ORACLE8I AND ORACLE DESIGNER: A POWERFUL COMBINATION FOR DATA WAREHOUSING

Muralidhar (Murli) Manickam, Hencie Consulting Services, Inc.

Sameer Jejurkar, Hencie Consulting Services, Inc.

INTRODUCTION

Today, data warehousing has evolved to a point where it is generally accepted that data warehousing is an excellent approach for organizations to transform existing data, in order to make it available for analysis leading to decision making. Building and using a data warehouse is probably the best way to gain access to reliable information that can be used for making decisions. The question is "how do I build the data warehouse and what tools can I use to do so effectively and efficiently?". This paper takes a look at how Oracle 8i and Oracle Designer have simplified the process of data warehousing design, implementation and maintenance. Both these products provide a variety of features that facilitate the data warehousing process. Oracle 8i is a powerful RDBMS that has a lot of features that support data warehousing. Oracle Designer can be used to assist in the design, implementation and maintenance of a data warehouse.

ORACLE 8I

Oracle8i – the latest release of Oracle RDBMS, like its predecessor Oracle8, continues the tradition of providing a rich environment for data warehousing. Significant features like table and index partitioning, bit-map indexing, better processing for star schema queries, support for distributed databases etc. have been available since Oracle8 (some from later versions of Oracle7). Oracle8i enhances this support with new features like materialized views, summary advisor and transportable tablespaces.

ORACLE DESIGNER

Oracle Designer is a tool for modeling, designing and generating client/server databases and database applications. Although Designer has been primarily used for developing database applications, its strong and extensive support for modeling and forward and reverse database engineering make it highly useful for the data warehousing process. It provides a rich set of analysis and design tools that can be used to model the data warehouse requirements and its design. Designer can also *design capture* (reverse engineer) existing databases to create models using the same tools. Oracle Designer adopts a repository driven approach – all data is stored in a central repository, enabling the members of a team to work together and efficiently, thus making the task of project management easier. Oracle Designer also provides organizations the code re-usability feature. The proprietary relationship between any piece of code and its programmer doesn't exist anymore with Oracle Designer. Oracle Designer supports a variety of methodologies including rapid-prototyping. One can use designer to quickly build a prototype to validate the feasibility of the DW. A prototype can help to make informed decisions about estimating the costs and resources. It can help to gain

insights at an early stage of the development life cycle, rather than detecting problems at a later stage of the project (eliminating the possibilities of incurring heavy costs).

REQUIREMENTS OF A DATA WAREHOUSE

The following technical requirements must be satisfied for any data warehouse:

- Rapid Access to Data:

  The users should be able to get acceptable response time from queries

- Adaptability:

  Businesses are changing at a rapid speed. The DW should be able to adapt easily, to keep up with such rapid changes.

- Scalability:

  As the DW is implemented the data volume will grow substantially. More users will start using it. It is necessary that the DW be scalable enough to handle changes (increase) in both data and user loads.

- Availability:

  As the DW becomes increasingly important for organizations, its availability, when needed, is a very high priority. This enforces organizations to adapt the right infrastructure and tools.

- Manageability:

  As the DW evolves and grows more complex, managing it becomes very critical and important. During the life of a DW it undergoes lot of changes, in terms of data as well as structure.

While designing and implementing a data warehouse, each of these requirements must be kept in mind.

DATA WAREHOUSING PROCESS

The data warehousing process is evolutionary. A data warehouse is typically implemented as a series of successive iterations. Every iteration in the process of building a data warehouse, involves the following stages (which is also shown in Figure 1)

- Gather requirements
- Design the warehouse
- Build the data warehouse
- Clean, transform and load Data
- Maintenance

