# Oracle Warehouse Builder 10*g* Release 2

# Data Modeling

*May 2006*

ORACLE

## Note:

This document is for informational purposes. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

# Oracle Warehouse Builder 10*g* Release 2

# Data Modeling

## INTRODUCTION

Consider the tasks of consolidating data, integrating data and analyzing data. Did you notice that none of those common tasks includes the word *modeling*? Are we not concerned about *how* we integrate, consolidate and analyze all of this precious data anymore? Or did we assume these tasks are trivial and that data integrators can attend to such undervalued tasks as partitioning data, determining the right grain of the fact tables, and performing advanced analytics?

A data model can add significant value if it is modeled to satisfy your business requirements and modeled using the appropriate technology. Unfortunately most people only realize this after they deliver some parts of the system and find themselves stuck in a model that neither performs nor solves business problems. In the worst case, a poorly defined model lacks the flexibility to address newly arising business problems

This paper evaluates Warehouse Builder as a tool that, in the context of data integration or consolidation projects, helps you build an appropriate data model. We discuss the business reasons and some unique and compelling features that Warehouse Builder offers to make your design experience easier and more efficient.

## THE SCOPE OF DATA MODELING

Our intention is to create a data model that enables us to represent integrated data from disparate sources. On that data model we also want to enable advanced analytics for our users to solve advanced business questions.

We are not trying to figure out which data modeling tool is the best tool in the market, we are trying to find the data modeling tool that would give us the best overall result for this data integration problem.

### A high level methodology

Many books have been written about data models and how to approach certain types of models. We are not trying to summarize these books or prescribe a different methodology. This paper describes a high level methodology upon which most data models seem to be based.

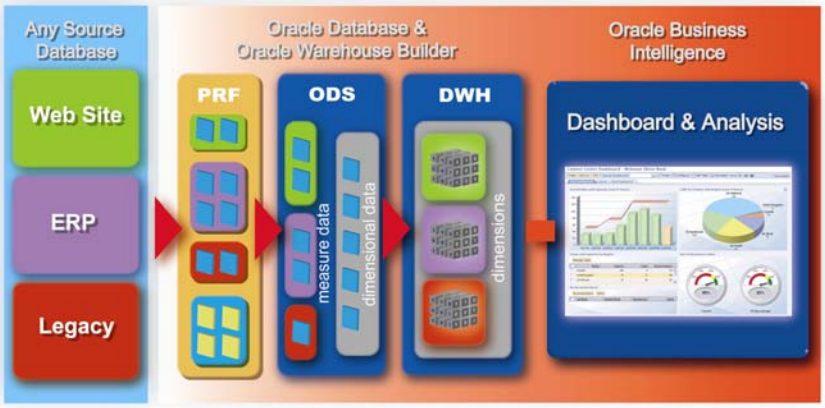


**Figure 1 A simplified methodology for data integration in Oracle**

In Figure 1, data from various sources and in a set of schemas flows into an analysis environment. Typical steps as shown above are to integrate data into a central schema of detailed data. In our case we refer to that schema as an operational data store (ODS). From the detailed data we move into several aggregated data elements represented here as cubes and dimensions in a data warehouse-like (DWH) set of objects. Advanced reporting occurs in the data warehouse, but can also transparently move down to the detailed ODS. Notice a first step in the integration flow labeled PRF. This is the profiling step and schema.

Since this methodology specifically looks at Oracle's infrastructure a number of components require a close look. Data Profiling, noted as PRD in Figure 1 is a crucial element in the methodology. As such this paper addresses data profiling in detail.

#### Embedded data profiling and data quality

One of the most crucial, but mostly overlooked topics, in data integration is data quality. In typical modeling schemas, data quality is mostly embedded via constraints. Domains are checked with check constraints, relationships are enforced with foreign keys etc.

In the Oracle methodology, we enable data quality at a higher level and allow data quality to penetrate each design component. We also drive data quality from the data.

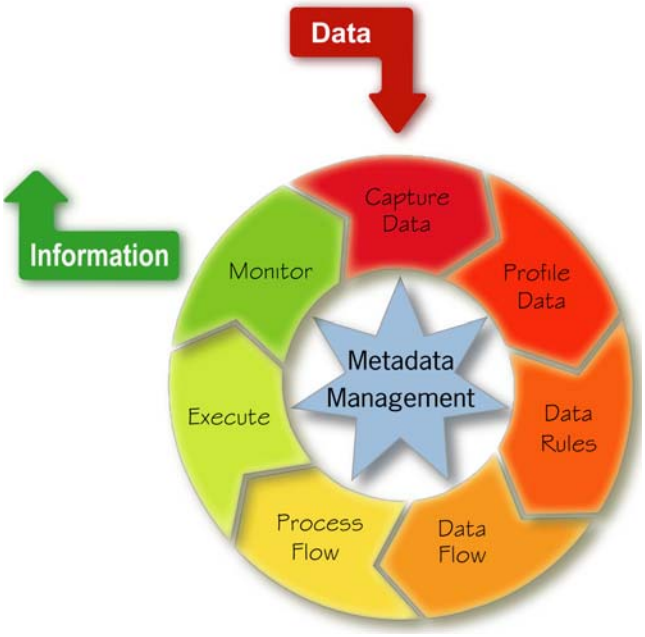


**Figure 2 Data drives information quality**

By allowing the data to drive the rules we are applying to it, we create better results. The results are not just better information but also better information delivery through better designed data models.

The learning phase, which is what data profiling is in its essence, is embedded in the methodology. Data profiling forms the first step to the processes of data integration and data modeling. After we learn the business rules from the data, we can generate a great number of data model elements directly from those rules. This directly increases the productivity of the data modeler and focuses the data modeler on the important rather than on the trivial.

#### Advanced Analytics

When you embed data quality in your designs, you make a major leap towards better information for your end users. Now, in what format do you present that information to end users? Most people in the industry, including probably your own end users disagree. Some information, especially at the detail level, can be dealt

with in printed or static lists. More advanced business functions require more advanced reporting methods.

This requirement means that traditional data models must go into previously uncharted territory. The days are gone that you could model data and think purely in terms of tables and relational objects.

Your end users are asking questions like what, where and why.

