PPDM 3.9: PPDM ARCHITECTURAL PRINCIPLES

The PPDM Association is a not-for-profit society for data managers.

The PPDM Data Model is a relational model specification used by the Oil and Gas industry as the function for effective management and stewardship of data related to Upstream Exploration and Production.

PPDM 3.9 was built by industry subject matter experts who have worked cooperatively over 24 years to develop an open data specification that is practical and useful for companies of all sizes.

Professional Petroleum Data Management Association (PPDM)

PPDM ARCHITECTURAL PRINCIPLES

- Data Model Version PPDM 3.9
- Version 2.0
- September 14, 2012
- Updated 2018

Contents

PPDM Architectural Principles 1
Architectural Principles: Tables 2
Architectural Principles Columns 8
Architectural Principles Constraints in PPDM 29
Architectural Principles Design Issues 38
Architectural Principles Versioning 39
Architectural Principles Vertical Tables 42
Architectural Principles Sequence Control 48
Architectural Principles Economics and Financial 49
Architectural Principles Column Precision 50
Architectural Principles Column Field Type 51
Architectural Principles Standardized Columns 51
Architectural Principles SQL View Definitions 54
Architectural Principles Super / Sub Types 55
Architectural Principles Reference Tables 57
Architectural Principles Domains 59
Architectural Principles Units of Measure 59
Currency Units of Measure 61
Architectural Principles Coordinates 61
Architectural Principles Extensibility and Subsetting 63
Architectural Principles Meta Tables and Meta Data 64
Additional Architectural Guidelines and Conventions 65
Alias Tables 66

ARCHITECTURAL PRINCIPLES: DATA DEFINITION LANGUAGE (DDL)

Structure

Data Model DDL is generated from the Data Model's internal tables:

- PPDM_SYSTEM
- PPDM_TABLE
- PPDM_COLUMN
- PPDM_CONSTAINT
- PPDM_CONS_COLUMN
- PPDM_CHECK CONS_VALUE

DDL Contents

PPDM DDL is segmented as follows (in alphabetic order):

File Extension	Description	Mandatory in PPDM 3.9?	Implementation
CCM	Create Column Comments	Yes	Not supported by all versions of SQL*Server
CK	Create Check Constraints	Yes	These may not be altered by the user.
FK	Create Foreign Keys	Yes	Some FK may not be implementable in some RBDMS without intervention. For example, some versions of MS*SQL Server may not handle tables that contain more than 253 constraints, or that are referenced by more than 253 constraints.
GUID	Add unique index to ROW_MANAGEMENT_ GUID	No	Needed in order to use the columns named PPDM_GUID correctly.
GUIDNN	Add Not Null constraint to ROW_MANAGEMENT_ GUID	No	Needed in order to use the columns named PPDM_GUID correctly.
OUOM	Create FK for columns Original Units of Measure	Yes	Not supported by all versions of SQL*Server, too many foreign keys against PPDM_UNIT OF MEASURE.

PK	Create primary keys	Yes	PPDM Primary keys are mandatory and may not be altered.
RQUAL	Create foreign keys for columns named PPDM_ROW_QUALITY (to R_PPDM_ROW_QUALITY)	Yes	Not supported by all versions of SQL*Server, too many foreign keys against R PPDM_ROW QUALITY
RSRC	Create foreign keys for columns named SOURCE (to R_SOURCE)	Yes	Not supported by all versions of SQL*Server, too many foreign keys against R SOURCE.
SQL	Command file that executes DDL components in an appropriate order.	No	You may run in any order you wish
SYN	Create table synonyms	Yes	Not supported by all versions of SQL*Server
TAB	Create table and columns	Yes	
TCM	Create table comments	Yes	Not supported by all versions of SQL*Server
UK	Create Unique Keys	Yes	Not used in PPDM 3.9.
UOM	Create foreign keys on columns storing units of measure	Yes	Not supported by all versions of SQL*Server, too many foreign keys against PPDM_UNIT_OF_MEASURE.

RDBMS Support

Any RDBMS language can be supported, provided the RDBMS is entry level SQL*92 compliant.

- Oracle DDL is generated for each release
- MS SQL*Server DDL is generated for each release
- Other RDBMS languages will be created by member request, provided that the requesting member is willing to work with the PPDM Association's staff to verify the syntax and datatypes, and to test the DDL before it is released.

ARCHITECTURAL PRINCIPLES: TABLES

Structure

Naming of tables should reflect the subject area, sub-area or natural grouping of tables. The structure of a table names is: SUBJECT AREA + MODIFIER1(sub-area) + MODIFIER2(grouping)... Example:

- WELL
- WELL_PRESSURE
- WELL_PRESSURE_AOF
- WELL_PRESSURE_AOF_4PT

Structure Rules

Maximum of 24 characters is used for PPDM table names.

Table names are singular and in present tense.

Example:

- WELL_TEST

Intersection / associate tables will be named according to business usage and by borrowing from the names of the intersecting tables.

Cross-reference tables created from a single parent table are named by adding an XREF qualifier to the name of the table.

Example:

- BA_XREF

Name Component Rules

- Names must only contain alphanumeric characters and underscore.
- Name components are separated by an underscore character.
- Names are not case sensitive, and are created in UPPER CASE by the PPDM Association.
- Complete names must not be in the Oracle, Sybase or PL SQL reserved words list (see Appendix A for a list of these words)
- Avoid using 'A', 'AN', 'AND', 'OF', 'OR', 'THE'.

Consistency

Use consistent subject area abbreviations.

Example:

- BA - Business Associate
- R_ - Reference Tables
- WELL_DIR_SURVEY - Well Directional Survey

- SEIS_ - Seismic

Use common acronyms or industry accepted abbreviations where useful and meaningful to the membership. These should be unique.

Example:

- AOF - Absolute Open Flow
- GOR - Gas Oil Ratio
- DLS - Dominion Land Survey
- LR – Land Rights
- PDEN – Production Entity
- RM – Records and Information Management
- M_B – Metes and Bounds

Naming Process

- Construct business name (fully descriptive).
- Choose appropriate subject area name.
- Choose appropriate modifier(s).
- Determine if name meets maximum length specifications (24 characters for table names)
- If too long, apply abbreviations
- Be consistent. Naming abbreviations or class words that have been used in past should be re-used when the context is similar. Remember that the objective is to ensure the model is as consistent and easy to understand as possible
- A column that requires a measured value should be named so that the OUOM and or UOM column names that accompany it can have the identical prefix components.

Example:

- MEASURED VALUEMEASURED VALUE OUOM
- MEASURED VALUE UOM

- If it isdifficult to arrive at a name that pleases everyone in the work group, use the table comments to clarify the meaning and usage of the table.

Synonyms

- Each table is assigned a name and a synonym in the CASE tool. Synonym names are released as an optional segment of the DDL in a file names *.SYN.
- Synonyms are named so as to group tables into logical components.

Synonym Prefix	Module (table set)
A	Areas
APP	Applications
BA	Business Associates

BALIC	Business Associate Licenses
C	Contracts
CAT	Catalogues
CLS	Classifications
CO	Consultations
CON	Consents
COT	Contests
CS	Coordinate systems
ENT	Entitlements
EQ	Equipment
EZ	Ecozones
FAC	Facilities
FIN	Finance
FLD	Fields
FOS	Fossils
HSE	Health Safety and Environment
INS	Instruments
INT	Interest Sets
LE	Legal Descriptions
LI	Lithology
LR	Land Rights
LS	Land Sale
NOT	Notifications
OB	Obligations
PA	Paleo Analysis
P	PPDM Data Management
PDE	PDEN
PL	Pools
PROD	Products
PROJ	Projects
RATE	Rate schedules
REST	Restrictions
RM	Records Management
R_	Reference tables (note that an Underscore must follow the letter R)
SA	Sample analysis
SF	Support Facilities
SS	Seismic
ST	Stratigraphy
W	Wells

- Synonyms are usually constructed from prefix and the first letter of each name component.
- Every synonym must be unique. If the construction method above would result in a duplicate, enforce uniqueness by adding a number or an additional character to the end of the synonym.
- Primary key and foreign key names are based on the synonyms.

Table comments

- Every table must have a complete description explaining what it is.

- Table long names always appear as the first part of the table comments (in upper case), followed by a colon (:).

Example:

- AIR DRILLING INTERVAL: The Air Drilling Interval table contains depth interval and air volume information where air drilling was utilized in a wellbore. Air drilling is rotary drilling that uses compressed air instead of a circulating mud system.

Table length

- Some relational databases impose a limit on the maximum size of a table. SQL Server 2000 limits the total allowed length to 8060 bytes. Even though you can successfully create a table in SQL Server 2000 that could theoretically contain more than 8060 bytes, you will be unable to actually add (instantiate) or update a record that has more than 8060 bytes of data.
- Table design must allow entry level SQL92 compliant relational database implementations to achieve 100% Gold level compliance; however, SQL Server is not entry level compliant. Despite this, the modeling committee has elected to support this limitation as this RDBMS is sometimes desired by implementation teams for other reasons. For this reason, the total length of a table may not exceed 8060 bytes.
- SQL Server limitations are posted here http://msdn2.microsoft.com/en-us/library/ms143432.aspx
- A discussion about this limitation is on the forums here http://www.ppdm.org/forums/viewtopic.php?t=304

Table deprecation

- Workgroup Deprecation:
  - From time to time, work group reviews and re-engineering the model will make some portions of the model invalid. While every attempt will be made to ensure that redundancy is cleared up during design, there are times when existing tabare left in the model for a period of time so that members can plan a migration. Note that this practice is only used when absolutely essential, and should be rare.
  - In some cases, workgroups may replace a table in a subject area that is being re-engineered. In these cases, the function of the dropped table will be mapped to new tables in the new version.
  - The replaced table may be marked for deprecation or not, as determined most appropriate by the workgroup.
- Change management deprecation
  - Tables that are included in a model release but scheduled for deprecation in the next release as a result of the change management process will be re-named with a Z_ prefix.
  - New implementations of the PPDM Models should avoid using these tables, and existing implementation should be revised to ensure that these tables removed from use.
  - Mappings from deprecated tables to new table functionality will be provided with the model release.
  - Tables named Z_% in a previous release will be removed from the next model release.

ARCHITECTURAL PRINCIPLES: COLUMNS

Structure

- Column names are made up of a collection of words, each of which has particular significance. The name components are:

PRIMARY + MODIFIER(s) + CLASS WORD

Primary Words

- The primary word is usually a noun describing the subject area of the name. Normally the subject area is implied by the context and table the column is in. If required, a subject area or partial table name may be used to explicitly state the column usage.

Example:

- WELL_NAME
- WELL_NUMBER

Modifier Words

- The modifier word is used to provide additional description of the column being named. Modifiers are used to further describe the primary word, class word or both.

Example:

- CURRENT_STATUS
- FINAL_DRILL_DATE
- SURFACE_LATITUDE

Class Words

- A class word identifies the type or category of data being described by the column name. This may be a general classification such as DATE or NUMBER or more specific such as TEMPERATURE or PRESSURE.
- Column names contain class words, which are used to define the type of information kept in the columns by referencing data value concepts such as depth, elevation, latitude, longitude, length, temperature etc.
- Certain class words have been defined and reserved, so that their use in the data model may be consistent in their meaning, structure and usage. In some cases, the format has also been defined, and a domain established to assist with managing how they are used. Here is a list of the **Class Words** in PPDM.
- The class word is usually the last component in the column name, except for columns containing _OUOM or _UOM. In these cases, the class word is usually second last in the column names.

Example:

- ENERGY_SWEEP_DURATION
- ENERGY_SWEEP_DURATION_OUOM

- Columns that represent units of measures will end in

- _UOM for storage unit of measure
- _OUOM for an original unit of measure.

- When a class word is used by itself it may be the full word used even if the abbreviation is acceptable for other column names.
- If a column name includes the name of a domain's class word, then the column must be part of that domain.

Example:

- All columns that end in DATE must be in the DATE domain.

Class Word	Full Name	Usage
ABBREVIATION	ABBREVIATION	Exists in all reference tables
AGE	AGE	References geologic age
ALIAS	ALIAS	Set of columns that reference alternate names and identifiers for an object or concept
AMOUNT	AMOUNT	Refers to a measured amount often as mass or volume
ANL	ANALYSIS	References the Analysis subject area
CATEGORY	CATEGORY	Mechanism used to refer to a type of concept of object.
COLOR	COLOR	References a color in the visible or invisible color spectrum. Many color pallets are used by industry, be sure to reference the correct one.
COORD or COORDINATE	COORDINATE	Refers to the location of an object
COST	COST	Refers to financial costs, used in conjunction with references to the currency type and conversion rate.
COUNT	COUNT	A whole number reference to the number of items.
DATE	DATE	A calendar date – used with DATE DESC to verity level of accuracy
DEPTH	DEPTH	Refers to depth measurements
DESC or DESCRIPTION	DESCRIPTION	A short narrative description about an object, activity, event etc.
DIAMETER	DIAMETER	The inside or outside measurement of the diameter of an object such as a pipe.
DIST or DISTANCE	DISTANCE	The distance between two measured points, such as KB to ground or in a survey system.
DURATION	DURATION	The total time of an event.
GUID	GLOBAL UNIQUE IDENTIFIER	Globally Unique identifier system used for PPDM data coordinate references
ID or IDENTIFIER	IDENTIFIER	Unique identifier associated with an object or a row of data. Care should be taken to clarify which concept is in use.
IND	INDICATOR	A column that may have a value of Y or N or NULL
LATITUDE	LATITUDE	Geographic latitude in decimal degrees
LONG_NAME	LONG_NAME	The full name assigned to an object or concept.
LONGITUDE	LONGITUDE	Geographic longitude in decimal degrees

MD	MEASURED DEPTH	The length measure to an object or activity, usually in the wellbore. This distance is measured within the wellbore from an identified reference point, such as a KB or DF
MNEMONIC	MNEMONIC	An abbreviated form for a curve type or parameter, used mainly in well logs
NO	NO	A number of something, in numeric form
NUM	NUM	A reference number, or serial number, that is used by a system for identification
OBS_NO	OBSERVATION NUMBER	Numeric reference to a unique measurement, that does not imply time or spatial occurrence.
OUOM	ORIGINAL UNIT OF MEASURE	The unit of measure in which data was originally captured or received.
PERCENT	PERCENT	A percentage as a fraction of 100
PRESS or PRESSURE	PRESSURE	A pressure measurement, usually as taken in a field situation in a reservoir or a measuring point.
QUALITY	QUALITY	A subjective or objective measure of the level of trust to be associated.
SEQ_NO	SEQ_NO	Numeric reference to a unique measurement, that does imply time or spatial occurrence.
SHORT_NAME	SHORT_NAME	A short name given to an object, concept or reference value for convenient reference.
SIZE	SIZE	The measured size of an object linearly, in area or in volume.
SOURCE	SOURCE	The source of a record. May be the original source, or the source from which original data was compiled. Be clear about which you mean.
STATUS	STATUS	The current state of an object or activity, usually from the perspective of a process or stakeholder group.
SYNONYM	SYNONYM	The technical synonym used in a data base system, not the same as alias.
TD	TD	Total depth, the final depth to a terminating point. Is a kind of Measured Depth.
TEMP or TEMPERATURE	TEMPERATURE	A measured or calculated temperature, must be associated with a unit of measure
THICKNESS	THICKNESS	References the vertical distance from top to bottom,, usually of a pay zone or formation.
TIME	TIME	References time of day, must be associated with a time zone.