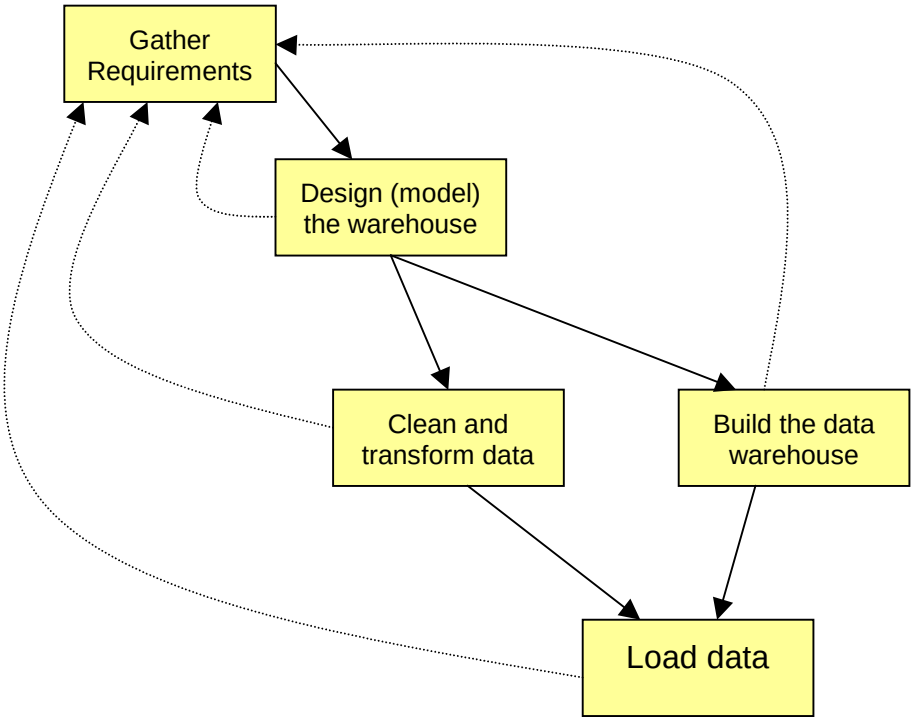

Figure 1

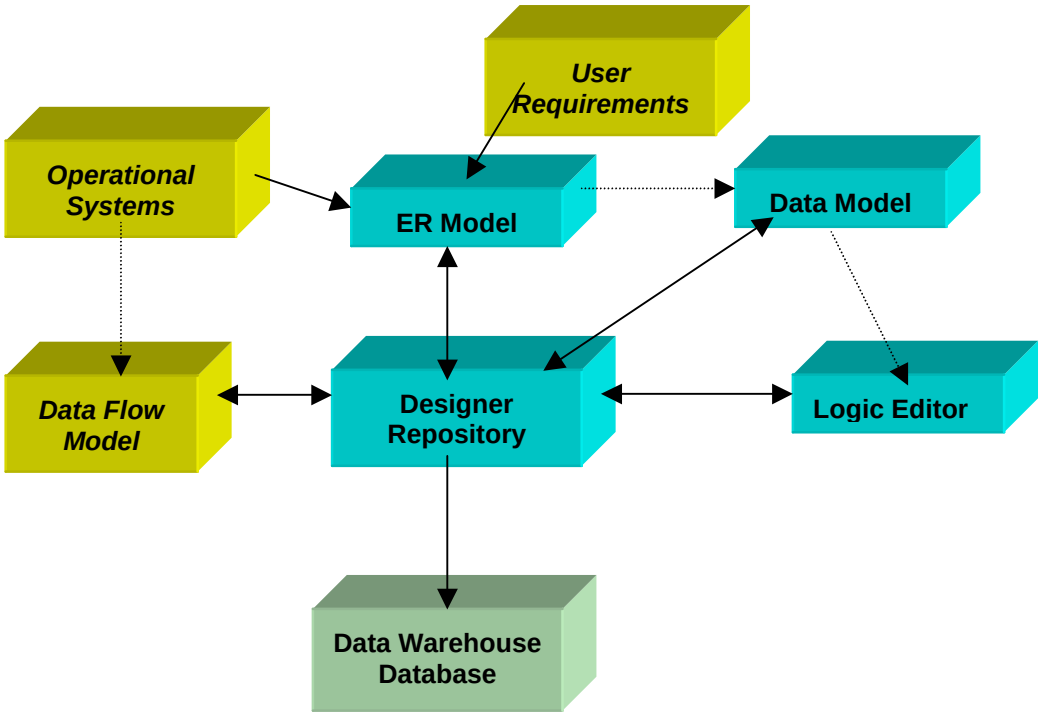

Figure 2

GATHER REQUIREMENTS

The methods for gathering requirements for a data warehouse are typically classified as either source-driven or user-driven. For source driven requirements gathering, the operational systems form the basis for defining requirements. Having or building a logical ER model based on the operational system helps to better understand, analyze and communicate the data feeds and source requirements necessary to support the multi dimensional data warehouse. The ER model is important to understand the various elements and their interrelationships and dependencies. The objective of such a model is to identify the common data between different business areas and to provide a single set of data definition. This model need not be very detailed. In fact, it can be simple and identify the primary entities and relationships among them. Such a model can also help in defining the scope of the data warehousing process. This process is shown in Figure 2.

ER MODELING IN DESIGNER

Entity Relationship Diagrammer in Oracle Designer provides:

- A diagramming tool for creating entity relationship diagrams including support for advanced features like arcs and super-type/sub-type entities
- Access to a multi-user Repository for creating and maintaining detailed definitions of entities, attributes and relationships

Figure 3 shows a simplified ER diagram for a order entry system in Designer.