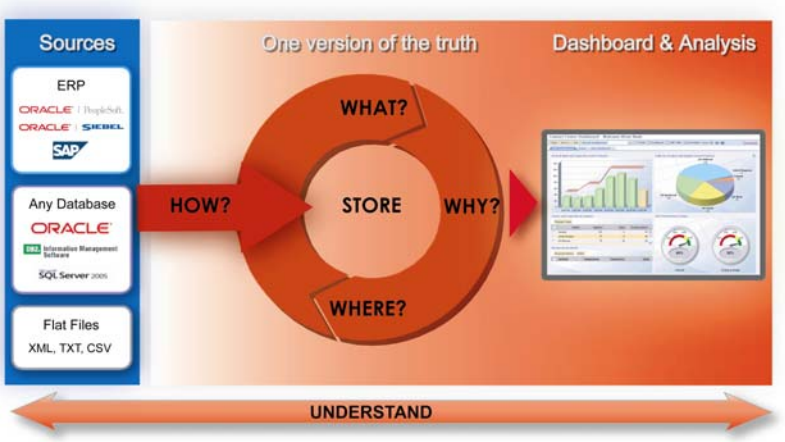


**Figure 3 Many different questions to answer**

To satisfy those questions you must enter the world of hierarchies, dimensions, x-y coordinates, drills and pivots. You also have to deal with fuzzy logic and predictions.

Additionally, after you deliver the data, your end users will want to understand where it came from. How was the data transformed? What rules were applied? In short, you need to add a lot of information and analytics into your model.

#### Best fit

With all of these requirements and the scope in mind, you can see why Warehouse Builder is such a good fit.

With Warehouse Builder you can understand the source metadata and data structures, integrate quality into the system via data profiling and data rules, and design a system that embeds all of the advanced analytics (OLAP, Spatial and Data Mining) into your model.

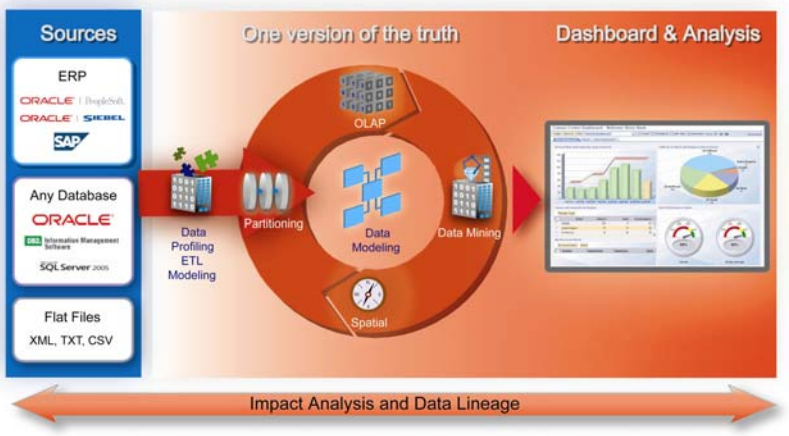


**Figure 4 Warehouse Builder, the ideal fit**

On top of that, due to the integration of all metadata Warehouse Builder provides end-to-end data lineage and impact analysis across the entire system.

## RELATIONAL MODELING FUNCTIONS AND FEATURES

To build a house you need a foundation. If the nuts and bolds are missing it will be a shaky foundation. The same is true for a data model. So we need to first understand basics and see whether they are provided.

In this section we look at those nuts and bolds, notably at defining the data objects (tables, materialized views etc.) and related parts (constraints, indexing, partitioning, storage etc.).

### Data Objects

The physical data objects in Warehouse Builder can be imported from a source, which can be a database catalog or a CASE tool. Or, you can create the data objects in the Warehouse Builder user interface or scripting language. From that metadata (Warehouse Builder works like a CASE tool) you generate DDL scripts to create the objects in the database.

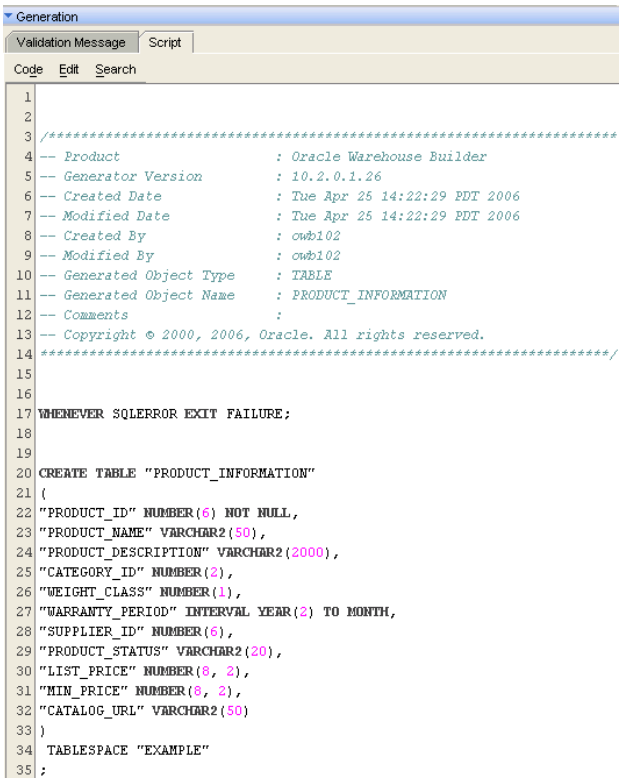


**Figure 5 Generated DDL from an imported table**

Use the data object editor to design and review relational objects dimensional objects, and business intelligence objects.

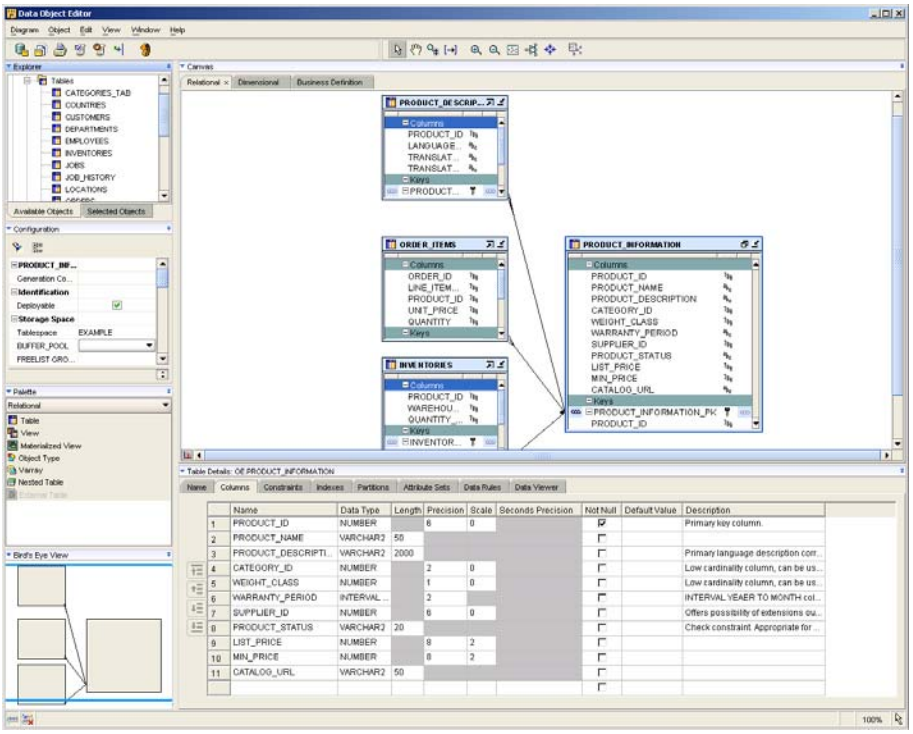


**Figure 6 The data object editor**

#### Materialized Views and logs

An object that was extended greatly in the new release is the materialized view object. Apart from designing a query to capture the data for the materialized view, you can configure the materialized view to perform exactly as you desire.

