

**Example:** For the table CUSTOMER with the alias CUST the primary key constraint will be named:

CUSTOMER\_PK

#### Unique Key Constraints

The naming convention for unique key constraints is:

<*table name*>\_UK<*n*>

Where <*n*> is the distinguishing key, starting from 01.

* Unique key constraint names should have a minimum of 6 characters and a maximum of 30.

**Example:** The table called INVOICE (INV) has two unique identifiers. The unique key constraints will be named:

INVOICE\_UK01

INVOICE\_UK02

#### Foreign Key Constraints

The naming convention for foreign key constraints is:

<*table name of the originating key*>\_FK<*n*>

**Example:** INVOICE\_LINE (INVOICE\_ID) >------------- INVOICE (INVOICE\_ID)

The foreign key constraint on INVOICE\_LINE will have the foreign key constraint name generated as INVOICE\_LINE\_FK01.

* Foreign key constraint names should have a minimum of 6 characters and a maximum of 30.

**Example:**

shipped from

INVOICE >----------------- LOCATION

INVOICE >----------------- LOCATION

returned to

The foreign key constraints on INVOICE will have the foreign key constraint names generated as:

INVOICE\_FK01

INVOICE\_FK02

#### Check Constraints

This standard applies to table level check constraints. The naming convention for check constraints is:

<*table name*>\_*CK<n>*

**Example:** The table called LOCATION\_HISTORY has a check constraint to enforce a list of values. The check constraint will be named:

LOCATION\_HISTORY\_CK01

**Example:** The table called COMPETITOR has a check constraint on the column END\_DATE to ensure it is greater than the start date. The check constraint will be named:

COMPETITOR\_CK01

#### NOT NULL Constraints

The naming convention for NOT NULL constraints is:

<*table name*>\_*NN*<*n>*

Where <*n*> is the distinguishing key, starting from 01.

* NOT NULL constraint names should have a minimum of 9 characters and a maximum of 30.

**Example:** The table called LOCATION\_HISTORY (LOCHIST) has just one NOT NULL constraint. The NOT NULL constraint will be named:

LOCATION\_HISTORY\_NN01

### Indexes

#### Constraint Indexes

Oracle7 and newer versions implicitly create the index when creating constraints in the database.

#### Unique Indexes

The naming convention for manually created unique indexes that are not related to any constraints or are candidate keys is:

<*table name*>\_UI01

* Unique key index names should have a minimum of 12 characters.

**Example:** To improve query performance, an index on the column MSO\_HOUSE\_KEY in table ADDRESS (ADDR) is created. The index will be named:

ADDRESS\_UI01

#### Non-unique Indexes

The naming convention for manually created non-unique indexes that are not related to any constraints is:

<*table name*>\_NI01

* Non-unique key index names should have a minimum of 12 characters.

**Example:** To improve query performance, an index on the column EMPLOYEE\_NAME in table EMPLOYEE (EMP) is created. The index will be named:

EMPLOYEE\_NI01

#### Function-Based Indexes

The naming convention for manually created function-based indexes is:

<*table name*>\_FI01

* Function-based index names should have a minimum of 12 characters.

**Example:** A function-based index on the column TRUNC(DATE\_CREATED) in table EMPLOYEE (EMP) is created. The index will be named:

EMPLOYEE\_FI01

#### Bit-mapped Indexes

The naming convention for manually created bit-mapped indexes is:

<*table name*>\_BI01

* Bit-mapped index names should have a minimum of 12 characters.

**Example:** A bit-mapped index on the column EMPLOYEE\_OFFICE in table EMPLOYEE (EMP) is created. The index will be named:

EMPLOYEE\_BI01

#### Partitioning

Partitioning can be used when a special need arises or, in the interest of table maintenance, it is determined by the Data Architect and the Database Administrator that index partitioning would be beneficial.

The naming convention for partitioned indexes is:

<*index name*>\_HPARTXX or \_HPXX for hash partitions

-or-

<*index name*>\_RPARTXX or \_RPXX for range partitions

-or-

<*index name*>\_LPARTXX or \_LPXX for list partitions

. -or-

<*index name*>\_PYYYYMM for automated partition maintenance scripts

**Example:** CUSTOMER\_PKI\_HPART01 or CUSTOMER\_PKI\_HP01

### Sequences

The naming convention for sequences is:

<*table name*>\_SEQ

Where <*table name*> is the table for which the sequence will provide values to the primary key.

* Sequence names should have a minimum of 8 characters.

**Example:** CUSTOMER\_SEQ

### Views

The naming convention for views is:

<*table name*>\_<*criteria*>\_VW

Where <*table name*> is the name of the root (or ‘master’) table the view is based on. The <*criteria>* qualifier is optional. The qualifier, if used, should give the end users a clear idea of the purpose and contents of the view. Use the <*criteria>* qualifier if:

* Using the table name alone is not unique
* The view is based on a join of 2 or more tables
* The view contains a where clause
* The view is unusually complex
* The view is a summary

**Example:** EMPLOYEE\_VW is a view on the EMPLOYEE table

**Examples:** CUSTOMER\_ACTIVE\_VW provides information on only active records in the CUSTOMER table.

EMPLOYEE\_DEPARTMENT\_VW is a view joining the EMPLOYEE table to the DEPARTMENT table.

CUSTOMER\_COMPLAINT\_VW provides information about CUSTOMER and their notes of type ‘complaint’.

CUSTOMER\_SALE\_TOTAL\_VW provides summary information on customer sales activity.

