Oracle Database Standards You Can Use Now

by

Jeffrey M. Stander

ANZUS Technology International
2401 N Northlake Way
Seattle, WA 98103 USA
1-800-945-7375
jstander@anzus-technology.com

Presented at IOUGA 1999

Prerequisite

For everybody. Attendees should have some knowledge of SQL and PL/SQL.

Objectives

To (1) provide you with a basic understanding of WHY a standards and practices document is a useful (nay, **essential**) part of the development toolkit; and (2) present a set of basic programming and design standards for Oracle which will get you started.

Abstract

Standards for the creation and maintenance of relational databases are as much a part of the infrastructure for software development as the computers and the database itself. Good standards do not constrain the creativity of designers and developers, but rather encourage the development of best practices. Also, as the name implies, an enterprise now has common practices, which all members of the IT team can easily understand and relate to.

Among Oracle professionals many of the practices presented here are *de facto* standards in common use in the industry, having evolved because of their obvious utility, but in many companies there is no Standards document and programmers and designers often leave incomprehensible code and data structures for those who follow. It pays to have standards and to follow them.

This paper will present an introduction to relational database naming standards, program objects naming standards, coding standards, file naming standards, and a discussion of standards as they apply to application design using Designer/2000.

TABLE OF CONTENTS

1. Introduction

1.1. Why Bother With Standards?

This is not a trivial question. Most IT sites I have worked at have no official standards and those that do exist are the personal working methods of whomever previously worked on their database systems.

Commonly database design evolves over time in response to pressing and immediate needs without careful planning or guidance. In a typical enterprise, different computer platforms and operating systems support applications that use disparate data models and database management systems. To further complicate matters, third party applications as well as in-house development have created incompatible data structures. Both syntactic differences (the same data is defined differently or represented in a different format) and semantic differences (different meanings for the same term) have created conflict between systems. [1]

Over the years certain "correct" methods have developed in the community as a whole, but these are often ignored by self-taught Oracle developers or those who have only recently come to use Oracle. I will try to present what I perceive as these *de facto* standards, as well as others learned on job sites and from professional colleagues. The standards presented are gathered from my professional experience, professional colleagues, code study, reading, and generally available information on the web and elsewhere. I have attempted to synthesize and select good practices. Keep in mind that what is presented here is only my opinion and you are quite welcome to disagree with it. Having a set of your own standards is more important than proving mine wrong. [2]

Standards for the creation and maintenance of relational databases should be as much a part of the infrastructure for software development as the computers and the database itself. Good standards do not constrain the creativity of designers and developers, but rather encourage the development of best practices. Also, as the name implies, an enterprise now has common and consistent practices that all members of the IT team can easily understand. If (even more rare) a standards document exists, then all contractors and new hires have a reference for how to interpret existing code and database objects. Even if the enterprise's standards are different from an individual's common practice, it still creates the consistent common environment. This can save hours of time and confusion and sometimes avoid costly mistakes.

Maintenance especially becomes much easier. As a small example, if a view name always ends in `_V` (*e.g.* `EMP_V`). Any developer will now know this is a view (with its inherent restrictions on updates and query optimizations) and will not assume it is a table.

1.2. How Far Should a Company Go?

I would say that many smaller companies (and some larger ones) have no standards at all. They might have a common practice for their site, which is enforced only by custom. As a consultant I can try to introduce standards, or just do the best job I can with what is there.

[1] Throughout this paper, *data* means information stored in a database, in purchased applications, or collected and stored by end users (engineering staff, for example). The term *database object* refers to anything created in the database, except the data itself (tables, indexes, triggers, procedures, for example).

[2] However, if you have a serious objection to a standard as I've proposed it, and a reason to back it up, please send me an email. As I proceed with this project I would be delighted to hear from other practitioners on their methods.

One can go too far with standards. Beware of this. They can become an impediment if taken to the extreme. I once chose to move on to a new job rather than use the pathologically convoluted standards adopted by a particular workgroup.

Standards are especially agonizing if they are

- Too far off the mainstream (i.e. not common practice).
- Too complex.
- Ignored by everyone but you.
- Used by everyone but you.
- Not applied consistently within the organization or (worse) within an application.
- Not supported by management.
- Not enforced.

Somewhere between *standards that are way too complex* and *no standards at all* is the Standards Middle Path for your company. This means you will have

- Industry-accepted standards (or close to industry-accepted standards)
- A company document which describes these standards

The scope of the standards will also vary. As a minimum, naming standards for database objects (entities, tables, columns, constraints, indexes, sequences, etc. etc.) will go a long way towards making it easier to use and modify your database. Beyond that, standards can be written or adopted for program design and coding (including SQL, PL/SQL, HTML, Java, etc.), GUI design, Database and System administration (e.g. SID names, control file names, database file names, etc.), other file naming conventions and enterprise modeling rules.

Once adopted, standards must be enforced. There must be a code review, design walkthrough, or architecture committee that will make sure the rules are followed.