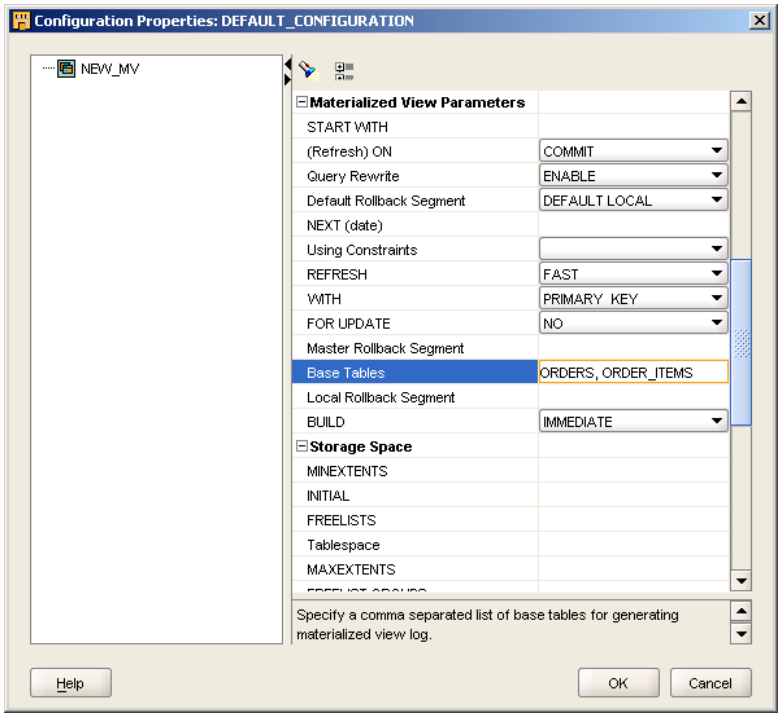


**Figure 7 Setting MV configuration**

The resulting code, here with a simple query example then reflects your choices. Any choice you do not explicitly make in configuration is defaulted to the database default setting.

```
21 CREATE MATERIALIZED VIEW LOG  ON ORDERS;
22
23
24 CREATE MATERIALIZED VIEW LOG  ON  ORDER_ITEMS;
25
26 CREATE MATERIALIZED VIEW "NEW_MV"
27  BUILD IMMEDIATE
28  REFRESH  COMPLETE
29  ON  COMMIT
30  WITH PRIMARY KEY
31  USING  DEFAULT LOCAL ROLLBACK SEGMENT
32  ENABLE QUERY REWRITE
33  AS select ord.customer_id customer
34 ,      sum(ord.order_total) total
35 ,      sum(otm.quantity) quantity
36 from   orders ord
37 ,      order_items otm
38 where ord.order_id = otm.order_id
39 group by ord.customer_id;
```

**Figure 8 A generated MV with its logs**

#### Data Types and User Defined Types

You also define or import user defined database objects in the data object editor. These user defined database objects greatly extent the already impressive list of data types supported by adding Varrays, nested tables and object types. You can nest types within other types.

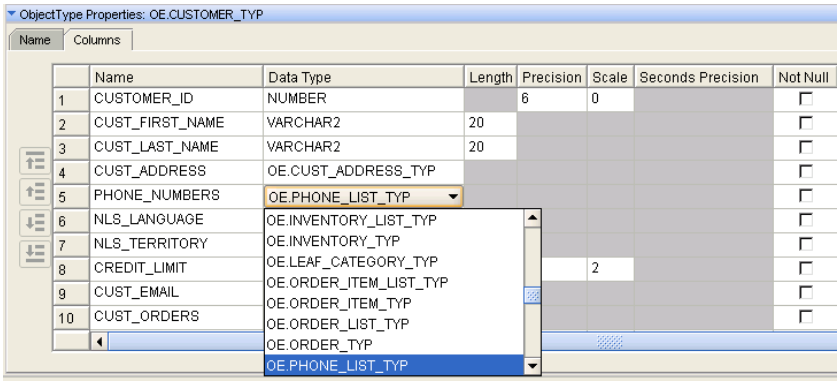


**Figure 9 Handling types - in this case nested within a type**

Warehouse Builder aims to support all data types in the database. Data types such as CLOBs are supported, as are the more common types like Date and Timestamp, Nvarchar2. Of course the classic data types such as number and varchar2 are supported. For a detailed list check the documentation.

### Indexing, Partitioning and Storage

The design aspects of a data model do not just involve the table structures and columns. A lot of the performance gains in certain applications are coming from the proper usage of indexes, partitioning and storage clauses.

#### Indexing and constraints

We will not attempt to describe how best to use indexing here, but we will highlight what is available in Warehouse Builder for these all-important but often overlooked objects. We will also discuss the constraints functionality here as well as it is closely linked to indexing.

Constraints and indexes are in the logical model and are created and maintained in the data object editor. When you import metadata, you receive all information for constraints and indexes as is shown below for the orders table.

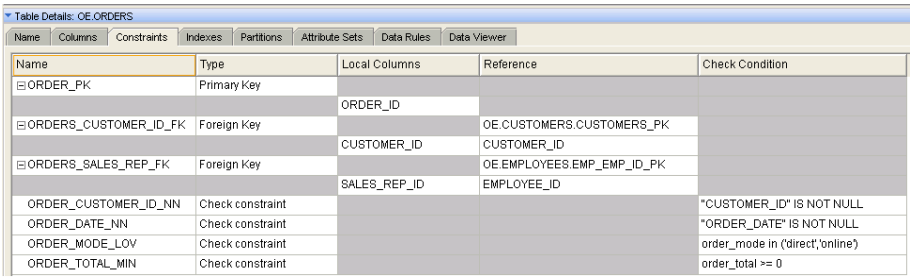


**Figure 10 Modeling constraints**

Constraints also have a physical implementation, which is dealt with in the configuration settings in Warehouse Builder. You have options as to how you want to store a constraint. For example, you can select the tablespace.

