An Oracle White Paper
June 2009

An Introduction to Oracle SQL Developer Data Modeler

ORACLE

Introduction

Oracle SQL Developer Data Modeler is a new, graphical data modeling tool that facilitates and enhances communication between data architects, database administrators, application developers and users, and simplifies the data modeling development process itself. Using SQL Developer Data Modeler users can create, browse and edit, logical, relational, physical, multi-dimensional, and data type models. The generation of DDL scripts improves productivity and promotes the use of standards.

The introduction of this tool illustrates Oracle's commitment to improve developer productivity and to facilitate the improved quality and standard of the work done by the database developer community and their users. SQL Developer Data Modeler is designed for all database data modelers, from business architects to DBAs and from database to application developers. The role of SQL Developer Data Modeler is to simplify data modeling development tasks and serves as a powerful communication tool between developers and business users.

This document highlights the main features and benefits of Oracle SQL Developer Data Modeler.

Oracle SQL Developer Data Modeler

Oracle SQL Developer Data Modeler is a complete model-to-implementation solution for data related modeling, whether for

- New operational or business intelligence related databases
- Capturing existing database implementations to provide a graphical representation and related metadata for documentation
- Adding and implementing new data requirements

The Oracle SQL Developer Data Modeler audience ranges from data architects to database administrators and developers, all users who need a clear, consistent and understandable picture of the data and the underlying data properties.

Architecture

Enterprise architecture is a term used to describe the art of documenting all the elements that make up an enterprise. Enterprise Architecture describes the structure and behavior of an enterprise and its information systems.

The best known Enterprise Architecture frameworks are the Zachman Framework, the framework from the Department of Defense (DoDAF) or the Federal Enterprise Architecture Framework (FEAF). SQL Developer Data Modeler follows the Zachman Framework for defining Data structures.

SQL Developer Data Modeler Layers

SQL Developer Data Modeler model comprises three tightly synchronized layers: a logical model, relational models, and physical models. There is a one-to-many implementation when moving from the logical to the relational to the physical layer, which means that you can create multiple instances at the next, lower level in the hierarchy. Properties set at the upper layers are reflected and preserved at the lower levels. This is useful when an application has more than one physical deployment or customer-facing implementation to support.

Synchronization

Tight synchronization between models is important, by carrying properties from one layer to the next, it preserves the organizational expertise added at each level. For instance, design decisions made in the relational model regarding important issues such as table splits and joins, key structures, and indexing are preserved even as the logical model above changes to reflect new business information needs. The physical layer beneath may also have platform specific options

set. These are preserved as the layers above adjust. This saves an organization's DBAs from extensive amounts of rework and repetitive re-coding of the database properties and parameters.

When working with DB2 mainframe, SQL Developer Data Modeler makes space and free-space parameter calculations based on the record insert-delete profile of each platform's instantiation, if this information is available. The built-in design expert rules engine provides design integrity assurances for the organization and the DBA, whose depth of knowledge in tuning a database on a specific platform may differ across the platforms she or he is required to support.

Importing

SQL Developer Data Modeler can import models from various formats, including the metadata from a variety of sources. This allows an organization to take advantage of existing operational databases across current platforms and importing models that may have been built using other tools.

SQL Developer Data Modeler imports

- Details from standard DDL scripts
- Schema and physical details directly from the database data dictionary
- Entity and Schema objects from the Oracle Designer repository
- Multi-dimensional models using Cube Views or XMLA files
- CA ERwin Data Modeler models complete with diagram layouts and the name standardization and abbreviations files for naming conventions

SQL Developer Data Modeler Distribution and Installation

Oracle SQL Developer Data Modeler is a standalone, independent product, available for download from the Oracle Technology Network (OTN). A SQL Developer Data Modeler Viewer is also available for download from OTN. The viewer provides users with the ability to open and view models previously created in the Data Modeler.

SQL Developer Data Modeler runs on Windows, Linux and Mac OS X. To install SQL Developer Data Modeler simply unzip the downloaded file. With SQL Developer Data Modeler users can connect to any supported Oracle Databases. There is also support for IBM DB2 LUW V7 and V8, IBM DB2/390, Microsoft SQL Server 2000 and 2005 or a standard ODBC/JDBC driver for selective import of database objects and data browsing and migration.

Models are stored as XML files and are easily shared or placed under source code control.

In the next section we describe the key concepts and features of SQL Developer Data Modeler.

Integrated Models

SQL Developer Data Modeler provides facilities to build sets of related and integrated models. At the core of SQL Developer Data Modeler is the logical model. The logical model provides a true implementation-independent view of enterprise information and acts as the mediator that maps definitions in the dimensional models to different physical implementations. A logical model, or a part of it (subject area or sub view), can be transformed to one or more relational models. Each relational model can have an unlimited number of physical implementations in the form of physical models (referred to as RDBMS Sites in SQL Developer Data Modeler), with each physical model based on a supported database.

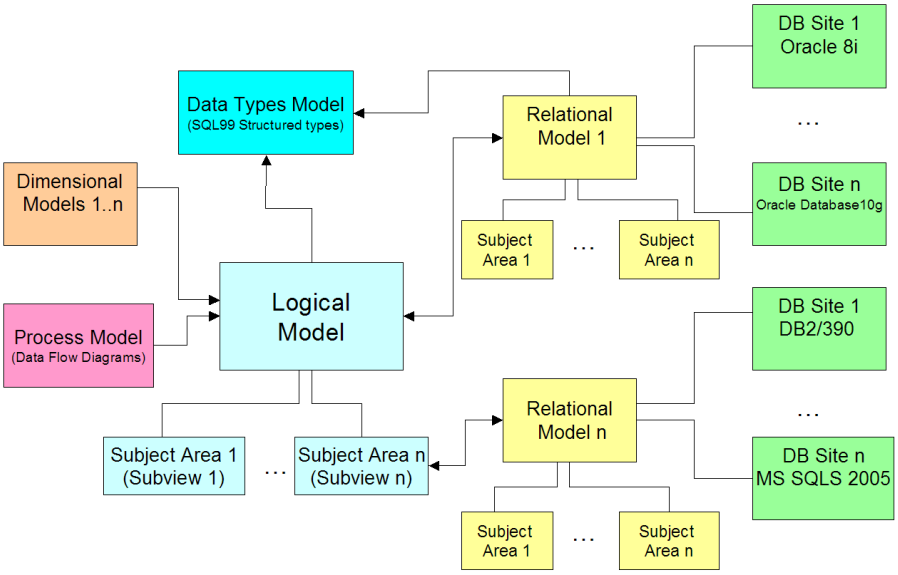

Figure 1: SQL Developer Data Modeler - Model Structure

Logical Models

The logical model in SQL Developer Data Modeler includes standard logical modeling facilities, such as drawing entities and relationships, plus the following key features:

- Box-in-box presentation for the super-type and sub-type hierarchy of entities
- Support for exclusive relationships (arcs)
- Barker or Bachman notation

The image below illustrates a logical model displaying an arc, sub-types and super-types and using the Barker notation.