No matter how different a standard may seem, if it is close to *de facto* industry standard practice and/or adopted for a rational reason, you can get used to it. That standard will soon become second nature to use. A good example of this is the *entity short name* and the *table short name*, which we will review later.

This document is organized along standards types, rather than functional areas. We will cover the following:

- Naming standards (in detail)
- Design Standards (overview)
- Programming Standards (overview)

The limited time and paper length limitation at IOUG-LIVE precludes covering more than a fraction of this subject. A more extensive version of this paper can be found on the web site at http://anzus-technology.com.

2. Naming Standards

The following naming standards are for use in developing ORACLE-based applications. These conventions should be used whenever possible in naming application objects.

Many of the naming standards specified below were chosen with Designer/2000 design and development work in mind. However, these standards can and should be applied to all Oracle-based application development work unless overriding arguments to the contrary can be provided.

2.1. General Naming Conventions for Oracle Databases

Conform to Oracle naming rules:

1. Begin all names with a letter.
2. Only the characters A-Z, 0-9 and _ (underscore) may be used (the use of $ and # are not allowed)
3. Do not use Oracle reserved words as names, e.g. DATE, COUNT, SUM, MAX or DESC.
4. A name should not be the name of another Oracle object of the same type.
5. Names should be as short as possible while remaining meaningful.
6. Use underscores as delimiters (except for entity names, where white space is used).
7. Avoid prepositions where possible (e.g. use VESSEL_LENGTH rather than LENGTH_OF_ VESSEL).
8. Develop and use a general abbreviation standard (see Appendix A, Page 23). Each application system may require additional abbreviations to be added to the standard set.
9. Avoid meaningless or redundant names (e.g. DATA_TABLE for a table name, ROW_CREATE_DATE for a column name).

2.2. Designer/2000 Naming Standards

2.2.1. Designer/2000 Application Systems

An *application system* is a collection of database elements and associations within the Designer/2000 repository. An application system typically corresponds to a database application that you are designing and building.

All Application Systems should have a name or mnemonic of up to 8 characters that is unique within the organization. In practice it is preferable to use a 2- or 3- character mnemonic for ease of reference and for use as an object prefix. For example:

ODW	Operational Data Warehouse
HRS	Human Resource System
CRS	Customer Response System
COR	Corporate Data System.
REF	Reference Data and Standard Objects.

Applications based on the database schemas should have the same name as the schema name.

2.2.2. Entities

Entities should be named in upper case, in the singular, using white space (not underscores as delimiters) and should usually be no more than three words. For example:

PERSON (not PEOPLE)
WAFER
MFG SPECIFICATION
PARTY
WORK ORDER

2.2.3. Entity Short Names

Every entity within an application will be given an *entity short name* or *alias*, consisting of a 4-character mnemonic, which is unique within the application. This will generally produce meaningful and unique names although a little variation may be appropriate. The *table short name* is the given the same name as the entity short name.

The general rule for deriving short names is:

1. If the entity name is one word, use the first four letters of the word
2. If the entity name contains two words, use the first two letters of each word
3. If the entity name contains three words, use the first letter of the first two words, plus first two letters of the third word
4. If the entity name contains four words, use the first letter of each word.

In the event of duplicate short names, a similar combination of letters should be used. Examples of entity short names are listed below:

ENTITY NAME	SHORT NAME
PROCESS	PROC
PROCESS REGISTER	PRRE
PROCESS PARM REGISTER	PPRE
PROCESS PARM REGISTER TYPE	PPRT

Occasionally exceptions will crop up. My favorite is the PARTY entity, which would abbreviate to PART, which is both misleading and possibly a duplicate name in a manufacturing or supply database. Therefor PRTY could be used, but I usually just call it PARTY, since the short name is hardly longer than the full name. I would accept another rule that if the long name is 5 characters make the short name equal to the long name. Occasionally collisions occur and then you just have to pick another alias.

Before you gag and reject this, please read on. I hated this when my friend (and Oracle Consultant) Margaret first proposed it to me. Why should a perfectly good name like MARKETING ANALYSIS, for instance, be given an alias of MAAN. Surely MKTA or MKAN or even MKT_ANAL would be better? But Margaret wouldn't hear of it. Always follow the rules, she said, you would get used to it and eventually swear by it. She was right. First, it is easy to create table or entity short names because there is a rule for doing it. MKT isn't marketing to everybody, they might pick MRK or MKTNG Secondly, because the rule is consistent, it is always easy to determine which table or entity is being referenced. Third, everybody else is doing exactly the same thing (YES!). You can live with MAAN and other strange sounding aliases and eventually you will like it. Trust me.

2.2.4. Attributes and Domains

Attributes and domains should be named as descriptively as possible. They should be in the singular, in upper case, using white space (not underscores) as delimiters, and should usually be no more than three words.