TIMEZONE	TIMEZONE	References the time zone used for time of day. Mixing multiple time zones is discouraged
TVD	TRUE VERTICAL DEPTH	Vertical depth from a reference elevation, usually to a point in a wellbore. May be used in other contexts as well.
TYPE	TYPE	Used as the primary key domain for columns that are validated by a reference table. Note that not all columns contain this class word, particularly where column length is long.
UOM	UNIT OF MEASURE	The unit of measure in which a measured value is stored. May or may not be the same as the original unit of measure. Mixing unit systems is discouraged.
VELOCITY	VELOCITY	A measured or calculated velocity, mainly used in the seismic subject area
VOLUME	VOLUME	A volume of substance, either liquid or gas. For hydrocarbons, should be associated with the appropriate pressure and temperature regime.
WEIGHT	WEIGHT	The measured or calculated weight of a substance, used mainly for additives and analysis.
WIDTH	WIDTH	A measured or calculated width of a physical object
YEAR	YEAR	The year in which something happened.

Structure Rules

- Maximum of 27 characters used for PPDM column names.
- Column names are singular and in present tense.
- Column names should proceed from general to specific, left to right.

Example:

- INVENTORY_CLOSE_BALANCE
- SEGMENT_LENGTH_UOM

Name Component Rules

- Names only contain alphanumeric characters and underscore.
- Names are not case sensitive.
- Complete names must not be in the Oracle or PL SQL reserved words list (see Appendix A for a list).
- Avoid the use of 'A', 'AN', 'AND', 'OF', 'OR', 'THE'.
- Be aware of and avoid possible confusion with abbreviated column names in the same or other modules, to avoid misinterpretation of column meaning.

Example:

- Is REC = RECOVERY, RECEIVER or RECORDED?
- Is COMP = COMPLETED or COMPANY?

- With few exceptions, all parts of a column name are either the full word or a PPDM recognized abbreviation.

Example:

- Use DIAMETER instead of DIA, DIAM, DMTR
- Use DEPTH instead of DEP, DPTH

- In a few instances, there may be no recognized full word, only an industry standard abbreviation or acronym.

Example:

- GOR used everywhere, gas oil ratio not used.
- DLS used instead of Dominion Land Survey
- UOM used instead of Unit of Measure
- OUOM used instead of Original Unit of Measure

- Where a column has an associated UOM or OUOM column, the name of the column should allow exact character string correspondence to the UOM or OUOM column.

Example:

- ACQTN_ SHOTPT_ INTERVAL
- ACQTN_ SHOTPT_ INTERVAL_ OUOM

- Columns present in multiple tables that have a common definition should be named consistently unless business confusion is created. If the choice is to abbreviate the column name then the name will be abbreviated in all occurrences. If the choice is not to abbreviate the column name then all occurrences will not be abbreviated.
- Columns that represent foreign keys in child tables should usually be given the same name as the column in the parent table unless there is more than one relationship or denormalization to the parent table. In this case, names that differentiate the relationships should be used.

Examples:

- Occurrences of Business Associates are often named to refer to the role played by the Business Associate, such as Operator, Address for Service etc.
  i. OPERATOR_BA_ID
  ii. CONTACT_BA_ID
- The WELL table has two relationships to the NODE table.
  i. SURFACE_NODE_ID
  ii. BASE_NODE_ID
- The SEIS_SUMMARY table has two relationships to the SEIS_POINT table
  i. FIRST_SEIS_POINT_ID
  ii. LAST_SEIS_POINT_ID

Reserved Words

- When naming columns, avoid using words that are reserved by other databases and functions. Note that some legacy use may be found for some reserved words; these are revised as requested by PPDM Members.
- Here is a list of the **Reserved Words** we currently watch. This list grows frequently, and is difficult to keep up with. For new table and column names, we recommend that the name consist of at least two parts (AAA_BBB).

Note: This list may not contain all reserved words

ORACLE	MYSQL	POSTGRES	SQLSRV	SQL
ACCESS	ACCESSIBLE	ABORT	ADD	ACCESS
ADD	ADD	ABS	ALL	ADD
ALL	ALL	ABSOLUTE	ALTER	ALL
ALTER	ALTER	ACCESS	AND	ALTER
AND	ANALYZE	ACTION	ANY	AND
ANY	AND	ADA	AS	ANY
ARRAY	AS	ADD	ASC	ARRAYLEN
ARRAYLEN	ASC	ADMIN	AUTHORIZATIO N	AS
ARROW	ASENSITIVE	AFTER	AVG	ASC
AS	BEFORE	AGGREGATE	BACKUP	AUDIT
ASC	BETWEEN	ALIAS	BEGIN	BETWEEN
AT	BIGINT	ALL	BETWEEN	BY
AUDIT	BINARY	ALLOCATE	BREAK	CHAR
BEGIN	BLOB	ALTER	BROWSE	CHECK
BETWEEN	BOTH	ANALYSE	BULK	CLUSTER
BY	BY	ANALYZE	BY	COLUMN
CASE	CALL	AND	CASCADE	COMMENT
CHAR	CASCADE	ANY	CASE	COMPRESS
CHECK	CASE	ARE	CHECK	CONNECT
CLUSTER	CHANGE	ARRAY	CHECKPOINT	CREATE
CLUSTERS	CHAR	AS	CLOSE	CURRENT
COLAUTH	CHARACTER	ASC	CLUSTERED	DATE
COLUMN	CHECK	ASENSITIVE	COALESCE	DECIMAL
COLUMNS	COLLATE	ASSERTION	COLLATE	DEFAULT
COMMENT	COLUMN	ASSIGNMENT	COLUMN	DELETE
COMPRESS	COLUMNS	ASYMMETRIC	COMMIT	DESC
CONNECT	CONDITION	AT	COMPUTE	DISTINCT
CRASH	CONNECTION	ATOMIC	CONSTRAINT	DROP
CREATE	CONSTRAINT	AUTHORIZATIO N	CONTAINS	ELSE
CURRENT	CONTINUE	AVG	CONTAINSTABL E	EXCLUSIVE
DATE	CONVERT	BACKWARD	CONTINUE	EXISTS
DECIMAL	CREATE	BEFORE	CONVERT	FILE
DECLARE	CROSS	BEGIN	COUNT	FLOAT
DEFAULT	CURRENT_DAT E	BETWEEN	CREATE	FOR
DELETE	CURRENT_TIME	BIGINT	CROSS	FROM

DESC	CURRENT_TIMESTAMP	BINARY	CURRENT	GRANT
DISTINCT	CURRENT_USER	BIT	CURRENT_DATE	GROUP
DROP	CURSOR	BIT_LENGTH	CURRENT_TIME	HAVING
ELSE	DATABASE	BITVAR	CURRENT_TIMESTAMP	IDENTIFIED
END	DATABASES	BLOB	CURRENT_USER	IMMEDIATE
EXCEPTION	DAY_HOUR	BOOLEAN	CURSOR	IN
EXCLUSIVE	DAY_MICROSECOND	BOTH	DATABASE	INCREMENT
EXISTS	DAY_MINUTE	BREADTH	DATABASEPASSWORD	INDEX
FETCH	DAY_SECOND	BY	DATEADD	INITIAL
FILE	DEC	C	DATEDIFF	INSERT
FLOAT	DECIMAL	CACHE	DATENAME	INTEGER
FORM	DECLARE	CALL	DATEPART	INTERSECT
FOR	DEFAULT	CALLED	DBCC	INTO
FROM	DELAYED	CARDINALITY	DEALLOCATE	IS
GOTO	DELETE	CASCADE	DECLARE	LEVEL
GRANT	DESC	CASCADED	DEFAULT	LIKE
GROUP	DESCRIBE	CASE	DELETE	LOCK
HAVING	DETERMINISTIC	CAST	DENY	LONG
IDENTIFIED	DISTINCT	CATALOG	DESC	MAXEXTENTS
IF	DISTINCTROW	CATALOG_NAME	DISK	MINUS
IMMEDIATE	DIV	CHAIN	DISTINCT	MODE
IN	DOUBLE	CHAR	DISTRIBUTED	MODIFY
INCREMENT	DROP	CHAR_LENGTH	DOUBLE	NOAUDIT
INDEX	DUAL	CHARACTER	DROP	NOCOMPRESS
INDEXES	EACH	CHARACTER_LENGTH	DUMP	NOT
INITIAL	ELSE	CHARACTER_SET_CATALOG	ELSE	NOTFOUND
INSERT	ELSEIF	CHARACTER_SET_NAME	ENCRYPTION	NOWAIT
INTEGER	ENCLOSED	CHARACTER_SET_SCHEMA	END	NULL
INTERSECT	ESCAPED	CHARACTERISTICS	ERRLVL	NUMBER
INTO	EXISTS	CHECK	ESCAPE	OF
IS	EXIT	CHECKED	EXCEPT	OFFLINE
LEVEL	EXPLAIN	CHECKPOINT	EXEC	ON
LIKE	FETCH	CLASS	EXECUTE	ONLINE
LOCK	FIELDS	CLASS_ORIGIN	EXISTS	OPTION
LONG	FLOAT	CLOB	EXIT	OR
MAXEXTENTS	FLOAT4	CLOSE	EXPRESSION	ORDER
MINUS	FLOAT8	CLUSTER	EXTERNAL	PCTFREE

MLSLABEL	FOR	COALESCE	FETCH	PRIOR
MODE	FORCE	COBOL	FILE	PRIVILEGES
MODIFY	FOREIGN	COLLATE	FILLFACTOR	PUBLIC
NOAUDIT	FROM	COLLATION	FOR	RAW
NOCOMPRESS	FULLTEXT	COLLATION_CATALOG	FOREIGN	RENAME
NOT	GOTO	COLLATION_NAME	FREETEXT	RESOURCE
NOTFOUND	GRANT	COLLATION_SCHEMA	FREETEXTTABLE	REVOKE
NOWAIT	GROUP	COLUMN	FROM	ROW
NULL	HAVING	COLUMN_NAME	FULL	ROWID
NUMBER	HIGH_PRIORITY	COMMAND_FUNCTION	FUNCTION	ROWLABEL
OF	HOUR_MICROSECOND	COMMAND_FUNCTION_CODE	GOTO	ROWNUM
OFFLINE	HOUR_MINUTE	COMMENT	GRANT	ROWS
ON	HOUR_SECOND	COMMIT	GROUP	SELECT
ONLINE	IF	COMMITTED	HAVING	SESSION
OPTION	IGNORE	COMPLETION	HOLDLOCK	SET
OR	IGNORE_SERVER_IDS	CONDITION_NUMBER	IDENTITY	SHARE
ORDER	IN	CONNECT	IDENTITY_INSERT	SIZE
OVERLAPS	INDEX	CONNECTION	IDENTITYCOL	SMALLINT
PCTFREE	INFILE	CONNECTION_NAME	IF	SQLBUF
PRIOR	INNER	CONSTRAINT	IN	START
PRIVILEGES	INOUT	CONSTRAINT_CATALOG	INDEX	SUCCESSFUL
PROCEDURE	INSENSITIVE	CONSTRAINT_NAME	INNER	SYNONYM
PUBLIC	INSERT	CONSTRAINT_SCHEMA	INSERT	SYSDATE
RANGE	INT	CONSTRAINTS	INTERSECT	TABLE
RAW	INT1	CONSTRUCTOR	INTO	THEN
RECORD	INT2	CONTAINS	IS	TO
RENAME	INT3	CONTINUE	JOIN	TRIGGER
RESOURCE	INT4	CONVERSION	KEY	UID
REVOKE	INT8	CONVERT	KILL	UNION
ROW	INTEGER	COPY	LEFT	UNIQUE
ROWID	INTERVAL	CORRESPONDING	LIKE	UPDATE
ROWLABEL	INTO	COUNT	LINENO	USER
ROWNUM	IS	CREATE	LOAD	VALIDATE
ROWS	ITERATE	CREATEDB	MAX	VALUES
SELECT	JOIN	CREATEUSER	MERGE	VARCHAR
SESSION	KEY	CROSS	MIN	VARCHAR2
SET	KEYS	CUBE	NATIONAL	VIEW

SHARE	KILL	CURRENT	NOCHECK	WHENEVER
SIZE	LABEL	CURRENT_DATE	NONCLUSTERED	WHERE
SMALLINT	LEADING	CURRENT_PATH	NOT	WITH
SQL	LEAVE	CURRENT_ROLE	NULL	
SQLBUF	LEFT	CURRENT_TIME	NULLIF	
START	LIKE	CURRENT_TIMESTAMP	OF	
SUBDATE	LIMIT	CURRENT_USER	OFF	
SUCCESSFUL	LINEAR	CURSOR	OFFSETS	
SYNONYM	LINES	CURSOR_NAME	ON	
SYSDATE	LOAD	CYCLE	OPEN	
TABAUTH	LOCALTIME	DATA	OPENDATASOURCE	
TABLE	LOCALTIMESTAMP	DATABASE	OPENQUERY	
THEN	LOCK	DATE	OPENROWSET	
TO	LONG	DATETIME_INTERVAL_CODE	OPENXML	
TRIGGER	LONGBLOB	DATETIME_INTERVAL_PRECISION	OPTION	
TYPE	LONGTEXT	DAY	OR	
UID	LOOP	DEALLOCATE	ORDER	
UNION	LOW_PRIORITY	DEC	OUTER	
UNIQUE	MASTER_HEARTBEAT_PERIOD	DECIMAL	OVER	
UPDATE	MASTER_SSL_VERIFY_SERVER_CERT	DECLARE	PERCENT	
USE	MATCH	DEFAULT	PIVOT	
USER	MEDIUMBLOB	DEFERRABLE	PLAN	
VALIDATE	MEDIUMINT	DEFERRED	PRECISION	
VALUES	MEDIUMTEXT	DEFINED	PRIMARY	
VARCHAR	MIDDLEINT	DEFINER	PRINT	
VARCHAR2	MINUTE_MICROSECOND	DELETE	PROC	
VIEW	MINUTE_SECOND	DELIMITER	PROCEDURE	
VIEWS	MOD	DELIMITERS	PUBLIC	
WHEN	MODIFIES	DEPTH	RAISERROR	
WHENEVER	NATURAL	DEREF	READ	
WHERE	NO_WRITE_TO_BINLOG	DESC	READTEXT	
WITH	NOT	DESCRIBE	RECONFIGURE	
	NULL	DESCRIPTOR	REFERENCES	
	NUMERIC	DESTROY	REPLICATION	

	ON	DESTRUCTOR	RESTORE	
	OPTIMIZE	DETERMINISTIC	RESTRICT	
	OPTION	DIAGNOSTICS	RETURN	
	OPTIONALLY	DICTIONARY	REVERT	
	OR	DISCONNECT	REVOKE	
	ORDER	DISPATCH	RIGHT	
	OUT	DISTINCT	ROLLBACK	
	OUTER	DO	ROWCOUNT	
	OUTFILE	DOMAIN	ROWGUIDCOL	
	OVERWRITE	DOUBLE	RULE	
	PRECISION	DROP	SAVE	
	PRIMARY	DYNAMIC	SCHEMA	
	PRIVILEGES	DYNAMIC_FUNCTION	SECURITYAUDIT	
	PROCEDURE	DYNAMIC_FUNCTION_CODE	SELECT	
	PURGE	EACH	SESSION_USER	
	RANGE	ELSE	SET	
	READ	ENCODING	SETUSER	
	READ_ONLY	ENCRYPTED	SHUTDOWN	
	READ_WRITE	END	SOME	
	READS	END-EXEC	STATISTICS	
	REAL	EQUALS	SUM	
	REFERENCES	ESCAPE	SYSTEM_USER	
	REGEXP	EVERY	TABLE	
	RELEASE	EXCEPT	TABLESAMPLE	
	RENAME	EXCEPTION	TEXTSIZE	
	REPEAT	EXCLUSIVE	THEN	
	REPLACE	EXEC	TO	
	REQUIRE	EXECUTE	TOP	
	RESTRICT	EXISTING	TRAN	
	RETURN	EXISTS	TRANSACTION	
	REVOKE	EXPLAIN	TRIGGER	
	RIGHT	EXTERNAL	TRUNCATE	
	RLIKE	EXTRACT	TSEQUAL	
	SCHEMA	FETCH	UNION	
	SCHEMAS	FINAL	UNIQUE	
	SECOND_MICROSECOND	FIRST	UNPIVOT	
	SELECT	FLOAT	UPDATE	
	SENSITIVE	FOR	UPDATETEXT	
	SEPARATOR	FORCE	USE	
	SET	FOREIGN	USER	
	SHOW	FORTRAN	VALUES	
	SMALLINT	FORWARD	VARYING	
	SONAME	FOUND	VIEW	
	SPATIAL	FREE	WAITFOR	
	SPECIFIC	FREEZE	WHEN	
	SQL	FROM	WHERE	