### Materialized Views

Materialized Views are named exactly the same way as regular views except with a \_MVW suffix instead. The naming convention for materialized views is:

<*table name*>\_<*criteria*>\_MVW

## Other Database Objects

This section describes naming standards for objects that will normally be encountered or defined during the Oracle ‘Build’ phase. These objects are primarily database objects therefore any deviations from these standards must be approved by the DA.

### Functions

PL/SQL functions will normally be sub-programs within a PL/SQL package. Function names should contain verbs that indicate **how** the function will obtain the value that it returns e.g. GET\_STATUS. A function name should also indicate **what** the return value will be and nothing more.

#### Packaged Functions

The naming convention for packaged functions is:

<*action*> <*noun*> or <*adjective*>

Where <*noun*> or <*adjective*> indicates what the function will return. Do not reuse the application name, any part of the package name or any suffix in the function name.

Functions contained in the same package can have the same name, provided that their arguments are not of the same number and data types. Creating multiple functions with the same name in the same package with different arguments is called *overloading* the function.

**Example:**  GET\_STATUS is a function within the package VESSEL\_PKG that returns the status of a vessel.

#### Stand-alone Functions

The naming convention for stand-alone functions is:

<*action*> <*noun*> or <*adjective*>\_F

Where <*noun*> or <*adjective*> indicates what the function will return. Do not reuse the application name or any part of the package name in the function name.

**Example**: GET\_TOTAL\_UNITS\_CHECK\_F is a stand-alone function that returns TRUE if the total unit count is greater than zero.

### Object Views

The naming convention for object views is:

<*object view name*>\_OV

Where <*object view name*> is the base name of the object on which the view is based.

Object view names must be singular just like tables.

* Object view names should have a minimum of 7 characters.

**Example:**

CREATE TYPE ot\_employee AS OBJECT

(employee\_no NUMBER(4),

employee\_name VARCHAR2(20),

job VARCHAR2(9)

)

CREATE OR REPLACE VIEW EMPLOYEES\_OV OF ot\_employee

WITH OBJECT IDENTIFIER (employee\_no)

AS SELECT employee\_no, employee\_name, job

FROM employees;

### Packages

The naming convention for packages is:

<*package name*>\_PKG

Where <*package name*> is the name of the table or other object on which the package operates.

Each package will contain procedures and functions that operate on a table or other object.

Where a package does not operate on data in a particular table or object, a name describing the purpose of the package should be used.

PL/SQL code should take advantage of the many benefits packages offer over stand-alone procedures and functions:

* Enforced information hiding
* Object-oriented design
* Top-down design
* Object persistence
* Performance improvement
* Package names should have a minimum of 8 characters.

**Examples**: VESSEL\_PKG is a package containing various procedures for the VESSEL table.

STRING\_PKG is a package that provides functions to handle string manipulations.

### Procedures

PL/SQL procedures will normally be sub-programs within a PL/SQL package.

#### Packaged Procedures

The naming convention for packaged procedures is:

<*action*> <*noun*> and/or <*verb*>

Where <*noun*> or <*verb*> provide a brief explanation of what the procedure does. Do not reuse the application name, any part of the package name or any suffix in the procedure name.

By default, packages used to manipulate data in a single table may be created with 5 packaged procedures which act as methods for all access to that table. The default packaged procedures are:

select\_all – Returns all records

select\_rec – Given the entire primary key and only the primary key returns complete record

insert\_rec – Inserts a record

update\_rec – Given the entire primary key and only the primary key updates a record

delete\_rec - Given the entire primary key and only the primary key deletes a record.

Procedures contained in the same package can have the same name, provided that their arguments are not of the same number and data types. Creating multiple procedures with the same name in the same package with different arguments is called *overloading* the procedure.

**Example:** RELATIVE\_SPEED is a procedure within a package used to calculate relative speed of a vessel.

#### Stand-alone Procedures

The naming convention for stand-alone procedures is:

<*action*><*noun*> and/or <*verb*>\_P

Where the <*noun*> or <*verb*> is a brief explanation of what the procedure does. Do not reuse the short name or any suffix in the procedure name.

**Example:** ELAPSED\_TIME\_P is a stand-alone procedure that calculates and processes the elapsed time between two events.

### SQL files

The naming convention for SQL files is:

<*noun*> and/or <*verb*>.SQL

Where <*noun*> or <*verb*> is a brief explanation of what the SQL file does.

The header for all SQL files should contain information similar to the following:

/\* Formatted on Date and Time

---Canoe Ventures SCM Header-------------------------------------

--- File Name :

--- SCM Revision :

--- Author :

--- Last Modified :

-----------------------------------------------------------------

-- TITLE :

-- PARAMETERS :

-- USAGE : script ARG1 ARG2

-- OUTPUTS :

-- DESCRIPTION :

-- NOTES :

--

-----------------------------------------------------------------

-- MODIFICATION HISTORY

-- DATE AUTHOR DESCRIPTION

------------ --------- ------------------------------------------

--

--

*\*/*

### Synonyms

Synonyms are created as aliases for tables, views, sequences, triggers, procedures, functions, packages, materialized views, Java class schema objects, or other synonyms. Synonyms are used for security and convenience to:

* Mask the name and owner of an object
* Provide location transparency for remote objects of a distributed database
* Simplify SQL statements for database users

The naming convention for synonyms is:

*<Synonym name>*

Where <*synonym name*> by default exactly matches the name of the underlying object that the synonym is referencing. An exception to this is where the underlying object has been renamed in which case the synonym needs to be redefined and applications based on the synonym will continue to function without modification. Exceptions will also be granted where the intent is to obfuscate the actual object’s name to provide a layer of abstraction.