In Designer/2000, utilizing domains can optimize the creation of attributes. Domains define a set of properties that apply to attributes. This property set usually consists of data structure definitions, but may be extended to include validation rules and format constraints. Before creating attributes during the Analysis stage, domains should already be defined such that when the attribute is entered, the appropriate domain definition can be associated to it. This way, the attribute can inherit the properties of the domain, and in the event of property changes, utilities are available to propagate those changes. Using domains appropriately can save a great deal of work on the part of the designer.[3]

2.2.5. Business Functions

Business function statements should always describe what is done rather than why it is carried out.

A function statement should always start with one to three active verbs carried out on an object. The name describes one activity, so the object of the verb is usually singular, e.g. *Calculate Month To Date Yield.*

The function statement should describe only one function. Avoid the use of AND, or the delimiter ";". Each function, if decomposed, must have at least two subordinate functions. The value and scope of a function must be equal to all of its subordinate functions. The functions should be referenced by a unique base 15 alphanumeric character string.

2.2.6. Module Names

Module names should be limited to a string of 8 alphanumeric characters. This allows for the 8-character file name limit imposed by MS-DOS and WIN3.1 (Curse you Bill Gates). This rule can be relaxed if only Windows NT or Windows 95/98 is the standard for client applications.

The module name should be in lower case, to enable it to be moved consistently between operating systems.

Module names should be of the form **aannnnpp** where:

aa	Sub-System Prefix	
nnnn	Unique program reference (could be based on a business function reference to convey some limited meaning and position to the program within the whole application or a number)	
pp	Program Type Code	
	sc	screen or Form
	rp	Report
	p	SQL
	l	PL/SQL?
	c	C
	j	Java
	o	Operating system script

[3] Hervé Deschamps has a method of creating default domains in a Designer/2000 repository. For this and other standards enforcement techniques see his web site: http://home.sprynet.com/~hdeschamps/homepage.htm.

2.3. Program Objects Naming Standards

Program objects include

- PL/SQL Variables
- Oracle*Forms Objects
- Oracle*Reports Objects
- Variables in other languages

2.3.1. Program names

Program names will usually be implementations of Modules that have been defined in the Designer/2000 repository; therefore their names will conform to the conventions laid down for Module names.

2.3.2. Oracle*Forms Objects Names

2.3.2.1 Blocks

Blocks will be named differently according to their usage. In cases where more than one block is required that would have the same name under the following standards, then all such blocks will be given a numeric suffix, beginning with 1, to uniquely identify each block.

Base Table	Standard Base table blocks will use the table or view short name as the block name.
Control	A control block is usually required for most forms, to allow entry of options and store local control values. When present, this block will be named CONTROL.
Non-Database	Non Database Block are typically required for some specific action or purpose, e.g. to enter search criteria or display additional information. These blocks will be named for their *action,* and if they operate on only one object, then *action_object*, e.g. SEARCH_ADDRESS, would be the block in which address search criteria is entered.
Special	There are a number of special purpose blocks that are available from the Standard Objects Form that must not be duplicated in any Form. These are TOOLBAR, CALENDAR, FOLDER_TOOLS and FOLDER_CONTROL.

2.3.2.2 Items

Since items that are mapped directly to database columns must be named the same as the column, it is logical to apply the same naming conventions to items as is applied to columns. The exception to this rule is *mirror items*, which will be named the same as their database equivalents with a '_MIR' suffix.

2.3.2.3 *Record Groups and LOV's*

Record groups and LOV's will be named for the *object* that they represent, or if they contain only a subset of the object the name will include a brief description of the criteria, in the form *object_criteria*. CUSTOMERS (all customers), CUSTOMERS_ACTIVE (all customers that are active).

2.3.2.4 *Windows and Canvasses*

Windows and/or Canvasses are purely graphical objects that are used to control and present information, their names therefore should identify, as clearly as possible, the information they are to present, e.g. CUSTOMER_SEARCH or CUSTOMER_DETAILS.

2.3.3. Oracle*Reports Object Names

Not included at this time.

2.4. DBA Naming Standards

DBA patterns of naming vary widely. The ojbects in question include:

- ControlFile
- Database
- Database Link
- Role
- Profile
- Rollback Segment
- Snapshot
- Snapshot Log
- Etc.

2.4.1. Database Name

A database name (SID) ideally should be only 3 characters in length, e.g. DEV, PRD, TST. With many databases longer names may be necessary, but names like "JONESCO.PROD1" could easily be P1 with no loss of meaning. The basic rule is that SIDs should be short, concise, and easily recognizable. You should avoid embedding host names with the database, e.g. (MYCOMPUTER.PROD1). If you have lots of instances try devising a naming scheme, e.g.

[Plant][Use][Env]

where Plant refers to the plant or department, Use might be **w** for warehouse, **o** for operational data, and Env **d**, **t**, or **p** for (you guessed it) development, test and production. Thus the M3B plant might have **m3bwd** as the SID for its warehouse development environment.

2.4.2. Database Control Files

Use control**nn**.ctl where **nn** is a two-digit integer starting at 01 to uniquely identify the copy of the control file. A minimum of 2 control files are created for each database. Control files will be stored in a directory of the form /u**xx**/oradata/<sid> where **xx** is another two-digit integer.