	SQL_BIG_RESULT	FULL	WHILE	
	SQL_CALC_FOUND_ROWS	FUNCTION	WITH	
	SQL_SMALL_RESULT	G	WRITETEXT	
	SQLEXCEPTION	GENERAL		
	SQLSTATE	GENERATED		
	SQLWARNING	GET		
	SSL	GLOBAL		
	STARTING	GO		
	STRAIGHT_JOIN	GOTO		
	TABLE	GRANT		
	TABLES	GRANTED		
	TERMINATED	GROUP		
	THEN	GROUPING		
	TINYBLOB	HANDLER		
	TINYINT	HAVING		
	TINYTEXT	HIERARCHY		
	TO	HOLD		
	TRAILING	HOST		
	TRIGGER	HOUR		
	UNDO	IDENTITY		
	UNION	IGNORE		
	UNIQUE	ILIKE		
	UNLOCK	IMMEDIATE		
	UNSIGNED	IMMUTABLE		
	UPDATE	IMPLEMENTATION		
	UPGRADE	IMPLICIT		
	USAGE	IN		
	USE	INCREMENT		
	USING	INDEX		
	UTC_DATE	INDICATOR		
	UTC_TIME	INFIX		
	UTC_TIMESTAMP	INHERITS		
	VALUES	INITIALIZE		
	VARBINARY	INITIALLY		
	VARCHAR	INNER		
	VARCHARACTER	INOUT		
	VARYING	INPUT		
	WHEN	INSENSITIVE		
	WHERE	INSERT		
	WHILE	INSTANCE		
	WITH	INSTANTIABLE		
	WRITE	INSTEAD		

	XOR	INT		
	YEAR_MONTH	INTEGER		
	ZEROFILL	INTERSECT		
	False	INTERVAL		
	True	INTO		
		INVOKER		
		IS		
		ISNULL		
		ISOLATION		
		ITERATE		
		JOIN		
		K		
		KEY		
		KEY_MEMBER		
		KEY_TYPE		
		LANCOMPILER		
		LANGUAGE		
		LARGE		
		LAST		
		LATERAL		
		LEADING		
		LEFT		
		LENGTH		
		LESS		
		LEVEL		
		LIKE		
		LIMIT		
		LISTEN		
		LOAD		
		LOCAL		
		LOCALTIME		
		LOCALTIMESTAMP		
		LOCATION		
		LOCATOR		
		LOCK		
		LOWER		
		M		
		MAP		
		MATCH		
		MAX		
		MAXVALUE		
		MESSAGE_LENGTH		
		MESSAGE_OCTET_LENGTH		
		MESSAGE_TEXT		
		METHOD		

		MIN		
		MINUTE		
		MINVALUE		
		MOD		
		MODE		
		MODIFIES		
		MODIFY		
		MODULE		
		MONTH		
		MORE		
		MOVE		
		MUMPS		
		NAME		
		NAMES		
		NATIONAL		
		NATURAL		
		NCHAR		
		NCLOB		
		NEW		
		NEXT		
		NO		
		NOCREATEDB		
		NOCREATEUSER		
		NONE		
		NOT		
		NOTHING		
		NOTIFY		
		NOTNULL		
		NULL		
		NULLABLE		
		NULLIF		
		NUMBER		
		NUMERIC		
		OBJECT		
		OCTET_LENGTH		
		OF		
		OFF		
		OFFSET		
		OIDS		
		OLD		
		ON		
		ONLY		
		OPEN		
		OPERATION		
		OPERATOR		
		OPTION		
		OPTIONS		

		OR		
		ORDER		
		ORDINALITY		
		OUT		
		OUTER		
		OUTPUT		
		OVERLAPS		
		OVERLAY		
		OVERRIDING		
		OWNER		
		PAD		
		PARAMETER		
		PARAMETER_MODE		
		PARAMETER_NAME		
		PARAMETER_ORDINAL_POSITION		
		PARAMETER_SPECIFIC_CATALOG		
		PARAMETER_SPECIFIC_NAME		
		PARAMETER_SPECIFIC_SCHEMA		
		PARAMETERS		
		PARTIAL		
		PASCAL		
		PASSWORD		
		PATH		
		PENDANT		
		PLACING		
		PLI		
		POSITION		
		POSTFIX		
		PRECISION		
		PREFIX		
		PREORDER		
		PREPARE		
		PRESERVE		
		PRIMARY		
		PRIOR		
		PRIVILEGES		
		PROCEDURAL		
		PROCEDURE		
		PUBLIC		
		READ		

		READS		
		REAL		
		RECHECK		
		RECURSIVE		
		REF		
		REFERENCES		
		REFERENCING		
		REINDEX		
		RELATIVE		
		RENAME		
		REPEATABLE		
		REPLACE		
		RESET		
		RESTRICT		
		RESULT		
		RETURN		
		RETURNED_LENGTH		
		RETURNED_OCTET_LENGTH		
		RETURNED_SQLSTATE		
		RETURNS		
		REVOKE		
		RIGHT		
		ROLE		
		ROLLBACK		
		ROLLUP		
		ROUTINE		
		ROUTINE_CATALOG		
		ROUTINE_NAME		
		ROUTINE_SCHEMA		
		ROW		
		ROW_COUNT		
		ROWS		
		RULE		
		SAVEPOINT		
		SCALE		
		SCHEMA		
		SCHEMA_NAME		
		SCOPE		
		SCROLL		
		SEARCH		
		SECOND		
		SECTION		
		SECURITY		

		SELECT		
		SELF		
		SENSITIVE		
		SEQUENCE		
		SERIALIZABLE		
		SERVER_NAME		
		SESSION		
		SESSION_USER		
		SET		
		SETOF		
		SETS		
		SHARE		
		SHOW		
		SIMILAR		
		SIMPLE		
		SIZE		
		SMALLINT		
		SOME		
		SOURCE		
		SPACE		
		SPECIFIC		
		SPECIFIC_NAME		
		SPECIFICTYPE		
		SQL		
		SQLCODE		
		SQLERROR		
		SQLEXCEPTION		
		SQLSTATE		
		SQLWARNING		
		STABLE		
		START		
		STATE		
		STATEMENT		
		STATIC		
		STATISTICS		
		STDIN		
		STDOUT		
		STORAGE		
		STRICT		
		STRUCTURE		
		STYLE		
		SUBCLASS_ORIGIN		
		SUBLIST		
		SUBSTRING		
		SUM		
		SYMMETRIC		
		SYSID		

		SYSTEM		
		SYSTEM_USER		
		TABLE		
		TABLE_NAME		
		TEMP		
		TEMPLATE		
		TEMPORARY		
		TERMINATE		
		THAN		
		THEN		
		TIME		
		TIMESTAMP		
		TIMEZONE_HOUR		
		TIMEZONE_MINUTE		
		TO		
		TOAST		
		TRAILING		
		TRANSACTION		
		TRANSACTION_ACTIVE		
		TRANSACTIONS_COMMITTED		
		TRANSACTIONS_ROLLED_BACK		
		TRANSFORM		
		TRANSFORMS		
		TRANSLATE		
		TRANSLATION		
		TREAT		
		TRIGGER		
		TRIGGER_CATALOG		
		TRIGGER_NAME		
		TRIGGER_SCHEMA		
		TRIM		
		TRUNCATE		
		TRUSTED		
		TYPE		
		UNCOMMITTED		
		UNDER		
		UNENCRYPTED		
		UNION		
		UNIQUE		
		UNKNOWN		
		UNLISTEN		

		UNNAMED		
		UNNEST		
		UNTIL		
		UPDATE		
		UPPER		
		USAGE		
		USER		
		USER_DEFINED_TYPE_CATALOG		
		USER_DEFINED_TYPE_NAME		
		USER_DEFINED_TYPE_SCHEMA		
		USING		
		VACUUM		
		VALID		
		VALIDATOR		
		VALUE		
		VALUES		
		VARCHAR		
		VARIABLE		
		VARYING		
		VERBOSE		
		VERSION		
		VIEW		
		VOLATILE		
		WHEN		
		WHENEVER		
		WHERE		
		WITH		
		WITHOUT		
		WORK		
		WRITE		
		YEAR		
		ZONE		
		False		
		True		

Primary Key Column length

- Primary key columns in reference tables are in the Case tool domain “TYPE”. They have a length and datatype of varchar2(40).
- Primary key columns in business tables that are named %_ID are in the Case tool domain “IDENTIFIER” and have a length and datatype of varchar2(40).
- Primary key columns in tables that are named %_SEQ_NO are in the Case tool domain “SEQ NO” and have a length and datatype of number(8,0).

- Primary key columns in tables that are named %_OBS_NO are in the Case tool domain "OBS NO" and have a length and datatype of number(8,0).

Column Comments

- Every column must have a complete description explaining what it is.
- In some cases, a few examples are helpful to explain how to use the column.
- Where comments are taken from existing glossary sources, the source of the comments will be attributed in the column comments.
- Column long names always appear as the first part of the column comments (in upper case), followed by a colon (:).

Example:

- SOURCE: The source of this row of data, either an agency or organization, a software application, physical document, manual key entry or other origin.

Naming Process

- Construct business name (fully descriptive).
- If required choose appropriate primary word.
- Choose appropriate modifier(s).
- Choose appropriate class word.
- Avoid using business terminology that is not used consistently as this will serve to confuse implementers.
- Determine if name meets maximum length specifications (27 characters for column names).
- If too long, apply abbreviations. Be consistent in how groups of column names are applied, however. For example, column XXX and XXX_OUOM should have the same XXX components.
- Be consistent. Naming abbreviations or class words that have been used in past should be re-used when the context is similar. Remember that the objective is to ensure the model is as consistent and easy to understand as possible.
- It may be difficult to arrive at a name that pleases everyone in the work group. Use the column comments to clarify the meaning and usage of the column.

Source As a column

- Most PPDM tables contain a reference to the SOURCE. Source is taken to mean the direct source of the data that was loaded into this row.
    - If Site A (a data vendor) gets their data from Source Q (a government), and Site B (an O&G company) purchases their data from site A, then the SOURCE for A is Q, and the source for B is A.
- A source may be a business associate, a project, an application or a published reference. The table referencing these sources has been added (R_SOURCE_ORIGIN), so that more complete attribution of data may be supported.
- Where the SOURCE column originates in the host table (as a FK from R_SOURCE), the column should be called SOURCE.

- Where the SOURCE column comes from another table, or there is more than one SOURCE column, each SOURCE column should be prefixed to ensure the columns are not confused during use. (ie: PDEN_SOURCE and PROD_STRING_SOURCE)

Table name references in PPDM

- Table names are used as references in the PPDM model, usually as a FK reference to PPDM_TABLE. Since users are permitted to add tables to the PPDM schema, according to the rules in this document, the length of column references to table names have been extended from 24 characters to 30 characters to allow member extensions to be accommodated.

ARCHITECTURAL PRINCIPLES: CONSTRAINTS

- Referential Integrity refers to the development of enforced (mandatory or optional) relationships between parent and child tables. In a relational database, the concept of Referential Integrity is critical in order to maintain the value of the information contained within the database. It ensures that database information is associated in a consistent and meaningful fashion.
- For example, Referential Integrity prevents a Well Test from being loaded unless the loader can determine which Well the test belongs to. It also forces data loaders to validate entries where needed, so that variations or typographic errors in names (Ex. Alberta, Ablerta, Alta, Ab) do not create data retrieval problems later on.

Architectural Principles Primary Key Constraints

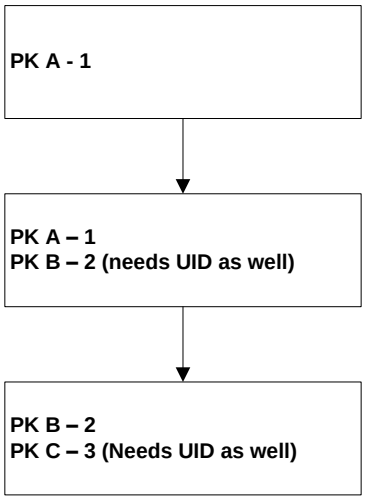
Objectives

- Every table in the PPDM Data Model contains a Primary Key.
    - The Primary key of a PPDM table consists of a column or group of columns that uniquely identify each new occurrence or row of data in the table.
    - Cascading Primary Keys In PPDM may also be used to carry key identification values that are commonly queried down into subordinate tables.
- Primary keys may be used to
    - Uniquely identify a row in a table that is used to enforce referential integrity (RI).
    - Uniquely identify an object “globally”, within a database schema, an instance of a Relational Data Base Management System (RDBMS), or globally.
    - Allow users to accurately identify a record in a table that is of interest. This function may also be satisfied by the use of a Unique Key and is of value for searches and retrievals or for duplicate detection.
    - Aid in efficient indexing and retrieval of data for queries and reports.
    - Reconcile multiple database instances with each other. These systems may be distributed through federation or by replication. They may also support aggregations and roll-ups between systems.
- Primary keys in PPDM should be enforceable using native SQL only, although triggers or procedures may be necessary to generate surrogate components. .

Guidelines

1. Since the Primary Key components of PPDM tables are mandatory, their data values must be known or capable of being created at load time. Values that may be added later, or that are not likely to be known at load time should not be part of the Primary Key.
    a. DATE fields should not be used as Primary Key components, since many sites may not know the value of a date at load time. Additionally, variations in the precision or certainty of dates make them unsuitable as primary key components.
    b. MEASURED VALUES, such as DEPTHS, should not be part of the primary key, as non integer numbers should not be in a primary key. Again, uncertainty or variations in the precision of measured values make them unsuitable as primary key components.

2. Primary keys should allow the implementer to choose a natural key or a surrogate key, as dictated by their business needs, for significant business objects or processes.
    a. RECOMMENDATION: The PPDM Association provisionally suggests that columns named as %_ID should be populated with a GUID whenever possible.
3. Surrogate Keys are unique values with little or no intrinsic meaning. A surrogate key should be used in concert with clearly identified natural values that serve to uniquely identify a row of data. A surrogate key or key component may be used in the following situations.
    - if the number of columns in the natural identifier becomes very large, it may be replaced (all or in part) by a surrogate key - this prevents cascading an unwieldy key. In this case, the natural identifier must be identified and a unique index created for the natural identifier.
    - if the value of the natural identifier is not known at the time the row is created or if there is no natural identifier, surrogate key components such as %_ID, %_OBS_NO or %_SEQ_NO may be added as a component of the Primary key.
4. All components of the Primary key are mandatory.
5.
6. The primary key of a child table may contain components that are inherited from its parent table. Not all components of the parent primary key must cascade into the primary key of the child table.
    a. For new tables in PPDM, 3.9, Primary keys may be created that cascade down one level only, as shown in the illustration. Successful implementation of this strategy will require the creation of unique identifiers for each database row.
    b. If the entire key of a parent is not cascaded into the PK of the child, the component that is cascaded must have a unique constraint in the parent table.



7. Non-static data fields, which are subject to change or modification, should not be used as Primary Key components. There is considerable overhead associated with managing tables with Primary Keys that may change over time.
8. Long, multiple component: Primary keys add some overhead to load and processing time, and can significantly increase the size of subordinate tables, to which the PK is

cascaded. Effort should be expended to make the PK contain fewer, rather than more, components. Do not make a column part of the PK simply to ensure it is mandatory.