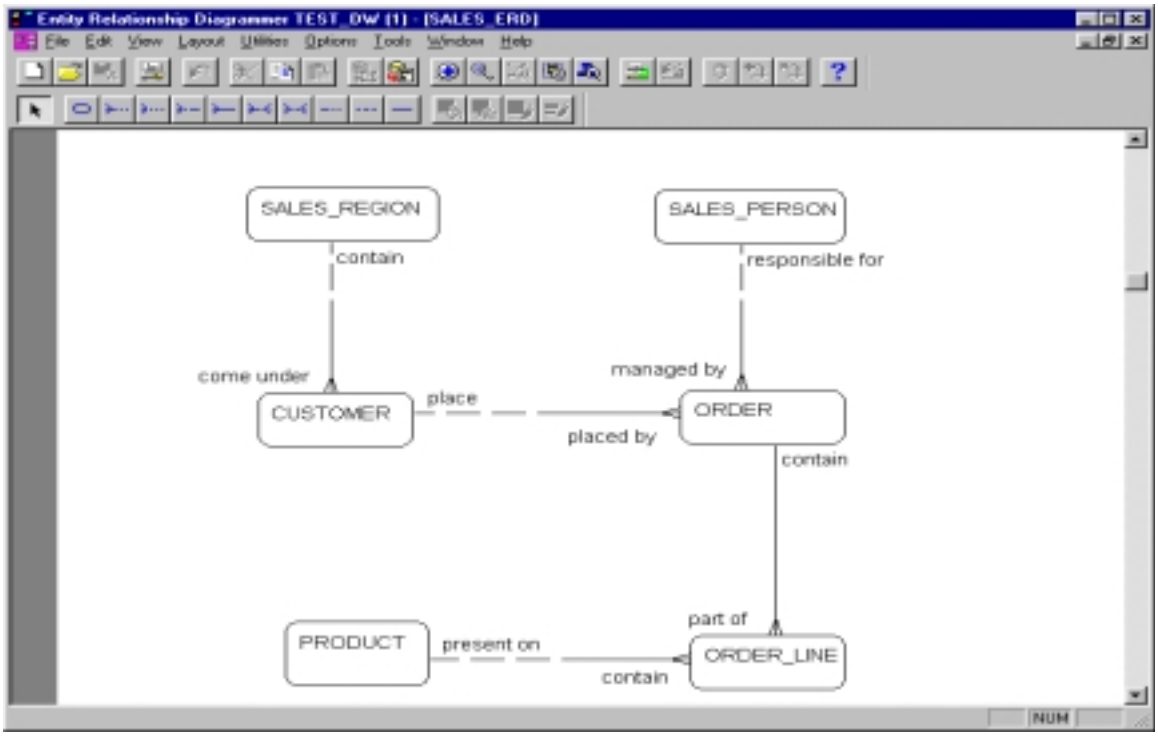

Figure 3

If for a relational system, the ER model does not exist or is outdated, as is often the case, the following steps can help to create an ER model based on the existing table structure.

- The Design Capture utility of Designer can be used to get the database structure into the repository
- The Table to Entity Retrofit utility can be used to produce the corresponding ER diagram
- This diagram can then be refined to depict the current requirements

In user driven gathering, the requirements are defined by investigating the functions the users perform. Traditional analysis meetings, interviews and JAD (joint application development) sessions are used here to drive out the requirements. In this case, the ER model can be build based on these investigations. To reduce complexity and strengthen understanding, the ER model should be split into multiple diagrams based on subject areas. Designer's repository driven approach enables sharing of entities across subject areas.

Note: You will have to create a new application system within the Designer repository with property Data warehouse? = 'Yes'.

DESIGN THE DATA WAREHOUSE

DIMENSIONAL MODEL

Once the logical model has been defined and well understood, it is necessary to define the dimensional models for the warehouse. Dimensional modeling has two primary purposes. First, to present the design of the data warehouse is a form that can be easily understood by the end users and second, to provide a framework that allows for high performance access.

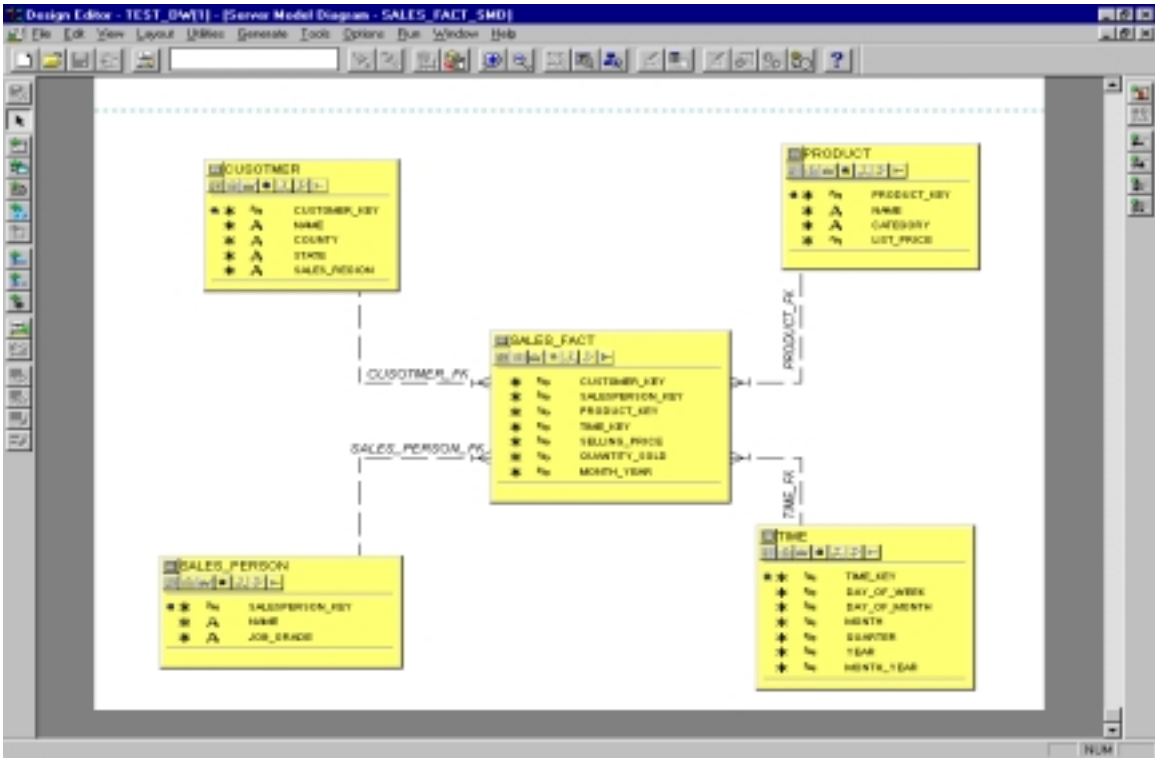

Figure 4

Figure 4 shows a simplified dimensional model created as a Server Model Diagram in designer based on the earlier ER diagram. Each dimensional model consists of a fact table and a set

of smaller tables called dimensions. The fact table is a collection of data items, usually numerical measures that represent “facts”. Each dimension represents a contextual background for facts. Dimensions are the parameters over which the analytical processing is performed. Each dimension table has a single-part key that corresponds exactly to one of the components of the multi-part key in the fact table.