2.4.3. Redo Log File

On-line Redo logs will be named redo**nn**.log where **nn** is a two digit integer to uniquely identify the redo log group. Only one redo log file will exist per redo log group: On production systems, the disks are already mirrored, and on development systems, redo log mirroring is not justified. Redo logs will be stored in a directory of the form /u**xx**/oradata/<sid> where **xx** is two-digit integer., and <sid> is the database system id.

Archive logs will have the archive log format specifier "%s.log" (see admin guide) and the archive destination specifier "/u**xx**/oraarch/<sid>/arch", where sid is the database system id. Archive log mode will be used for all databases that require backing up. No databases will be brought down on a scheduled basis for backups. A nightly export system will exist (and is currently in place, see MSA DBA Operations Guide) and will be used for databases where that type of backup is deemed appropriate by the DBA.

2.4.4. Tablespace

Use the format <*purpose*> or <*purpose*>**nn** where nn is an optional two-digit integer starting at 01. Examples of tablespace names are SYSTEM, RBS01, USER, CASE01. A tablespace is usually designated for indexes, e.g. USERX, or CASE_X. The standard set of tablespaces is USER and SYSTEM. Try not to use these for specific applications. Set up application-specific tablespaces which have the application acronym as the part of the name, e.g. MFG01.

With Designer/2000 Release 1.3 it used to be a good idea to specify Tablespaces as substitution variables, e.g. &&ASSET01, which could then be pointed to appropriate tablespaces using DEFINE statements. This facilitated maintenance of development and test environments that might have a different tablespace design than the final production system. In the current release of Designer/2000 it allows for associating a different storage scheme when generating DDL targeted at different environments.

Release 7.1 or later can automatically defragment the tablespace. The trick is to ensure that the PCTINCREASE in the default storage of a particular tablespace is a non-zero value. By default, the PCTINCREASE is 50%. Use PCTINCREASE 1.

2.4.5. Database File

Database tablespace data file names will be in lower case, and will have the format "<tbs>**nn**.dbf" where <tbs> is the tablespace name (optionally abbreviated) with underscores eliminated, and **nn** is a two digit integer. Examples of tablespace datafile names include "system01.dbf" and "casex03.dbf".

2.4.6. Rollback Segments

The standard set of rollback segments include rbs01, rbs02, rbs03, and rbs04. Optionally, an rbsbig may exist and be brought online by a batch process. Rollback segments in general are set up by a DBA and their names and uses are set by a DBA on a case by case basis.

2.4.7. Database Control Files

Use control**nn**.ctl where **nn** is a two-digit integer starting at 01 to uniquely identify the copy of the control file. A minimum of 2 control files is created for each database. Control files will be stored in a directory of the form /u**xx**/oradata/<*SID*> where **xx** is another two-digit integer.

2.5. Database Objects Naming Standards

Database objects include

- Tables
- Views
- Columns
- Indexes
- Synonyms
- Procedures
- Packages
- Triggers
- Schemas (Users)
- Sequences
- Roles
- Foreign Key Constraints
- Primary Key Constraints
- Database Control Files
- Redo Log File
- Cluster (in conjunction with DBA only)

2.5.1. Tables

Table names should be short but meaningful. Although Oracle allows up to 30 characters, table names should normally be between 12-18 characters and never more than 26. Use standard abbreviations[4] where possible and use underscores to separate words (e.g. MFG_SPECS instead of MANUFACTURING_SPECIFICATIONS or MFGSPECS).

Avoid generic meaningless table names like DETAIL_DATA, TMP3 or NEWSTUFF.

The CASE repository default is to use plural table names, e.g. ORDERS based on the entity ORDER, and this should be followed.

Table names must be unique throughout an application and describe their full significance within the organization, not just within the application. For example, within a purchasing system, use PURCHASE_ORDER_SENT or PURCH_ORDER_SENT instead of just ORDER.

Within an organization a table should be uniquely named for practical reasons of managing the database. The logical reason for this is if two applications contain tables with the same name they should logically contains the same information and should be sharing the same data. However, in the real world it may not always be possible to closely integrate tables in this way and the naming conventions must allow for this.

Table names should follow these rules:

- Table names should be plural (correctly spelled)
- Table names should be prefixed by application acronym
- Table names will use underscores between words
- Table names should not be longer than 26 characters
- Tables generally should not have more than 20 - 30 columns
- Each table should have comments
- Table storage definitions should be appropriate to anticipated table use; it is normal for

[4] See Appendix A.

storage clauses to be adjusted over time
- Every table should have a primary or unique key, preferably numeric single column
- If no reasonable key can be constructed from data, use surrogate enforced by Oracle sequence. If loading legacy data, the sequence can start high to allow for externally setting the key in the initial data load.

2.5.2. Table Short Names

Table Short Names or *Table Aliases* (I use these interchangeably) will usually be identical to the entity short names that the tables implement. The exceptions to this are tables that implement more than one entity, are a vertical or horizontal partition of an entity or did not exist as entities at all.

Table names are given a shorter alias (*table short name*) for use in prefixes or as components of index, primary key or foreign key names.