Architectural Principles Detecting Unique Objects

The primary role of a database is the storage and management of information (data) that is important to a business. Applications that are built on these data stores must ensure that good data management practices are followed; highly performant applications that use duplicate, corrupt or suspect data add little value to business users.

Functions

- Unique identifiers serve two essential functions in support of this need:

1. Identify duplicates, so they can be consolidated and integrated. In the Oil and Gas business, the combination of data values that is used to determine entries that are unique may vary geographically or over time.
    - For every implementation of PPDM, a team of business users, data managers and systems specialists must determine what combination of business attributes will uniquely identify a row of data.
    - Note – assigning a new system ID to a row of data may not mean it is unique.
2. Identifying data appropriately for retrieval. PPDM allows many columns to be NULL, at the discretion of the implementation. Only the PK columns are mandatory; if these values are system assigned, it may be difficult to build a database that will help users consistently and completely identify objects that meet certain criteria.
    - For every implementation of PPDM, a team of business users, data managers and systems specialists must determine what business attributes should be mandatory for each table.
    - Values that are mandatory must be consistently known at INSERT time.
    - Assigning dummy values to mandatory columns when they are not known at load time is a data management practice that should be discouraged. If the contents of a column cannot be consistently created with appropriate values at INSERT time, they should be left NULL (not made mandatory). Do not assign dummy values to columns!

Best practices

- It is often helpful to create a staging area into which data can be loaded and prepared for insert into a database. If you have incoming data that is created by more than one process before an INSERT into the PPDM database, it may be appropriate to stage it until all the required data is available.
    - Note that rules about what values are mandatory for a data type may vary by the source of the data, by the region in which the data is found or the vintage of the data. In these cases, rules about what is mandatory should be enforced with RULES, rather than database constraints.
- If you are inserting data that has more than one version, discuss the data with the business to determine an appropriate course of action
    - Sometimes data genuinely has more than one version; in this case, it may be appropriate to use one of the versioning methods described in this document. Directional surveys, well headers fall into this category.

- All versions of all values for all data may be managed in the Data Management subject are of PPDM 3.9, along with descriptive meta data about the changes.

- Impact on compliance Your PPDM compliance measure allows you to create additional NOT NULL constraints in PPDM if:
  - You demonstrate satisfactorily that you have effective duplicate detection rules in place. These rules must be documented in the PPDM Rules tables, and an enforcement mechanism must be documented and in place.
  - You demonstrate that you are not creating "dummy" data values during any batch or interactive load processes to satisfy a NOT NULL constraint.
  - You will be asked to provide this information to the PPDM Association, and allow inspection of data systems, before this latitude may be exercised.

An implementation of PPDM that uses IDENTIFER columns as varchar2(20) will be measured as PPDM compliant, provided that this rule is documented in PPDM_RULE, and the values are migrated into PPDM 3.9 and above without shortening the column length.

Architectural Principles Foreign Key Constraints

Objectives

- Foreign Keys are constructed to ensure that foreign keys can be implemented and enforced using only native SQL whenever possible.
- To support good data management practices.

Guidelines

- 1. In some cases, the Foreign Key structure of tables may result in creation of duplicate columns (i.e. 2 or 3 UWI columns). Typically, a business rule exists that requires all of the UWI columns to refer to the same well. In these cases, the model will be reconciled to a single UWI column that is referenced in multiple constraints; this will resolve the data management risk of having each column refer to a different well. This will result in columns that are a component of more than one constraint.
- For example, some tables contain relationships to all of R_COUNTRY, R_PROV_STATE and R_COUNTY (based on variable user needs). The projection process would normally generate the following columns when tables are created:

Note: This LEGACY example is provided for clarification only – these tables and relationships do not exist in PPDM 3.9.

Example:

- R_COUNTRY constraint
  i. COUNTRY3
- R_PROV_STATE constraint
  i. COUNTRY2
  ii. PROV_STATE2
- R_COUNTY constraint
  i. COUNTRY
  ii. PROV_STATE
  iii. COUNTY

- The final modeled result should contain only three columns, with the COUNTRY column constrained three times, the PROV_STATE column constrained twice, and the COUNTY column constrained once.

Example:

- R_COUNTRY constraint
  i. COUNTRY
- R_PROV_STATE constraint
  i. COUNTRY
  ii. PROV_STATE
- R_COUNTY constraint
  i. COUNTRY
  ii. PROV_STATE
  iii. COUNTY

Example:

- WELL_TEST_PRESS_MEAS
  i. This table contains three constraints, each of which includes the column named UWI. All three constraints must reference the same column (this is because each pressure measurement in a well test should reference the recorder used on the same well test, and each period referenced must reference the same well test as used by the recorder.).
  ii. The table also contains three constraints to WELL_TEST. For the same reason as above, all three references must be to the same well test, so each constraint references the same columns.

- Unless the columns referenced by a foreign key are also part of the Primary key, they should always be defined as NULLABLE.
- It is acceptable for the PPDM tables to have columns that are bound by more than one constraint. However, it is generally not desirable for a PPDM table to contain multiple columns that refer to the same value. An exception is granted for COMPONENT tables, which may contain foreign key references to many tables.
-

Implementation Considerations

The data model contains a very large number of foreign keys; a number of issues can potentially arise that should be kept in mind by the implementer.

- 1. Tables contain a primary key that includes a column which is referenced from another table. AREA for example, contains the 2 part PK AREA_ID and AREA_TYPE. The column AREA_TYPE is validated by the reference table R_AREA_TYPE. When entering data into AREA, the FK to R_AREA_TYPE is explicitly enforced. However, AREA is used as a FK in many other tables - in these tables, the reference to R_AREA_TYPE is indirect.
  - **IMPLICATION** - When updating reference table values (or any other table that contains a PK that cascades into other tables) be careful to make sure that both direct and indirect relationships are checked and updated.

- 2. Tables that have lots of foreign keys can be difficult for query optimizers to work with. This is particularly true for the COMPONENT tables. In these cases, use hints in your query tool that will help the optimizer select the correct method. There are a few tips for Optimizing Queries that may help.
- 3. In the PPDM DDL, indexes are only provided for FK columns. Remember that you are responsible for deciding which indexes to use (or not use).

Architectural Principles Check Constraints

Objectives

- Check constraints can be used to enforce the structural integrity of the data model, or to provide validation for certain kinds of values.

Guidelines

All check constraint values should be provided in UPPER CASE only. PPDM work groups may, based on the following principles, define constraints:

- 1. The check constraint may be used to enforce a super-sub type implementation as described in the section on arcs.
- 2. Values that the column may contain are part of a small set, known at design time, and not subject to change. For example, columns named %_IND may only contain the values Y or N (or NULL).
- 3. Tables that are projected at the supertype and have a column to specify the subtype that is described. PPDM 3.9 does not contain any such columns.

Architectural Principles Indexes

Unique Indexes

- Singleton columns with supporting unique indexes are useful in many circumstances. For example, the spatially enabling methodology created in 2002 – 2003 requires the use of a unique identifier for use in ESRI's Geodatabase. This method was deprecated in PPDM 3.9.
- The PPDM data model allows every table to contain a singleton (one column) unique column. This column supports important functions in the data management and other subject areas. These columns are named PPDM_GUID (varchar2(38)). If your implementation will use this column, implementers must add unique indexes and NOT NULL constraints to use it correctly. DDL commands to do this are included in the PPDM data model deliverables. The PPDM compliance measurement process supports this usage.

Index on Foreign Keys

- PPDM provides a starter set of indexes for every foreign key constraint defined in PPDM. These Indexes should not be considered a final solution, but as a starter to assist in developing a suite of indexes based on site requirements.
- We recommend that indexes be placed on the foreign keys, but recognize that too many indexes create their own set of problems. Please examine your implementation requirements and make decisions based on your need for query and update functions.

Performance

- The PPDM Association does not create indexes that are intended to help improve performance, although the structure of Primary and Foreign keys attempts to take some overarching needs into consideration.
- We recommend that you investigate the capabilities of your RDBMS to
    - Index data
    - Partition tables
    - Create views or materialized views

Guidelines

- 1. PPDM will not provide a set of performance indexes.
- 2. PPDM will provide index DDL for foreign key columns; use of these indexes is not mandatory. 3. PPDM will not provide index DDL for non-foreign key columns.

Architectural Principles Arcs

There are two kinds of arcs – exclusive and non-exclusive.

- 1. Exclusive arcs - a table contains FK constraints to a set of tables;The columns in the foreign key are contained in the primary key of the table. Only one of the foreign key relationships may be populated for each row in the table.
- 2. Non-exclusive arcs - a table contains FK constraints to a set of tables; one or more of the parent relationships may be populated for each row in the table.

Guidelines

- 1. PPDM tables may only contain non-exclusive non-mandatory arcs unless the arc is managed as a super-sub type implementation, as are PDEN and SEIS SET.
- 2. PPDM tables may use arcs that manage explicit super/sub type implementations in the data model provided that a SUBTYPE column is added to manage the arc.
    - The primary key of each table in the super/sub type implementation will include a %_SUB TYPE column that is defined as varchar2(30).
    - A check constraint is to be added to the SUBTYPE columns as follows:
        i. In the super type, the values of the SUBTYPE column name equal the table name of any of the subtypes.
        ii. In each of the sub type tables, the value of the TYPE column name may only be equal to the name of the table in which it is found.

Example:

- PDEN : The values of PDEN_SUBTYPE must be one of
    i. PDEN_AREA
    ii. PDEN_BUSIBESS_ASSOC
    iii. PDEN_FACILITY
    iv. PDEN_FIELD
    v. PDEN_LAND_RIGHT

vi. PDEN_LEASE_UNIT
vii. PDEN_OTHER
viii. PDEN_POOL
ix. PDEN_PR_STR_FORM
x. PDEN_PROD_STRING
xi. PDEN_RESENT
xii. PDEN_WELL

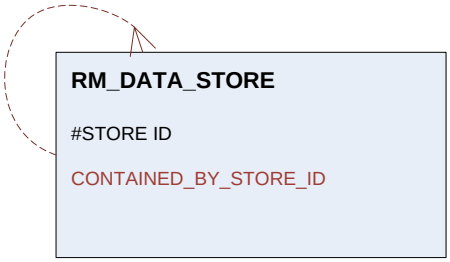
- The Value of PDEN_SUBTYPE must be the name of the table in which the column PDEN_SUBTYPE is contained.

i.

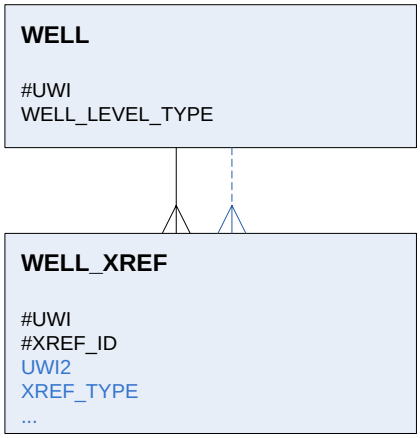
Architectural Principles Recursive Relationships

Recursive relationships, which allow entries in a PPDM tables to refer to other entries in the same table, occur in two forms:

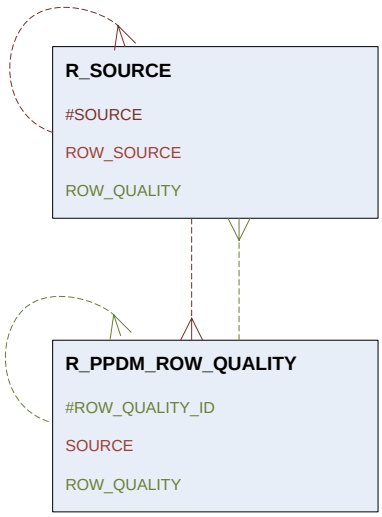
- Recursive relationships (Pig's ears) - these relationships are stored in the parent table. For example, the RM_DATA_STORE table contains a recursive reference to itself. This column is used to capture the containing relationships between data stores.



- Breakout tables - these relationships are stored in a separate table, with two relationships / constraints to the same parent table.



- Each of these constructs creates challenges with the insert process, as it may be necessary to first create the parent row of data, and then update the necessary child rows.
- This problem is evident between the tables R_SOURCE and R_PPDM_ROW_QUALITY. Special care must be taken to ensure that these tables are loaded correctly.



Architectural Principles Constraint Naming Conventions

- Constraint names are created by PPDM using the synonyms assigned to tables. This helps implementers diagnose certain kinds of problems during database operations, such as insert or update errors.

Guidelines

- 1. Constraint name may not be the same as a table name.
- 2. All constraint names must be unique.
- 3. FK constraint name equals the short name (Synonym) of the host table and the Synonym of the source table concatenated with a key type designation (i.e. BA_R_SRC_FK for a foreign key).
- 4. Foreign key constraint names end in _FK. If a table contains more than one relationship with a parent table (as may happen when a table contains several measured values, each of which is referenced by a unit of measure), add a number to the end of the constraint name. (XX_PUOM_FK1)5. Primary key constraint names should be named as SYNONYM _PK. They can be a maximum of 20 characters long.
- 6. Unique index constraint names should be named as SYNONYM _UKn.
- 7. Check constraint names should be named as SYNONYM_CKn

ARCHITECTURAL PRINCIPLES DESIGN ISSUES

The following sections deal with specific modeling issues, and lay out the recommendations of the modeling committee for handling each.

Architectural Principles NULL Data

Data values that are unknown at load time should present as few problems to users of the data model as possible. Consideration must be given to the international composition of the membership, and the variability of data availability and business processes this implies.

Guidelines

- All components of the Primary Key are mandatory. During design, care must be taken to ensure that the values of all components can be determined at load time.
- All other columns in a table must be optional. User sites should enforce optionality through use of procedures or triggers.
- The meaning of a NULL value should be considered during design, and requirements accommodated in the structure.
- The PPDM Meta model may be used to fully describe the history of columns that are NULL, using the tables PPDM_AUDIT_HISTORY and PPDM_QUALITY_CONTROL to capture information about how information was searched and quality controlled.

Architectural Principles Dates

- Data, particularly archival or historical data, is often incomplete and may be missing values that are present for new data. Even new data may be incomplete. This is particularly a problem for date values in PPDM; it is common for dates to be missing or inexact (i.e. year or month and year only).
- PPDM queries often require use of "search between dates" clauses. These queries are greatly benefited by the use of the complex data type DATE. However, when the date known is only approximate, creating dummy values in a date field in order to facilitate queries may present data integrity or veracity problems. Clearly, there are cases when both search functionality and accuracy of retrieval are needed.

Modeling Guidelines

- Consideration must be given to the impact on business queries if the DATE field may be NULL. NULL date fields negatively impact search between dates queries; on the other hand, populating columns with meaningless dummy data leads to mistrust by users. Recommendations for population procedures should be made during the modeling process.
- PPDM offers three methods for modeling dates:
    - DATE as a native database data type. All date values should be modeled in this form, to enable searches and indexing.
    - DATE_DESC as a varchar2 (8) field. Where the date is inexact and it is desirable to indicate the level to which it is known, DATE_DESC may be used in addition to the DATE field. This column indicates the level of precision to which the date is known (YYYY, YYYYMM, YYYYMMDD, YYYQQ)
    - XXX_DATE_STRING. Date as a text string representation of the date. Supported for legacy tables in the production reporting module. Should not be used in the future.

- Time should be modeled as:
  - TIME_ELAPSED: to indicate that the value represents a total amount of elapsed time.
  - TIME: Time of Day. This column must also be accompanied by TIMEZONE (and foreign key).

Use of DATE fields in Primary Keys

- DATE fields are not used as part of any Primary key. A unique component, such as an %_OBS_NO, %_SEQ_NO or %_ID field is used instead.
- In many cases, dates are not known when loading data into PPDM. If dates are part of a Primary key, this forces users to enter false (dummy) data into the PK. This introduces a level of uncertainty into the quality of the Primary Key fields – a highly undesirable situation.