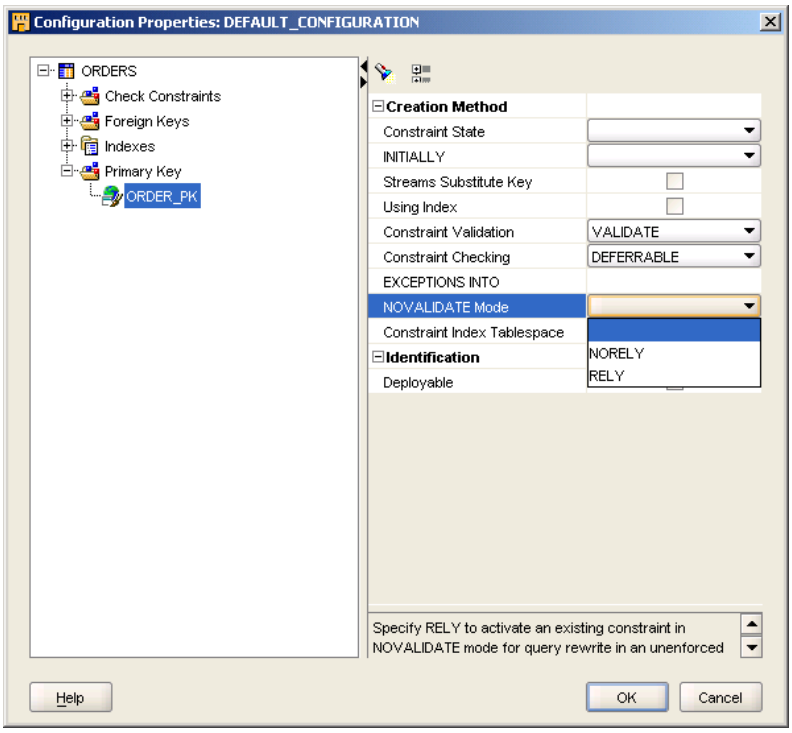


**Figure 11 Configuring a primary key constraint**

You can also choose to use an index for a constraint, determine validation and determine an error table if you want to remove records from the table that cause a constraint violation when enabling the constraint.

If you use an index, you can define these in Warehouse Builder as well. Again, this is done in the data object editor. You can determine the scope of an index and partition the index.

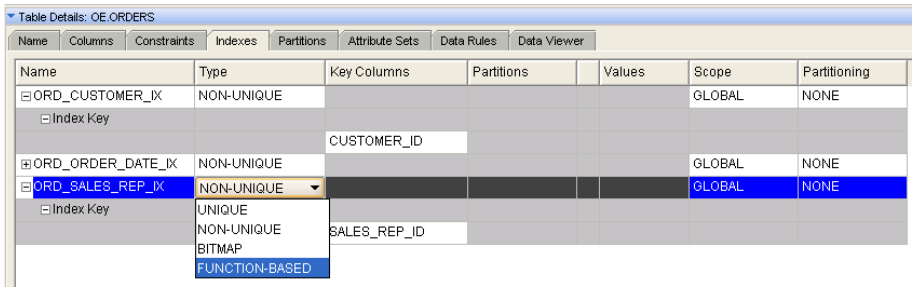


**Figure 12 Modeling indexes**

As with constraints, you can configure the indexes with regards to storage and all other details.

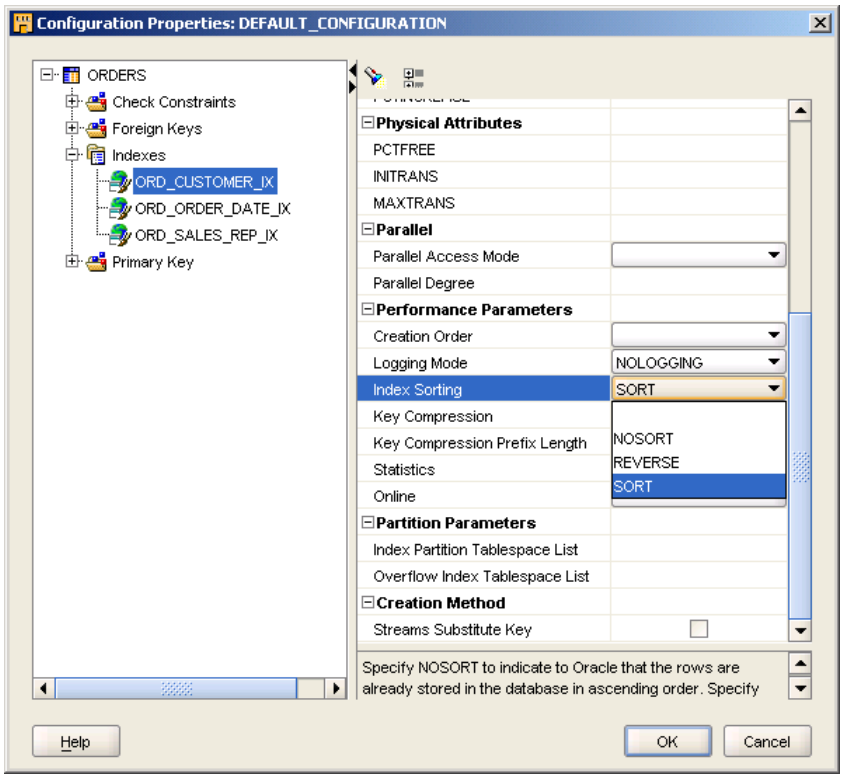


**Figure 13 Configuring physical properties on an index**

Decide on parallel or no parallel; decide on the sort strategy on the index and many other options.

#### Partitioning

We already discussed partitioning in the index panels, so it should be clear the Warehouse Builder supports partitioning not just at the table level, but also at the index level.

As with indexing and constraints, you use the data object editor to do all your work on partitions, and use configuration properties to determine the physical implementation.