Some of these models may share some dimensional tables. You can use the Server Model Diagrammer in the Design Editor to create and maintain the dimensional models. It is important to document design considerations behind each dimension and fact table as well as documenting the relationship these models have with the logical ERD. Designer allows the documentation of denormalization, validation and derivation logic whilst keeping track of source entity/attribute information.

B-TREE INDEXES

In an Oracle database, B-tree indexes can be used to improve the performance of queries that retrieve a small proportion of rows in a table. The cost-based optimizer uses the statistics in the data dictionary to make intelligent decisions about determining which indexes to use. The data warehouse must be optimized for queries. So multiple indexes must be defined and created based on the analysis already performed. Designer should be used to create each index definition and document the reasons behind why this index was created.

BITMAP INDEXES

Bitmap indexes have been available since Oracle version 7.3. Bitmap indexes are appropriate in cases where the column contains a few distinct values (low cardinality). Examples include **State**, **Gender** (or **Sex**), **Sales Region** and so on. You can create bitmap indexes that can enhance performance of queries based on low cardinality columns. Those queries that allow multiple bitmaps to be used for restricting the resulting set of rows can be executed very efficiently. In Oracle8 and above, bitmap indexes are used to improve the performance of star joins – joins connecting dimension tables to the fact tables.

TABLE PARTITIONING

As the tables in the data warehouse grow very large they become less manageable. The performance of queries that access these tables can deteriorate considerably. Such tables should be considered for partitioning. Typically, in a data warehouse it is the fact tables can become very large; therefore they are good candidates for table partitioning. Oracle has supported range table range partitioning since release 8. A single table can be partitioned along ranges of values of the **partition key**. For example, a table with a date column can be partitioned by month – the rows for each month will reside in a separate partition. This partitioning is transparent to the applications, tools and end-users. Oracle handles the movement of data into and querying from partitions automatically. Each partition can be stored in a separate tablespace.

Careful thought should be given to the decision of selecting the partition key. The partition key should consist of columns that are frequently used in the where clauses of queries. This will allow the query optimizer to eliminate several partitions outright restricting the search to fewer portions of the table. This can lead to dramatic improvements in query performance. For example, consider a sales fact table that has been partitioned month-wise and contains sales data for two years. A query to find the total sales during the second quarter of this year will look at only the three relevant partitions. Assuming fairly uniform distribution of sales, this means that approximately only one-eighth of table is accessed. There are additional benefits to using table partitioning. Partitions can be backed up and recovered individually. They can also be reorganized individually. Partitions can be added or dropped individually. Thus, if the sales fact table should contain only five years of data, every month the partition for the oldest month can be dropped and that for the new month added.

All this can be done without affecting any of the other partitions. This effect of a rolling window can be accomplished relatively easily.

Oracle8i introduces **partition-wise joins**. Joins between equi-partitioned tables – tables having the same partition key – can result in better performance, as the optimizer considers new methods that leverage these partitioning characteristics.

Table partitioning can provide substantial benefits including better query performance, better data reorganization, backup and maintenance. Estimates about the sizes and growth of such large tables can help you make partitioning decisions during the design stage. Indexes, like tables, can be partitioned too. Partitioned indexes can be created on partitioned as well as non-partitioned tables. In case of partitioned tables, the partitioned index may (prefixed local index) or may not (non-prefixed local index) have the same partition key as the table. Due to the partition elimination feature the prefixed local indexes usually provide the most improvement in query performance.

Summaries, Aggregates and Materialized Views

Many data warehouses create summary and aggregate tables that store pre-computed results of aggregated data such as sums, averages etc. The primary purpose of these tables is to provide better user response time. Usually, accessing these pre-computed results is more efficient than doing these aggregations on the fly for each user request. However, this leads to the overheads of managing these tables and populating/updating them on a regular basis. A related issue is anticipating the user needs and creating the necessary summary tables in advance.

Oracle8i introduces a new feature called **materialized views**. When so requested, Oracle materializes the view by running the view query and storing the result in the database. The queries against this view will now access a physical table. In most cases this will lead to a huge improvement in query performance. Summary and aggregate tables can be implemented in Oracle8i as materialized views.

Another related feature called **automatic query rewrite** allows Oracle to consider materialized views automatically when evaluating queries. This enables the optimizer to determine if accessing the materialized view is more efficient than accessing the underlying tables. The use of materialized views is transparent to the user because Oracle automatically rewrites the SQL to use the physical table. The DBA can create views with the necessary SQL and materialize the view. The biggest advantage of using materialized views is that it requires no change in the database design or the SQL statements used in queries and applications.

With materialized views it is critical that they be synchronized with their underlying tables. Oracle8i automatically keeps these views updated. There are three available options – full refresh, partial refresh and deferred refresh. Full refresh the view is rebuilt completely following a change in the base table. Incremental refresh allows the view to be changed only as required to remain consistent with the underlying table. Deferred refresh can be used to refresh the view at a later point in time, such as once every week.