Recommendation: Population of unknown dates

- Policy should be to populate in a way that will include as much data in date searches as possible.
- RDBMS products handle null date components differently. This can have a significant impact when you are integrating systems that use or have used different RDBMS products or versions of the same product. It's very important to have and implement consistent business rules for dates.
- PPDM recommends this business practice:
  - If the month is unknown set to 01. DATE_DESC should be set to YYYY
  - If the day is unknown set to 01. DATE DESC should be set to YYYYMM
  - If the year is unknown column must be NULL. DATE_DESC should also be NULL

Recommendation: Populating Time and Date

- Although the DATE complex data type can support dates to a high degree of precision, they usually default to a set value if they are left NULL on insert. This results in some uncertainty about the difference between a real time and a default time. In cases where time of day is important, a separate column is created to capture the time component. If time is not known, this value must be left empty (NULL).
- Time of day must always be accompanied by TIMEZONE. Do not leave TIMEZONE null if the time has been inserted.

ARCHITECTURAL PRINCIPLES VERSIONING

Versioning occurs when multiple copies of an entity attribute need to be stored in order to reflect different values of the data. There are four types of versioning:

Source Versioning

- Data values may differ among different information sources. The preferred version is stored in the primary table and other versions are in an auxiliary table.

Example:

- WELL and WELL_VERSION

Guidelines

- A main table, which contains the 'preferred' values for the object is created. It does not contain a SOURCE reference. (i.e. WELL)
- A version table is created to track all versions of data for the object.
- The version table should be named XXX_VERSION, where XXX is the object this is a version of.
- SOURCE is part of the Primary Key for the VERSION table. (i.e. WELL_VERSION)

Inherited Versioning

- The values in child tables are referenced to the source version of the parent table; a different source is not explicitly defined in the child tables.

Example:

- WELL_CORE and its children

Guidelines

- SOURCE is part of the Primary Key, and is carried down to child tables from the table at the 'top' of the family of tables.
- Child tables inherit the Primary key of the parent.
- A new and different SOURCE may not be defined for child tables in the family of tables.

Example:

- For the WELL_TEST set of tables, SOURCE is part of the PK for WELL_TEST - additional sources cannot be defined for its subordinate tables.

Chronological Versioning

Information about an object, such as its status, may vary in time, but the historical information is required for auditing, legal or other purposes.

Example:

a. WELL and WELL_STATUS

Guidelines

- Current value may be retained in the host table for queries in legacy tables, but is discouraged for new tables.
- Complete set of values is stored in the status table.
- Status tables contain status information in two columns
    - STATUS_TYPE is used to reference the kind of status (such a financial status, construction status, legal status, vendor status, regulatory status etc)

- STATUS is used to contain the status value
- See the PPDM web site on Well Status and Classification for examples on how status information can be usefully decomposed into atomic constituent parts for effective query and retrieval.

- DATE should not be part of the Primary Key for the version table. Use an OBS NO column for the Primary Key component in the versioned table.

Example:

Well status history is kept in the WELL_STATUS table.Alias Versioning

Identification of an object, such as a well or a seismic line, may vary.

Example:

- SEIS_SET and SEIS_SET_ALIAS

Guidelines

- For legacy tables, you may maintain the preferred version of the name, which is normally used for reporting and retrieval, in the object table. This is discouraged unless necessary for performance reasons or to uniquely identify an object.
- All names, codes and other identifiers are kept in a separate table, which should be named XXX_ALIAS, where XXX is the name of the object this is an alias for.
- R_SOURCE is used as a foreign key (often part of the Primary key) in the _ALIAS table to indicate what the source of the alias was.
- In some cases, dates when the alias was created, and the REASON the alias exists are important pieces of information.

Column Names

- PK columns
- XXX_ALIAS_ID
- ACTIVE_IND
- ALIAS_CODE
- ASIAS_FULL_NAME
- ALIAS_REASON_TYPE (and FK)
- ALIAS_SHORT_NAME
- ALIAS_TYPE (and FK)
- AMENDED_DATE
- CREATED_DATE
- EFFECTIVE_DATE
- EXPIRY_DATE
- LANGUAGE_ID (and FK)
- ORIGINAL_IND
- OWNER_BA_ID (and FK)
- PPDM_GUID
- PREFERRED_IND

- REASON_DESC
- REMARK
- SOURCE (and FK)
- SOURCE_DOCUMENT (and FK)
- STRUCKOFF_DATE
- SW_APPLICATION_ID (and FK)
- USE_RULE
- ROW_CHANGED_BY
- ROW_CHANGED_DATE
- ROW_CREATED_BY
- ROW_CREATED_DATE

ARCHITECTURAL PRINCIPLES VERTICAL TABLES

About Vertical Tables

A vertical table is a table that allows a column to contain values that reference more than one kind of information. These tables generally follow the structure *Property type* and *Property value*. For example, the table EQUIPMENT_SPEC allows you to store any kind of specification about a piece of equipment. For pumps, you might store the capacity of the pump and the speed at which it can operate. For microscopes, you might store the resolution that it is capable of resolving. For and engine, you might store the horsepower rating and number of cylinders. Each of these values can be stored by defining an appropriate Property Type.

Vertical tables are used when the number of possible data values cannot be determined at design time, or the number of data value types is very large. Vertical tables should be designed with care to ensure that best data management practices can be supported. Vertical tables are often subject to degradation of the content they contain; this is because the tables are so flexible that they are difficult to manage. Our objective is to create vertical tables that allow the user the highest degree of control possible, so that the data is less subject to decay.

Specific design templates have been created for vertical tables, please be sure to use them.

Property type control

Each vertical table is controlled by a reference table that controls the type of property being stored. In an equipment specification table, for example, the property type table would contain the list of specification types (diameter, length, mass etc).

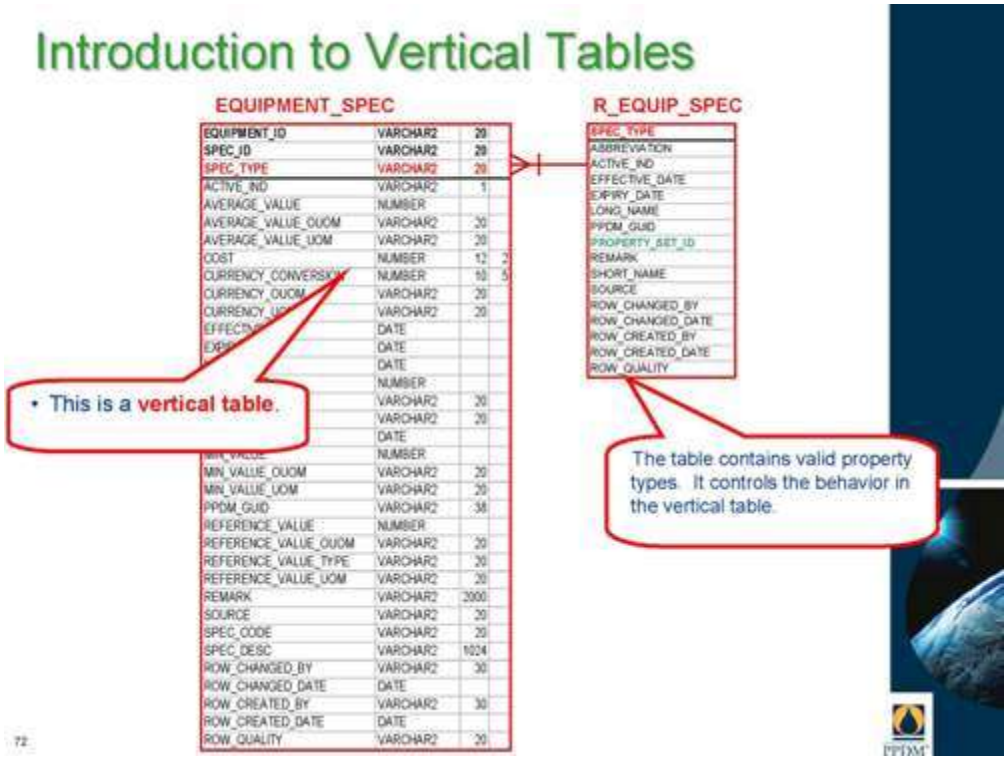
The Property type reference table will contain a foreign key from PPDM PROPERTY SET. The PROPERTY SET determines which columns should be used to capture information about each possible property type.

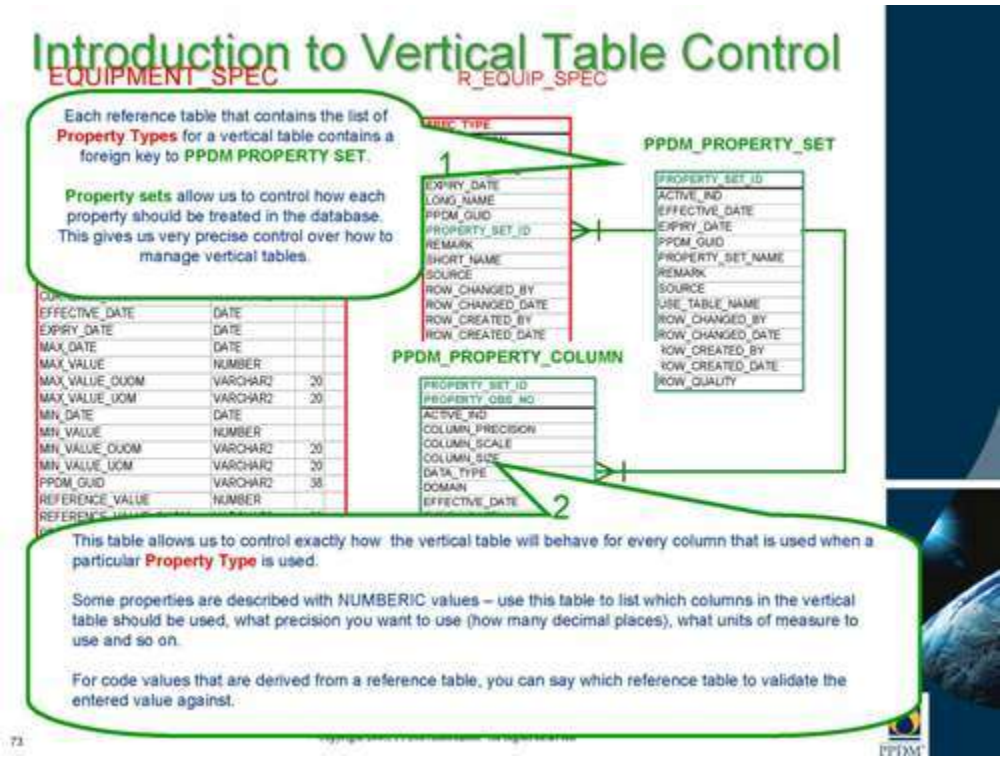
PPDM PROPERTY SET

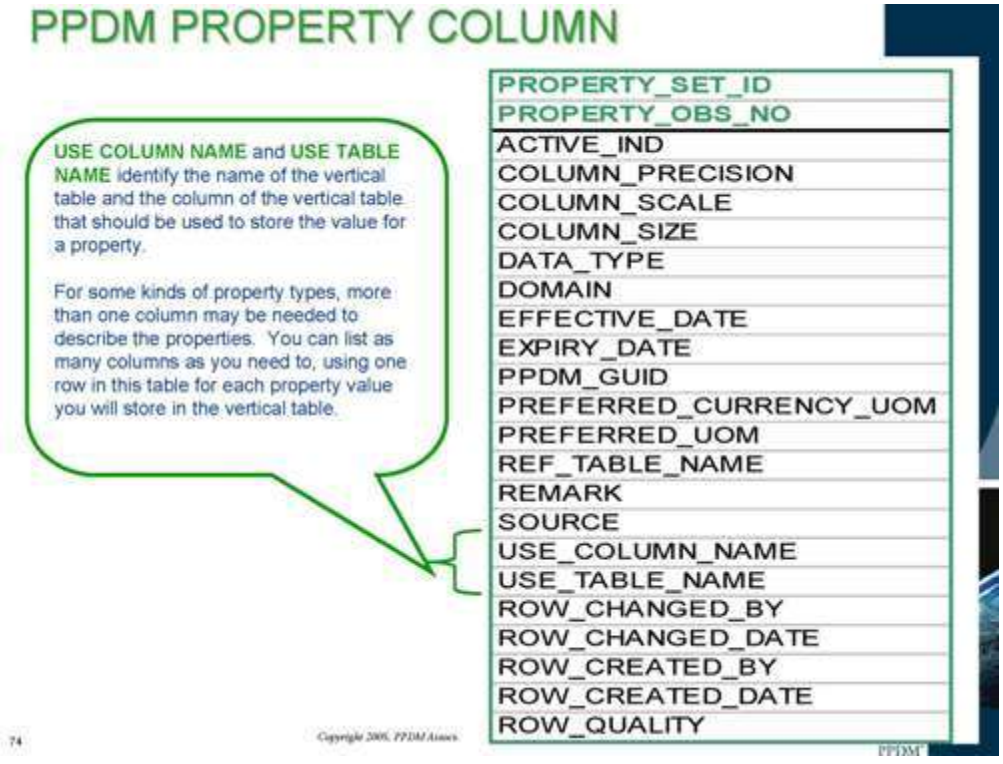
This table and its child PPDM PROPERTY COLUMN control the behavior of each kind of property in the business (vertical) table.

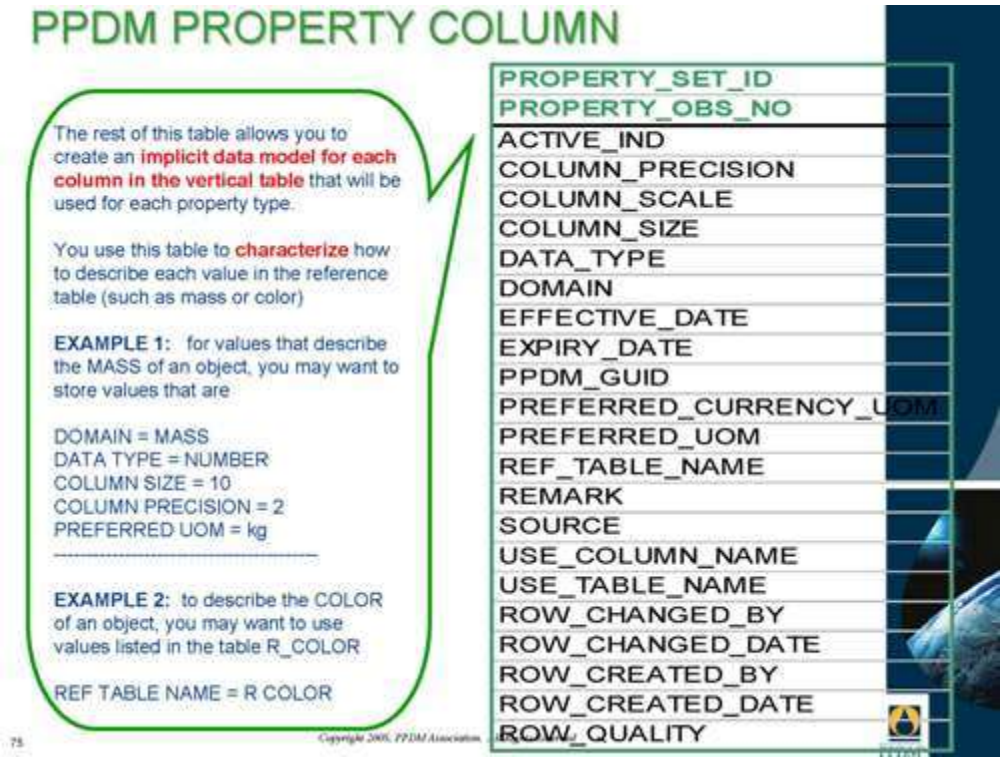
Example:

- Value in the reference table is MASS
  i. In the business table, this information should be stored in the AVERAGE VALUE column, with a preferred unit of measure of kilograms and 2 decimal places of precision.
  ii. Note that this cannot be controlled by the database, and will need to be managed procedurally.
- Value in the reference table is COLOR
  i. In the business table, this information should be stored in the SPEC CODE table and validated against the reference table R_COLOR.
  ii. Note that in this case, the validation cannot be controlled by database constraints, and will have to be managed procedurally.