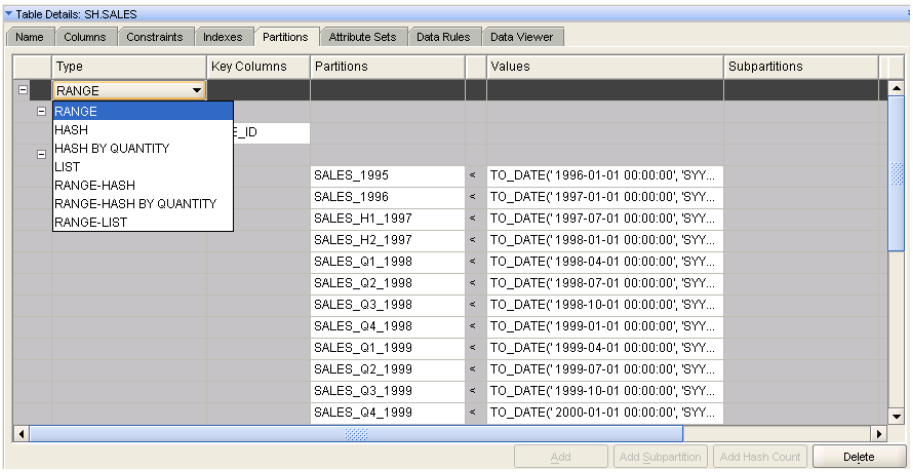


**Figure 14 Modeling partitions**

Warehouse Builder supports all partitioning models as supported in the Oracle Database. This includes modeling something called partition exchange loading (PEL).

#### Partition Exchange Loading

In PEL, data is loaded from the source table in an empty table that is an exact match in columns and types to the overall table that the data will eventually merge into eventually. Then indexes are added in parallel. Because these are only added to the table and not to the much larger end target, this is a lot faster. Then the table is swapped with an empty partition in the actual reporting table. The key is that there is no DML, hence there is no data moved which makes this extremely fast.

#### Traditional Partitioning Models

Back to the partitioning models, to allow faster definition of hash partitions in the Warehouse Builder user interface, an option called Hash by Quantity was added. This is not a type but merely a way to drive the UI for hash partitions allowing you to specify just the number of partitions.

Configuration takes care of some additional parameters as you can see below.

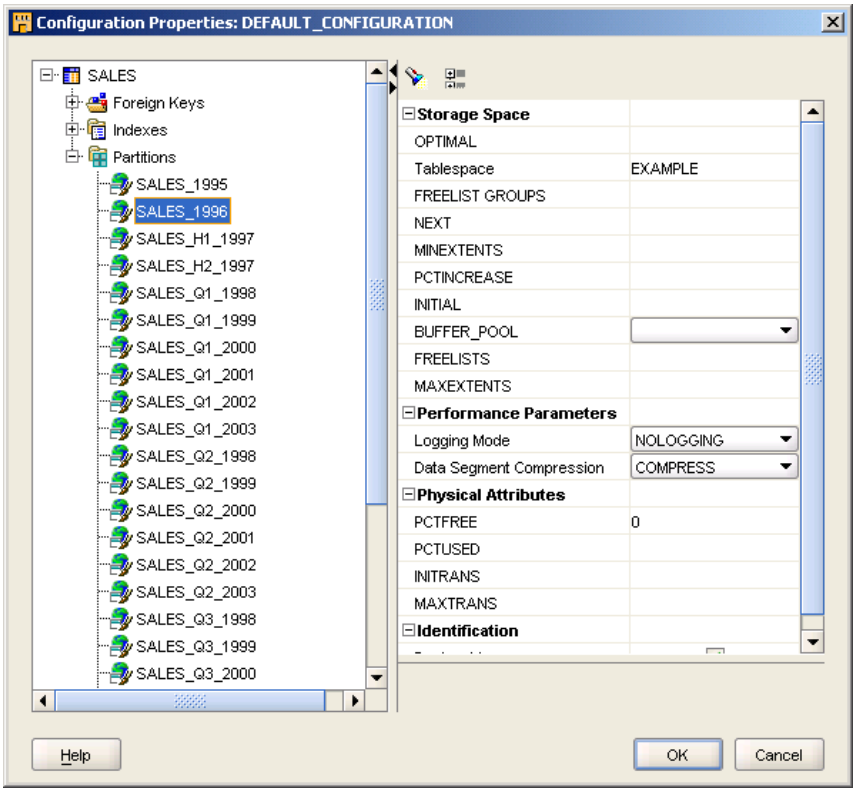


**Figure 15 Configuring partitions**

### Storage

We already talked about tablespaces in the index and constraint section. Obviously this also applies for the data storage objects such as tables and materialized views.

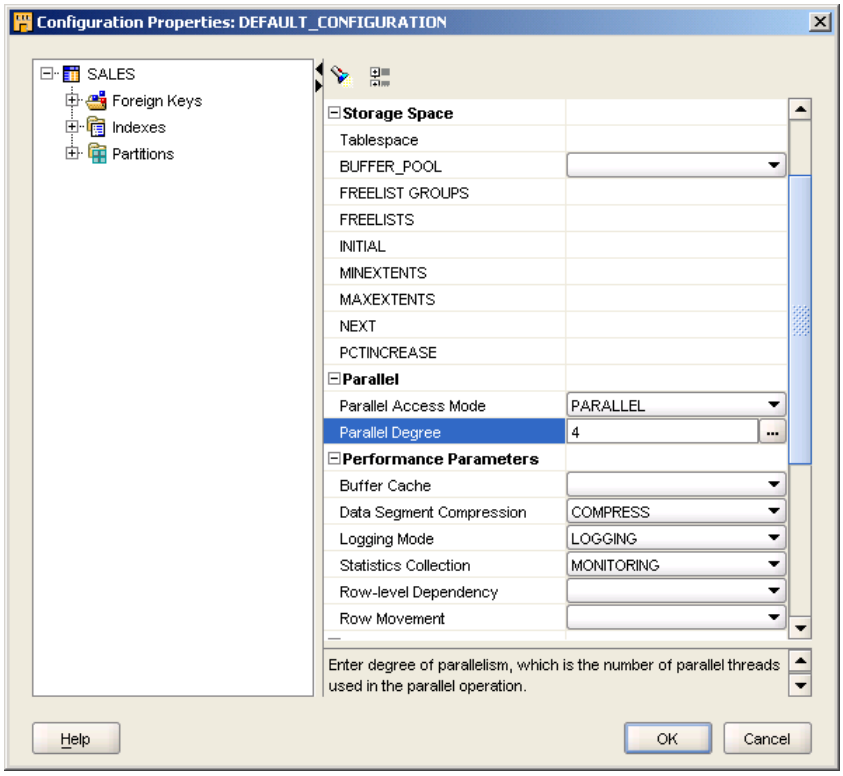


**Figure 16 An example of table configuration**

To determine other storage options you typically use the tablespace to drive that for the objects in it. Should you however insist on setting per table or even partition and index, you can do so with Warehouse Builder. The configuration properties for each object have the applicable details.