* Synonyms should have a minimum of 4 characters.

**Example:** VESSEL is a synonym for the table VESSEL

### Triggers

A trigger is a stored PL/SQL block associated with a table, a schema, or database Triggers should be used sparingly due to several factors:

1. Since triggers are global to all users, when performing mass updates the trigger needs to be disabled which necessitates an application downtime.
2. Triggers increase database overhead.
3. Triggers can be inconsistent over database links.

When a trigger is deemed to be needed the naming convention for them is:

<*table name*>\_ <*when*><*type*><*level*>\_TRG

Where:

<*table name>* is the name of the table on which the trigger is based

<*when*> refers to whether the trigger is executed **B**efore or **A**fter an insert, update, or delete

<*type*> is the action (**I**nsert, **U**pdate or **D**elete) performed by the trigger

<*level*> indicates if the trigger is fired for every **R**ow in the table or once for every DML **S**tatement executed

* Trigger names should have a minimum of 10 characters.

**Examples:** LOCATION\_BUR\_TRG = A before update trigger to save previous location information for every row updated.

LOCATION\_AIUDS\_TRG = An after insert, update or delete trigger to change the status code in an audit table for each DML statement.

## Diagramming Guidelines

### General

#### Introduction

This section of the standards document specifies standard practices for elements and characteristics of Diagramming. By implementing consistent diagramming practices confusion will be minimized when interpreting diagrams and attempting to understand business rules.

#### Objectives

The intent of this section is to establish Diagramming standards and guidelines that will accomplish the following:

* Increase consistency
* Improve readability
* Improve pattern recognition

#### Applicability

This document applies to all database development at Canoe-Ventures.

### Entity Relationship (ER) Diagrams

These guidelines should be implemented when creating ER Diagrams via the Oracle SQL Developer Data Modeler, Erwin or any other ERD tool.

#### Multiple Modular Diagrams

Rather than creating one large diagram, create many smaller diagrams (In Erwin, these are called Subject Areas). This will eliminate the need to maintain one large diagram that will over time become unreadable, unmanageable, and overwhelming to non-technical users. If you need to answer questions such as, what entities are related to other entities, use the Repository Object Navigator or the Entity Definition report. Do not rely on one large diagram.

#### Avoid Overcrowding

Do not crowd many entities onto a single diagram. In general, one diagram should contain no more than 15 to 20 entities.

If a single diagram contains more than 15 entities, consider a logical refinement (e.g., the CUSTOMER INFO diagram would be subdivided into two diagrams: CUSTOMER INFO1 and CUSTOMER INFO2).

#### Display a Legend on the Diagram

The following is helpful if displayed: Diagram, Title, Created, Last Modified, Author and Application.

While the diagram name may be somewhat cryptic, the diagram title should be indicative of the business area being modeled and be understandable by business users.

**Example:**

Diagram Name = ERD-CUST INFO

Title = Customer Definition Information.

#### ER Diagram Layout Convention #1

**In this section, please refer to the example diagram; Figure ERD-1 Example of correct use of standards.**

Following are the guidelines for to be used for determining the layout of an ER diagram:

Entity Placement

* Place fundamental entities to the right of the diagram
* Size entities to reduce clutter and provide clear paths for relationships

Drawing Relationships

* Face crow’s feet to the left (West)
* Avoid line crossings if at all possible
* Keep relationship lines horizontal; do not angle them
* Minimize the use of bent lines (straight lines are preferred)

Relationship name placement

* Place relationship names in a clockwise pattern

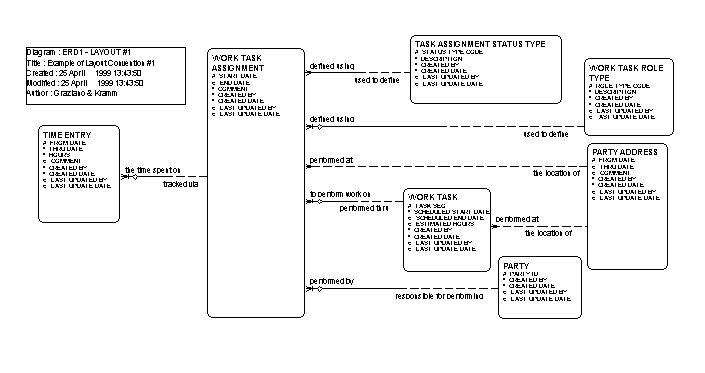


Figure ERD-1 Example of correct use of standards

#### ER Diagram Layout Convention #2

**In this section, please refer to the example diagram; Figure ERD-2 Example of correct use of standards.**

Following are the guidelines for to be used for determining the layout of an ER diagram:

Entity Placement

* Place fundamental entities to the right or bottom of the diagram
* Size entities to reduce clutter and provide clear paths for relationships

Drawing Relationships

* Face crow’s feet to the left (West) or up (North)
* Avoid line crossings if at all possible
* Keep relationship lines horizontal or vertical; do not angle them
* Minimize the use of bent lines (straight lines are preferred)

Relationship name placement

* Place relationship names in a clockwise pattern

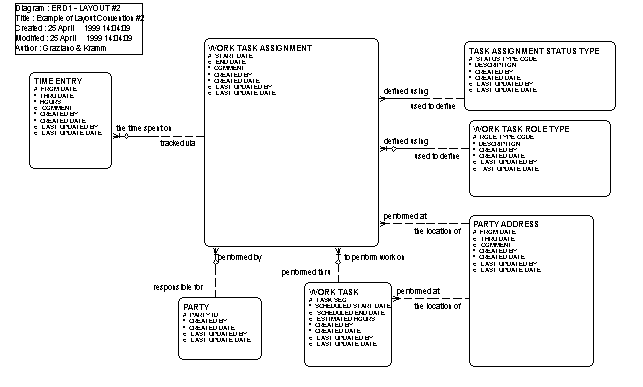


Figure ERD-2 Example of correct use of standards

#### ER Diagram Layout Convention #3

Following are the guidelines for to be used for determining the layout of an ER diagram:

Entity Placement

* Place fundamental entities at the top of the diagram
* Size entities to reduce clutter and provide clear paths for relationships

Drawing Relationships

* Always face crow’s feet to the bottom (South)
* Avoid line crossings if at all possible
* Keep relationship lines horizontal or vertical; do not angle them
* Minimize the use of bent lines (straight lines are preferred)

Relationship name placement

* Place relationship names in a clockwise pattern

#### Consolidating Diagrams

In general, it is a good practice to always use your tool’s repository to consolidate and version your diagrams.

#### Deleting Un-needed Diagrams

If a diagram is no longer needed, delete it from the Repository.

#### Drawing Relationships to Sub-types

When drawing relationships to a sub-type entity, ensure that the line(s) between the sub-type and other entities are clearly connected to the sub-type and not its super-type.

Uses of sub-types imply that they will have attributes that are specific to them. If this is not the case, do not use sub-type constructs where simple examples will do. Instead, include examples as text on the drawing, where clarification is necessary.

#### Color Coding Diagrams

Use color coding or shading to distinguish the following:

* Entities shared from other applications
* Entities that appear on more than one diagram
* Code type entities
* Subject area groupings

## PL/SQL Standards

This section describes the standards used for PL/SQL. As a general rule, all PL/SQL elements including prefixes should be in lowercase.

* PL/SQL elements should have a maximum length of 30 characters.

### PL/SQL Program Data

#### Constants

The naming convention for PL/SQL program constants is:

c\_<*constant name*>

Where <*constant name*> should be a logical, meaningful, and concise name representing the use of the value the constant will hold. Where the constant will be used to hold a value to be assigned to a column in a SQL statement, then <*constant name*> must be the same as the column name.

When declaring constants their scope should always be package specification or package body. Avoid declaring constants locally in a procedure or function. If constants are kept at the package level they can be declared together in the form of a list which will make it easier to review and maintain the values that are assigned to the constants. If we limit ourselves to the package level there will be no need to create a scope identifier prefix for local versus packaged constants thereby reducing the complexity of the prefixing system.

All constants should be in lower case, including the prefix.

**Example:** c\_date\_mask CONSTANT VARCHAR2(20) := ‘DD-MON-YYYY HH24:MI:SS’

#### Exceptions

The naming convention for PL/SQL program exceptions is:

e\_<*exception name*>

User defined exceptions should be limited to package specification or package body scope for the same reasons as constants.

**Example:** e\_time\_expired

#### Record types

The naming convention for PL/SQL program record types is:

<*name of type*>\_rec

**Example:**

TYPE employee\_rec IS RECORD (

name\_first VARCHAR2(25),

name\_last VARCHAR2(25)

);

#### Index-by table types

The naming convention for PL/SQL program index-by table types is:

<*name of type*>\_tbl

**Example:** TYPE name\_tbl IS TABLE OF VARCHAR2(25) INDEX BY BINARY INTEGER;

Consistent with the use of singular table names, collection types (index-by table, nested table and varray) should have singular names. If only a single value is needed then a scalar variable should be used.

#### Nested table types

The naming convention for PL/SQL program nested table types is:

<*name of type*>\_ntbl

**Example:** TYPE column\_name\_ntbl IS TABLE OF VARCHAR2(30);

#### Varray types

The naming convention for PL/SQL program varray types is:

<*name of type*>\_vtbl

**Example:** TYPE county\_vtbl IS VARRAY(50) OF VARCHAR2(30) NOT NULL;

#### Ref cursor types

The naming convention for PL/SQL program reference cursor types is:

<*name of type*>\_refcur

**Example:** TYPE part\_refcur IS REF CURSOR RETURN parts%TYPE;

#### Object types

The naming convention for PL/SQL program object types is:

<*name of type*>\_obj

**Example:**

CREATE TYPE employee\_obj AS OBJECT

(employee\_no NUMBER(4),

employee\_name VARCHAR2(20),

job VARCHAR2(9));

#### Subtypes

The naming convention for PL/SQL program subtypes is:

<*name of subtype*>\_stype

**Example:**

v\_post\_code VARCHAR2(10)

SUBTYPE post\_code\_stype IS v\_post\_code%TYPE;

#### Cursors

The naming convention for PL/SQL program cursors is:

<*cursor name*>\_cur

Where <*cursor name*> is a logical, meaningful, and concise name representing the function the cursor serves.

Cursors declared in a package body have an additional prefix of g\_ to indicate package body scope. Cursors declared within a procedure or function will have no scope prefix added.

All cursors must be in lower case, including the prefix.

**Example:**

CURSOR title\_cur

IS

SELECT title

FROM book

ORDER BY title;

**Example:**

CREATE OR REPLACE PACKAGE BODY VESSEL\_PKG IS

g\_owner VARCHAR2 (30); --The owner of the table

g\_name VARCHAR2 (30); --The name of the table

CURSOR g\_column\_cur

IS

SELECT column\_name, data\_type, data\_length, data\_precision, data\_scale

FROM dba\_tab\_columns

WHERE owner = p\_v\_owner AND table\_name = p\_v\_name;

...