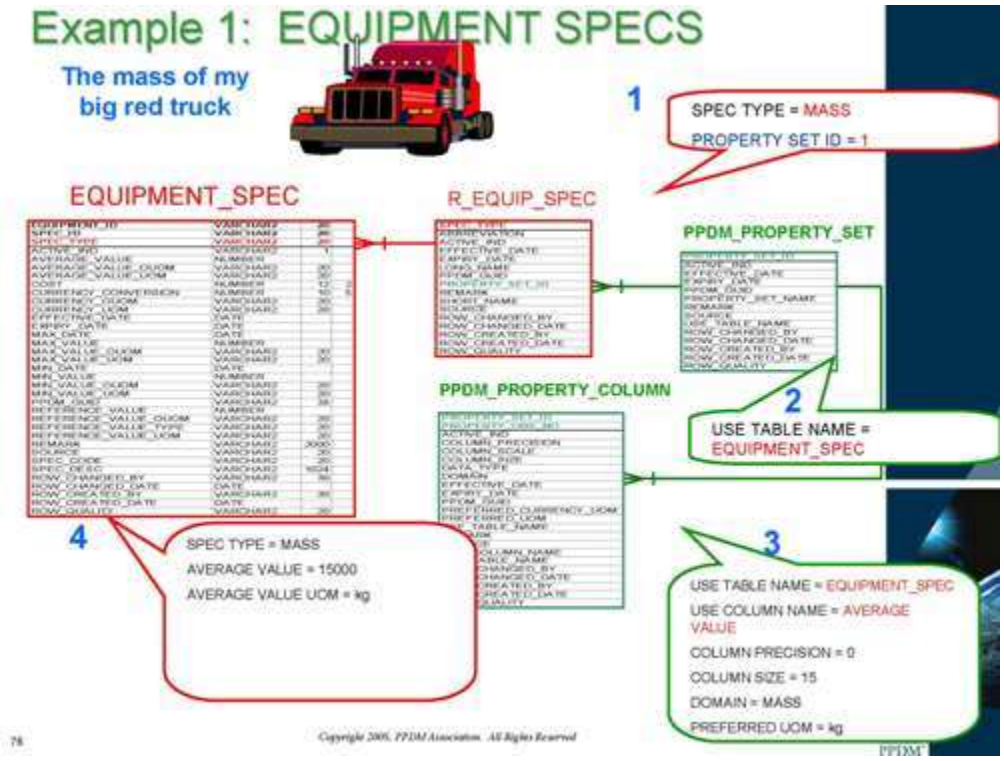


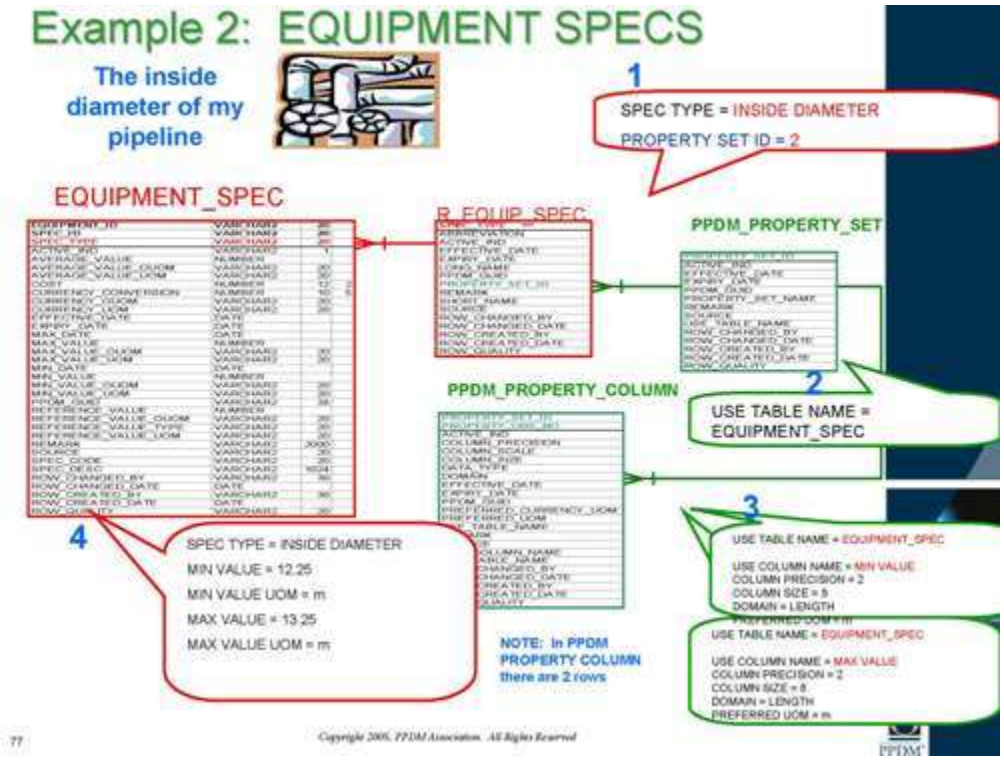


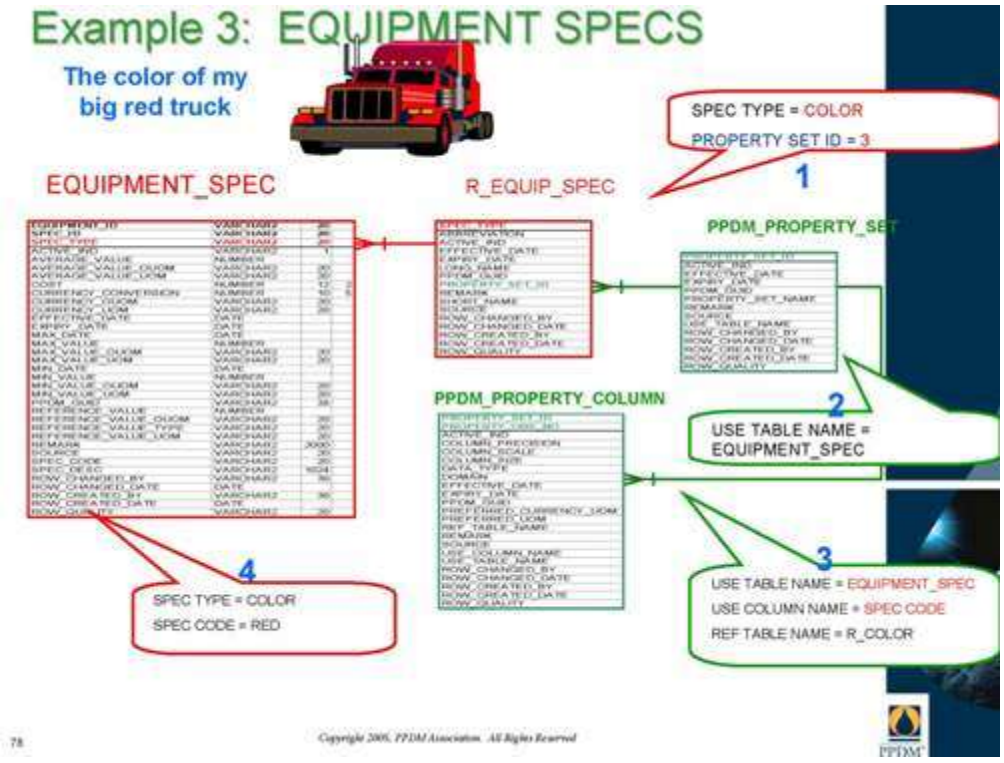


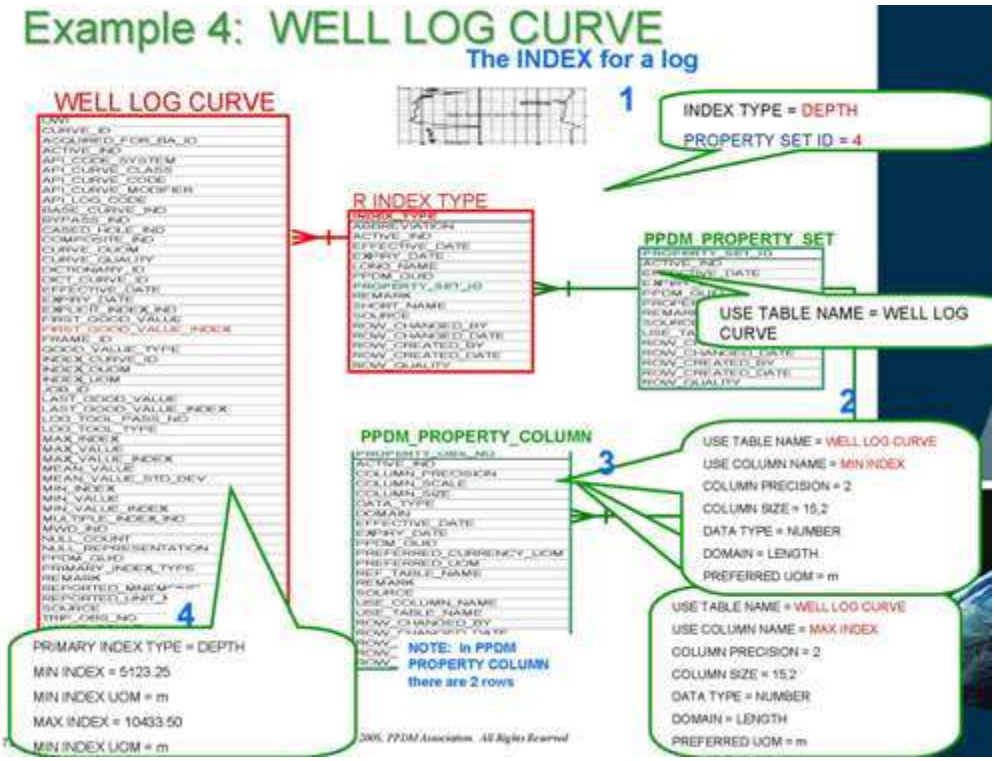


EXAMPLES:









ARCHITECTURAL PRINCIPLES SEQUENCE CONTROL

Sequences in the database may be ordered by observation, or by some other factor, such as a spatial relationship or an event. There are two types of sequences:

Observation Sequence

- Each occurrence is based on a separate observation, such as for well production tests. The order of events or occurrences is not significant.

Example:

- WELL_PRESSURE.PRESSURE_OBS_NO

Guidelines

All occurrences of observation sequences will be named using the class word OBS_NO:

- XXX_OBS_NO
- The format will be number 8.0

Example:

- AOF_OBS_NO

- PRESSURE_OBS_NO

Ordered Sequence

Occurrences are ordered based on sequential information, such as spatial location, sequence or chronology.

Example:

- SEIS_POINT.POINT_SEQ_NO

Guidelines

- All occurrences of ordered sequences will be named using the class word SEQ_NO:
  - XXX_SEQ_NO
  - The format will be number 8.0

Example:

- POINT_SEQ_NO
- WELL_SEQ_NO
- REMARK_SEQ_NO

Chronological Event

Occurrences are ordered based on time, but the date or time may not always be known. In the case, a unique identifier is added to the PK, and the date is a nullable column in the table. Dates should not be added to primary keys.

Example:

- WELL_STATUS
  i. UWI
  ii. SOURCE
  iii. STATUS_ID

ARCHITECTURAL PRINCIPLES ECONOMICS AND FINANCIAL

PPDM does not, at this time, provide an economic model to handle AFE's and other financial information other than at a high level. Feedback from the membership indicates that most members have financial systems in place, and merely wish to obtain summary information or pointers into the financial databases.

Guidelines

- 1. Currency values should be stored in a single unit of measure for a column in a table to prevent scalability problems from developing. In addition, as many currency-based questions are likely to combine data from more than one column, it is desirable to encourage all values in a table or even in a subject area (CONTRACTS) to utilize the same stored currency type .
  - Use the currency domain (NUMBER 12,2).

- 2. The column CURRENCY_OUOM should accompany each currency. In many cases, one set of these is sufficient for a row in a table, even if there is more than one currency value stored.
    - Indicates the units of currency as originally received by the payee.
- 3. The column CURRENCY CONVERSION should accompany each currency field. In many cases, one set of these is sufficient for a row in a table, even if there is more than one currency value stored. Use the following column information:
    - Use the currency conversion domain (NUMBER 10,5)
- CURRENCY CONVERSION RATE: the rate applied to convert the currency to its original monetary UOM from the stored UOM. This value is valid for this row in this table only. When this value is multiplied by the STORED currency value, the original value of the transaction in the original currency is obtained.

Modeling Financial Information

- 1. At this time, PPDM does not attempt to fully model financial or accounting information. Intent of modeling should be to capture summary information and a connection to an accounting system only.
- 2. AFE or cost center numbers may be captured in the data model:
    - Using the FINANCE module. Relationships to FINANCE are captured in the table FIN COMPONENT.
    - As a column in a business table (see the OBLIGATION subject area for some examples).

ARCHITECTURAL PRINCIPLES COLUMN PRECISION

Number columns in PPDM should be defined in terms of their total length and the number of decimals allowed, except for vertical tables, where the total length and precision are defined in PPDM_PROPERTY_COLUMN. This enforces use of a standard level of precision in the values stored.

Guidelines

- 1. For numeric columns that capture a single type of numeric information (such as depth), the total field length and the number of decimal places should be defined.
- 2. For numeric columns in a vertical table, the column length and precision should be NUMERIC UNRESTRICTED. In these cases, the total length and precision for each kind of numeric value should be defined in the vertical control table using PPDM PROPERTY SET.
- 3. Column types used frequently should be controlled through a CASE domain, to ensure consistency of use. Following is a partial list of number domains in PPDM 3.8:
    - AGE 15,5
    - CURRENCY 12,2
    - CURRENCY_CONVERSION 10,5
    - DEPTH 10,5
    - DIAMETER 8,3
    - ELEV 10,5
    - LAT 12,7

- LONG 12,7
- OFFSET 8,2
- PARAMETER_VALUE 10,5
- PARCEL_SIZE 13,5
- PERCENT_INT 15,12
- VELOCITY 10,5
- Vertical tables NUMBER UNRESTRICTED (no precision defined, allows user to define size)
- VISCOSITY 8,3
- VOLUME 14,4

ARCHITECTURAL PRINCIPLES COLUMN DATA TYPES

Guidelines

The following data types are supported in PPDM version 3.9:

- 1. DATE
- 2. CHAR
    - VARCHAR2 (Oracle DDL)
    - VARCHAR (SQL*Server DDL)
- 3. NUMBER
- 4. BLOB information. It should be noted that occurrences of this type should place the column in a separate table with only primary keys and standard audit columns, rather than embedding it in a business table with other attribute information. The type of BLOB to implement should be determined by the work group.
    - CLOB etc. (Oracle DDL only)
    - TEXT / IMAGE / VBINARY (SQL*Server only)

ARCHITECTURAL PRINCIPLES STANDARDIZED COLUMNS

All PPDM tables will contain the following columns. Many of these columns are intended to provide audit tracking capabilities and to indicate which values are current and which are not. However, the EFFECTIVE and EXPIRY date columns are intended for business information and should not be absorbed by system or data loading functions.

SOURCE

This row indicates the source from which the data was obtained. Note that this may be a business associate, an application, a service provider, original paper documents etc. Use common sense business rules when populating this information.

This column (with a constraint to R_SOURCE) should be added to every table, unless the work group determines that the SOURCE of the parent table will always equal the source of the child table.

Example: SEIS_SET contains the SOURCE that determines source for SEIS_POINT

ROW CHANGED BY

Should be added to every table in the data model. In many cases, this will be the system assigned userID of the person who changed the data, but business rules may dictate the use of other information.