Physical Database Design in Designer

Using Designer you can specify the definitions for almost all database objects including tables, columns, constraints, indexes, views, snapshots, sequences etc. (as shown in Figure 5). Designer supports the new Oracle8 features including table partitioning, bit-map indexes etc.

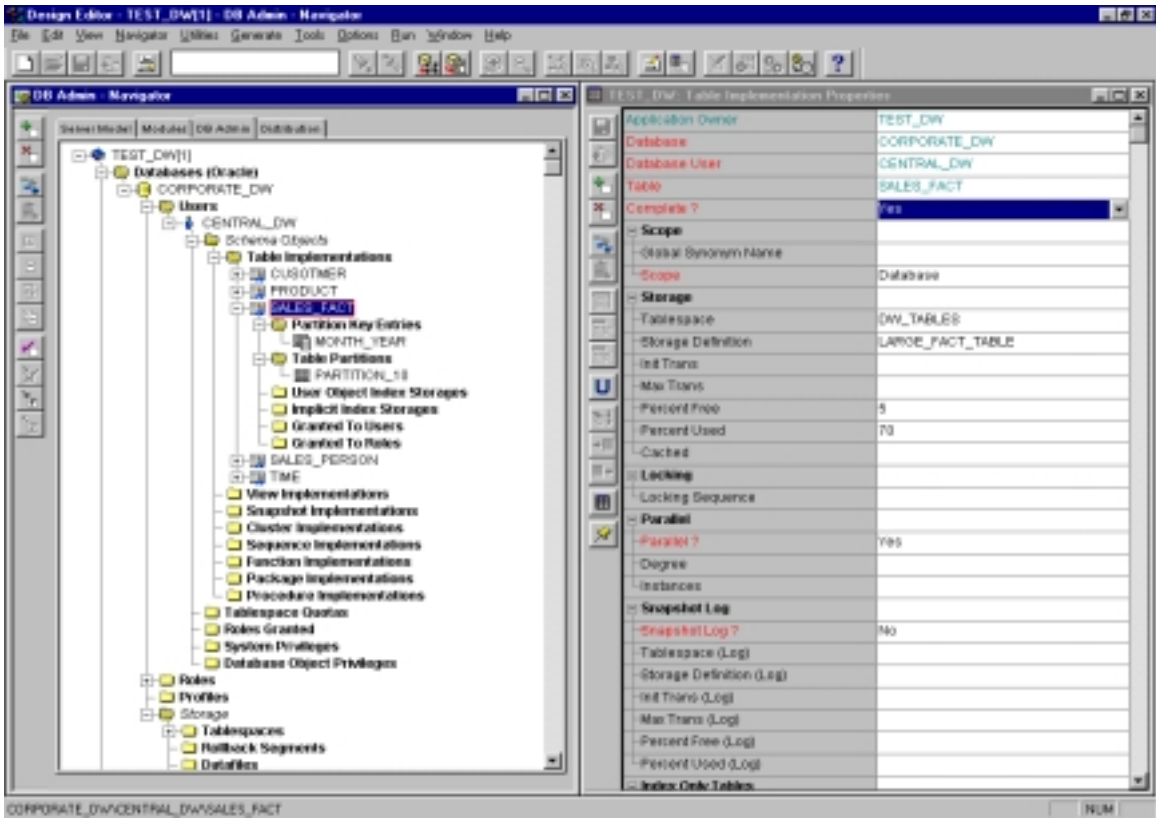

Figure 5

DOCUMENTING DATA SOURCES

Before you build the warehouse and load data into, it is necessary to identify and document, in detail, the sources of data. These can exist in current operational systems, including legacy systems, or external sources. You can create database definitions for oracle as well as non-oracle relational databases in Designer. Definitions for Non-Oracle databases can be created as ANSI database definitions. You can easily reverse engineer (capture design of) objects from such databases into Designer. The design of flat file structures cannot be captured in Designer. However, designer does provide a way to document this. Flat files are usually composed of multiple records and records are composed of fields. The following information can be documented (as shown in Figure 6):

- Type, organization, access method, storage medium and file size for the file
- Format and size of the record
- Data type, length, storage format of the fields in the record
- Usage of fields, that is, you can associate a field with a column of a table already defined in the repository