#### Variables

The naming convention for PL/SQL program variables is:

v\_<*variable name*> - local to procedure or function

g\_*<variable name>* - global to the package body

Where the data type for the variable is scalar and the scope is local only the v\_ will be needed to prefix the <*variable name*>. If the variable data type is a composite data type then appropriate suffix should be supplied for the variable as listed **in Appendix E – PL/SQL Element Format Chart**. Having the type identified in the suffix should help code readability. Additionally, the type suffix helps resolve conflicts between the object type variable and the scalar variable that have the same or similar base names.

<*variable name*> should be a logical, meaningful and concise name representing the value the variable will hold. Where the variable will be used to hold the value of a column in a SQL statement, then <*variable name*> must be the same as the column name.

Variables declared in a package body have a prefix g\_ to indicate global package body scope. Variables declared within a procedure or function will have the v\_ added. Variables should never be declared in a package specification as this violates the principles of encapsulation and information hiding.

**Example:** g\_header\_id VARCHAR2(2) ; – scalar variable declared in package body

#### Parameters

The naming convention for PL/SQL program parameters is:

*<parameter mode>*\_<*parameter name*>*\_<parameter\_type>*

A parameter has one of three modes: IN, OUT or IN OUT. An IN parameter passes a value into the module, but its value cannot be modified. An OUT parameter passes a value out of the module, but its value cannot be referenced in the module. An IN OUT parameter can be both referenced and modified in the module. Parameter modes are abbreviated as follows:

IN pi

OUT po

IN OUT pio

By incorporating the parameter mode directly into the parameter name, its purpose in the module is self-documenting. Whenever that parameter is encountered in the code, you know that you are looking at a parameter, not a local variable and you know the ways in which the parameter can be use and/or changed.

<*parameter type*> is the parameter type as listed in **Appendix E – PL/SQL Element Format Chart.**

<*parameter name*> is a logical, meaningful, and concise name representing the value that will be passed to the program unit. Where the parameter will be used to hold the value of a column in a SQL statement, then <*parameter name*> must be the same as the column name.

**Example**:

PROCEDURE combine\_name

(pio\_name\_first IN OUT VARCHAR2,

pio\_name\_last IN OUT VARCHAR2

po\_name\_full OUT VARCHAR2);

#### FOR Loop Index Names

The naming convention for numeric FOR loops is:

<*loop index name*>\_index

The naming convention for cursor FOR loops is:

<*loop record name*>\_rec

## Bibliography

## Appendix A – Object Format Chart

Unless otherwise stated the maximum length of Oracle objects is 30 characters. Lengths shown include suffixes and spaces or underscores.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Property** | | **Minimum Length of Object Name** | | **Maximum Length of Object Name** | **Required Suffix** | **Usage** |
| **Applications** | | | |  |  |  |
| Short Name | 3 | | | 5 |  | Used for application schemas. |
| **Entities** | | | |  |  |  |
| Name | | 3 | | 25 |  | Used to derive default entity singular. |
| Short Name (Alias) | | 3 | | 10 |  | Used to generate table alias, part of FK column name to indicate source entity, part of table unique ID name. |
| **Domains** | | | |  | | |
| Name | | 3 | |  |  | Attribute/Column Domain property. |
| **Attributes** | | | |  | | |
| Name | | | 3 |  |  | Used to generate column name. |
| **Unique Identifiers (UIDs)** | | | |  | | |
| Name | | | 5 | 12 |  | Used to generate primary key. |
| **Relationships** | | | |  | | |
| Name | | |  | 60 |  | May be used as part of the FK constraint name (if necessary) to ensure uniqueness of name. |
| **Server Model Objects** | | | |  | | |
| Table Names | | 4 | | 25 |  | Singular of entity name. |
| Table Aliases | | 3 | | 10 |  | Must equal entity short name. |
| Data Archive Tables | | 8 | | 26 | \_AUD or \_HIST or \_TXN or \_STG |  |
| Journal Tables | | 8 | | 26 | \_JNL |  |
| Primary Key Constraints | | 6 | | 13 | \_PK |  |
| Unique Key Constraints | | 8 | | 30 | \_UK*<n>* |  |
| Foreign Key Constraints | | 10 | | 30 | \_FK*<n>* |  |
| Check Constraints | |  | |  | \_CK*<n>* |  |
| NOT NULL constraints | | 9 | | 16 | \_NN<*n*> |  |
| Non-unique Indexes | | 12 | |  | \_NI<*n*> |  |
| Bit-mapped Indexes | | 12 | |  | \_BI<*n>* |  |
| Sequences | | 8 | |  | \_SEQ |  |
| Views | |  | |  | \_VW |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Property** | **Minimum Length of Object Name** | **Maximum Length of Object Name** | **Required Suffix** | **Usage** |
| **Other Database Objects** | |  | | |
| Functions |  |  | \_F | Suffix only used with stand-alone functions. |
| Packages | 8 |  | \_PKG or \_DAL or \_DIL | \_DAL used for Data Abstraction Layer; \_DIL used for Data Integrity Layer or \_PKG |
| Procedures |  |  | \_P | Suffix only used with stand-alone procedures. |
| Synonyms | 4 |  |  | Same name as underlying object. |
| Temporary Objects |  |  | \_TMP | Suffix |
| Triggers | 10 |  | \_TRG | Suffix |

## Appendix B - Approved Acronym List

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| DAL | Data abstraction layer |
| DIL | Data integrity layer |
| SPA | SYS / PRIN / AGENT |
| wm | web methods |
| WLS | Web logic |

## Appendix C - Approved Abbreviation List

|  |  |
| --- | --- |
| **Abbreviation** | **Full Word** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Appendix D- Application Abbreviation List

|  |  |
| --- | --- |
| **Abbreviation** | **Full Word** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Appendix E– Oracle Reserved and Key Words (11.1)

This appendix lists Oracle reserved and key words including PL/SQL reserved words. Reserved words have a special meaning to Oracle and cannot be redefined. For this reason, you cannot use them to name database objects such as columns, tables, or indexes. Keywords also have a special meaning to Oracle but are not reserved words and so can be redefined. However, some might eventually become reserved words. PL/SQL keywords may require special treatment when used in embedded SQL statements.