Many users populate this value by trigger on INSERT and on UPDATE. Others populate this value by trigger only on UPDATE (in this method, the column value may be NULL, making certain types of query more difficult). An implementation must create a business rule about which method they will use and enforce consistency throughout their systems.

ROW CHANGED DATE

Should be added to every table in the data model. Usually the system date of the change.

Many users populate this value by trigger on INSERT and on UPDATE. Others populate this value by trigger only on UPDATE (in this method, the column value may be NULL, making certain types of query more difficult). An implementation must create a business rule about which method they will use and enforce consistency throughout their systems.

ROW CREATED BY

Should be added to every table in the data model. In many cases, this will be the system assigned userID of the person who changed the data, but business rules may dictate the use of other information.

Some members may choose to implement this column using a NOT NULL constraint. In this case, you may use an optional procedure provided by the PPDM Association that alters these columns. However, you should note that some companies cannot enforce a not null constraint of this type for legal or policy reason. Care should be taken to research the implications before implementing this script.

Many users populate this value by trigger on INSERT.

ROW CREATED DATE

Should be added to every table in the data model. Usually the system date that the data was added.

Some members may choose to implement this column using a NOT NULL constraint. In this case, you may use an optional procedure provided by the PPDM Association that alters these columns. However, you should note that some companies cannot enforce a not null constraint of this type for legal or policy reason. Care should be taken to research the implications before implementing this script.

Many users populate this value by trigger on INSERT.

ROW EFFECTIVE DATE

Should be added to every table in the data model. Used for system or data management functions to indicate when a row of data came into effect as a live (or used) row.

ROW EXPIRY DATE

Should be added to every table in the data model. Used for system or data management functions to indicate when a row of data is no longer in effect as a live (or used) row.

REMARK

Should be added to every table in the data model, unless there is a %_REMARK subordinate table. Note that not more than one remark column should be added to any table. If more than one remark column is needed, create a subordinate %_REMARK table.

EFFECTIVE DATE

Should be added to every table in the data model. Indicates the date that this data first came into effect from a business perspective. In business transactions, this date may be before or after the date the data is added to the database. For reference tables, this is often the date that a value is included for use (approved by the business).

Please note that this column is NOT intended to be used for information aboaut the technical implementation or management of the data in a table. It is a business value, and should be derived from business information. If a system date is to be used, the system date should be based on the needs of the business, rather than on dates that system transactions occur (even if those dates may turn out to be the same date).

EXPIRY DATE

Should be added to every table in the data model. Indicates the date that this data is no longer in effect. In business transactions, this date is often the date that the information becomes obsolete, invalid or is replaced, such as the date that a contract expires, or a reference value is not to be used any longer. Note that expiry dates may exist in the future.

Please note that this column is NOT intended to be used for information aboaut the technical implementation or management of the data in a table. It is a business value, and should be derived from business information. If a system date is to be used, the system date should be based on the needs of the business, rather than on dates that system transactions occur (even if those dates may turn out to be the same date).

ACTIVE IND

Should be added to every table in the data model. Indicates whether this row of data is currently valid. Setting this flag to N allows users to retain archival data but disallow its use in current application.

This column is implemented as varchar2(1). Note that a char format would be equally effective, but the destructive change to the model is such that the modeling committee has recommended against using the char data type.

ROW MANAGEMENT GUID

Should be added to every table in support of GUID practices. An optional set of DDL is provided that will change the GUID columns to NOT NULL and add unique indexes to each. This column is implemented as a surrogate value to uniquely identify a row of data within the table, the database instance, the RDBMS instance or globally, as implemented by the member.

- For implementation, the PPDM modeling committee recommends the use of a GUID as an alpha-numeric string.
- Database assigned ROW IDENTIFIERS are not recommended for use in this column.
- A GUID should be assigned to a row of data and propagated with that row in all of its iterations. For example,
    - A row of data that exists in a test, development and production environment should use the same GUID.
    - A row of data that is federated should use the same GUID.
    - A row of data that is migrated into another system with a process of transformation should normally be assigned a new GUID, although some judgment is needed to determine whether the transformed row identifies the same object or a new object.
    - These decisions should be clearly identified and documented using the PPDM_RULE or PPDM_MAP_RULE tables.

It is recognized that some members may chose to use unsigned integers to uniquely identify rows. We recommend that these users EXTEND the data model to include a new column with an INTEGER datatype, and add NOT NULL and UNIQUE constraints to the extended column. If clearly identified and documented, this addition will not result in loss of compliance points.

Member recommendation for PPDM 3.9

PPDM 3.9: The table comment for each column will reference the full name "ROW MANAGEMENT GUID", but the column name will not be altered

PPDM 3.10 or 4.x: the column name will be changed to ROW MANAGEMENT GUID

Best Practice

The value in this column should be populated with a global identifier, not a row ID. Generate the GUID when a row of data is created, and propagate this value as the data is disseminated across federated or versioned systems (test, dev, prod etc). In this sense, the value is not fully unique, as it may be replicated across many systems; the purpose is to uniquely identify that row of data for data management purposes.

If you transform or otherwise modify the row of data as it is moved from system to system, a new ROW MANAGEMENT GUID should be created.

ROW QUALITY

Should be added to every table in the data model. Indicates the quality of a row of data in a table. Information about every attribute in a row may be stored in other tables in PPDM.

ARCHITECTURAL PRINCIPLES SQL VIEW DEFINITIONS

Data models, starting with version 3.5, may include view definitions as part of the model deliverables. These view definitions may be particularly useful in cases where a generic table can be made easier to understand or use through views.

XREF Tables

- Cross reference tables, particularly where they are used to cross reference values in a super-sub type implementation, may be used to view special types of relationships.

Example:

- LR_XREF views for relationships Title to lease, lease to part of lease, unit to unit tract etc.

Spatial Views

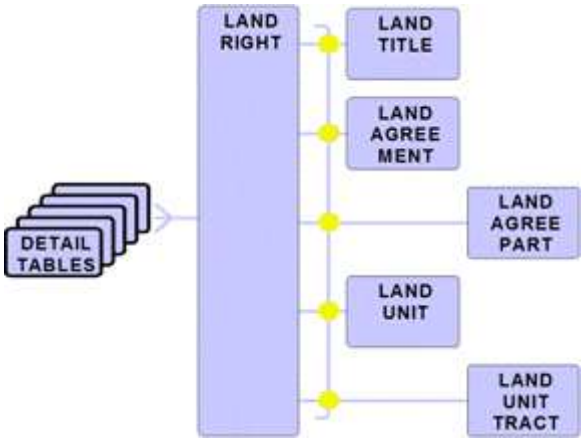
Spatially enabling information comes with a cost; specifically you can only effectively query information contained in the business table that has been spatially enabled. Views that flatten information for query purposes may be created. Note that PPDM Lite contains definitions for flattened representations of data that have been created through the PPDM work group process.

ARCHITECTURAL PRINCIPLES SUPER / SUB TYPES

When Does PPDM 3.8 Use Super Sub Types?

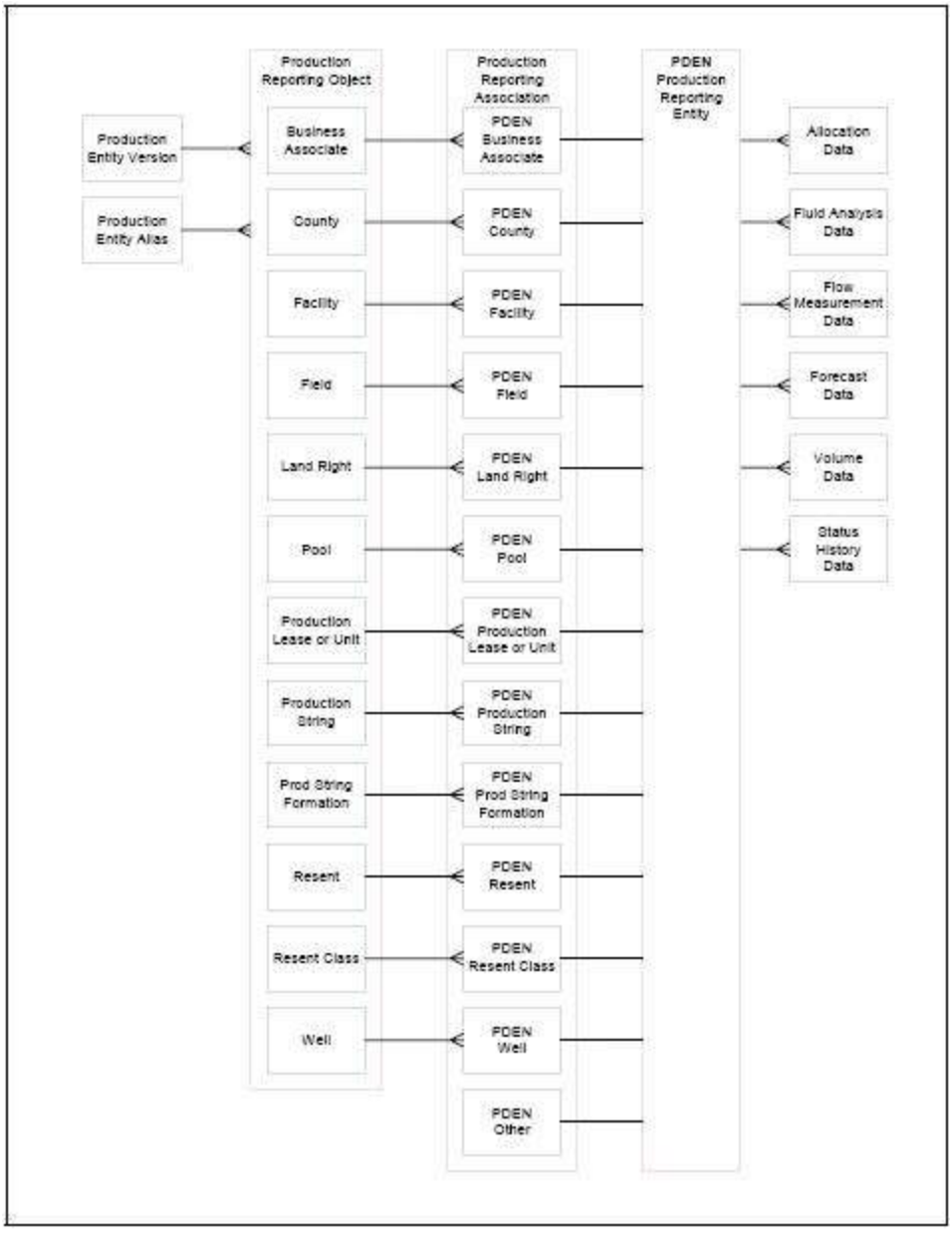
Super Sub type relationships are used in PPDM under certain conditions:

1. **The data model must manage a business object or entity that occurs in various forms.** While each form may be described using slightly different attributes, the overall behavior of the business object is much the same. For example, the LAND RIGHTS module is divided into land rights that are owned, leased or combined with others (pooled). Information that describes each kind of land right is stored in each sub type table. Information that may describe any land right is stored in the super type table. The super type table manages all relationships to other parts of the data model; this reduces the number of foreign keys that might otherwise be in the model.



2. An entity in the data model must take on the identity of a business object that is described and defined in another part of the data model. For example, the PRODUCTION ENTITIES module in PPDM 3.8 is used to capture information about production volumes. Production

may be captured for many kinds of business object, including WELL, BUSINESS_ASSOCIATE, POOL, FIELD and more. In this case, the PDEN table takes on the identity of the necessary business object via the sub type tables, which contain a foreign key to the object that the PDEN is using.



Guidelines

- The relationship between the super and sub type table is always 1:1 (the PK in both tables contain the same columns)
- One of the PK columns is defined as varchar2(30), and is validated by a check constraint

- In the **Super Type table**, the Check Constraint requires the value of the column to equal the name of ANY of the sub type tables.
- In each **Sub Type table**, the Check Constraint requires the value of the column name to equal the name of the table the column is in. In all cases, this value must also exist in the check constraint for the super type table.
- Relationships to other business modules in PPDM should be given at the Super Type level
- Relationships to detail tables in the subject areas should be given at the Super Type level, un less the detail relates only to a specific sub type.

Using Super Sub type tables

- It is important to be sure that you populate BOTH the super type table and the sub type table when creating data. If you do not, you may lose important information or relationships.
- For example, if you populate PDEN and PDEN_SUBTYPE = 'PDEN_WELL', you must also put a row of data into PDEN_WELL. The FK relationship to the well whose identity the PDEN is assuming is in PDEN_WELL.

ARCHITECTURAL PRINCIPLES REFERENCE TABLES

Reference tables provide a valid list of values for a particular column(s). These are sometimes referred to as look-up tables, code tables or decode tables.

Objective

- To encourage inter-operability through the usage of standard structures and population of reference tables while encouraging the use of referential integrity.
- To provide a mechanism that uses disk space efficiently, which improves data quality and consistency and which facilitates simple and consistent data retrieval.
- A reference table may be used to validate data input; it contains a set of authorized values for a particular field.

Guidelines

- These tables are to be named using R_ as a prefix.
- Reference entities will be implemented as separate tables rather than one large table with a code type column.
- Individual tables must be separately defined; this facilitates model uptake and comprehension.
- Relationships can be enforced using foreign key constraints (this cannot be done using one large code table with code type as part of the primary key).
- Every reference table will include a foreign key relationship to the reference table R_SOURCE, to indicate the source of each value in the reference table.
- Some reference tables have relationships to other reference tables (i.e. R_WELL_STATUS contains a FK from R_WELL_STATUS_TYPE)).
- Other entities in the model exhibit reference behavior but are not identified as reference entities. (For example BUSINESS ASSOCIATE, FORMATION, ELLIPSOID etc.) Generally, if a table is used for any purpose other than validation of entries, it should not be called a reference table.

Reference table use

- R_CODE_VERSION enables members to handle multiple sources of reference values. Multiple sources must then be rationalized into a single set of values.
- The meaning of a NULL value in PPDM must be considered (unavailable, not applicable, lost data, expected to arrive later ….). Reference table values should contain default values which are consistent in form and which articulate the differences between meanings.
- Reference tables that are used to control the behavior of vertical tables contain a foreign key to the table PPDM_PROPERTY_SET. This reference may be used to control the behavior of different kinds of reference values.

Population of Reference Tables

- The Association may, on occasion, input some selected sample entries into reference tables where the information has specific value to the model, or to understanding the model.
- The PPDM Association's Data Content project seeks out and publishes (or references) valid content for reference tables where this is appropriate. This information may be found here http://www.ppdm.org/standards/content/index.html.

Structure

- Table structure template to be used for reference tables is as follows. Note that all columns are controlled by domains in the CASE tool.
- CREATE TABLE R_SAMPLE_TABLE(

```
sample_table_type VARCHAR(20) not null,
abbreviation VARCHAR(12),
active_ind VARCHAR(1),
effective_date DATE,
expiry_date DATE,
long_name VARCHAR(255),
PPDM_GUID VARCHAR(38),
short_name VARCHAR(30),
source VARCHAR(20),
remark VARCHAR(2000),
row_quality VARCHAR(20),
row_changed_by DATE,
row_changed_date VARCHAR(30),
row_created_by DATE
row_created_date VARCHAR(30)
CONSTRAINT r_sample_table_pk PRIMARY KEY (sample_table_type));
```

Related Column Naming Conventions

- Columns representing the relationships to the reference tables are generally named after the reference table name except where common business usage provides an alternate name or a qualifier is required to explicitly define the column usage. For example, columns that reference BUSINESS_ASSOCIATE may be called OPERATOR, COMPANY, BROKER etc.