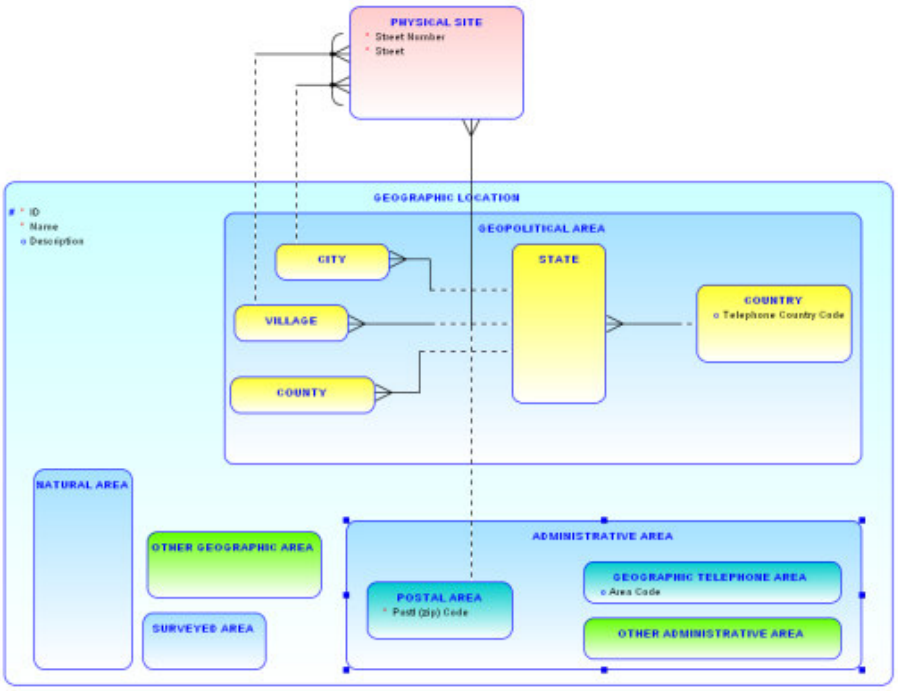

Figure 2: Arcs and box-in-box representation for super-types and sub-types

Relational Models

The SQL Developer Data Modeler relational model is an intermediate model between the logical model and the physical models. It supports relational design decisions independent of the constraints of the target physical platform(s). All many-to-many relationships and all super-type/sub-types entity hierarchies are resolved during forward engineering (transformation) of the logical model, or part of it, to a relational model.

Name standardization is also applied during forward engineering.

Physical Models

SQL Developer Data Modeler supports most Oracle physical objects. It exposes many elements of an object's structure and definition (for example, partitions and subpartitions) in the object browser. Users can access these through the property dialogs invoked directly from the browser without having to open intermediate dialogs. SQL Developer Data Modeler also provides easy access to related definitions; for example, partition properties for local indexes, which are directly accessed from the dialog of the related table partition.

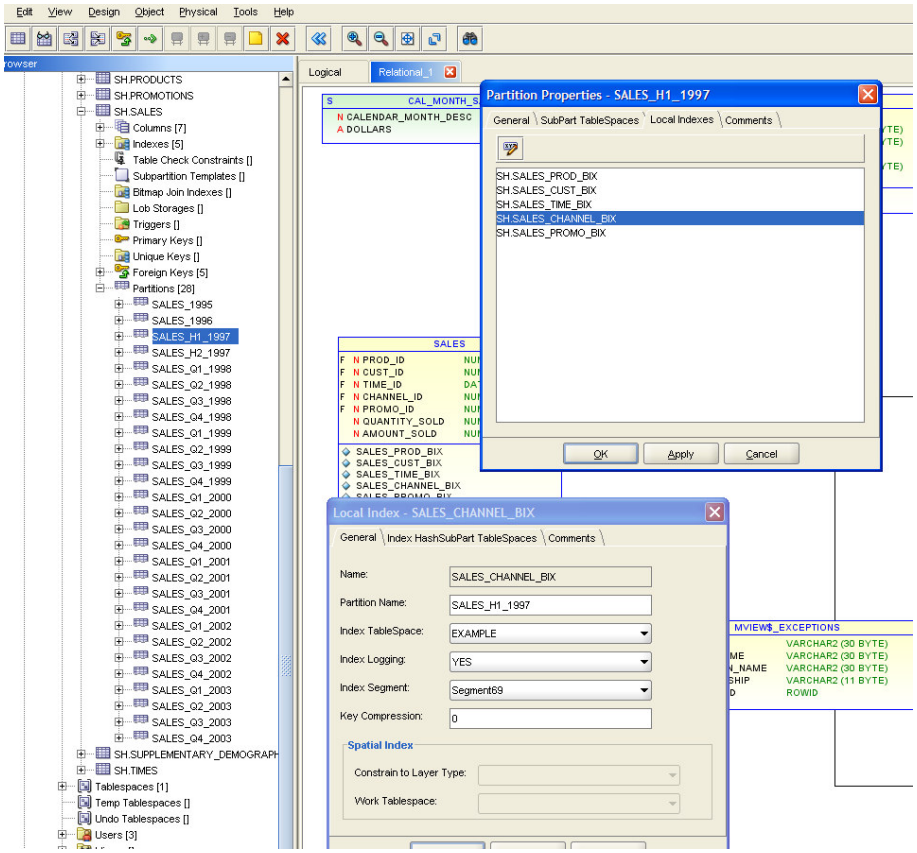

Figure 3: Partition Property Window

The Data Modeler provides support when users modify definitions, as in the following examples:

- All table partitions are kept when a partitioning type is changed from "Range" to "Composite", with the result that partitions do not need to be defined again. As "Composite" partitioning is an extension of "Range" partitioning it is logical that the current partition definitions should be reused.
- Matching is provided for the number of table partitions and the number of defined partitions for local indexes. As is often the case, only a few partitions (from among 100) have settings that differ from the defaults. SQL Developer Data Modeler requires only the two different partitions to be defined. It will generate valid DDL statements and will automatically add the 98 additional index partitions with the partition name only.

Two important Oracle features are also covered by SQL Developer Data Modeler's physical model for Oracle:

- Support for SQL level dimensions, levels, attributes and hierarchies. Without these definitions, Query Rewrite cannot be used with materialized views, thus limiting the benefits to be gained from creating materialized views with aggregations.