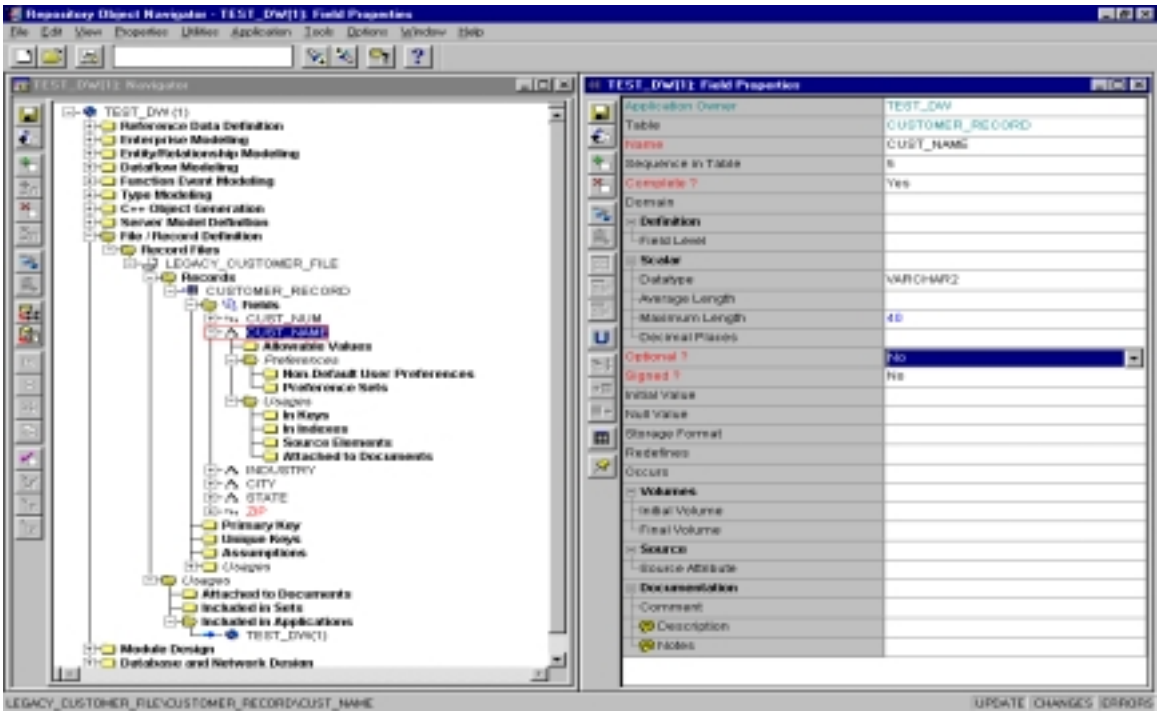

Figure 6

DEFINING THE MOVEMENT, EXTRACTION AND LOADING OF DATA

Traditionally data flow diagrams (DFDs) have been used in the design of OLTP systems to model the processes and flow of data in the system. In data warehousing process data undergoes a great deal of transformation between its movement from its source operational system to its destination – the data warehouse. DFDs can be used to effectively model this movement, transformation and extraction of data (shown in Figure 7).

A DFD consists of four notations:

- **Sink** is source or destination of data that resides outside the boundary of the system. In data warehousing, sinks can be used to denote the sources of data outside the warehouse. For example a legacy accounting system.
- **Process** depicts a task that processes the incoming data flows to transform it into an outgoing data flow. You can use processes to depict the extraction and transformation steps. The definition of the process can include detail logic or pseudo code for it.
- **Data flow** represents the flow of data between processes, sinks and data stores. Data flows can be used to define the movement of data between their sources, the extraction/transformation processes and data stores.

- **Data stores** are used depicts temporary or permanent stores of data. In the warehousing process they can be used to denote the temporary tables or files that are used to hold data before it is loaded into the warehouse tables.