In addition to the following reserved words, Oracle uses system- generated names beginning with ‘SYS\_’ for implicitly generated schema objects and sub-objects. Oracle discourages you from using this prefix in the names you explicitly provide to your schema objects and sub-objects to avoid possible conflict in name resolution. Additionally, the word DUAL should not be used as a name for an object or object part as DUAL is the name of a dummy table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Oracle Reserved Words** | | | |
| ACCESS | ELSE | MODIFY | START |
| ADD | EXCLUSIVE | NOAUDIT | SELECT |
| ALL | EXISTS | NOCOMPRESS | SESSION |
| ALTER | FILE | NOT | SET |
| AND | FLOAT | NOTFOUND | SHARE |
| ANY | FOR | NOWAIT | SIZE |
| ARRAYLEN | FROM | NULL | SMALLINT |
| AS | GRANT | NUMBER | SQLBUF |
| ASC | GROUP | OF | SUCCESSFUL |
| AUDIT | HAVING | OFFLINE | SYNONYM |
| BETWEEN | IDENTIFIED | ON | SYSDATE |
| BY | IMMEDIATE | ONLINE | TABLE |
| CHAR | IN | OPTION | THEN |
| CHECK | INCREMENT | OR | TO |
| CLUSTER | INDEX | ORDER | TRIGGER |
| COLUMN | INITIAL | PCTFREE | UID |
| COMMENT | INSERT | PRIOR | UNION |
| COMPRESS | INTEGER | PRIVILEGES | UNIQUE |
| CONNECT | INTERSECT | PUBLIC | UPDATE |
| CREATE | INTO | RAW | USER |
| CURRENT | IS | RENAME | VALIDATE |
| DATE | LEVEL | RESOURCE | VALUES |
| DECIMAL | LIKE | REVOKE | VARCHAR |
| DEFAULT | LOCK | ROW | VARCHAR2 |
| DELETE | LONG | ROWID | VIEW |
| DESC | MAXEXTENTS | ROWLABEL | WHENEVER |
| DISTINCT | MINUS | ROWNUM | WHERE |
| DROP | MODE | ROWS | WITH |