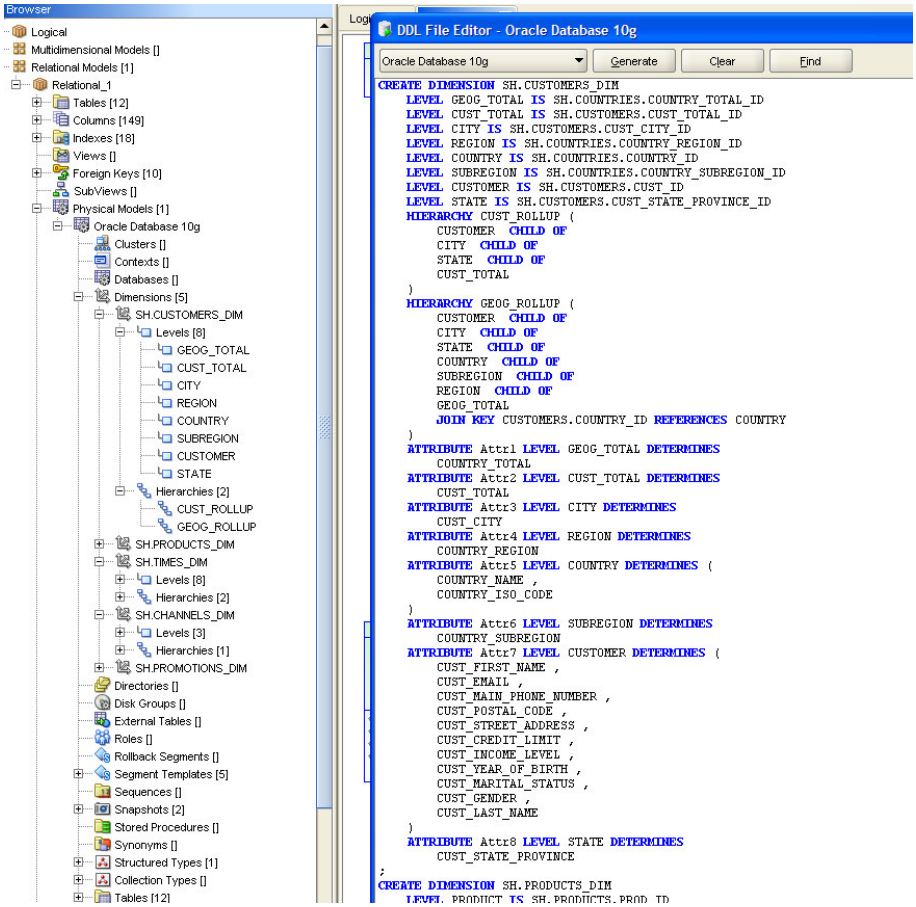

Figure 4: Dimensions, Levels and Hierarchies in an Oracle Physical Model

- Support for bitmap join indexes, which in some cases produce better results than materialized views and can thus be an alternative to using materialized views.

Multi-dimensional Models

SQL Developer Data Modeler has predefined classifications that allow each entity and table to be classified as fact, dimension, summary, temporary, or logging for multi-dimensional modeling.

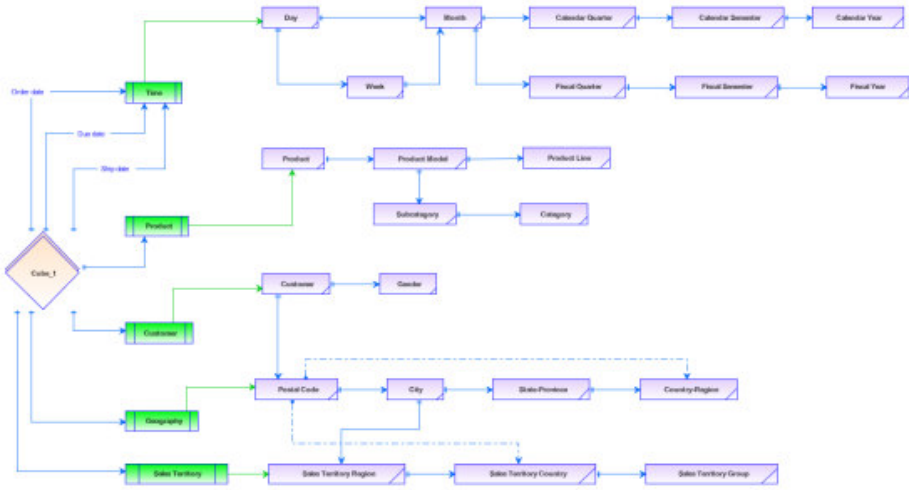

Figure 5: Compact Dimensional Diagram

This is merely an extendable classification schema. Users can also associate different colors with each predefined or user defined classification type, so that entities, and tables are colored depending on their classification type.

SQL Developer Data Modeler provides a dedicated, full-featured dimensional model on a separate abstract layer and shown on a separate diagram. SQL Developer Data Modeler follows Model-Driven Architecture (MDA) guidelines and provides facilities for building and expressing a pure logical multi-dimensional model using the Common Warehouse Model (CWM) categories like cube, slice, measure, attribute, level, dimension, and hierarchy in order to provide proper multi-dimensional presentations of business data. Support for merging dimensions, role-playing and fact dimensions, different types of measures, calculated attributes, level-based and value-based hierarchies, and level-based ragged hierarchies allow all dependencies, aggregation rules, and roll-up (navigational) paths to be defined correctly and in an expressive way. This is provided in both a compact and as a detailed dimensional model diagram – see Fig. 5 and Fig. 6.

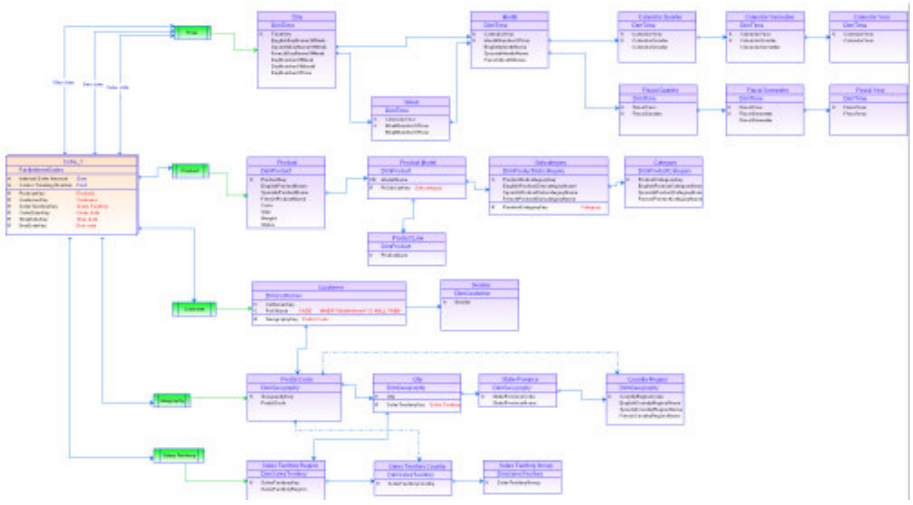

Figure 6: Detailed Dimensional Diagram

The dimensional model is built on top of the logical model. It maps levels, attributes, and measures to the entities and attributes defined in the logical model. These definitions can be mapped to an unlimited number of relational models and their implementations in different RDBMS sites, thus providing great flexibility in deployment of a dimensional model.

Apart from all the dimensional facilities (including the Oracle OLAP measure editor), there is a query wizard (a built-in Slice object – see Fig. 7), which allows users to generate SQL SELECT statements from the dimensional definitions.