The table short name (alias) should be formulated in exactly the same fashion as used to determine entity short names. (See Page 7).

For example:

TABLE NAME	ALIAS
PROCESSES	PROC
PROCESS_REGISTERS	PRRE
PROCESS_PARM_REGISTERS	PPRE
PROCESS_PARM_REGISTER_TYPES	PPRT

2.5.3. Column Names

2.5.3.1 Columns should honor the following guidelines:

- Column names must be unique within a table.
- Columns names should not be longer than 26 characters
- Column names should be meaningful, use standard abbreviations, and use the correct datatype
- Always use VARCHAR2 for text fields, even for columns of with a data length of 1 – VARCHAR2(1). The CHAR datatype should not be used unless COBOL programs are involved (and even then, why not make the programs handle the translation rather than using CHAR instead of VARCHAR2?).
- NUMBER columns should use correct precision, e.g. NUMBER(14). Don't use NUMBER without a precision. This is because the NUMBER datatype defaults to NUMBER(*max*), where *max* is system dependent. Migration to a new machine could affect the database.
- Queryable columns should be forced to upper case.

Column names will NOT be prefixed by the table short name. Not that the Designer/2000 (1.3) Database Design Wizard will prefix all column names by default unless this function is switched off. Designer/2000 2.1 has an option for prefix generation for column names. This should be off.

2.5.3.2 *Table-specific column prefixes*

Table-specific column prefixes (Table Alias Prefix) should not be used because:

- They lengthen the column name unnecessarily
- Not using column prefixes strongly encourages adherence to naming standards
- Adherence to SQL programming standards can eliminate the need for column prefixes if the use of aliases is required within each SQL statement.

Columns prefix is an option that can be set when you run the Database Design Transformer in Designer/2000. Make sure it is deselected prior to running the DDT.

2.5.3.3 *Standard Suffixes*

It is useful to use a standard suffix to identify common types of fields. The use of these is optional, but a consistent approach should be used throughout a given application. They should always be used in "Corporate Tables".

`_CODE`	a code that is known and used by a user to identify an object (usually an alphanumeric code), e.g. MFG_CODE.
`_NBR`	A number code that is known and managed by the user, e.g. PART_NBR.
`_ID, ID`	A system generated unique number used internally within the application and not usually known by the user. This is usually generated by a sequence. For primary key columns, the column name will be <*table alias*>`_ID`; for foreign key columns, the column name will be <*referenced table alias*>`_ID`.
`_FLAG`	A field used as a Flag field to indicate a specific sub-type or role as identified during the analysis phase.
`_FLAG_YN`	A flag for a field in which 'Y' or 'N' are the only allowed values (default Y)
`_FLAG_NY`	A flag for a field in which 'Y' or 'N' are the only allowed values (default N)
`_DATE`	`Always postfix a date field, i.e. use SHIP_DATE, not DATE_SHIPPED`

2.5.3.4 *Primary Key Column Names*

Numeric single-column primary key names should be <*table alias*>_ID, e.g. EMP_ID.[5] For a surrogate key column, this is the default setting in Designer/2000. If this option is deselected, no prefix is generated and the surrogate key column name is set to 'ID'.

[5] I used to prefer simply ID because in a SQL statement the table alias is generally employed, e.g. `SELECT .. FROM EMP.ID ..` is clear and less messy than `SELECT .. FROM EMPL.EMPL_ID ..` Also this distinguishes primary keys from foreign keys which always have the table alias prefix. However, as it is common practice to name a PK column with the table alias prefix I will bow to that custom. Either way is acceptable as long as it is consistently used throughout the enterprise.

2.5.3.5 *Foreign Key Constraint Column Names*

Foreign key columns will be the referenced primary key column prefixed with the referenced table's short name: *<referenced table alias>_<referenced primary key column>*, e.g. a foreign key column from EMP to DEPT should be named EMP_DEPT_ID. This is the default setting for the "Database Design Transformer" in the current Designer/2000 release.

An advantage of including referenced (parent) table names in the foreign key name is so that potential joins can be spotted quickly and the columns involved identified. This aids system maintenance and QA significantly.

In the case where a primary key occurs in a table as the foreign key more than once, you must ensure that some indication of the relationship is added to the column name so that they can be distinguished.

2.5.3.6 *Audit Columns*

Audit Columns, often called the "Fab four" or "Who" columns should exist on every table unless the table is read-only. They may have other names then those listed below. Just be consistent. These columns are as follows:

```
CREATE_DATE          DATE         DEFAULT SYSDATE   NOT NULL
CREATED_BY           VARCHAR2(30) DEFAULT USER      NOT NULL
UPDATE_DATE          DATE                           NULL
UPDATED_BY           VARCHAR2(30)                   NULL
```

Note: A trigger on the table can be created to prevent unauthorized changes to the CREATE_DATE and CREATED_BY fields and to automatically populate the UPDATE_DATE and UPDATED_BY fields.