|  |  |  |  |
| --- | --- | --- | --- |
| **Oracle Keywords** | | | |
| ADMIN | CURSOR | FOUND | MOUNT |
| AFTER | CYCLE | FUNCTION | NEXT |
| ALLOCATE | DATABASE | GO | NEW |
| ANALYZE | DATAFILE | GOTO | NOARCHIVELOG |
| ARCHIVE | DBA | GROUPS | NOCACHE |
| ARCHIVELOG | DEC | INCLUDING | NOCYCLE |
| AUTHORIZATION | DECLARE | INDICATOR | NOMAXVALUE |
| AVG | DISABLE | INITRANS | NOMINVALUE |
| BACKUP | DISMOUNT | INSTANCE | NONE |
| BEGIN | DOUBLE | INT | NOORDER |
| BECOME | DUMP | KEY | NORESETLOGS |
| BEFORE | EACH | LANGUAGE | NORMAL |
| BLOCK | ENABLE | LAYER | NOSORT |
| BODY | END | LINK | NUMERIC |
| CACHE | ESCAPE | LISTS | OFF |
| CANCEL | EVENTS | LOGFILE | OLD |
| CASCADE | EXCEPT | MANAGE | ONLY |
| CHANGE | EXCEPTIONS | MANUAL | OPEN |
| CHARACTER | EXEC | MAX | OPTIMAL |
| CHECKPOINT | EXPLAIN | MAXDATAFILES | OWN |
| CLOSE | EXECUTE | MAXINSTANCES | PACKAGE |
| COBOL | EXTENT | MAXLOGFILES | PARALLEL |
| COMMIT | EXTERNALLY | MAXLOGHISTORY | PCTINCREASE |
| COMPILE | FETCH | MAXLOGMEMBERS | PCTUSED |
| CONSTRAINT | FLUSH | MAXTRANS | PLAN |
| CONSTRAINTS | FREELIST | MAXVALUE | PLI |
| CONTENTS | FREELISTS | MIN | PRECISION |
| CONTINUE | FORCE | MINEXTENTS | PRIMARY |
| CONTROLFILE | FOREIGN | MINVALUE | PRIVATE |
| COUNT | FORTRAN | MODULE | PROCEDURE |
| PROFILE | SAVEPOINT | SQLSTATE | TRACING |
| QUOTA | SCHEMA | STATEMENT\_ID | TRANSACTION |
| READ | SCN | STATISTICS | TRIGGERS |
| REAL | SECTION | STOP | TRUNCATE |
| RECOVER | SEGMENT | STORAGE | UNDER |
| REFERENCES | SEQUENCE | SUM | UNLIMITED |
| REFERENCING | SHARED | SWITCH | UNTIL |
| RESETLOGS | SNAPSHOT | SYSTEM | USE |
| RESTRICTED | SOME | TABLES | USING |
| REUSE | SORT | TABLESPACE | WHEN |
| ROLE | SQL | TEMPORARY | WRITE |
| ROLES | SQLCODE | THREAD | WORK |
| ROLLBACK | SQLERROR | TIME |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Oracle PL/SQL Reserved Words** | | | |
| ABORT | BETWEEN | CRASH | DIGITS |
| ACCEPT | BINARY\_INTEGER | CREATE | DISPOSE |
| ACCESS | BODY | CURRENT | DISTINCT |
| ADD | BOOLEAN | CURRVAL | DO |
| ALL | BY | CURSOR | DROP |
| ALTER | CASE | DATABASE | ELSE |
| AND | CHAR | DATA\_BASE | ELSIF |
| ANY | CHAR\_BASE | DATE | END |
| ARRAY | CHECK | DBA | ENTRY |
| ARRAYLEN | CLOSE | DEBUGOFF | EXCEPTION |
| AS | CLUSTER | DEBUGON | EXCEPTION\_INIT |
| ASC | CLUSTERS | DECLARE | EXISTS |
| ASSERT | COLAUTH | DECIMAL | EXIT |
| ASSIGN | COLUMNS | DEFAULT | FALSE |
| AT | COMMIT | DEFINITION | FETCH |
| AUTHORIZATION | COMPRESS | DELAY | FLOAT |
| AVG | CONNECT | DELETE | FOR |
| BASE\_TABLE | CONSTANT | DELTA | FORM |
| BEGIN | COUNT | DESC | FROM |
| FUNCTION | NEW | RELEASE | SUM |
| GENERIC | NEXTVAL | REMR | TABAUTH |
| GOTO | NOCOMPRESS | RENAME | TABLE |
| GRANT | NOT | RESOURCE | TABLES |
| GROUP | NULL | RETURN | TASK |
| HAVING | NUMBER | REVERSE | TERMINATE |
| IDENTIFIED | NUMBER\_BASE | REVOKE | THEN |
| IF | OF | ROLLBACK | TO |
| IN | ON | ROWID | TRUE |
| INDEX | OPEN | ROWLABEL | TYPE |
| INDEXES | OPTION | ROWNUM | UNION |
| INDICATOR | OR | ROWTYPE | UNIQUE |
| INSERT | ORDER | RUN | UPDATE |
| INTEGER | OTHERS | SAVEPOINT | USE |
| INTERSECT | OUT | SCHEMA | VALUES |
| INTO | PACKAGE | SELECT | VARCHAR |
| IS | PARTITION | SEPARATE | VARCHAR2 |
| LEVEL | PCTFREE | SET | VARIANCE |
| LIKE | POSITIVE | SIZE | VIEW |
| LIMITED | PRAGMA | SMALLINT | VIEWS |
| LOOP | PRIOR | SPACE | WHEN |
| MAX | PRIVATE | SQL | WHERE |
| MIN | PROCEDURE | SQLCODE | WHILE |
| MINUS | PUBLIC | SQLERRM | WITH |
| MLSLABEL | RAISE | START | WORK |
| MOD | RANGE | STATEMENT | XOR |
| MODE | REAL | STDDEV |  |
| NATURAL | RECORD | SUBTYPE |  |

## Appendix F - PL/SQL Element Format Chart

|  |  |
| --- | --- |
| **Property** | **Required Prefix** |
| **Constants & Exceptions** | |
| constant | c\_ |
| exception | e\_ |
| **Types** | |
| record type | *<type\_name>\_*rec |
| index-by table type | *<type\_name>*\_tbl |
| nested table type | *<type\_name>*\_ntbl |
| varray type | *<type\_name>*\_vtbl |
| ref cursor type | *<type\_name>*\_refcur |
| object type | *<type\_name>*\_obj |
| sub type | *<type\_name>*\_stype |
| **Cursors & Variables** | |
| cursor | *<var\_name>*\_cur |
| variable of scalar | No Suffix needed |
| variable of record type | *<var\_name>*\_rec |
| variable of table%row type |
| variable of index-by table type | *<var\_name>*\_tbl |
| variable of nested table type | *<var\_name>*\_ntbl |
| variable of varray type | *<var\_name>*\_vtbl |
| variable of ref cursor type | *<var\_name>*\_refcur |
| variable of object type | *<var\_name>*\_obj |
| variable of object reference | *<var\_name>*\_refobj |
| **Parameters** | |
| **Parameter Modes** |  |
| IN | pi\_ |
| OUT | po\_ |
| IN OUT | pio\_ |
| **Parameter Types** |  |
| cursor parameter | *<parameter\_name>*\_cur |
| parameter of scalar | No suffix needed |
| record type | <*parameter\_name>\_*rec |
| table%row type |
| index-by table type | *<parameter\_name>*\_tbl |
| nested table type | *<parameter\_name>*\_ntbl |
| varray type | *<parameter\_name>*\_vtbl |
| ref cursor type | *<parameter\_name>*\_refcur |
| object type | *<parameter\_name>*\_obj |
| object reference | *<parameter\_name>*\_refobj |
| **FOR Loop Index Names** | |
| FOR loop index (numeric) | \_index |
| FOR loop index (cursor) | \_rec |

## Appendix G – Domain Examples

The following domains are intended as examples only and are **not** approved domains.