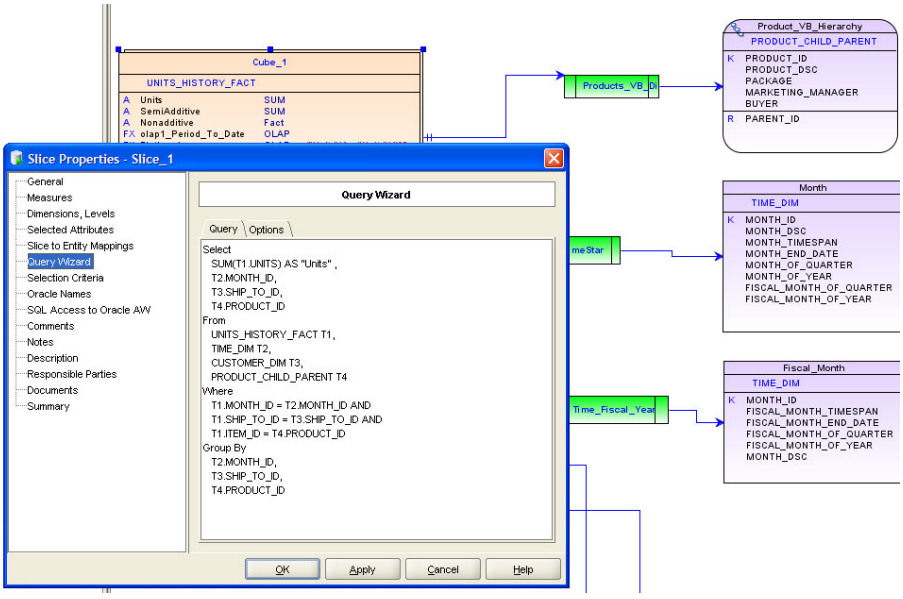

Figure 7: Query Wizard built into the Slice Object

Oracle SQL Engine

There is bidirectional transfer of metadata between the dimensional model and SQL Developer Data Modeler database sites. Dimension, level, and hierarchy definitions from the dimensional model can be engineered as SQL, thus providing metadata for query rewrite to the database engine. Going in the opposite direction (almost the complete dimensional model can be built using database definitions), SQL dimensions, levels, and hierarchies can be engineered to their dimensional counterparts, and cube and measure definitions can be created if certain conditions are met (such as the presence of foreign key definitions between fact and dimensional tables). See Figure 6 showing the Oracle Sales History (SH) sample model.

Data Type and Structured Type Models

SQL99 introduced the concept of a structured type as a named user-defined composite type with the ability to add a super-type and sub-types inheritance hierarchy.

The Data Types model allows users to create and visualize structured types and the inheritance hierarchies of structured types, defining distinct and collection (array) types. These definitions are mapped to Oracle object types, VARRAYs, and nested tables in an Oracle physical model.

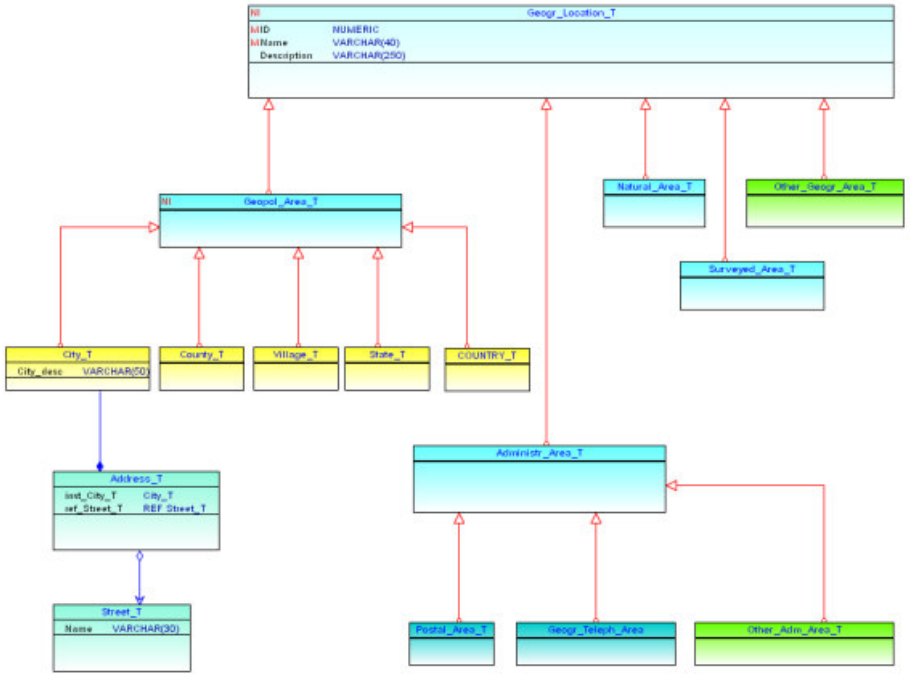

Figure 8: Data Types Model

The logical model and any relational models can use definitions from the data types model to specify the data type for attributes and columns or to specify that a table (entity) is of a certain structured type.

Type Substitution

SQL Developer Data Modeler has "type substitution" at row (table and view) level as a graphical operation and presentation. The type substitution constraint defines instances of which subtypes can be accommodated by a table or view based on a structured (object) type, as shown in the image below using with green lines.

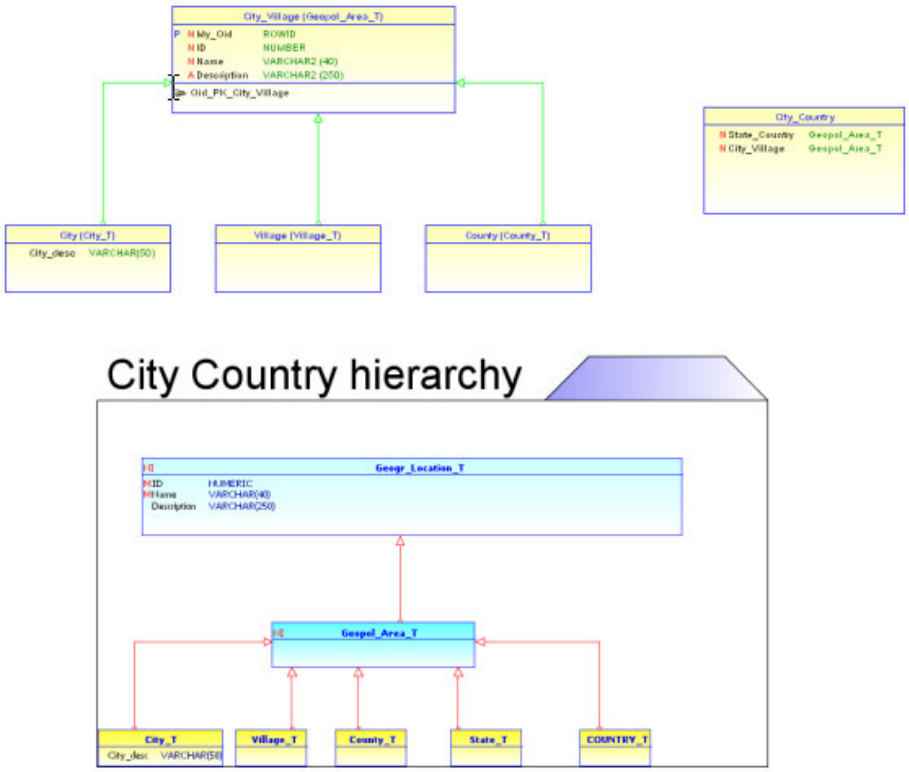

Figure 9: Relational Model with Nested subview from DataTypes Model

Type substitution at the column level and a "scope" table for reference types are among the SQL99 features that are supported in the relational model. Database triggers are generated to force type substitution definitions when these constraints cannot be expressed in a declarative way.

Spatial Models

SQL Developer Data Modeler supports spatial models by allowing users to define spatial tables. Users can define one or more spatial columns for columns with an SDO_GEOMETRY data type or a function with the result of type SDO_GEOMETRY, and can also define the related spatial indexes. These definitions can be further extended in the physical model.

All Oracle Spatial-related definitions and required registrations in the USER_SDO_GEOM_METADATA view are included in the generated schema, and are imported during database import.

