**DOCUMENT CONTROL**

Data Architecture Standards and Guidelines

Version # : Draft A

Released on :

Author :

**DOCUMENT HISTORY**

| Version | Date | Additions / Modifications | Prepared / Revised By |
| --- | --- | --- | --- |
| 1.00 |  | Initial |  |
| 1.01 |  | Added Domains and Housekeeping |  |
| 1.02 |  | Added DataStage & Outbound File sections,  Updated intended audience |  |
| 1.03 |  | Updated Datastage section and added KSH section |  |
| 1.04 |  | Added naming conventions for DataStage jobs |  |

**Contents**

1. Introduction 4

1.1. Purpose 4

1.2. Scope 4

1.3. Intended Audience 4

2. PL/SQL Coding Conventions 5

2.1. Documentation Standards and Formatting 5

2.2. Comments 7

2.3. Naming Conventions 7

2.4. Error Handling 8

2.5. Temp Tables/Work Tables 8

3. Modeling Standards 9

3.1. Logical 9

3.1.1. Entity 9

3.1.2. Attribute 10

3.1.3. Class Words 11

3.1.4. Data Type 11

3.1.5. Domains 12

3.2. Primary Key and Unique Constraints 12

3.2.1. Primary Key format 12

3.2.2. Foreign Key format 13

3.2.3. Alternate Key format 13

3.2.4. Index format 13

3.2.5. Check Constraint format 13

3.3. Relationships and Cardinality 14

3.3.1. Relationships 14

3.3.2. Cardinality 14

3.4. Abbreviations 17

3.4.1. Attribute Names Abbreviated 17

3.4.2. Column Names Abbreviated 17

3.5. PHYSICAL 18

4. DataStage 21

4.1. Comments 21

4.2. Variables 22

4.2.1. DataStage Stage Variables 22

4.2.2. DataStage Variables Standards 22

4.3. .DSX Naming 22

4.3.1. DataStage Job Naming 22

5. Unix KornShell 24

6. Outbound Files 25

6.1. Encryption 25

7. File Naming conventions 26

# Introduction

## Purpose

This document describes development standards and guidelines for PL/SQL coding and data modeling.

## Scope

The Scope of this document is intended for BCBSRI. It is not specific to a particular team or application. It should be used for all new development and, whenever possible, applied to maintenance.

## Intended Audience

The target audience is comprised of Database Administrators, developers and reviewers of PL/SQL, KSH and DataStage.

# PL/SQL Coding Conventions

## Documentation Standards and Formatting

PL/SQL can be used to code stored procedures, packages or standalone program units. The following standards would complement the standards for stored procedures and packages. Standards already listed for SQL query language would take effect whenever SQL code is used with in PL/SQL:

**The standard header for all PL/SQL programs should contain:**

* Program Name
* Program Purpose – a description of why the program was written
* One revision history line for each revision. Each line should contain the author, date, version number, and description of the change

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Program Name : <name of package, procedure, etc>

Purpose : <why was the program written>

Revision History:

<author> <date> <version> <description EX:Initial release>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**A sample package and procedure shell is shown below:**

CREATE OR REPLACE PACKAGE P\_LOAD\_AGE\_RANGE\_STG AS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Program Name : P\_LOAD\_AGE\_RANGE\_STG

Purpose : this package holds all procedures used to create the age range staging table

Revision History:

C. Hogan 1/21/07 1.0 Initial release

C. Hogan 2/22/07 1.1 Added new proc sp\_calc\_infants

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- put global var's here that can be used in any of the procedures

-- EXAMPLES SHOWN BELOW:

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

gn\_JobId NUMBER;

gs\_ProcedureStatus VARCHAR2(4);

gs\_FieldName VARCHAR2(30) := 'RECORDS';

gs\_CtlFileHandle UTL\_FILE.FILE\_TYPE;

gs\_FileLocation VARCHAR2(30) := '/ora\_ftp';

gs\_FileExtension VARCHAR2(15) := '.TXT';

gs\_FileMode VARCHAR2(1) := 'r';

gdt\_HIGH\_DATE DATE;

gi\_CursorId INTEGER; -- Cursor Id

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- list procedures included in this package (example below)

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Program Name : SP\_LOAD\_AGE\_RANGE\_STG

Purpose : this proc loads ages from a cursor to a staging table

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ PROCEDURE SP\_LOAD\_AGE\_RANGE\_STG;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- this is the end of the package header

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

END P\_LOAD\_AGE\_RANGE\_STG;

/

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- this is the beginning of the package body

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CREATE OR REPLACE PACKAGE BODY P\_LOAD\_AGE\_RANGE\_STG AS

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- this is the beginning of the procedure listed in the package header