Perhaps the more important section of configuring a table is shown in Figure 16, where you can determine the parallel degree and the compression on the object. It enables you to transparently make use of the parallel support Oracle provides in both loading and in querying. It also enables you to reduce the storage size required for your objects by applying the compression on the object.

### Multi-Configuration

You set the physical characteristics for a logical design, but in many cases the production system and the development system have different physical aspects. Tablespaces for example are probably named differently. Since this impacts the generated code, Warehouse Builder allows you to have multiple configurations with physical settings. By simply switching the configuration you can generate code for the respective system without changing properties.

### Auto Completion

One of the interesting productivity boosters in Warehouse Builder is the auto completion of columns. Most organizations have certain standards, where columns

that have identifiers are always numeric, columns with codes are always char(3) and cannot be null etc.

You can customize the auto completion feature such that, for example, any column that a user enters with ID in it defaults to number. Or each column with CODE in its name defaults to char(3) and sets the column to Not Null. Needless to say that defaulting this will reduce the error rate of definition.

```
# SchemaEditor.properties
# Property File for setting sensible defaults for grid cells based on the name entered
#
# Summary of regular expression syntax.
#
# Name matches are case sensitive.
# The period (.) will match any single character.
# The asterisk (*) will match 0 - any number of the preceding character.
#
# Complete rules for regular expression matching are equvalent to those of the java.util.regex.Pattern class.
# http://java.sun.com/j2se/1.4.1/docs/api/java/util/regex/Pattern.html
#
# List of supported tags
# datatype,
# length,
# precision,
# scale,
# isnull
#


# ID Pattern
NameMatch = .*_ID
datatype = NUMBER
precision = 22
scale = 0

# ID exact Pattern
NameMatch = ID
datatype = NUMBER
precision = 22
scale = 0
```

**Figure 17 Naming configuration file**

Auto completion is achieved by using a driver file. You can edit the driver and enter your standards. A sample is shown here.

## DIMENSIONAL MODELING

We discussed a lot of the building blocks in Warehouse Builder to do relational modeling. Many dimensional models are implemented using these basic building blocks. Rather than giving you just the building blocks, Warehouse Builder adds value in the dimensional area by accelerating common tasks and enabling you to model without too much emphasis on the physical implementation.

### Relational and OLAP

Warehouse Builder enables you to model dimensions and cubes, also known as fact tables, at a slightly abstract and logical level. Rather than concentrating on table structures, Warehouse Builder encourages you to focus on the business problems first. After the solutions to these business problems are modeled, you should consider the implementation.

This means that you first analyze your business and, for example, model the product dimension and its hierarchies. After that model is complete, you press the "implementation" button to generate the underlying physical storage objects such as tables or analytic workspaces for OLAP. Warehouse Builder supports both a relational implementation based on tables and an implementation for the Oracle OLAP option.

#### Why is this model useful?

The main benefit from this dual implementation mode is found in data integration. By allowing a logical dimension (or cube) to be the target of a mapping, the implementation of this dimension (or cube) is shielded from the developer. There is no need for the developer to understand the differences between relational and OLAP loading, making the process a lot simpler. Warehouse Builder simply chooses the appropriate code to load the system.

### Relational models

If you choose a relational implementation, you layer a dimensional model on top of implementation objects. The dimensional model will drive the objects and how they are implemented. This process is called binding in Warehouse Builder.

#### Star or snowflake?

Warehouse Builder supports three types of binding, using a star model (a single table per dimension around the cube), a snowflake model (a table per level for each dimension around the cube) and a custom binding intended for reverse engineered models.

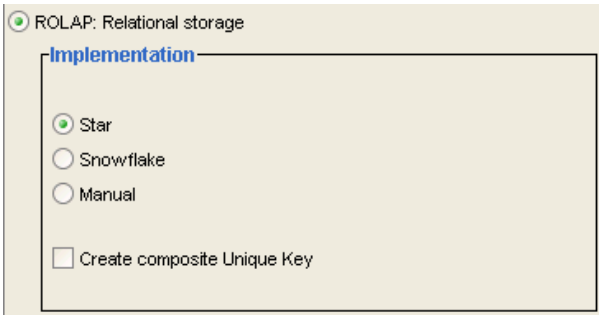


**Figure 18 Setting the storage options for a relation implementation**

With this flexibility and the binding implementation, which generates the appropriate tables for you, you save a lot of work in the definition and in the data integration steps.

#### Solved dimensions

A solved dimension is a structure where a table is used to store at least a single record to represent each member in a dimension hierarchy with a unique record.

The reason for creating a solved dimension is the common requirement to link fact data to a dimension at *any* level, not just the lowest level.

Imagine a time dimension for actuals going to the day level, but for budgets the data is created at the month level. In a normal star table it is not possible to link into this star table at a unique month.

The business case mentioned above is best explained by a detailed example. Consider a simple Product dimension

Line Of Business
|
^
Product Group
|
^
Product

In previous releases, Warehouse Builder created a dimension populated as shown below. Notice that there is no direct option to join product group fact data to this dimension.

| PK | LOB | Product Group | Product |
|---|---|---|---|
| 1 | Home | Video | VCR |

| | | | |
|---|---|---|---|
| 2 | Home | Video | Handicam |
| 3 | Home | Personal audio | Walkman |
| 4 | Pro | Audio | Microphone |
| 5 | Pro | Audio | Headphone |
| 6 | Pro | Audio | Amplifier |

Beginning in the new release, Warehouse Builder can alternatively populate the dimension like this:

| PK | LOB | Product Group | Product |
|---|---|---|---|
| 1 | Home | *null* | *null* |
| 2 | Home | Video | *null* |
| 3 | Home | Video | VCR |
| 4 | Home | Video | Handicam |
| 5 | Home | Personal audio | *null* |
| 6 | Home | Personal audio | Walkman |
| 7 | Pro | *null* | *null* |
| 8 | Pro | Audio | *null* |
| 9 | Pro | Audio | Microphone |
| 10 | Pro | Audio | Headphone |
| 11 | Pro | Audio | Amplifier |

Against a dimension populated like this, it is possible to join data at any level. This is therefore the ideal solution for this very common business problem. All of this is generated out of the box when you load data into a dimension with Warehouse Builder.

#### Slowly Changing Dimensions

Every ETL developer struggles with Slowly Changing Dimensions, mostly because no data modeling tool allows these objects to be modeled easily.

In Warehouse Builder, slowly changing dimension logic is designed in the actual dimension metadata. The dimension captures all logic to be applied to the data coming into the dimension.