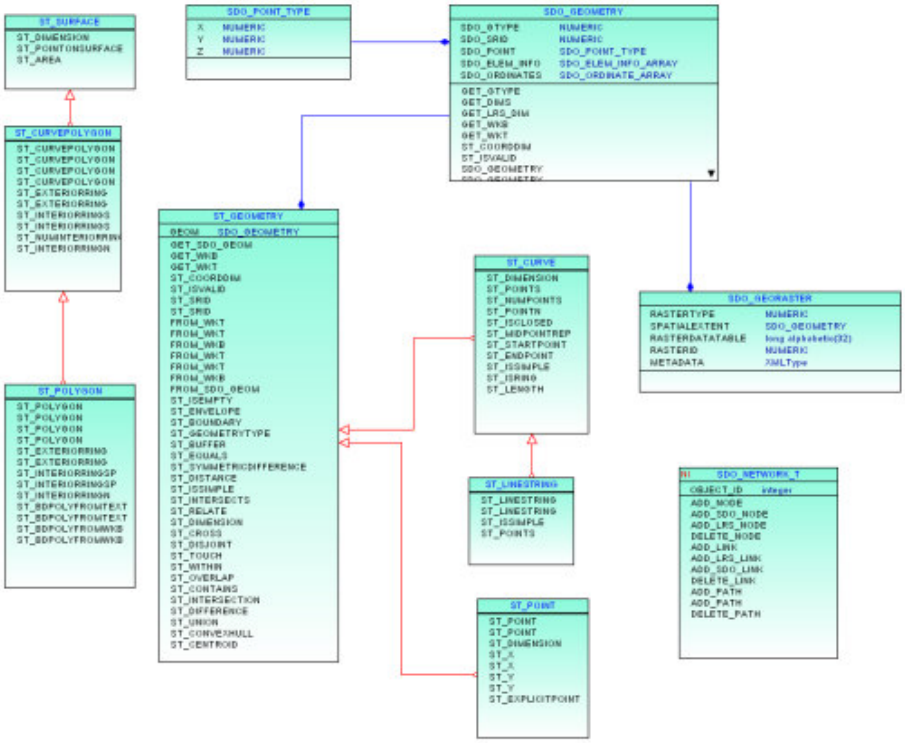

Figure 10: Data Types in MDSYS Schema

SQL Developer Data Modeler recognizes all data types declared in the MDSYS schema as predefined; and are therefore never included in generated DDL scripts. This allows data types from the MDSYS schema to be imported into a data type's model when they are required for modeling purposes.

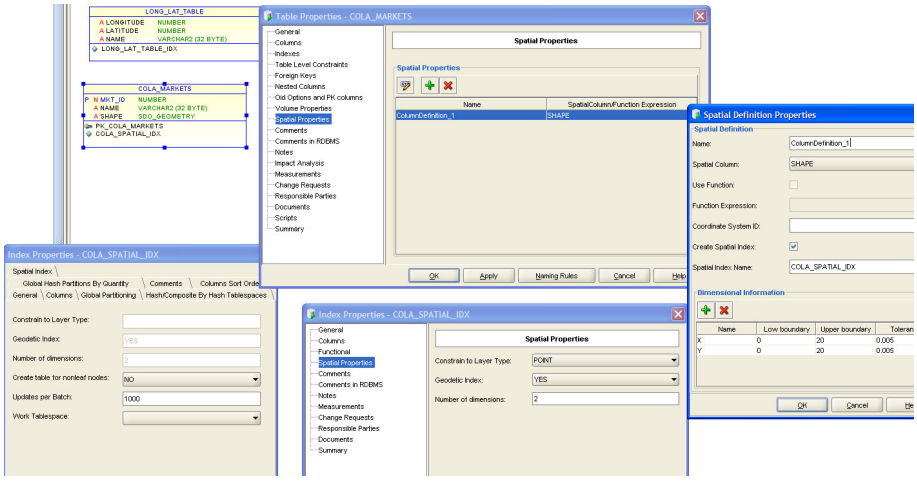

Figure 11: Spatial Definitions in a Relational and Physical Model

Creating Models

Users import existing logical models, multi-dimensional, or relational models or create them from scratch. Relational models are also created by importing script files (DDL) or by importing directly from the data dictionary. SQL Developer Data Modeler can import directly from the Oracle Designer repository or CA ERwin Data Modeler logical models. Multi-dimensional models can be imported using Cube Views or XMLA files, or created from Oracle SQL dimensions imported from database or DDL script. Data types models can also be created using import from database or DDL script, for Oracle and DB2/UDB.

Extensive and wizard-led engineering capabilities allow you to re-engineer a relational model to a logical model or to engineer a logical model to one or more relational models, where both models can be kept synchronized.

Creating New Models

On opening SQL Developer Data Modeler empty logical and relational model pages are automatically opened. To start creating a new model, users can use the associated toolbar, clicking the appropriate button and then the model to start. Each action invokes the associated dialog, allowing the user to create or update the elements on the diagram.

Importing from the Data Dictionary

Users can select the objects to be included in the import or reverse engineering process. SQL Developer Data Modeler supports

- Selection lists for all supported objects

- The ability to filter out secondary tables
- Smart import for Oracle object types, VARRAYs, and nested table. They are imported only if they are included in the selection list or are used in the definition of other selected objects (table, view, column, method parameter or result).
- Import from multiple schemas, creating a single diagram for all tables and views and a separate subview for each schema imported.

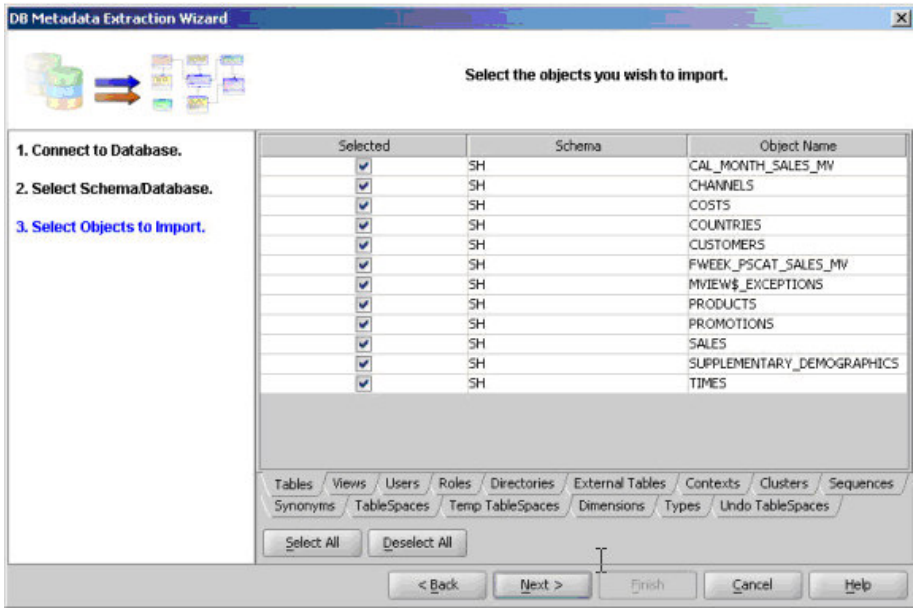

Figure 12: Data Dictionary Import

Importing from Oracle Designer

SQL Developer Data Modeler has a wizard that extracts selected objects directly from the Oracle Designer repository. It allows users to select the Work Area and Application Systems and then to select individual objects of each selected object type.