2.5.4. Column Order In Table

The ordering of columns within tables should be:

- not null columns should be first in table (usually PK, then FK)
- Audit columns (not null first)
- LONGS or VARCHAR2 with a length > 80 should be last in table.

If you are expecting to populate a lot of NOT NULL columns after initial row insertion it can be efficient for space usage to put those columns before the last NOT NULL column in the table.

2.6. Constraints

2.6.1. Primary Key Constraints

- Primary key constraint names will be of the form *<table alias>***_PK.**
- Each table should have a primary key – database will enforce with unique index (DDL to create key should specify using index)

- Where possible, primary keys should be numeric non-intelligent columns populated by sequences
- Rules for non-numeric keys will be covered in the expanded document.
- Composite primary keys are discouraged
- Rules for composite keys: type of columns, number of columns, order of columns will be discussed in the expanded document.
- Rules for use of primary keys vs. unique keys will be discussed in the expanded document.
- The database max of 16 key components is way too high
- Scripts to create primary key constraints should be created separately as ALTER TABLE scripts rather than in-line constraints.

2.6.2. Unique Key constraints

- Unique key constraint names will be of the form <*table alias*>**_UK** or <*table alias*>_<*meaningful name*>**_UK**.
- A unique constraint allows null column values so long as the other component remains unique across the entire data set.
- As with primary keys, the database creates indexes to enforce the unique constraint.

2.6.3. Foreign Key Constraints

- Foreign key constraint names will be of the form <*referencing table alias*>_<*referenced table alias*>**_FK**
- For multiple keys append number to end of name: <*alias*>_<*alias*>**_FK**.
- The foreign key column would be <*referenced table alias*>_ID
- Scripts to create foreign key constraints should be created separately as `ALTER TABLE` scripts rather than in-line constraints.
- rules for creating indexes on FKs will be covered in the expanded document.

2.6.4. Check Constraints

Check constraint names will be of the form <*table alias*>_<*meaningful name*>**_CC**. The meaningful name is a mnemonic for the function of the for the check key constraint, e.g. PROC_START_B4_END_CC to enforce a start date before an end date.

2.7. Indexes

Primary Key and Unique Indexes will be created as the direct result of the appropriate referential integrity constraint and will therefore be named for the constraint. Any additional indexes created will be named in a similar fashion to constrain names, in the form <*table alias*>_<*meaningful name*>**_X**. The meaningful name says something about the purpose of the index. In those cases where the index is provided to access the table by a specific foreign key use be the short name of the table for which this is a foreign key.

- Unique indexes that are used to enforce primary or unique constraints should be created by the database via then USING INDEX clause on constraint definition
- Enforcing index names will have the same name as the constraint that they enforce

- Non-unique indexes should be named: <*table_alias*>**_I**
- Non-unique indexes should be established for query purposes
- There is a need to establish boundaries for when to allow indexes, balance query performance vs. insert, update
- Index storage definitions should be appropriate to anticipated table/index use; it is normal for storage clauses to be adjusted over time
- Indexes should be created in a separate tablespace from the associated data
- **Note** if the index is created with no data in the table then the storage parameters, PCTFREE and PCTUSED are ignored. Index storage needs to be reviewed after installation.

JS: This gets into tuning & use of indexes. Some stuff is in white paper, e.g. overloaded index, bit-mapped index and optimizer usage, etc.

2.8. Views

View names conform to the same general requirements as table names**. Their names should reflect the function of the view, not the names of the underlying tables**, and will be suffixed with `_V` to distinguish them from tables, unless the view defines a subtype of the base table.

Where a view has been created to implement a sub-type, or to provide a horizontal and/or vertical partition of a table (typically as a result of security requirements), a synonym will be created on the view, omitting the `_V` suffix. The application should always access the synonym, rather than directly accessing the view, in order to avoid change if the view is ever instantiated. For example EMPLOYEE can be a subtype of the PARTY entity and is a synonym to the `EMPLOYEE_V` view on the PARTIES table.

It will be usually be necessary to create a number of complex join views to enhance application and network performance. These views will always be accessed using the _V suffix, in order to alert users to the fact they are using a view.

- View names should be prefixed by application acronym and suffixed with `_V`
- View names will use underscores between words
- View names should not be longer than 26 characters
- Views should be limited in the number of columns they return
- View should not have a large number of columns
- Each view should have comments.

2.9. Temporary and Working Tables

Temporary tables are created and dropped for temporary usage by applications and their names should be of the form <*meaningful name*>**_TMP,** e.g. INV_SEARCH_TMP.

Working tables are permanent tables that hold transient data. They should have names of the form <*meaningful name*>**_WRK,** e.g. INV_SEARCH_WRK. It is expected that working tables be maintained by a given application which is responsible for populating and deleting rows. There can be significant system overhead on loading and unloading working tables. Use TRUNCATE whenever possible instead of DELETE FROM.

Temporary and work table names should be prefixed with the application system acronym.

2.10. Synonym Naming and Use

2.10.1. Public Synonyms

Public synonyms should be the primary means by which the application system locates application database objects.