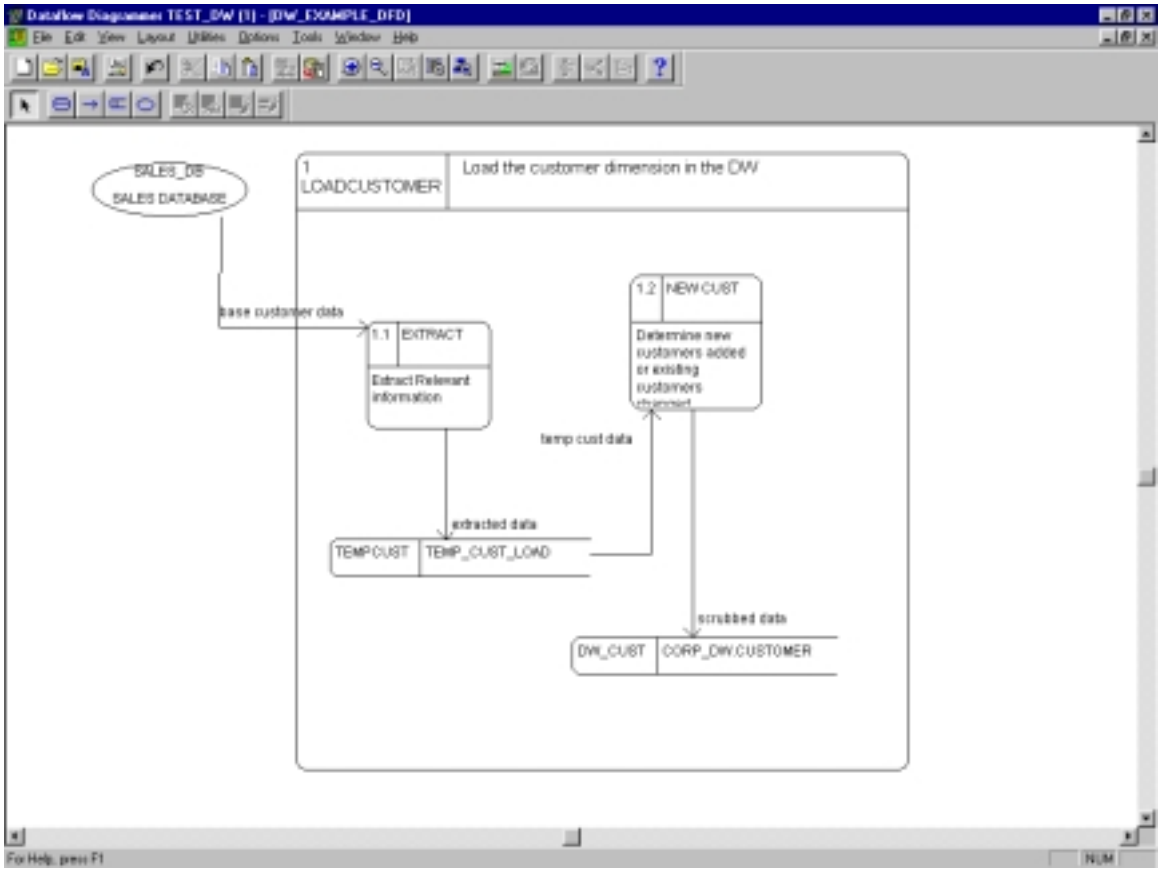

Figure 7

Designer provides a very good data flow modeling tool with comprehensive data dictionary support. You can attach detailed data items (fields, columns, pieces of data) to each data flow and data store and define the frequency, resources and logic or pseudo code for the process. Thus, DFDs provide a robust and formal way to understand, define and document a complex network of otherwise fragmented processes. The DFD can help to visually represent the transformation process and the data dictionary can help to document the details at data element levels.

BUILD THE DATA WAREHOUSE

Before you build the actual database it is useful to have an estimate of the size of objects in the warehouse. This allows you to allocate sufficient space to those objects when they are created. In Designer, when you define the tables, you can also specify the estimates on number of rows, initial size and growth predictions. Designer has reports that can help you in estimating the table and index sizes based on these estimates. These can help you to decide the storage parameters for the tables and indexes. Designer can generate the SQL to create the tables, views, snapshots, indexes and sequence based on the definitions stored in the repository. You can also define tablespaces, data files, and physical storage characteristics of objects in Designer and use its server generator to generate the SQL based on these definitions.

Because Designer can generate all the necessary SQL based on the information stored in its repository the need to maintain SQL scripts in the file system is eliminated. This process is shown in Figure 8.

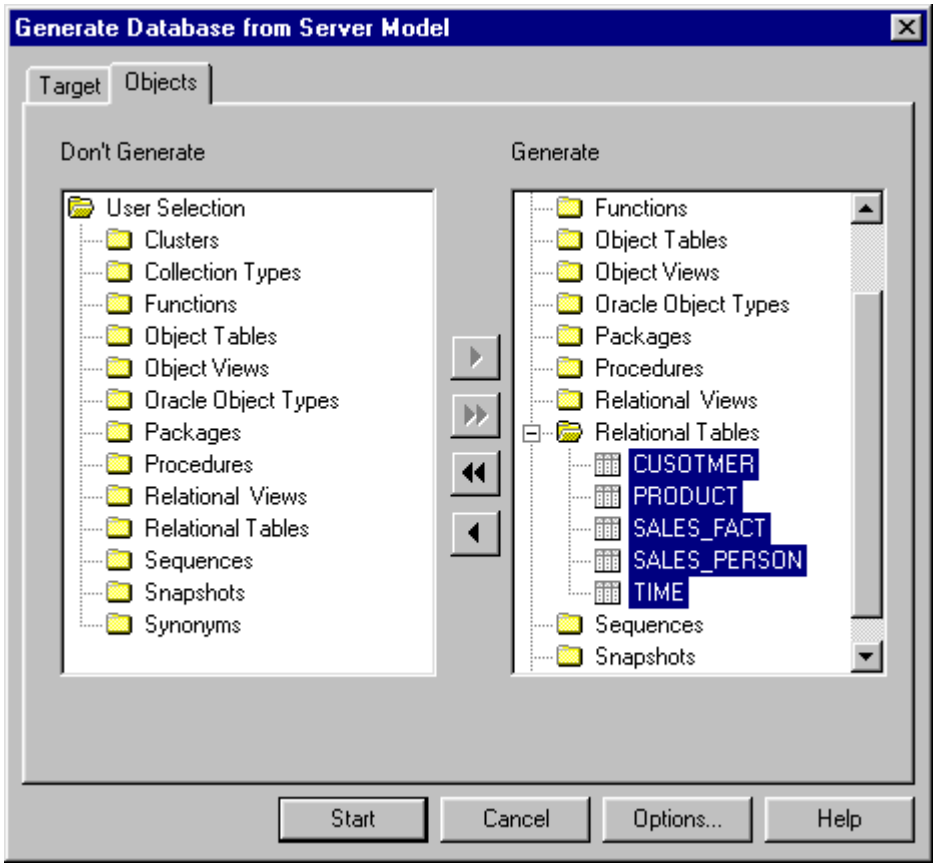

Figure 8

CLEAN, TRANSFORM AND LOAD DATA

At this stage it is necessary to create the extraction and transformation routines needed to populate the data warehouse. Some routines might be PL/SQL routines and can be created as stored programs; others might be 3GL or even assembly code routines. Each routine can be defined as a PL/SQL definition in Designer. Since Designer supports the notion of user-defined module languages, all of these details can be captured in the tool.

If you have documented many of these in detail in the DFD you need not capture this information. However, if you haven't done it, modules are a good way to capture these details. For PL/SQL routines you can use the Logic Editor to actually build these PL/SQL modules. Logic Editor provides a graphical environment for developing stored programs (server-side PL/SQL procedures, functions and packages) as well as database triggers. It provides capabilities for entering, editing, importing, exporting and implementing code. All the code entered via the logic editor is stored in the repository. Its features also include drag and drop editing, automatic text formatting and syntax checking. The syntax checking is performed against the definitions of database objects in the repository, so it is possible to validate the stored programs even before the tables are actually created in the database. The server generator of Designer can generate the necessary scripts to create these stored programs in the database. This is shown in Figure 9.