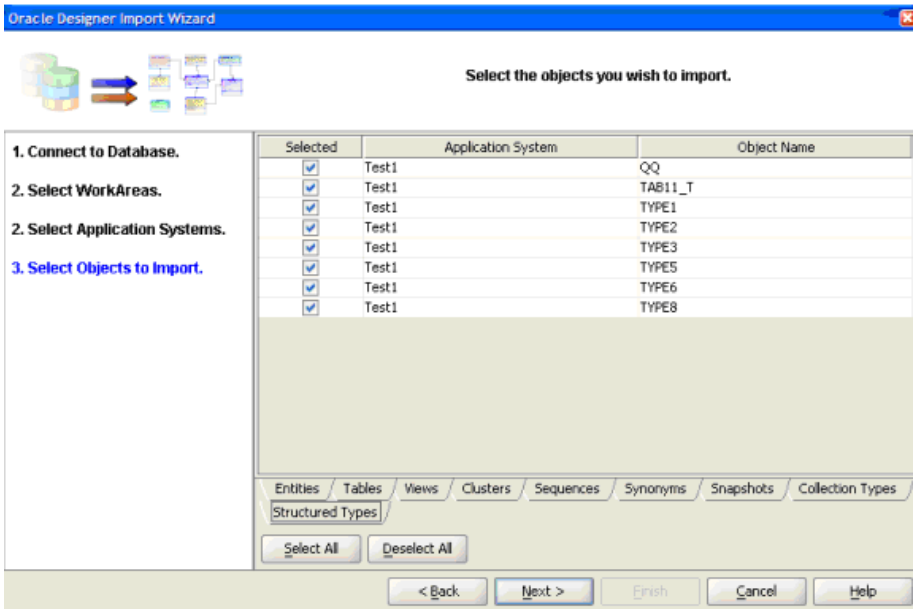

Figure 13: Import from Oracle Designer repository

In addition to supporting standard objects, SQL Developer Data Modeler imports definitions for arcs (exclusive relationships), Oracle objects, domains and collection types.

All diagrams belonging to selected application systems are also imported as subviews in their respective SQL Developer Data Modeler models (logical, relational, and data types).

Generating Scripts

SQL Developer Data Modeler generates DDL scripts for Oracle, DB2 and SQL Server, providing a number of export options. Users can generate:

- Database specific DDL scripts from the physical models
- Cube Views Metadata or XMLA files for multi-dimensional models
- XML files for Oracle OLAP or directly to create Oracle Analytical Workspace

Generating DDL and Self Managing Scripts

In addition to regular DDL scripts, SQL Developer Data Modeler generates advanced scripts that offer self-managing functionality:

- By defining "start and stop" steps, the Execution window can be defined when the scripts are executed. Each DDL statement represents an execution step and, in case of failure, the script can be restarted from the point (step) of failure.

- The set of Oracle errors to be masked can be defined (see Fig.14). Script execution is interrupted if any of these errors appear. Execution is stopped for all non-maskable errors that occur.

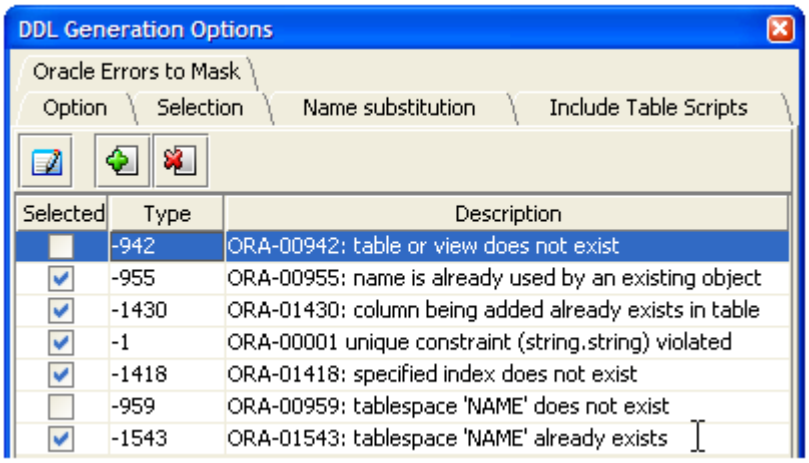

Figure 14: Defined Oracle errors ignored during script execution

- Logging script execution with three levels of logging.

Generated advanced Oracle DDL scripts are Oracle SQL*Plus scripts and can be executed directly in SQL*Plus or in Oracle SQL Developer. They are generated in two variants, depending on the way the input parameters are provided – either interactively (in SQL*Plus or Oracle SQL Developer) or using the command-line to pass input parameters to SQL*Plus. In the latter case, advanced scripts are easily integrated into any general scheduling or change management environment in order to propagate database changes.

SQL Developer Data Modeler has name substitution facilities to ease database migration tasks. For example, you can have the same set of database objects exist in different databases (test, development, production environments, etc.) but with different naming templates.

Updating the Database through DDL Scripts

SQL Developer Data Modeler enables safe and effective through its generated DDL scripts.

The typical sequence of actions generated for tables that need to be recreated includes the following:

1. Rename the existing table.
2. Create the new table with the new structure.
3. Copy the contents of the old table into the new one.
4. Delete the old (renamed) table.

If an unidentified data error occurs in step 3, then the old table will be deleted in step 4 and this will result in data being lost. The cause of the error could be varied; possibilities include an error occurring during the copy step when a column data type is changed or by tablespaces becoming overloaded. Even this simple test can show the risk: how to properly handle the change of a column's data type from DATE to VARCHAR2.

SQL Developer Data Modeler provides the following framework for safe database modifications:

- Advanced scripting with execution window, error masking, and logging
- Two backup options for tables that are to be recreated – "Rename" or "Unload to file system"
- Global and/or per-table settings for the "Unload" directory
- Each partition (for partitioned tables) is unloaded to a separate file
- A data type conversion expression can be defined for columns with a changed data type

Oracle Analytic Workspaces

Using the AW/XML API, a dimensional model can be exported in an XML file or directly in Oracle Analytic Workspaces (AW). The export itself can be done in two modes:

- The whole dimensional model, or just part of it, can be exported as a new Analytic Workspace. The slice object allows only part of cube metadata (including dimensionality) to be exported, thus providing additional flexibility in deployment to different production systems.

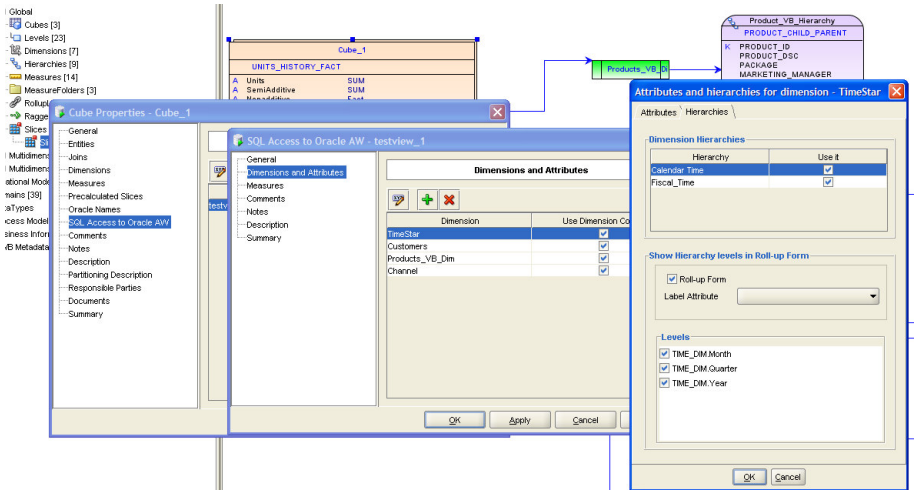

Figure 15: SQL Access to Oracle Analytic Workspace Wizard

Export of calculated measures – new (or all) calculated measures could be exported to an already existing AW. The current structure of measure folders (including subfolders from all levels of nesting) is also synchronized with AW.