2.10.2. Private Synonyms

Private synonyms are used principally within development and testing databases where the likelihood of multiple copies of application objects existing in multiple schemas is high and hence the value of public synonyms is low and potentially transient. (Remember private synonyms take precedence over public synonyms.)

2.10.3. Naming Conflicts

Synonyms will be used as required to resolve naming conflicts between custom-developed and third party database objects that reside in the same Oracle database.

2.11. Sequences

Sequence names will be of the form *<table alias>*_*<attributed name>*_**SEQ**, e.g. MASP_ID_SEQ.

- sample select statement
- What about dev to test to prod?

2.12. Database Triggers

Trigger names will be of the form *<table alias>*_*<trigger type>*_**TR***n*, where **n** is an optional identifier starting at 1, e.g. PART_BUT_TR1 (*PARTY table before update trigger #1*; future releases of Oracle will allow multiple triggers of the same type).

Trigger types execute on database events:

- Before or After Statement (B or A)
- Insert, Update or Delete (I, U or D)
- For each Row or for the entire Table (R or T)

Therefore *trigger type* would be one of the following: BIR, BIT, AIR, AIT, BUR, BUT, AUR, AUT, BDR, BDT, ADR, ADT.

Note: Definitely avoid putting code directly in triggers unless it is very simple, i.e. limited to 10 lines or less. Use packages instead and call these from the trigger.

2.13. PL/SQL Packages

Packages will be named to reflect their utility, e.g. DW_UTILS.

See Feuerstein

2.14. Domains vs. Codes

2.14.1. Guidelines for Defining Domains

Domains should not be defined for highly volatile codes and/or descriptions. Instead, more traditional code tables should be used to maintain these values.

Further, domains should not be defined for code tables that will hold more than 250 values. In such cases, the List of Values processing which would be performed on domains would result in unsatisfactory performance.

2.14.2. Guidelines for Generating Domains

The domain values MUST BE loaded before a module, which refers to the domain, can be run.

The lengths of several columns have been shortened, to agree with the maximum lengths that can be specified via CASE.

2.15. Coding Standards (SQL and PL/SQL)

3. Programming Standards

3.1. Coding Principles

1. Code must be readable (and well-commented) to be maintained.
2. Tools such as Oracle Forms, Oracle Reports and PL/SQL are used whenever possible. Avoid using PRO*C, user exits, and other complexities or other languages.
3. Fast performance is critical on a network, particularly a WAN.
4. Software must be properly versioned and archived.
5. Avoid platform-specific code unless absolutely necessary.
6. Platform-specific code should be modularized and the interface well documented.
7. All programs should be subject to design walk-through prior to coding and code walk-through after coding.
8. Employ reusable code wherever possible.

3.2. Coding Standards

1. When coding in PL/SQL always use packages, not procedures and functions. Explain further.
2. Use standard prefixes for sub-system names, then use one, two or three words to name the package, like SPEC_NAME_SEARCH. Try to use descriptive verb or noun procedure and function names, like SEARCH or GET.
3. Always define PRAGMA RESTRICT_REFERENCES for all packages.
4. Use uppercase for PL/SQL and SQL verbs, lower case for everything else.
5. Include a Version function in every package that returns the Version string (e.g. from PVCS, $Header$).
6. Do not create unwieldy cross-reference packages. If there are common routines, pull them out into common packages. Most normal packages should have only a few entry points and only a few variables exposed.
7. Always populate the "Fab Four" (often called the “Who” fields) audit columns, unless the table is strictly insert only, read only, or for some other overriding reason

4. Design Standards

5. File Naming

5.1. File Extensions

Always use the default file extensions expected by the Oracle Products where appropriate. The following list is not exhaustive (yet).

Tool/Application	File Type	File Extension
SQL*PLUS	procedures	**.sql**
	spooled files	**.lst**
	table definition source files	**.tab**
	constraint definition source files	**.con**
	index definition source files	**.ind**
	sequence definition source files	**.seq**
SQL*REPORT	Definition files (binary)	**.rdf**
	Definition files (text)	**.rpf**
	list files	**.lis**
Reports 2.5	source files - binary	**.rdf**
	source files - textual	**.rex**
	Executables	**.rep**
Forms 4.5	source files - binary	**.fmb**
	source files - textual	**.fmt**
	Executables	**.fmx**
	menu source files - binary	**.mmb**
	menu source files - textual	**.mmt**
	menu executables	**.mmx**
PL/SQL	library files - binary	**.pll**
	library files - textual	**.pld**
	library files - executable	**.plx**
	Database procedure source files	**.prc**
	Database trigger source files	**.trg**
	Database packages source files	**.pkg**
	Database packages definition source files	**.pls**
	Database packages body source files	**.plb**
SQL*Loader	control files	**.ctl**
	log files	**.log**
	input data files	**.dat**
	'bad data' files	**.bad**
	discard files	**.dsc**

Appendix A. Some Abbreviations and Acronyms

All sites should establish something like this. You don't have to use these, but you do have to use something, although I am particularly partial to avoiding _NO for "number" and using _NBR instead.