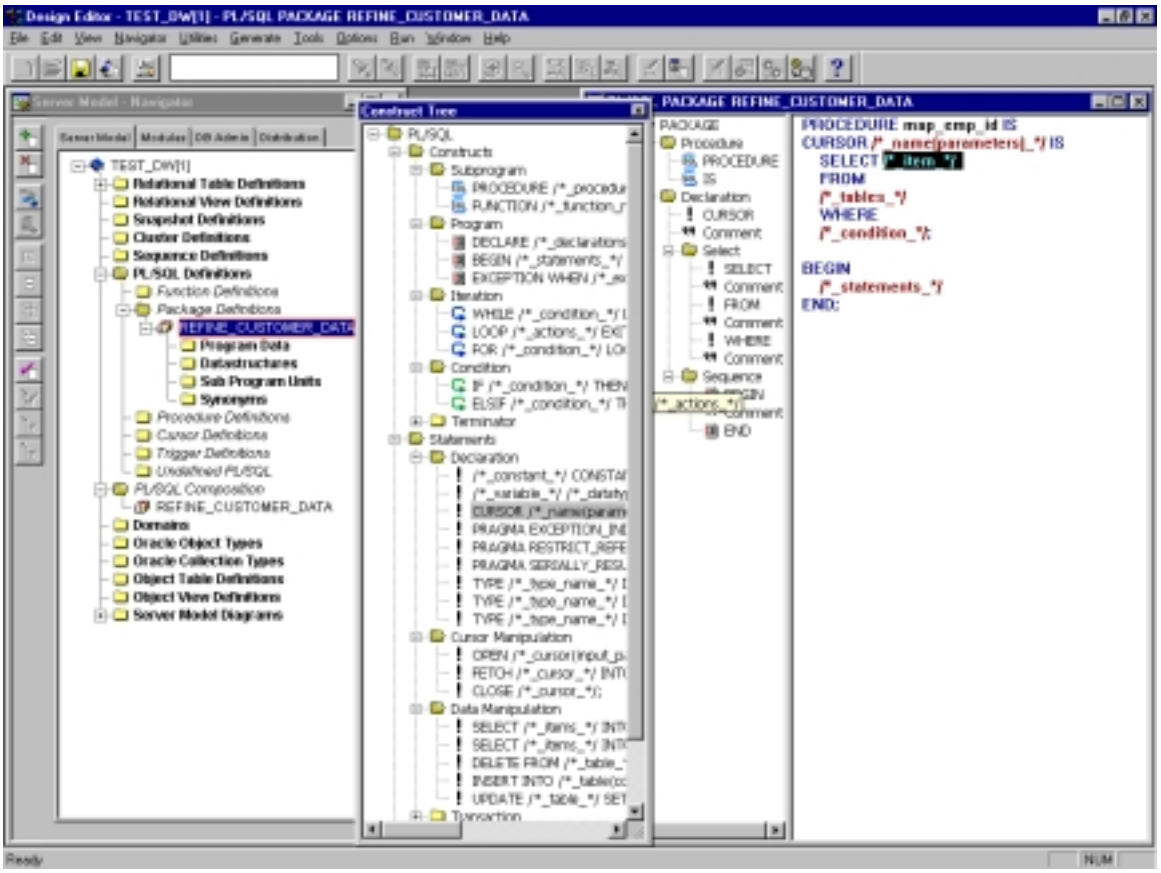

Figure 9

Designer has another useful utility called Matrix Diagrammer that can show – using a spread sheet – how each module interacts with the data sources and the data warehouse, showing module to column associations. This will provide invaluable impact analysis reports if the warehouse or source data changes.

TRANSPORTABLE TABLESPACES

Many times it is necessary to move large amounts of data from one Oracle database to another. For example, you may need to transfer data from an operational database to the data warehouse or transfer data from an enterprise warehouse to a departmental data mart. Traditionally, this transfer is accomplished by exporting data from the source database and importing it into the target database. Oracle8i provides a new feature called **transportable tablespaces** that allows a tablespace to be moved from one Oracle database to another. The complete tablespace can be moved from one Oracle database to another, provided both are running on the same operating system platform. No unloading or loading of data is required. The process is quick and efficient and can save substantial time and resource when the amount of data is huge.

MAINTENANCE

Once the first iteration of the data warehouse is complete and the users start using it, it is usually necessary to get ready for the next iteration. Since the data warehouse must cater to the needs of rapidly changing business, the database structure is likely to change frequently. In the case tables, you can simply modify the definitions in the Designer repository and use the server generator to generate the SQL to make the necessary modifications to the database. Similarly, you can define new tablespaces, data files etc. and have Designer make the changes to the database.

CONCLUSION

To implement a data warehouse successfully, organizations need to exploit the information already available in their operational systems. Oracle Designer provides a robust repository based approach to document the requirements and specifications for the data warehouse. It has some excellent modeling tools to support this. Its server generator can generate all the SQL scripts necessary to build the database for the warehouse. Oracle8i provides an excellent database environment to support a scalable data warehouse. We can conclude that when used effectively Oracle8i and Oracle Designer form an unbeatable combination for data warehousing.