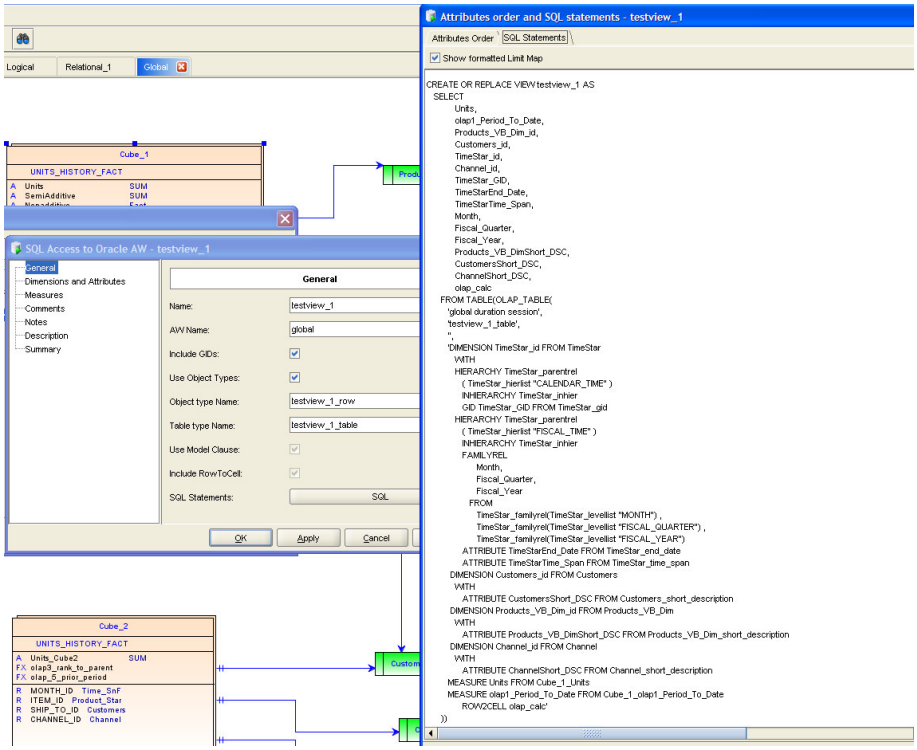

Figure 16: SQL Access to Oracle Analytic Workspaces - Generated SQL

There is a built-in wizard that acts as an enabler for SQL access to dimensional data in Oracle AW (using the OLAP_TABLE interface). Through this, all required object types and view definitions could be quickly and easily defined.

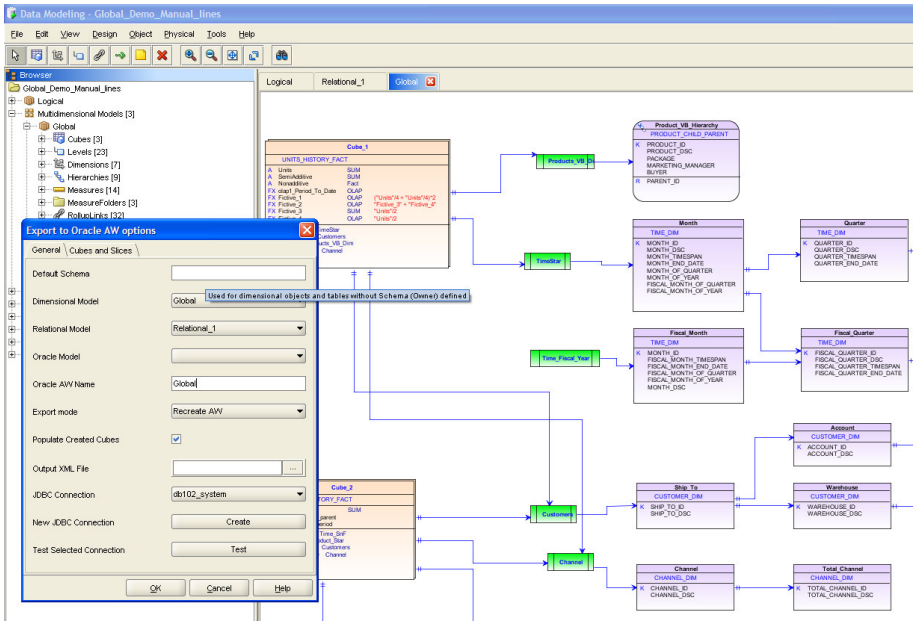

Figure 17: Export of multidimensional model to Oracle AW options window

Cube Views

SQL Developer Data Modeler supports bidirectional exchange of metadata with Cube Views. Dimensional models can be built using import of Cube Views metadata version 8.1 and 8.2, and a dimensional model can be exported in a Cube Views 8.2 XML file.

Many vendors of BI platforms (including Cognos, Microstrategy, Business Objects, and Oracle Hyperion) have declared support for import of Cube Views metadata. A SQL Developer Data Modeler dimensional model can be transferred to all platforms capable of importing Cube Views metadata.

MS SQL Server Analysis Services 2005

SQL Developer Data Modeler can export a dimensional model as an XMLA file and can import XMLA files with a SQL Server 2005 model.

General Features

SQL Developer Data Modeler supports a variety of additional features that support the processes described and models defined above. A few of these are discussed below.

Formatting, Sub views and Displays

Users can control colors, fonts and the dimensions of a single or collection of objects. A subview is a group of, often related, objects on a diagram. Any changes in the subview reflected in the main model. A relational model subview is automatically created per database schema when several schemas are imported at once. Subviews make it easier to maintain larger models.

Diagram Linking

In addition to standard diagram facilities, SQL Developer Data Modeler has support free diagram nesting (linking). "Free" means that the diagrams from different models can be linked together in order to have better presentation of business concepts, rules, and data. For example, a data flow diagram can be nested in a relational subview in order to represent data load and transformation. Fig. 9 above shows "City/Country hierarchy" sub view from the data types model, nested in a relational model.

Nested diagrams can be visualized as an icon or in a composite view (as in Fig9) and if a nested diagram has more than one display, one of them can be selected to represent the diagram.

Naming Standards

SQL Developer Data Modeler has basic support for naming standard glossaries, and also provides flexible and unrestricted naming rules, by supporting combinations of Prime word, Class word, Modifier and Qualifier.

In addition, SQL Developer Data Modeler allows the definition of naming rules for table elements (Fig.19). Name patterns for indexes and constraints (including Primary Key and Unique key) can be set using a combination of predefined variables, e.g {table}, {child}, {parent}, {column}, {seq_nr}, {model} and alphanumeric constants. Combining these, optionally with the SUBSTR function to restrict the total length of the word, users can define more flexible patterns.