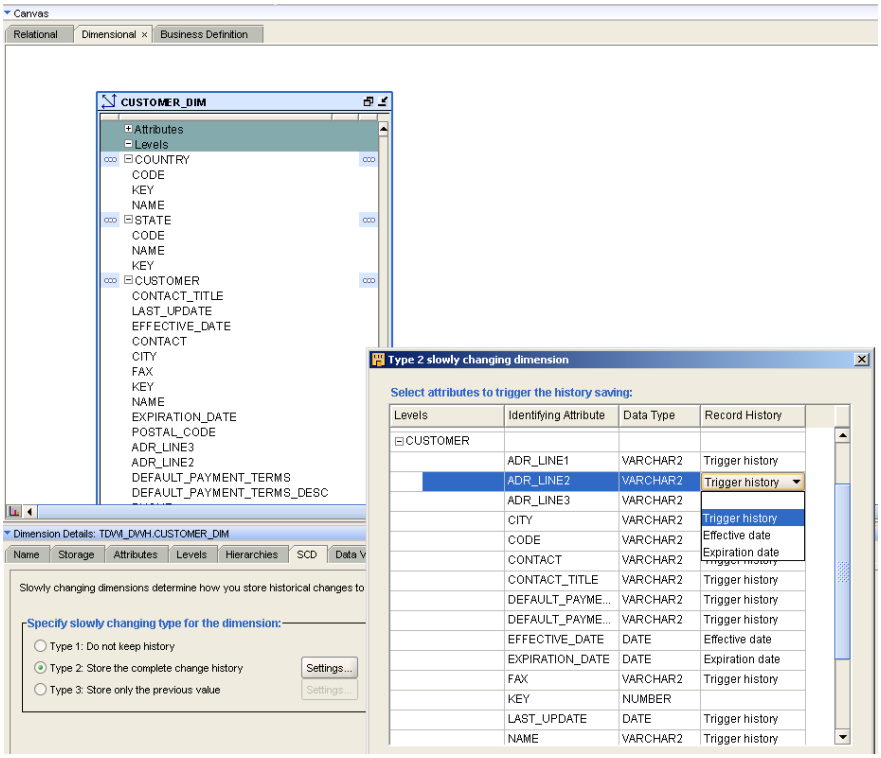


**Figure 19 A single implementation for Slowly Changing Dimensions**

After the business user has decided which attributes are important, the data modeler designs the dimension. Within the ETL steps, the developer now sees the dimension as any other dimension. This developer does need to worry about how to handle changes and updates. Warehouse Builder automates changes based on the dimension definition. The combination of these design and standardization steps makes the process of effectively dealing with slowly changing dimensions much faster.

This is a unique value proposition showing the synergy between a data modeling and an ETL environment.

### Cubes

Most people think of fact tables when they model a relational warehouse. Because Warehouse Builder supports both fact tables and cubes, the term cube is used for the logical object. Once you bind a cube to a table, you have in effect created a fact table.

Warehouse Builder automates the process by creating all the foreign keys to the dimensions. Creating the bitmap indexes on the foreign key columns and ensuring that the lookups in ETL to the dimensions are done automatically.

### Storage settings

Because the underlying objects that are used with this modeling technique are the same as you would usually use in relational modeling, you have all options available

to you. You can now place indexes, partitioning and constraints on the physical model.

## OLAP models

With an OLAP implementation – using Oracle OLAP – you can achieve tremendous results for business users. While OLAP seems like magic sometimes it becomes really easy in Warehouse Builder.

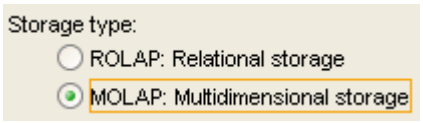


**Figure 20 Choosing storage for a dimension**

You simply model the dimensions at the logical level and then Warehouse Builder generates the appropriate code for you. The same goes for an ETL job, Warehouse Builder takes care of generating the correct code to load the dimensions and cubes, all using the native XML language for Oracle OLAP.

### Dimensions

The important objects in OLAP are dimensions and cubes at the logical level. The implementation is hidden from the user in the engine and there is no level of indirection to tables.

Because dimensions (and cubes) are stored differently, it is much easier to create advanced hierarchies such as value-based hierarchies and unbalanced hierarchies. Warehouse Builder supports these models in the modeling tool, and enables you to combine the hierarchies in a single dimension. ETL is done transparently, again making it easy and fast to load data into Oracle OLAP.

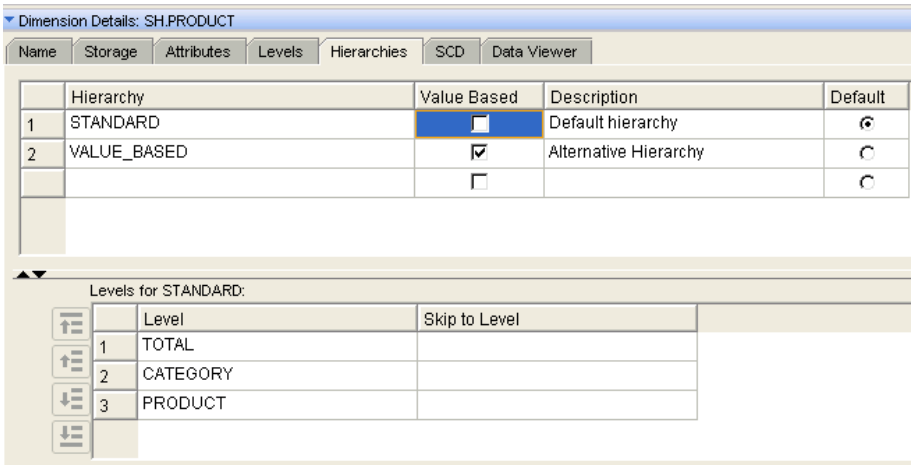


**Figure 21 Combining multiple hierarchy types**

### Cubes

While dimensions are interesting topics for discussion in a business situation, from a modeling perspective they are relatively simple in OLAP. Cubes are the real fun bits to discuss from a Warehouse Builder and a modeling perspective.

#### *Calculated measures*

Calculations or formulas are the strong point of an OLAP engine. You can do complex calculations ad hoc by using the capabilities in the engine. Typical calculations include ranking, share and time series analysis.

Warehouse Builder makes the modeling of these calculations very simple and fast by enabling you to generate the model for these types. This is a huge time saver, enabling you to generate instead of manually creating all these measures by hand.