Acronym or Abbreviation	Meaning/Full Word
ABBR	Abbreviation
ACC	Account
ACK	Acknowledgement
ADDR	Address
ADMIN	Administration
AI	Activity and Inventory
AMT	Amount
ASS	Asset
AVG	Average
BAL	Balance
CLI	Client
CNT	Count
CORP	Corporate
CTRY	Country
CTY	County
DOB	Date of Birth
EFF	Effective (e.g. EFF_DATE)
EMP	Employee
EXT	Extension
FRQ	Frequency
HIST	History
INC	Income
INIT	Initial
INSP	Inspection
IS	Information Services
IT	Information Technology
LEN	Length
MES	Manufacturing Execution System

Acronym or Abbreviation	Meaning/Full Word
MFG	Manufacturing
MON	Month
NBR	Number
NO	"**NO**" as in Negative (Number is abbreviated NBR)
ORIG	Original
PARTY	Party
PCT	Per Cent
PHONE	Telephone
PMT	Payment
PREV	Previous
PROD	Product
QUAL	Quality
RCPT	Receipt
RES	Residence
SEQ	Sequence
SPEC	Specification
TLA	Three Letter Acronym
TOT	Total
VAL	Value
YR	Year

Appendix B. Code Tables

Where a number of lookup codes are required in an application system a single master table should be used instead of many small tables. Views are used to reference distinct code types. A Master Code Table is often a Corporate table and its contents are subject to strict change control. Maintaining a single code table centralizes the maintenance of corporate codes.

A sample master code table follows:

```
CREATE TABLE CODES
(
  CODETYPE          VARCHAR2(10)                         NOT NULL,
  CODE              VARCHAR2(10)                         NOT NULL,
  CREATED_ON        DATE          DEFAULT SYSDATE        NOT NULL,
  CREATED_BY        VARCHAR2(30)  DEFAULT USER           NOT NULL,
  LAST_UPDATED_ON   DATE,
  LAST_UPDATED_BY   VARCHAR2(30),
  DESCR             VARCHAR2(100),
  ALIAS             VARCHAR2(10),
  END_DATE          DATE
)
```

Each distinct CODETYPE groups codes into a generic group. The CODETYPE='**CODE**' contains the names of all generic code types contained in the CODES table. A discontinued code has an end date. A discontinued code that may be in the table for referential purposes, but is superseded for INSERT, would have an ALIAS defined that pointed to the up-to-date code value. Note that a unique constraint must exist on ALIAS and CODETYPE.

Constraints on the CODES table are that all CODETYPES are members of the CODETYPE='**CODE**' group and that each ALIAS exists as a valid code and has an end date. Note the presence of the “who” columns used for an audit trail. Codes should always be modified by proper authority after impact analysis and should never be deleted once in production. The Primary Key for this table is CODETYPE + CODE. ALIAS along with its CODETYPE is a self-referencing foreign key constraint.

As needs require other columns could be a part of this table, e.g. an ALLOWABLE_LENGTH, LONG_DESCR or a DISPLAY_FLAG.

A particular code type often represents a domain, and is referenced from a view, e.g. an LOV for all currently valid CUSTOMER TYPE codes would be populated by a ‘SELECT *’ from the following view.

```
CREATE OR REPLACE  VIEW cust_type_v
AS
   SELECT code,
          descr
     FROM codes
    WHERE codetype = 'CUST_TYPE'
      AND alias IS NULL
      AND end_date IS NULL
/
```

An example of a code table is:

```
SQL> select codetype,code, descr, alias from codes
  2  order by DECODE(CODETYPE,'CODE','  ',CODETYPE)
  3  /

CODETYPE   CODE       DESCR                ALIAS
---------- ---------- -------------------- ----------
CODE       CUST_TYPE  Customer Type
CODE       CITY       City Code
CITY       SEA        Seattle
CITY       PDX        Portland
CITY       LAX        Los Angeles
CUST_TYPE  GOOD       Good Customer
CUST_TYPE  BAD        Bad Customer
CUST_TYPE  UGLY       Very Bad Customer    BAD
```

The SELECT statement below will return the correct code and description even if an obsolete code is used as the target code.

```
SELECT code, descr
  FROM codes
 WHERE codetype = 'CUST_TYPE'
   AND code = :target
   AND alias IS NULL
   AND end_date IS NULL
UNION ALL
SELECT code, descr
  FROM codes
 WHERE codetype = 'CUST_TYPE'
   AND code IN ( SELECT alias
                   FROM codes
                  WHERE codetype = 'CUST_TYPE'
                    AND alias IS NOT NULL
                    AND code = :target)
```

It is easy to see that this concept can support a number of variations on the theme including triggers, a package and stored procedures for query and insert. It is not unreasonable to restrict a code table like this to query and insert only, keeping the audit trail for all discontinued codes. Triggers could enforce particular constraints (e.g. an ALIAS must also exist as a valid CODE, new codes must conform to a particular length, or be uppercase, etc.).