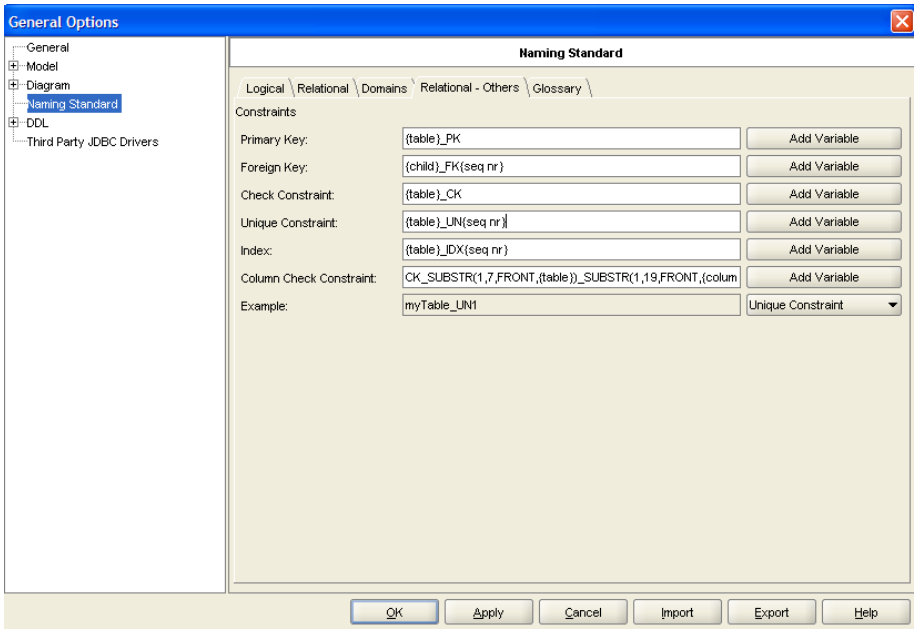

Figure 18: Naming rules for table elements

Impact Analysis

There is an Impact Analysis panel in the property windows for objects in the logical and the relational models. Impact Analysis shows the usage of an object by other objects in the same model. It also shows mapping information between logical model objects and related relational models objects and, through mapping, it shows the use of the object in a multi-dimensional and process model. Since logical models can be mapped to more than one relational model, users can track the dependencies between different relational models.

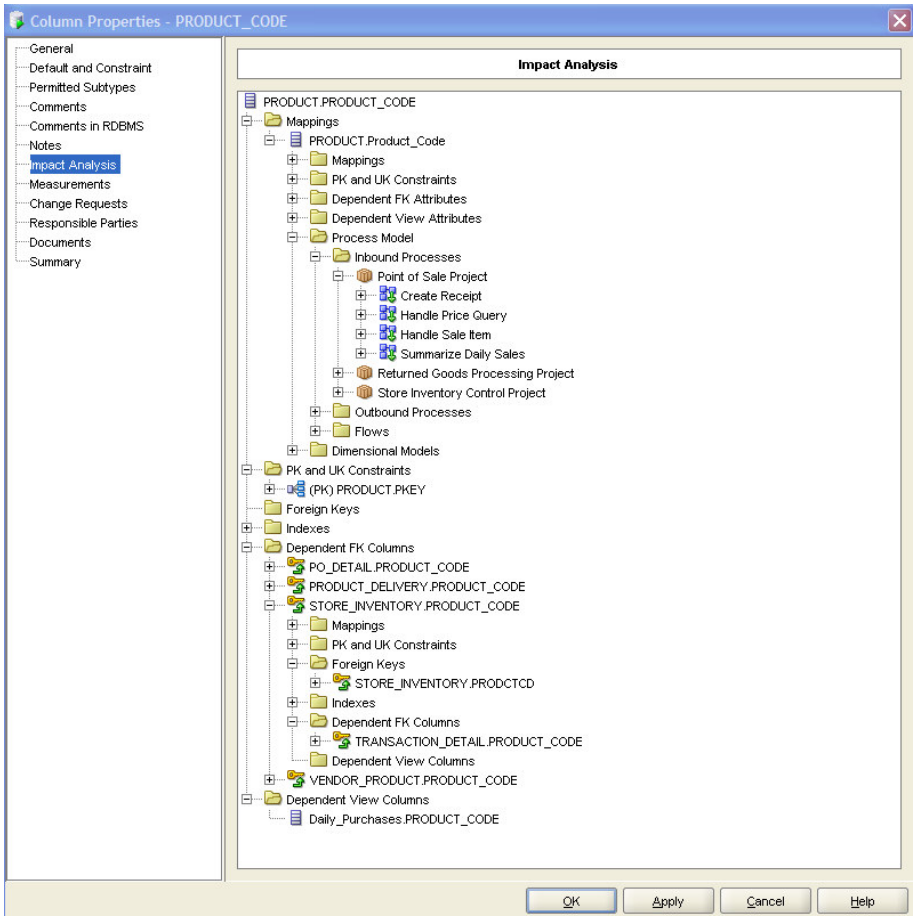

Figure 19: Impact Analysis Property Window

For domains, the Domains property window has a Where Used property window:

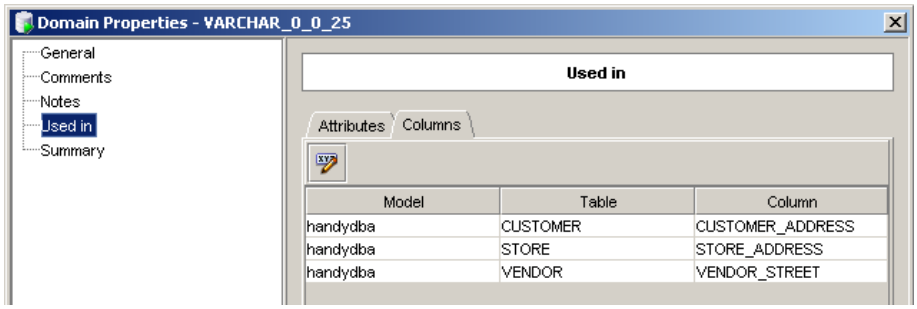

Figure 20: Where Used for Domains

Using the Reporting Repository

SQL Developer Data Modeler supports a reporting repository, allowing users to save and run SQL queries to gather details of the designs. A set of predefined reports are available as an extension to Oracle SQL Developer, where users can run the reports against their designs or write their own SQL query reports.

Summary

In this paper, we have illustrated many of the key features of Oracle SQL Developer Data Modeler. Oracle SQL Developer Data Modeler provides a toolset to help anyone involved in the data modeling and data design process. It supports logical or conceptual modeling (including multi-dimensional modeling for Business Intelligence), relational database modeling and the final detailed physical implementation.

Resources

For further information regarding Oracle SQL Developer Data Modeler, see http://www.oracle.com/technology/products/datamodeler/index.htm

There is a dedicated white paper on Naming Standards in SQL Developer Data Modeler.

http://www.oracle.com/technology/products/database/datamodeler/pdf/DataModelerNamingStandards.pdf

Oracle SQL Developer Data Modeler
June 2009
Author: Philip Stoyanov, Sue Harper
Contributing Authors: Chuck Murray

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Oracle is committed to developing practices and products that help protect the environment

0109