ARCHITECTURAL PRINCIPLES DOMAINS

- PPDM uses two kinds of domains
    - The PPDM Data Management model allows columns to be grouped into logical sets based on the type of value contained by the column. For example, a Data Management domain might group all Latitude and Longitude values.
    - The CASE tool allows CASE domains to be defined; they can be used to ensure consistent usage of column format and size in the model.

Objectives

- CASE domains
    - Use of CASE domains greatly facilitates the process of ensuring column name and format consistency during the data modeling process.
    - CASE Domains are to be established whenever common attribute fields are used repetitively throughout the model.
- CASE domains are to be used as appropriate in all discipline areas of the model.

Identifiers and Types

- Most tables in PPDM 3.9 contain a value that is contained in the CASE "Identifier" or "Type" domain. Identifiers are used as primary key identifiers in business tables. Types are used as primary key identifiers in reference tables.
- Identifiers in the Data Management module are designed to store table names, system identifiers, column names, constraint names etc.
- Identifiers usually include the class word %_ID in the column name
    - In PPDM 3.8, these columns were varchar2(20)
    - In PPDM 3.9 increase this to varchar2(40)
- Types often include the class word %_TYPE in the column name
    - In PPDM 3.8, these columns were varchar2(20)
    - In PPDM 3.9 increase this to varchar2(40)

Guidelines

- For a list of domains in current use, contact the PPDM office. Appendix B contains a snapshot of domains in use for version 3.8 of the Data Model.
- Existing domains should be used when possible. There should be one Lat/Long domain, for example.
- Additional domains should be established as needed.
- New domains should be communicated with all work groups.

ARCHITECTURAL PRINCIPLES UNITS OF MEASURE

- Numeric, measured quantities in the database must be qualified with a "units of measure" indicator. At times, the original units of measure must also be retained for regulatory, audit or tracking purposes.

PPDM Data Model 3.9 Architectural Principles

Objectives

- The Units of Measure module in PPDM allows the member to:
  - Provide and identify the UOM for all numeric value in the database.
  - Capture the original UOM where necessary.
  - Identify and adopt a standard set of conversions. A consistent method of conversions is necessary to change values between different systems of units.
  - Provide standard set of conversion values and algorithms for consistent conversion that does not introduce value creep as conversions are repeatedly applied.
  - Provide a standard set of mnemonics.

Guidelines

- Provide UOM for numeric values at the Meta table level (PPDM_COLUMN. DEFAULT_UOM_SYMBOL).
- Where the UOM for a column depends on the data (such as a production volume) provide UOM on a table column basis.
- Provide the Original UOM (OUOM) in the host table where business needs dictate. Regulatory or legal requirements may dictate that the original value and original UOM be available for certain numeric values.
- Both the UOM and OUOM must be included for measured values in a vertical table.
- The name of the OUOM column (and the UOM column, where used) should contain the same prefix as the column containing the value. (i.e. MEASURED_DEPTH and MEASURED_DEPTH_OUOM)

Method

- There is a reference guide in the PPDM WIKI describing how Units of Measure are handled in PPDM.
- All table names are stored in PPDM_TABLE.
- Column names are stored in PPDM_COLUMN.
- Recursive foreign key relationships within PPDM_COLUMN associate measurement columns with the appropriate UOM or OUOM columns.
- PPDM_COLUMN_GROUP: This serves as the mechanism to group together columns logically. This can be done as a "domain" or as a totally different type of grouping. We may need a lookup table.
- PPDM_MEASUREMENT_SYSTEM: Defines valid systems of measure. For example, Metric, US Imperial, British Imperial and so on.
- PPDM_UNIT_OF_MEASURE: Defines the valid units of measure within a measurement system.
- PPDM_DATA_STORE: This is intended to allow the grouping of tables into areas that may use different units of measure as defaults. It has been suggested that it may work well for project databases. This is still open for discussion.
- PPDM_UOM_ALLIAS: Intended to be an aid during data loads. Other names for units of measure.
- PPDM_QUANTITY: Defines the quantity - length, volume, area and so on.
- PPDM_UNIT_CONVERSION: Defines valid conversions between units of measure.

Currency Units of Measure

- Currency values in PPDM should all be stored as a single currency (i.e. all US$ or CDN$) to avoid performance degradation through on-line conversions and calculations or full table scans. However, there may be a need to restore the original value of the currency in the units originally received as. To accommodate this, currency values should be modeled as:
    - XXX (cost value) number (12,2)
    - CURRENCY_OUOM varchar2 (12)
    - CURRENCY_CONVERSION number (10,5)
- Currency conversion rates are not static, and consequently cannot be managed as other conversion factors (in the UOM Module). Every row of data must be accompanied by the conversion factor relevant at the time of the transaction. The currency conversion factor is the value which, when multiplied by the cost value, results in the original value of the cost in the Original units of currency.
- Usually, it should only be necessary to store one conversion value per table in PPDM. However, if necessary, additional columns for currency conversion may be specified and named to indicate the values to which they refer.

ARCHITECTURAL PRINCIPLES COORDINATES

- This section defines a consistent method for positioning all objects from the database in multi-dimensional space. The horizontal components of those coordinates typically are treated together as a pair of values, while the vertical coordinate is treated independently.
- Horizontal coordinates include northing, easting or x, y values (referred to as Grid coordinates) and latitude, longitude pairs (referred to as Geographic coordinates). Vertical coordinates usually refer to height, elevation or depth.

Objectives

- Provide consistent positional information for objects.
- Allow multiple versions of positional information.
- Support a "selected" row of data that contains the preferred version.
- Provide sufficient geodetic information including geodetic datum and coordinate projection to identify the positional data.
- Provide sufficient acquisition information to identify the acquisition method and estimated accuracy of the positional information.

Spatial tables

- Spatial coordinate information is stored in the tables named SP_%, except for well and seismic locations. For these two location types, in PPDM 3.8 you may use either the module specific location tables or the Spatial tables. Note that this use will be integrated into a single method in PPDM 3.9
- A link between spatial information in the SP_% tables and the business table must be captured in SP_COMPONENT. If you fail to populate this relationship, the spatial information will not contain proper contextual references to its parent business object.

Coordinate Values

- Horizontal coordinate values should be stored as Geographic values (latitude, longitude). Wherever possible, only the Geographic values should be kept; by keeping only one set of horizontal values, data integrity issues are minimized (you don't have to keep two sets of coordinates in synch with each other).
- Only store Grid values (northing, easting) where these represent the original units of survey, or where a compelling business case exists to store them (seismic processing often uses local coordinates, for example).

  Metes & bounds values are required by some agencies. These can be stored in PPDM; remember that these coordinates may be original values measured in the field (as a consequence they may or may not match the geographic coordinates)
- Latitude, Longitude values must be referenced to a coordinate reference system (CRS). The coordinate system references datum, spheroid and ellipsoid information, and transformations between CRS.
- Northing, Easting values must be qualified by including an appropriate map projection code.
- Coordinate values may also include acquisition activities that will demonstrate the method and estimated accuracy of coordinate capture. This information is captured in COORDINATE_ACQUISITION_METHOD

Guidelines

General

- Business objects (such as wells or seismic sets) in PPDM may be described spatially using point, linear or polygonal geometries).
- At design time, the workgroup should think about which kind(s) of geometry is appropriate. Sometimes, more than one kind of geometry will be appropriate. For example, seismic sets may be described as points, lines or polygons, depending on the needs of the users.

Versions

- Coordinates should be stored in the SP subject area, selecting the appropriate geometry type(s).
- Some coordinates (wells and seismic) are stored in special tables inside their own subject area. Refer to the subject reference guides for details.
- Keep alternate versions of coordinate data in a SP_%_VERSION tables. These data versions reflect different vintage, units, sources, methods of calculation, and/or methods of acquisition.

Groups of positions

- Each set of coordinate information should be stored in the same CRS. For example, the points in a seismic set should be referenced to the same CRS.

Polygons

- All objects that have polygonal spatial representations should use the SPATIAL module to describe those locations. Provide a FK from the business object into the table SP_COMPONENT. .

Elevations and Depths

- Regional Vertical Datums
  - All elevations are defined by a reference to a vertical datum, whether directly or by convention. Keep vertical datum information at the Meta table level.
- Local Vertical Datums
  - Currently elevation datums are not kept but understood to be a reference to a local vertical datum (i.e. NAVD29 relative to mean sea level), with few obvious problems. Where vertical datums are mixed, vertical datum information needs to be kept at the row level. Note that international requirements have increasingly driven this information to the table level.

Depth Reference

- Maintain reference measurement point for depths. All depths are defined relative to some reference point. (i.e. measured depths in a wellbore are defined relative to a surface reference point such as the Kelly bushing) Keep depth reference point at the row level in the WELL table. For seismic information, keep with the seismic tables.

ARCHITECTURAL PRINCIPLES EXTENSIBILITY AND SUBSETTING

- Extensibility and subsetting refers to the ability of an individual data model user to customize the model to meet their individual member business needs. In some cases, extensions are needed to manage proprietary or application specific information.

Objective

- To assist in the implementation of extensions or subsets so that changes to the model are easily identified to external users of the model (i.e. application vendors).

Guidelines

- Additional tables and columns may be added but the Primary Keys and Foreign Keys of PPDM tables must remain intact.
- Tables and columns may be subsetted but the Primary Keys and Foreign Keys of remaining PPDM tables must remain intact. This means that you may not drop a parent table if you are including the child table in your implementation.

Naming Extensions

Tables

- A total of 6 characters are available for naming private table name extensions. These characters should be added as a prefix to the usual table naming structure.

Example:

- ABC OIL COMPANY wants to extend the model by adding a table containing special core analysis information. They would name the table AB_CORE_ANALYSIS where AB_ identifies this as a private extension.

Columns

- A total of 3 characters are available for naming private column name extensions. These characters should be added as a prefix to the usual column naming structure.

Example:

- ABC OIL COMPANY wants to extend the model by adding a column containing proprietary Seismic line information. They would name the column AB_PROPRIETARY_FIELD where AB_ identifies this a private extension.

ARCHITECTURAL PRINCIPLES META TABLES AND META DATA

- There is a growing requirement to capture general information about the structure of the data model and the implementation of global issues such as UOM and coordinates. The proposed guidelines and the implementation of these guidelines provide a start that is expected to evolve in future releases of PPDM.

Objectives

- To define a set of Meta tables to hold information about the structure of the PPDM data model that includes major components such as tables, columns, constraints, domains, and units of measure.

Usage

Data Model Documentation

- Capture full names of table or columns (entity or attributes). Table and column names may be abbreviated to meet naming conventions or DBMS requirements. This provides a place for the complete name. Provide domains and identify to which domain a column belongs Provide common UOM, data type, data widths, value constraints.

Identify columns that are grouped

- Groups of columns, such as those sharing common data domain information (depths), those used by an application or those affected by a business process can be defined to help manage the model and its usage.

Document user extensions

- Provide a central repository for local extensions and/or subsets. This facilitates local managing of migration issues, as well as installation of PPDM-compliant systems.

Data management functions

- The data management work group has provided a substantial set of extensions to the PPDM meta model that support improved data management practices. Refer to the work group documentation for details.

PPDM DDL DELIVERY

Structure

PPDM data model DDL is generated as a series of files that separate the various components of the DDL

PPDM 3.8 (mandatory elements are bold)

- xxx.ccm – column comments
- **xxx.ck – check constraints**
- **xxx.fk – foreign keys**
- xxx.guid – create index on PPDM_GUID
- xxx.guidnn – make PPDM_GUID NOT NULL
- xxx.ouom – original units of measure constraints
- **xxx.pk – primary keys**
- **xxx.rqual – foreign keys for PPDM_ROW_QUALITY columns**
- **xxx.rrrc – foreign keys for SOURCE columns**
- xxx.sql – control comments – run this script
- xxx.syn – assign table synonyms
- **xxx.tab – create tables**
- xxx.tcm – create table comments
- xxx.uk – create unique keys
- xxx.uom – create unit of measure constraints

PPDM 3.9 (mandatory elements are bold)

- xxx.ccm – column comments
- **xxx.ck – check constraints**
- **xxx.fk – foreign keys**
- xxx.guid – create index on PPDM_GUID
- xxx.guidnn – make PPDM_GUID NOT NULL
- **xxx.ouom – original units of measure constraints**
- **xxx.pk – primary keys**
- **xxx.rqual – foreign keys for PPDM_ROW_QUALITY columns**
- **xxx.rrrc – foreign keys for SOURCE columns**
- xxx.sql – control comments – run this script
- **xxx.syn – assign table synonyms**
- **xxx.tab – create tables**
- xxx.tcm – create table comments
- xxx.uk – create unique keys
- **xxx.uom – create unit of measure constraints**

Note that some RDBMS limitations may make 100% compliance difficult for complete implementations of PPDM 3.9. In particular, some databases have limits:

- Number of foreign keys that can be in one table
- Number of times a table can be referenced by foreign keys (this is a limitation in SQL*Server)

ADDITIONAL ARCHITECTURAL GUIDELINES AND CONVENTIONS

ALIAS TABLES

What are ALIAS tables?

- In our industry, important business objects, such as wells, facilities, land rights, contracts, licenses and business associates are frequently assigned many names over their life. Since much of the rest of the data we work with may use these old or previous names, a name variant, a misspelling or a system assigned code when referencing these objects it's important to keep track of all the names, codes and identifiers that may be assigned.
- Many of the modules (subject areas) in PPDM 3.8 contain ALIAS tables. Although the structure of the ALIAS tables is not exactly identical in each case (allowing for some speciallized needs and to support legacy functions), they have been largely harmonized.

Why use the ALIAS tables?

- Often, a PPDM implementation can be used to integrate multiple systems. For example, information about wells may be loaded into many financial, production accounting, geotechnical or GIS applications. Each application may assign its own identifier to the well. In addition, identifiers for wells may change (in some jurisdictions, this is common place!).
- If every name,code and identifier associated with the well can be stored in one location, it is much easier to connect the master well with each of the replicated versions. Once this is done, data synchronization and updates are possible.

Normalize ALIAS values if you can

- Although names are often denormalized into the parent table of each module (LAND_RIGHT, SEIS_SET etc) in ways that may assist your query performance, it is generally best to leave these columns empty unless needed to improve your queries.

Store Names and Codes

- Each row in %_ALIAS tables can store multiple versions of an alias - this is very useful when you are mapping a Master Data Store into PPDM; in this case, the data source may use more than one version of the alias. Each alias may have a
  - CODE - codified version of the alias, usually a number or number-letter reference.
  - FULL NAME - the complete name of the object, as given by the source
  - SHORT NAME - a shortened version of the name for the object, as given by the source. Often, the short name is used in reporting or on displays such as maps.

ALIAS TYPE and ALIAS REASON

- Each alias should be referenced by the type of alias that has been given (REGULATORY, SW APPLICATION, INTERNAL are common). In addition the REASON why the alias has been captured is important.
- Unlike the ALIAS_TYPE column, the ALIAS_REASON states why the alias was created or captured (REPORTING, INTEGRATION, NAME CHANGE are common reasons).

Contextual Information

- The %_ALIAS tables allows you to capture a great deal of contextual information about each alias.
- For example, if the alias is a key or identifier attached to the license in a software application, you can capture the name of the application that uses the alias in the column SW_APPLICATION_ID.
- Use the LANGUAGE_ID to indicate the language this alias is in; this function is very useful for international uses.
- The FK column SOURCE DOCUMENT allows you to capture the place in literature where an alias version is used. This is very useful for some kinds of alias names, such as stratigraphic (formation) names or fossil names.
- A foreign key to BUSINESS_ASSOCIATE, usually called OWNER_BA_ID, allows you to identify the business associate who owns an alias (such as a regulatory agency).

Indicator Columns

- Three indicators in this table will help you sort out which alias version to use for various functions:
    - ACTIVE_IND is set to Y when the alias is still in use.
    - ORIGINAL_IND is set to Y when the alias is the original or first that was assigned to an object.
    - PREFERRED_IND is set to Y when the alias is the one that should be used by default when creating output, or for certain kinds of searches.