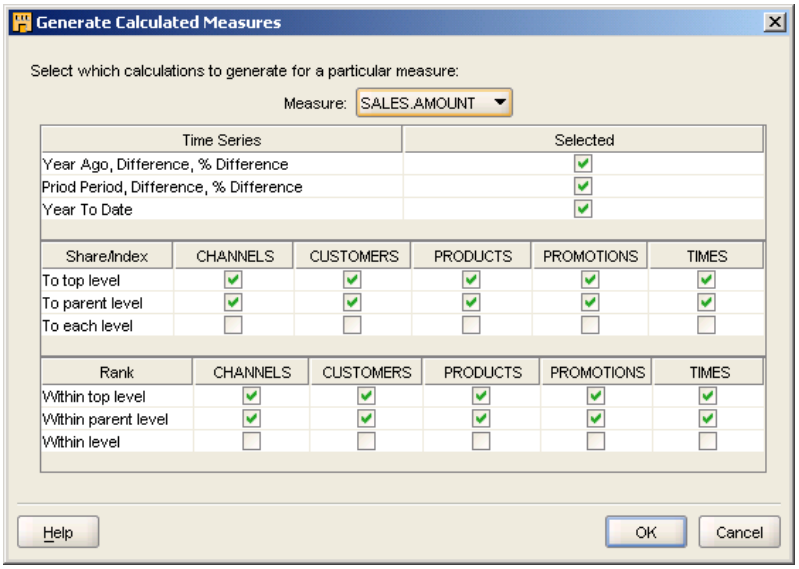


**Figure 22 Advanced analytics are generated**

On top of this functionality, you can create your calculations using either the templates that are supplied or by adding your own code in a calculation.

#### *Tuning*

When you define cubes there are a few things that impact the performance. These are however part of the design of the cube, rather than part of the physical properties.

To be really fast you must decide how much data is pre-calculated when the cube is aggregated. You do so per hierarchy and per measure. You also determine which aggregation you choose for each of the measures. This is done in the data object editor for cube objects.

To gain more performance in this aggregation step and reduce the space the cube occupies on disk, you should work with partitioning and compression as is shown below.

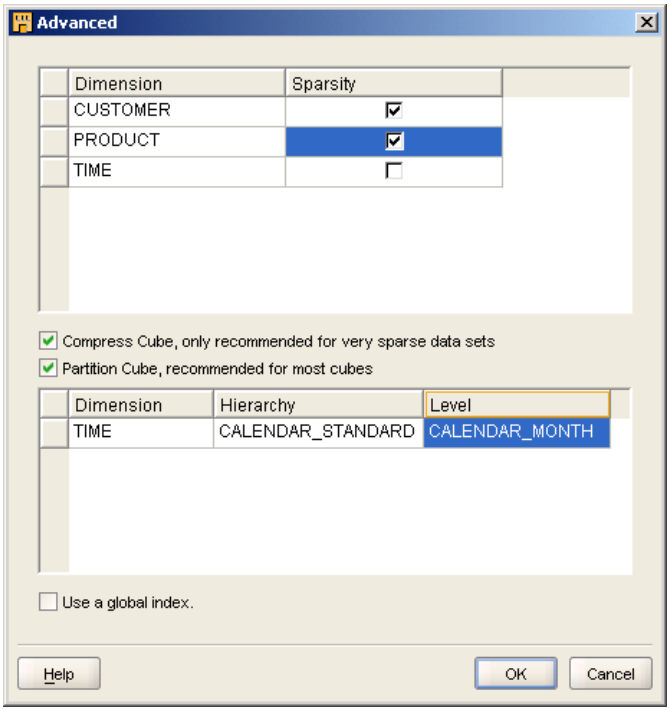


**Figure 23 Advanced storage options**

Partitioning enables you to use multiple processes to aggregate the data. Compression enables you to condense the cube to a smaller size.

### Time Dimensions

A separate mention should be made about time dimensions. Warehouse Builder generates a complete solution for time dimensions including mappings that generate data.

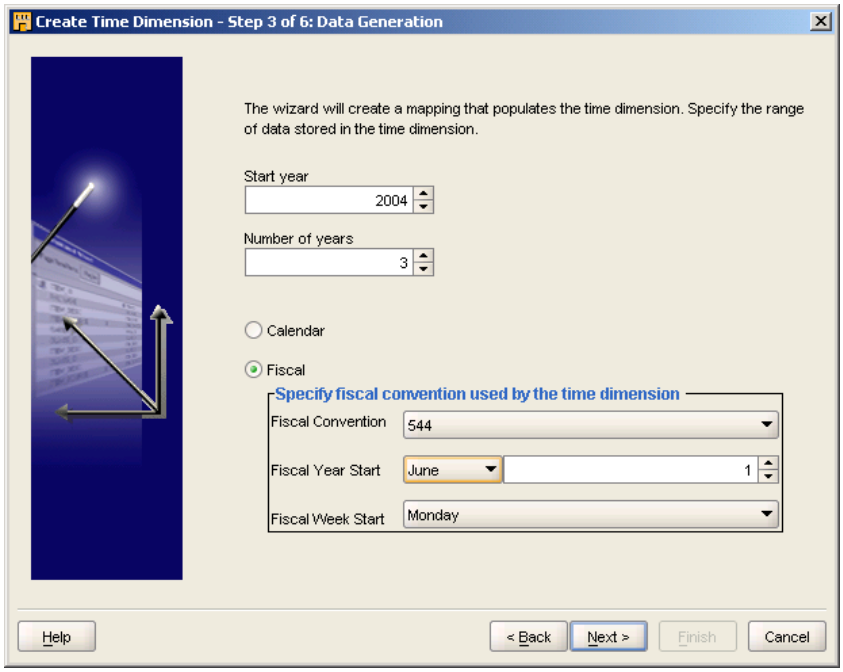


**Figure 24 Generating a fiscal time dimension**

You can change the definition and regenerate the mappings for those changes, and you can load both OLAP and relational time dimensions.

Warehouse Builder allows you, as you can see in Figure 24, to generate common fiscal time dimensions. You can mix and match fiscal and calendar hierarchies in a single dimension enabling maximum flexibility.

## SUMMARY AND CONCLUSION

Oracle Warehouse Builder enables both data modeling and data integration with impressive results. The embedded data quality and data profiling functionality gives you insight into your data and assists you in more rapidly building a relevant and flexible data model.

You can design dimensions in an abstract fashion. That is, first represent business data in the form of dimensions. Determine what questions you have of that data, and then decide upon the best answer to those questions by selecting from the various relational or OLAP implementations.

As you build your data model, take advantage of a number of Warehouse Builder features that greatly improves performance such as indexing, partitioning, and Partition Exchange Loading. Other features improve productivity such as Multi-Configuration and Auto Completion. And other features such as Slowly Changing Dimensions enable you to design more easily and efficiently.

**Oracle Warehouse Builder 10*g*R2 - An Overview**
**May 2006**
**Author: Jean-Pierre Dijcks**

**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**
**www.oracle.com**