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Program Name : SP\_LOAD\_AGE\_RANGE\_STG

Purpose : this proc loads the ages from a cursor to a staging table

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ PROCEDURE SP\_LOAD\_AGE\_RANGE\_STG AS

DATE\_MISMATCH EXCEPTION;

FILE\_NOT\_FOUND EXCEPTION;

OUT\_OF\_BALANCE EXCEPTION;

REFRESH\_INCOMPLETE EXCEPTION;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- actual start of the procedure

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BEGIN

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- put local var's here

-- EXAMPLES SHOWN BELOW:

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ls\_ProcedureNum := '0500'; -- !!!

ldt\_HIGH\_DATE := TO\_DATE('99991231', 'YYYYMMDD');

li\_RowStartCount := 0;

ls\_PkgProcName := 'P\_LOAD\_TABLE\_NAME\_STG.SP\_LOAD\_TABLE\_NAME\_STG;'; -- !!!

lb\_DATE\_ERROR := FALSE;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- put procedure logic here

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

...

...

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- generic wrapper exception handling covers entire procedure

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EXCEPTION

WHEN FILE\_NOT\_FOUND THEN

...

WHEN OUT\_OF\_BALANCE OR REFRESH\_INCOMPLETE THEN

...

WHEN FATAL\_ERROR THEN

...

WHEN OTHERS THEN

...

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- this is the end of the procedure listed in the package header

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

END SP\_LOAD\_AGE\_RANGE\_STG;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- this is the end of the package body

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

END P\_LOAD\_AGE\_RANGE\_STG;

/

## Comments

**Block Comments**

Use block comments to indicate key elements or sections of the code such as loops. Example:

**/\***

**\*** If the hash table is empty, populate it first before extracting

**\*** the field value this will only happen once, and all further

**\*** queries will simply hit the hash table

**\*/**

**Single-line comments**

For comments such as quick notation not intended to span more than one line, use the following form:

**--** Position the cursor variable

**Inline Comments**

These type of comments are placed usually on the same line as a statement and are particularly useful in lists such as:

* Variable declarations
* List of arguments to procedure/function where these are vertically aligned

Additionally, inline comments should be used for heavily nested code to clarify the termination of a block. For example:

END LOOP  **--**  End of while loop

END LOOP **--** End of for loop

END IF **--** End of if test

END LOOP  **--** End of outer loop

## Naming Conventions

* All the package names should start with *P\_*.
* All the procedure names should start with *SP\_*.
* All the Functions should start with *SF\_*.
* All the IN argument names should start with *pi\_*, OUT argument names should start with *po\_*, and all the IN OUT argument names should start with *pio\_*.
* All the local variable names should start with the letter *l*,

*for example: li\_ for NUMBER( n)*

*ln\_ for NUMBER(n,m)*

*ls\_ for VARCHAR2(n)*

*lb\_ for BOOLEAN*

*ldt\_ for DATE*

*lrec\_ for ROWTYPE*

* All the TYPE variable names should start according to above guidelines and end with *\_type*.
* All the global variables should start with *g*.
* All cursor names should start with *cur\_.*

## Error Handling

* At the very least, the outer most block of a PL/SQL program should always have a WHEN OTHERS handler. This would insure that all possible exceptions are handled.
* Always code a WHEN OTHERS exception last.
* Always use exception handling code around DML statements (inserts, updates, deletes, selects, etc.)

## Temp Tables/Work Tables

* Temporary and/or work tables should be truncated as the related processing finishes

# Modeling Standards

The following standards were developed as a control document while creating the Logical and Physical models, the DDL and database(s).

The following agreed on standards are generic, not tool specific, so they can be used regardless of whether a modeling tool is used or which tool is used.

## Logical

The logical model will contain the underlying structure of an enterprise’s data and the business rules governing it. Like any model the logical model is the representation of the real thing; it addresses the business and functional requirements of a system. The ultimate goal is to create a data application that can support some or all of your enterprise.

It is important to clearly design data structures and the relationships between them. A data model of your business can help define operational aspects, be an invaluable communications tool assisting in the orientation of goals and operations to employees, and ultimately the development of a physical model.

### Entity

An Entity is an object that defines for the business a person, place, thing, object or department.

Entities are real objects, such as people, places, a concept or things for which the organization wishes to keep that are relevant to your enterprise and which can be uniquely identified. In their physical implementation entities become tables.

Entities are the starting point for a data model. An entity will have a descriptive name, a definition that clearly explains why this entity is needed and how it is to be used. Since an entity is an aggregation of attributes, once an entity’s purpose has been carefully defined its attributes should become apparent.

An entity shall have the following characteristics:

* Be of interest to the organization
* Represent a single object based on its normalized attributes
* Represent more than a single instance of data
* Follow the rules of normalization appropriate for the level and type of model
* Be uniquely identifiable

##### Entity Name

The entity name will tie to the business. The name of the entity should describe what and or how it is used in its name. The name shall be a singular noun and may contain one or more modifiers for clarification. The more you understand about its use and what will be held in it the easier it is to name an entity. For example the MEMBER entity contains information about an applicant and all his/her eligible dependents. It will contain information about the member that is needed by the company to do business with them. (Name, coverage dates, address, birth date etc.)

The format of an Entity name will be:

* A Noun and may include one or more modifiers for clarification
* CAPITALIZED
* Words separated with spaces
* Maximum length of 65 characters

Examples:

**PROVIDER**

**RITE CARE PROVIDER** (contains subset of data specific to only RiteCare applications)

**MEMBER**

**CONTRACT TYPE**

**GROUP ACCOUNT**

**GROUP PACKAGE PRODUCT**

**GROUP PACKAGE CONTRACT**

##### Entity Definition

The definition of an Entity is an explanation of the meaning and importance of the entity to the business in sufficient detail to differentiate from other entities. The definition shall be composed of a scope and qualifier. The scope identifies the overall definition of the entity while the qualifier narrows the definition to the specific details. Give examples where appropriate. For reporting ease the entity name shall appear in the definition and be capitalized.

Examples:

**PROVIDER**

PROVIDER is a person, or entity, which provides a service to healthcare, contracted members.

**GROUP ACCOUNT**

GROUP ACCOUNT is an enrolled population of subscribes to which health legal care programs are marketed.

Example: 009900=Blue Cross Blue Shield.

**MEMBER**

MEMBER is an applicant and all his/her eligible dependents.

**CONTRACT TYPE**

CONTRACT TYPE identifies the different Contract Types that a MEMBER can select for coverage.

Example: S1=Single, F1=Family.

### Attribute

An Attribute (attribute type) is a descriptor of an entity. As an entity is an aggregation of attributes an attribute is a piece of the whole (entity). Attributes contain data used by the corporation, internal, or external users. This data could be used to perform a function, needed for calculations or to query for information. The name of the attribute is derived from describing the attribute.

##### Attribute Name

The format of an Attribute name will be:

* Begin with a noun and may include one or more modifiers for clarification
* End with a class word.
* Title Case, the beginning of each word upper case
* Spaces between words
* Maximum length of 50 characters

Examples:

**MEMBER NUMBER**

**MEMBER EFFECTIVE DATE**

**TOTAL EXCLUDED CLAIM AMOUNT**

**PROVIDER TYPE DESCRIPTION**

DATA WAREHOUSE UPDATE DATE

##### Attribute Definition

An attribute shall have only one meaning, and only one attribute within the model may represent that meaning. The definition of an Attribute will contain the Attributes name and clearly define how this attribute will be used within constraints of the Entity, the company and this application. Examples of what would reside are helpful. If an entity is referenced by name in the definition it should be capitalized.

Examples:

MEMBER NUMBER

Member Number is a unique number assigned to a MEMBER for identification purposes.

**MEMBER EFFECTIVE DATE**

The Member Effective Date stores the date, and time where available, where the instance became true within the business.

**PROVIDER TYPE DESCRIPTION**

Identifies the different types of Provider's. Valid Values are Person, Institution, or Ancillary.

**CONTRACT TYPE DESCRIPTION**

Contract Type Description is a free form textual narrative identifying the different contract types for a member's benefit selection.

**DATA WAREHOUSE UPDATE DATE**

The Data Warehouse Update Date will define the date, and time where available, where the occurrence was extracted from the source system. There will be an additional attribute created (Data Warehouse Write Date) if the actual warehouse population date is needed.

### Class Words

A Class word is a way of building classification into the attribute and column names. All attributes / columns will end with a class word. There is relationship between the class word and type of data that resides in this attribute. Such as the class word “Date” indicates the data in this field will be a date and the domain will be DATE. The Class word and the domains often share the same word.

Examples:

**Class Word Definition**

**Code** Character string representing another value of information.

**Description** Free form textual narrative of the field’s characteristic.

**Amount** Currency

**Name** Word or words that defines(s) what something is commonly known as.

**Date** Represents a calendar date.

### Data Type

The data type indicates the type of data and the format of data to be stored in an attribute / column within an entity / table. The data type is a template of what could reside in this field: characters, numbers, date/time and allowing for the maximum size.

Examples:

**VARCHAR2(100)** - contains alphanumeric (a-z, 0-9, or special characters) and the field could be 1 character long or up to 100 characters long.

Product Description PROD\_DESC (Dental – Enterprise – Full Platinum)

**CHAR(50)** - contains alphanumeric (a-z, 0-9, or special characters) and the field will always reserve space for 50 characters.

Member Last Name MBR\_LST\_NAM (Smith)

**DATE** - format mm:dd:yyyy

Member Effective Date MBR\_EFF\_DT (06/12/1994)

**NUMBER(7,2)** – contains numbers 7 in length with two decimals (99,999.99)

Rate Rider Amount RATE\_RIDR\_AMT (1250.00)

### Domains

Domains are reusable templates that promote consistent datatype definitions. Generic domains would be Datetime, ##Number, *Abc* String. You can construct domains as you would attributes, specifying a name, datatype properties, null status, default values, and validation rules. After creating domains, you can re-use them in your data model by applying them to attributes and table columns. By defining a domain, you also gain the power of propagating changes to the underlying definition to all attributes and table columns that reference it.

The main advantage of using established domains is consistency; secondary is ease of setting up the diagram. Predefined domains eliminate guessing about format and or lengths of fields. The Class Word helps establish the domain name.

Examples:

**(Attibute …. ending with Class Word Domain (for a Class word) Sample Datatype**

**Identifier** ID(10) VarChar (10)

IDNumeric38 NUMBER(38)

**Date** DATE (date only)

**Code** CD Char (1)

**Datetime** TS (date and time)

**Name (**choose the length needed) NAME50 Char (50)

NAME35 Char (35)

**Flag** FLAG Char (1)

**Amount** AMOUNT7,2 NUMERIC (7,2)

**Description** (choose the max length needed) TEXT VarCharxxx VarChar100

**Standard Domains to be used to ensure consistency within our organization**

**Create Timestamp**

**Update Timestamp**

**Create User Identifier**

## Primary Key and Unique Constraints

Each table must have at least one column or combination of columns used to uniquely identify a row in the table. This will be identified as the Primary key(s) for the table. In addition to displaying primary keys diagrammatically, named placeholders needed to enforce primary key and unique constraints will appear as index constraints. Tables could also have Alternate Keys identifying a secondary key that uniquely identifies a row(s) in the table. Primary keys and Alternate keys can be enforced with unique indexes or primary key constraints depending on the target database.

### Primary Key format

The format of the Primary Key name is as follows:

Table Name, with out the prefix "TB\_", followed with the suffix \_PK.

Example:

The primary key(s) for PRODUCT TYPE table (TB\_PROD\_TYPE) is the column PROD\_TYPE\_CD and is defined as PROD\_TYPE\_PK (PROD\_TYPE\_CD).

The primary key(s) for the GROUP PACKAGE table (TB\_GRP\_PKG) are the following columns GRP\_NUM, SUBGRP\_NUM, and PKG\_NUM they are defined as GRP\_PKG\_PK (PKG\_NUM, GRP\_NUM).

### Foreign Key format

The format of the Foreign Key name is as follows:

1. If the FK (the PK from the parent table) consist of only one column then the FK name is the column name followed with the suffix *FK##.*

Example:

The foreign key for the following column PROD\_TYPE\_CD from the PRODUCT TYPE table (TB\_PROD\_TYPE) is defined as PROD\_TYPE\_FK4 - Foreign Key (PROD\_TYPE\_CD) references TB\_PROD\_TYPE. [The number "4" indicates there are a least three other foreign key relationships from this table.]

1. If the FK parent table's PK consists of multiple columns then the FK names is the parent table's name without the prefix TB\_ and followed with the suffix \_FK##.

Example:

The foreign key for the following columns GRP\_NUM, SUBGRP\_NUM, and PKG\_NUM from the GROUP PACKAGE table (TB\_GRP\_PKG) is defined as GRP\_PKG\_FK4 -Foreign Key (GRP\_NUM, SUBGRP\_NUM, PKG\_NUM) references TB\_GRP\_PKG. [The number "4" indicates there are a least three other foreign key relationships from this table.

### Alternate Key format

The format of the Alternate Key name is as follows:

Table Name, with out the prefix TB\_, followed with \_AK and a number as needed.

Example:

The Alternate key for the GROUP TYPE table (TB\_GRP\_TYPE) is the column GRP\_TYPE\_DESC, it is defined as GRP\_TYPE\_AK1 references TB\_GRP\_TYPE.

### Index format

The format of the Index name is as follows:

IX\_<table name without the TB\_ prefix>\_<optional: column name that the index is on or an abbreviation for concatenated columns>\_<$BMP if it’s a bitmap index>.

Example:

The index for the GROUP PACKAGE table (TB\_GRP\_PKG) made up of the columns

GRP\_NUM and SUBGRP\_NUM is defined as IX\_GRP\_PKG. A primary key index consisting of the columns GRP\_NUM, SUBGRP\_NUM, PKG\_NUM and RATE\_EFF\_DT is defined as IX\_GRP\_PKG\_PK.

### Check Constraint format

The format of the Check Constraint name is as follows:

Column Name, followed with \_CK(###).

Example:

The constraint name for column CORP\_ID is CORP\_ID\_CK56, defined as CORP\_ID in ("C","B").

The constraint name for column MBR\_AGE is MBR\_AGE\_CK1, defined as MBR\_AGE >=0.

## Relationships and Cardinality

A relationship is an association between two entities based on a definable set of business rules. The relationship and cardinality between a pair of entities or tables is important to document. A line drawn between the two entities or tables represents relationships. Referential integrity can be enforced through foreign keys, constraints, or triggers.

### Relationships

It is important to document the relationship between two entities or tables. This is graphically shown by a line connecting one entity / table to another entity / table to identify the parent to child relationship. A relationship can be identifying or non-identifying. An identifying relationship becomes a primary foreign key in the associated entity where a non-identifying relationship passes just a foreign key to its associated entity. *Note in some tools non-identifying relationship lines are represented by broken lines. (- - - -).*

Equally important is the verbal description of the relationship between the two entities or tables. Business rules are helpful in creating the verbal descriptions and the model’s relationships clarify the business rules.

##### Relationship names

Use business rules to define verb phrases that describe the relationship between a pair of entities. The format for naming relationships is the combination of cardinality and the business rules. Using the Entities name, the cardinality and the business rules a sentence is created describing the interaction between two Entities.

Example:

The GROUP ACCOUNT is defined by one to many GROUP PACKAGE(s). And the reverse a GROUP PACKAGE defines one and only one GROUP ACCOUNT.

The PRODUCT TYPE defines zero to many GROUP PACKAGE PRODUCT(s). And the reverse a GROUP PACKAGE PRODUCT is defined by one and only one PRODUCT TYPE.

### Cardinality

Cardinality describes the quantitative dimension in the relationship between two entities as viewed from the parent entity. Fundamentally the following question is “How many instances in the child entity might I find for any given primary key value of the parent entity?”

Example: A GROUP PACKAGE is part of one and only one instance of a GROUP ACCOUNT and a GROUP ACCOUNT may hold zero to many instances of a GROUP ACCOUNT.

The two most commonly used diagramming techniques to show relationship and cardinality is IDEF1X and IE notation. Both use a line drawn between the two entities to show there is a relationship. Both document identifying and non-identifying relationships and zero to many, one to many, zero or one to exact number.

Where they differ is in the graphical representation of cardinality.

* IDEF1x uses filled in circle points to the parent, line alone represents zero to many and alpha characters represent one, zero, or a number to specify an exact number of instances.

See the following graphic example “Figure 1 - IDEF1X Notation”.

* IE uses vertical line to represent one, a “0” to represent zero and three lines pointing toward the entity (referred to as crows feet) to represent many.

See the following graphic example “Figure 2 - IE Notation”.

IDEF1X uses circles and characters – open circle “O” = zero, P = one or more, Z = zero or one, N = represents exact number, the filled in circle terminates at the child entity / table.

##### Figure 1 - IDEF1X Notation

**Identifying Relationship Non- Identifying Relationship**

One to zero or more One to zero or more

P

One to one or more Zero or one to zero or more

Z

One to zero or one One to one or more

P

N

One to exactly N Zero to one to one or more

Z

One to zero or more

Z

Zero or one to zero or more

N

One to exactly N

N

Zero or one to exactly N

IE notation uses 1, 0, N and crow’s feet. IE uses 1 = one and only one, 0 zero to one, 0 and Many is represented by crows feet. A solid line represents identifying relationship where non-identifying relationship are represented by a broken line.

##### Figure 2 - IE Notation

**Identifying Relationship Non- Identifying Relationship**

**A B A B**

A zero to many B; A zero to many B;

B one & only one A B one & only one A

A one to many B; A zero to many B;

B one & only one A B many not have any A

A may/may not one B; A one to many B;

B one & only one A B one & only one A

N A exactly N of B; A one to many B;

B may not have any A

N A exactly N of B;

B one & only one A

N A exactly N of B;

B may not have any A

Inheritance

Inheritance refers to the process of automatically migrating foreign keys between related entities. It is an essential feature of any data-modeling tool, as it ensures referential integrity in your logical and physical designs. Without automatic inheritance, you can accidentally omit foreign keys or define them inconsistently in different entities.

Two types of relationships propagate foreign keys:

* Identifying relationships add the parent entity's primary key as primary key attributes of the child entity. The migration of foreign keys as primary keys, can create a long chain of relationships that cascade primary keys down several levels.
* Non-identifying relationships contribute the parent entity's primary key as non-key attributes of the child entity. Not all tools show non-identifying relationships with a dotted line. In this situation only by looking at the keys in the child entities to see if the parents PKs are propagated as FK indicating this is non-identifying relationship. (ER/Studio using IE notation does not show dotted lines.)

Example:

A GROUP PACKAGE defines GROUP PACKAGE PRODUCT(s). A GROUP PACKAGE PRODUCT defined by GROUP ACCOUNT. Note the Primary Keys (PK) [identifying] in GROUP PACKAGE become Primary Foreign Keys (PK)(FK) in GROUP PACKAGE PRODUCT. But the primary key (PK) PRODUCT TYPE is only a Foreign Key (FK) [non-identifying] in GROUP PACKAGE PRODUCT.

See Figure 3 for graphic representation of above example.

##### Figure 3 - Inheritance identifying and non-identifying relationships

GROUP PACKAGE GROUP PACKAGE PRODUCT

Packager Number (PK) Product Identifier (PK)

Group Package Number (PK) defines SubGroup Number (PK)(FK)

SubGroup Number (PK) is defined by Group Package Number (PK)(FK)

Packager Number (PK)(FK)

Product Type Code (FK)

PRODUCT TYPE

Product Type Code (PK) defines

Is defined by

## Abbreviations

Modeling abbreviations will reside on BRIPNFS02\infosvcs:\DW\ERwin Abbreviation Glossaries\BCBSRI Abbreviation Glossary.nsm.

Physical model abbreviations are created by following these guidelines as closely as possible:

* Each physical field name will end in a class word
* Drop vowels from left to right, but leave the first letter if it is a vowel
* If a word is four characters or less, keep the entire word.
* Maximum length of a field name is 18 characters. If the field name is longer than 18, drop underscores from right to left.

### Attribute Names Abbreviated

Attribute names will only be abbreviated if the field exceeds 50 characters. Determine the least important word as the target abbreviation while maintaining a clear understanding of what this field is from the name. Check the corporate standard abbreviations and then the existing Abbreviation List.

### Column Names Abbreviated

Column names will be abbreviated if any word exceeds 4 characters. Corporate standard abbreviations will have first priority. Such as: Line Of Business (LOB), Treatment – TREAT. If there is not an existing abbreviation then a new one will be created using the following format and added to the Abbreviation List for future use.

Abbreviation format – the following steps explain how to create new abbreviations

* Abbreviate only words that exceed 4 characters

Exceptions-

Company or industry standard abbreviations such as:

Total (TOT), Explanation Of Benefits (EOB), Paid (PD), Status (STAT), or Year To Date (YTD)

The number of words in the attribute / column force judicial limitations in the originating attribute and the resulting abbreviations.

* Starting at the right -

Eliminate vowels unless a vowel is necessary for interpretation

(Example: Renewal (RNWL), Benefit (BNFT), Network (NTWRK),

Summary (SUMRY), or Designation (DESGNTN)

Eliminate duplicate characters unless is necessary for interpretation

(Example: Mapping (MPNG), Suffix (SFX), or Illness (ILLNSS), Title (TTL)

## PHYSICAL

The purpose of most data modeling is to generate a functional database design that meets performance and business requirements. The physical model is the interface between the logical model and the target database. The output of the physical model is the SQL / DDL source code documentation that should streamline the development process.

The physical design focuses on performance and tuning issues for a particular database platform. It is important that changes made to the physical database are documented in the physical model and synchronized with the logical model.

##### Table

A table is the physical representation of the Logical Entity, it is an object that defines for the business a person, place, thing, object or department. The physical design addresses the technical implementation of the logical model.

Tables store data about particular objects that users can query or modify. Physical characteristics are documented such as table volumes, growth patterns, Referential Integrative, (Primary and Foreign Keys), and Indexes. All of which are used to create the database source code. Relational databases are, in their physical implementation, tables are entities in first normal form or higher. Star and Snowflake schema are used to represent dimensional database. Dues to needs of the database the physical model may differ from the logical model. Sub-type entities are often absorbed in the Super-type entity or table. For ease of use views may be created consolidating a few columns from multiple tables.

Table Name

A Table name will be derived from the Entity name. All tables will have a prefix that identifies that this object is a Table and further defines the type of table. It may reference a specific subset or application with in the table name to further clarify and distinguish its usage.

The format of table names will be two to three characters followed with an underscore preceding the actual table name. Relational database the table name(s) will be preceded with "TB\_". Dimensional Models the Dimensional tables name will be preceded with "TD\_" and Fact tables will be preceded with "TF\_". Temporary tables needed for load purposes will be preceded with "LD\_". Error table names will be preceded with "TBE\_". Standard abbreviations will be used to construct the table name. A Views name will be preceded with "VW\_".

The format will be:

* Tables and views will precede the table name with two characters (with the exception of error tables which have three characters) and an underscore.
* A prefix of 2 characters followed with underscore “TB\_” will identify it as Relational table format
* "LD\_" to identify load tables.
* "VW\_" to identify it as a view.
* “TR\_” to identify reference tables.
* Dimensional models
* Dimensional tables format "TD\_"
* Fact tables format "TF\_"
* Error tables will follow the above format adding an additional "E" before the underscore. ("TBE\_")
* Abbreviating the Entity name using the standard modeling abbreviations will create the Table name.
* Multiple words will be separated with underscores.
* Maximum length of 18 characters
* Backup tables will follow the conventions above, suffixing the name with \_OLD

Examples:

TB\_PROV

**TB\_MBR**

**TB\_CNTRCT\_TYPE**

**TB\_GRP\_ ACCT**

**TB\_GRP\_PKG\_PROD**

**TB\_GRP\_PKG\_CNTRCT**

**TB\_LRSP\_GRP\_CNTRCT**

**LD\_ACCT\_PRMTR**

**TD\_RC\_CLM\_EXPNS**

**TD\_PD\_DT**

**TF\_RC\_CAPTN**

**TF\_RC\_DIV**

**VW\_ACCT\_ATTRBT\_LVL**

**TBE\_GRP\_ ACCT**

**TBE\_GRP\_PKG\_CNTRCT**

**TBE\_LRSP\_GR\_CNTRCT**

**TB\_PROV\_OLD**

Table Definition

The definition of a Table will be the same as it’s parallel Entity. If this table does not have “a” parallel entity in the logical then the definition will contain the Table’s name and clearly define how this table will be used within the company and this application. Give examples where appropriate.

Examples:

TB\_PROV

PROVIDER is a person, or entity, who provides a service to healthcare, contracted members.

TB\_BNFT\_PKG

BENEFIT PACKAGE is a contract holder's benefit package. Included only for Amisys mapping.

TB\_MBR

A member is an applicant and all his/her eligible dependents.

TB\_BNFT\_RLTN

BENEFIT RELATION defines the possible relationships such as Parent Title equals "L" and Child Title equals "B".

##### Column

A Column is the physical name for an Attribute. Standard abbreviations will be used to create the column name. The column holds data that is to be used in some way by or for the company.

Column Name

The format of the name of a Column will be:

* All UPPER CASE
* No spaces between words, instead use underscore “\_” to separate the words or abbreviations.
* Standard abbreviations will be used for the appropriate attribute conversion
* Will always end with a class word
* Maximum length of 18 characters

Examples:

**GRP\_ACCT\_NUM**

**MBR\_LST\_ NAM**

**BNFT\_PROD\_ID**

**MRKTNG\_GRPNG\_ORD\_NUM**

Column Definition

The definition of a Column will be the same as its Attribute; it will describe how and why this column is being used.

Examples:

**MBR\_NUM**

Member Number is a unique number assigned to a MEMBER for identification purposes.

**MBR\_EFF\_DT**

The Member Effective Date will store the date, and time where available, where the instance became true within the business.

**PROV\_TYPE\_DESC**

Identifies the different types of Provider's. Valid Values are Person, Institution, or Ancillary.

**CNTRCT\_TYPE\_DW\_UPDT\_DT**

The Data Warehouse Update Date will define the date, and time where available, where the occurrence was extracted from the source system. There will be an additional attribute created (Data Warehouse Write Date) if the actual warehouse population date is needed. (The definition of Data Warehouse Update Date is the same regardless of the table it resides. So the definition not reflect the tables name in the definition as it is used globally.

# DataStage

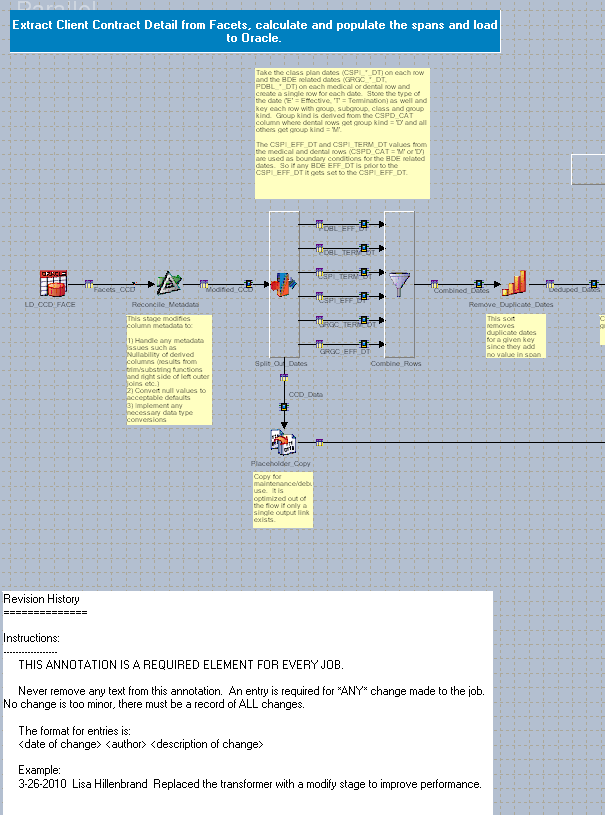
## Comments

Three text boxes should exist in each piece of DataStage code (jobs, sequencers, etc.) for comments.

Blue box at the top - full job description

Yellow middle boxes – descriptions for individual stages or processes

White bottom box – Revision history



The DataStage templates (parallel job, server jobs, sequencer) can be found in the project CBIDEV/Jobs/EDR\_TEMPLATES/

## Variables

### DataStage Stage Variables

* Database connections should reflect the table name it is pulling from or going to. If it is pulling from multiple tables, the name should reflect the type of data… Group, member, eligibility, claims… etc
* Transformers, lookups, modifies, funnels, joins, sorts and any other kind of processing stage should reflect what the particular stage does or processes.
* Link names should reflect what is being transferred through the link, this can be generally associated with the previous job to the link.

### DataStage Variables Standards

* File Name stage properties must be configured using two parameters, one for directory path and the second for file name. The directory path delimiter must be specified in the property to avoid errors. Do not assume the runtime value of the directory parameter includes the appropriate delimiter. If the user supplies it the operating system accepts *//* as a delimiter, and if it is not provided, which is common, the file name property specification is correct.
* Example of standard practice for file name properties:
* **#Dir\_Path#/#File\_Name#**
* Similar to directory path delimiters, database schema names, etc. should contain any required delimiter.
* Example of standard practice for table name properties:
* **#DatabaseSchemaName#.TableName**
* User Accounts and passwords must be specified as environment variables.
* Passwords must be set to type encrypted, and the default value maintained using the DataStage Administrator.

## .DSX Naming

The naming of a DSX should reflect the project name, the main folder name with the subfolder name.

For the project CBI<dev,sys,uat,prd>

/ODS\_EXTRACTS/CCD - CBI-ODS-CCD.dsx

/EDR\_STAGE/ELIG - CBI-EDRCERT-ELIG.dsx

/FACETS\_EXTRACTS/PCC/ CBI-FAC-PCC.dsx

### DataStage Job Naming

When a new job is created in DataStage, along with following all of the naming conventions for an individual job, there are naming conventions for sequential jobs; the “prefix” of the name indicated the subject area (e.g. CCD, PCC, FACHDR), followed by an ascending number pattern, e.g. \_00, \_10, \_20 etc, showing the order the jobs run in and followed by a brief description explaining what is happening in each step e.g. FacExtract, FacITSHost\_Extract



# Unix KornShell

The ksh script should include

1. Shell language indicator
2. Script name
3. Usage of script and parameters
4. History log
5. Error code design
6. Initialization of parameters
7. Setting of parameters
8. Testing of parameters
9. Creating private variables
10. Body of code



The Template is in CVS in /cbi/UnixScripts/KSH Template.ksh

# Outbound Files

## Encryption

It is mandatory that all outbound files (to any outside source, also including other departments within the company) use PGP encryption.

# File Naming conventions

Global File naming Format =

<environment>.<team>.<vendor>.<filetype>.<frequency>.<direction>.<datetime>

where:

NodeAcceptable Values

<environment>prd, uat, trn, int, sit, dev

<team> we will use CBI

<vendor> we will use the vendor names BCBSA and BCBSRI

<filetype[grouping]>Filetype : membership / claims / provider

Grouping: Additional grouping info if the vendor can transfer multiple files per type

<frequency>daily / weekly / monthly / yearly / bimonthly / biyearly / quarterly

<direction>inbound, outbound, or internal we will use internal for ODS to EDR

<datetime>File creation date and time

CC+YY+MM+DD+HH+MM

CC = Century

YY = Year

MM = Month

DD = Day

HH = Military time hour

MM = Time minute

Example: dev.cbi.bcbsri.**ods\_pcc**.daily.internal.200703222034