|  |  |  |  |
| --- | --- | --- | --- |
| **Domain Name** | **Format** | **Size** | **Description** |
| **AMOUNT** | Number | 9,2 | Monetary amounts. |
| **CODE** | Varchar2 | 20 | Character ID for alphanumeric unique identifiers by which the users recognize the entity. |
| **DATE** | Date | N/A | Generic date domain |
| **DESCRIPTION** | Varchar2 | 40 | Description text for a code. |
| **Ind** | Number | 1 | Yes/No indicator |
| **APP\_CREATED** | Varchar2 | 61 | Application or Package name and Procedure that performed the insert. |
| **USER\_CREATED** | Varchar2 | 30 | Oracle user ID of the person who inserted this record |
| **DATE\_CREATED** | Date | N/A | Date this record was inserted |
| **APP\_MODIFIED** | Varchar2 | 61 | Application or Package name and Procedure that updated the record. |
| **USER\_MODIFIED** | Varchar2 | 30 | Oracle user ID of the person who last updated this record |
| **DATE\_MODIFIED** | Date | N/A | Date this record was last updated |
| **Long text** | Varchar2 | 1000 | A long string of text |
| **Name** | Varchar2 | 40 | A text string to indicate the formal name for a person, place or thing. |
| **Rate** | Number | 6,2 | Used for value rates - includes a min and max rate. |
| **Seq id** | Number | 4,0 | Integer identifier for use as a line item (sequence within parent). |
| **Small text** | Varchar2 | 30 | Small string of text. |
| **Countries** | Varchar2 | 3 | Valid Alpha-3 ISO country codes. |
| **Text** | Varchar2 | 240 | Generic mid-length text string. Size based on legacy Oracle restrictions. |
| **Type** | Varchar2 | 10 | An attribute that contains an alphanumeric value to represent a categorization of an entity (i.e., a sub-type). |
| **UID** | Number |  | System generated unique identifier. |
| **Very long text** | Varchar2 | 2000 | A very long string of text. |

## Appendix H – Relationship Names Examples

The following relationships are intended as **examples only**.

|  |  |
| --- | --- |
| **Crows Foot Side (many)** | **Single Side (one)** |
| A condition for | Subject to |
| A modification of | Modified by |
| A usage of | Used for |
| Attached to | Inclusive of |
| Based on | The bases for |
| Based on | The originator of |
| Billed to | Billed for |
| Characterized by | The characteristics for |
| Classified by | The classification for |
| Copied from | The original for |
| Defined as | Used to define |
| Defined by | Used to define |
| Described by | The definition for |
| Described by | The description of |
| Described by | Used to describe |
| For | Subject to |
| Fulfilled by | Used to fulfill |
| Further defined by | A further definition of |
| Further described by | Used to describe |
| Further described by | A further description of |
| Included in | Comprised of |
| Located at | The location of |
| Made from | Composed of |
| Made by | The creator of |
| Manufactured by | The producer of |
| Part of | Composed of |
| Part of | Detailed by |
| Rated by | The rating for |
| Requested by | The requestor for |
| Returned from | Returned to |
| Sent from | Sent to |
| Sent from | The source of |
| Sent to | The destination of |
| Stored in | The storage for |
| The purpose for | Used for the purpose of |
| Used by | In need of |

## Appendix I – Standard Processing Log Table Structure

The following table definition is intended as a baseline **only**. Modifications may be necessary for your application.

CREATE SEQUENCE “OWNER”.”SCHEMA\_PREFIX”\_log\_seq

/

CREATE TABLE “OWNER”.”SCHEMA\_PREFIX”\_processing\_log

(log\_id NUMBER NOT NULL,

trace\_level VARCHAR2(20) DEFAULT 'ERROR' NOT NULL,

sid NUMBER DEFAULT 0 NOT NULL,

serial# NUMBER DEFAULT 0 NOT NULL,

username VARCHAR2(30) DEFAULT 'ORACLE' NOT NULL,

osuser VARCHAR2(30) DEFAULT 'ORACLE' NOT NULL,

source VARCHAR2(61) NOT NULL,

start\_time DATE NOT NULL,

end\_time DATE,

parent\_log\_id NUMBER,

transaction\_result VARCHAR2(30),

error\_code NUMBER,

code\_location VARCHAR2(200),

message VARCHAR2(2000) NOT NULL)

TABLESPACE “OWNER”\_sml\_d

/

ALTER TABLE “OWNER”.”SCHEMA\_PREFIX”\_processing\_log

ADD CONSTRAINT ”SCHEMA\_PREFIX”\_processing\_log\_pk PRIMARY KEY (log\_id)

USING INDEX TABLESPACE ”SCHEMA\_PREFIX”\_sml\_i

/

ALTER TABLE “OWNER”.”SCHEMA\_PREFIX”\_processing\_log

ADD CONSTRAINT ”SCHEMA\_PREFIX”\_processing\_log\_fk FOREIGN KEY (log\_id)

REFERENCES “OWNER”.”SCHEMA\_PREFIX”\_processing\_log (log\_id)

/

## CNV - Ben’s notes:

**Oracle SQL Developer “Tip of the Day**

**SQL Worksheet Bookmarks**

If you have many SQL Worksheets open, you can assign a bookmark number to each and then easily navigate among them. To create a bookmark, click the worksheet's tab and press Alt+Shift+number (for example, Alt+Shift+1). The number now appears as a small superscript in the tab.

To switch to a worksheet that has a bookmark, press Alt+number (for example, Alt+1).

My local SQL Plus client (executable):

C:\oraclexe\app\oracle\product\10.2.0\server\BIN\sqlplus.exe

Optionally {/nolog}

My local SQL Plus shortcut:

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Oracle Database 10g Express Edition

TOAD installation notes, 06/15/2011:

