

Leveraging Business Intelligence Tools with the OLAP option to the Oracle10g Database

*Bud Endress, Oracle Corporation
Hemant Verma, Oracle Corporation*

EXECUTIVE SUMMARY

Many organizations utilize both relational and multidimensional technologies to form an overall business intelligence infrastructure. The relational solution typically includes a relational database, a data warehouse and several different query and reporting tools. The multidimensional solution typically includes a stand-alone multidimensional database and query and reporting tools that are used with that particular multidimensional database.

While functional, there are significant issues with the deployment and maintenance of separate relational and multidimensional systems. Some costs are obvious – two databases must be licensed, two servers might be needed, additional query tools are required and there are additional administrative costs. Other costs are less obvious and more damaging to the decision making and planning processes of an organization. These costs result from information fragmentation.

Deploying and managing two parallel systems is clearly expensive. Each system requires software and hardware, and the attention of a database administrator. The problem is not limited to the databases since different sets of query and reporting tools are usually required for each system. These must also be licensed and supported.

It is certainly possible for an organization to spend enough money to deploy parallel data warehouse and OLAP systems. Spending money cannot solve problems related to information fragmentation. Where there are two systems there will be two copies of the data and business rules. Data can become unsynchronized and business rules can be defined differently in each system. To make matters worse, business rules are often defined in each tool rather than each database. The result is inconsistent interpretation of the data across each tool in each system.

One of the most important trends in the practice of business intelligence is the consolidation of data from multiple, disconnected data warehouses to - in the ideal - a single data warehouse that can offer a complete view of the business. The reason for this trend is very simple: it is nearly impossible to obtain a global view of your business if data are highly fragmented across multiple data warehouses.

The fragmentation of data across relational and multidimensional systems is equally as large of a problem as the fragmentation of data across multiple data warehouses. In many ways it is even more of a problem because not only are the data fragmented, but differences in analytic capabilities encourage variations in business rules across each system. Geographic, subject matter and technological fragmentation can compound upon each other and result in complete disintegration of the business intelligence process.

The OLAP option to the Oracle9i Release 2 Database was designed to eliminate the need to implement two separate databases, one relational and the other multidimensional. Oracle9i Release 2 included a full-featured multidimensional engine and true multidimensional data types in the Oracle Database kernel. As such, it was the first (and is still the only) relational-multidimensional database. The result was the ability to join relational and multidimensional data types to form a single, analytically complete view of your business. The trend continues with the OLAP option to the Oracle10g Database.

The remainder of this paper reviews the architecture and capabilities of the OLAP option to the Oracle10g Database and illustrates the application of this technology through the use of both relational and multidimensional business intelligence tools. It will show how the Oracle database can reduce or eliminate the need to replicate data across relational and multidimensional data types, how business rules can be defined in a single location in the database and how the data and business rules can service multiple types of business intelligence applications.

INTRODUCTION

There are two sections to this paper. This first reviews the architecture and capabilities of the OLAP option to the Oracle10g Database. It will focus on the multidimensional engine and data types, and how applications can with them. The first section will also discuss the problem information fragmentation in more detail.

The second part of the paper demonstrates how several different types of tools commonly used in a business intelligence system can use a single database instance containing a single copy of the data and the business rules.

REVIEW OF THE OLAP OPTION TO THE ORACLE 10G DATABASE

In the context of this discussion, the best way to describe the Oracle 10g Database is as a relational-multidimensional database. The Oracle 10g Database includes both relational and multidimensional (OLAP) capabilities in a single database instance. Oracle 10g is not a compromise between the two technologies - it is both a full featured relational database and a full featured multidimensional database. Perhaps most importantly, the Oracle 10g Database allows both relational and multidimensional data types work together. For example, they can to be joined in a single query using either an OLAP API or SQL.

CAPABILITIES OF THE OLAP OPTION

The OLAP option to the Oracle 10g database is a full-featured multidimensional engine within the context of the Oracle Database. It features:

- An industrial strength multidimensional calculation engine.
- Multidimensional data types.
- An OLAP API.
- A SQL interface.
- The OLAP catalog.

The multidimensional calculation engine provides support for a wide variety of multidimensional calculations and planning functions. Multidimensional queries are characterized as being calculations that cross multiple dimensions within a single query. For example, a query such as "what is the change in net profit resulting from the top 20% of customers for year to date this year as compared with a similar period last year for a grouping of my top 10 brands" crosses three dimensions (product, customer and time). While this might be very difficult to express in SQL, it would be simple using the OLAP options OLAP DML (a dimensionally aware data manipulation language).

Planning functions include statistical forecasts, models, allocations and projections or 'what-if' scenarios. The OLAP option includes an impressive library of planning functions and supports what-if scenarios through a read-repeatable, session isolated transaction model.

The OLAP option provides true array-based multidimensional data types within the Oracle database. Some data types are used for data storage – for example, dimension lists, relations and variables. Others, such as models, formulas and aggregation maps are used for persisting calculations and business rules. Calculations are defined using the OLAP DML, a dimensionally aware data manipulation language supported by the multidimensional engine. The OLAP DML is a high level procedural language that is accessible to DBAs and developers alike.

It is important to note that these multidimensional data types are stored in Oracle data files (in contrast to stand alone multidimensional databases that store multidimensional data in separate data files). Because multidimensional data types are stored in Oracle data files, they are administered along with all other Oracle data (for example, backup and restore).

The OLAP API is a Java, object oriented API that provides a multidimensional object model, metadata discovery and support for multidimensional data selection, navigation and calculations. The OLAP option supports both relational and multidimensional data types as data sources. As such, it supports both relational OLAP (ROLAP) and multidimensional OLAP (MOLAP).

The OLAP option's SQL interface provides SQL access to multidimensional data types. This allows SQL based applications such as report generators and ad-hoc query tools to access data and calculations managed by OLAP

option.

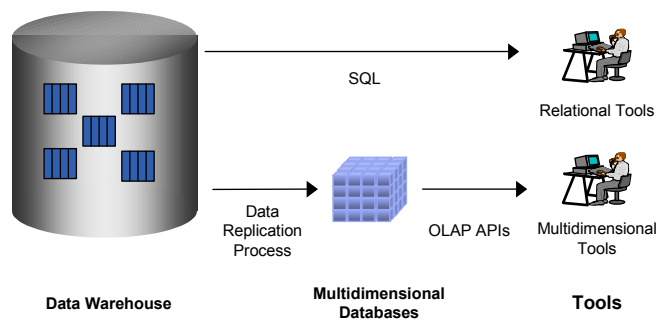
Finally, the OLAP catalog is the companion to the Oracle data dictionary. It includes a repository and API that supports the definition and discovery of logical multidimensional models and mappings to physical data sources.

OLAP AS PART OF THE DATA WAREHOUSE

Multidimensional databases have almost always been views as being external to the data warehouse. In his widely read 1996 book The Data Warehousing Toolkit¹, Ralph Kimball, an early data warehousing expert, described OLAP vendors as selling "proprietary, nonrelational decision support products that compete as replacements for relational databases and SQL-based front end tools". Mr. Kimball's basis for this opinion was quite reasonable. "At the time of this writing, these are the most serious issues for an IS department relative to OLAP products

- The OLAP products are not open. That is, they do not process standard SQL.
- The OLAP products do not scale to enterprise-sized data warehouses. These products cannot store and query the equivalent of a billion-row fact table"

Largely because of these problems, but also because multidimensional databases were less mature in terms of high availability, security and other important areas of functionality, multidimensional databases have never really been accepted as part of the core data warehouse. After all, if they didn't scale to warehouse sized data sets and they could not support the business intelligence tools typically used to access the warehouse, it is difficult to consider them part of the data warehouse.



Typical Environment With Both Data Warehouse and Stand Alone OLAP

To various degrees, vendors have solved scalability problems with multidimensional databases and it is not uncommon to see vary large multidimensional data sets. Until Oracle9i Release 2, when the OLAP option's multidimensional engine and data types were introduced to the Oracle Database, no vendor had solved the problems related to high-availability, security and openness to SQL based applications.

With the OLAP option to the Oracle Database, it is now possible to consider multidimensional OLAP data as being part of the data warehouse. The following points support this proposition:

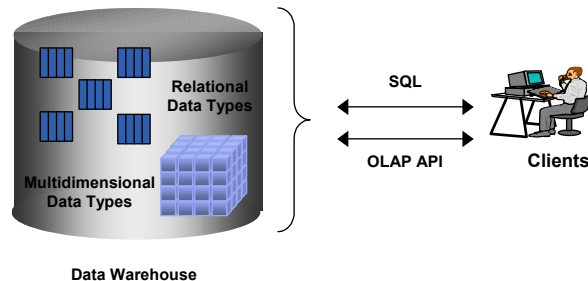
- The OLAP option supports SQL access to multidimensional data and calculations.
- The OLAP option supports very large multidimensional data sets. Data sets in excess of a billion rows are now common.
- Multidimensional data sets stored in the Oracle Database enjoy the same high availability and security features as all other data in the Database. It is now possible to trust the enterprise's mission critical data to multidimensional data sets.

With the Oracle Database, you can now consider a data warehouse environment that includes both relational and multidimensional data types. Rather than building a relational data warehouse and a separate multidimensional

¹ The Data Warehouse Toolkit, Ralph Kimball, 1996, John Wiley & Sons, Inc., New York

database for OLAP, you can simply choose between relational and multidimensional data types within the Oracle Database.

With the Oracle Database, the choice of data types is simply based on the characteristics of the data and calculations requirements. Two systems can be combined as one with the resulting improvements to the end user experience, the elimination of data and analytic fragmentation and lower costs.



Data warehouse with both relational and multidimensional data types

HOW APPLICATIONS QUERY OLAP DATA

Understanding the interfaces provided by the OLAP option – SQL and the OLAP API – is key to understanding how a wide variety of business intelligence tools and applications can be used with the OLAP option.

SQL INTERFACE TO MULTIDIMENSIONAL DATA TYPES

The SQL interface to multidimensional data types uses the Oracle Database's object technology to represent multidimensional data types to the Database's relational engine. The `OLAP_TABLE` table function, an implementation of the Database's object technology, is used to (a) transform `SELECT` statements from SQL to the OLAP option's OLAP DML and (b) transform multidimensional data sets being returned by the multidimensional engine to the format of a relational row set.

The process of querying multidimensional data sets using the `OLAP_TABLE` table function can be made completely transparent to the client application. Or, the application can be aware that the source of the data is multidimensional and leverage this knowledge by interacting with the multidimensional engine from within a `SELECT` statement or through PL/SQL.

The process of enabling SQL access to multidimensional data types is relatively simple – it is only necessary to define two abstract data types. A one abstract data type identifies the structure of the row set; the other indicates that the data source can be queried if it is a table. Optionally, a relational view can be defined over the multidimensional data set to make the multidimensional source of the data completely transparent to a SQL application.

In order to understand the SQL interface, it is useful to view a simple implementation. The following illustrates a common implementation where a multidimensional data set is revealed to an application as a star schema. Two views, one for a dimension view and other for a fact view, are shown.

DEFINING ACCESS TO A TIME DIMENSION

Anytime SQL will be used to access multidimensional data types two abstract data types – one as an `OBJECT` and the other as a `TABLE` – must be created. The `OBJECT` abstract data types defines the columns and data types. For example:

```
CREATE TYPE time_type_row AS object (
    month          varchar(5) ,
    quarter       varchar(5) ,
    year          varchar(5)
);
```

The corresponding `TABLE` abstract data type follows:

```
create type time_type_table as table of time_type_row;
```

While the creation of these two abstract data types are the only requirements for setting up access, it is common to create views to make the multidimensional data types transparent to the application. The following example creates a view for a time dimension lookup table.

```
CREATE OR REPLACE VIEW time_view AS
SELECT *
  FROM TABLE(OLAP_TABLE('global DURATION SESSION',
    'time_type_table',
    'limit time KEEP time_levelrel ''MONTH''',
    'DIMENSION time WITH
      HIERARCHY time_parentrel
        LEVELREL year, quarter, month
      FROM time_familyrel USING time_levellist'));
```

Note that the FROM clause selects from the OLAP_TABLE table function rather than a table or view. The OLAP_TABLE table function accepts a number of arguments that map it to the analytic workspace and objects within it. These arguments are briefly described below:

- GLOBAL is the user-defined name of the analytic workspace.
- DURATION describes the transaction model (either read repeatable view of the analytic workspace during the session, or a view that immediately recognizes any changes to the data in the analytic workspace).
- TIME_TYPE_TABLE binds the view to the abstract data type that was created above.
- 'limit time KEEP time_levelrel "MONTH"' is an OLAP DML command that is executed as part of the SELECT statement. (This command constrains time dimension in the analytic workspace to members at the month level. This is done so that only months are returned at rows. Higher-level members – quarters and years – are returned as columns.)
- The DIMENSION clause maps relational columns (described in the abstract data type) to multidimensional data types in the analytic workspace. The example shown illustrates some hierarchical structures (TIME_FAMILYREL and TIME_LEVELLIST) in the analytic workspace being used to select quarter and year members as columns in the view).

DEFINING A ACCESS TO FACTS

The process of defining access to fact data is very similar to that for the time dimension data. Again, two abstract data types are required:

```
create type sales_type_row as object (
  time_id          varchar2(5) ,
  channel_id       varchar2(5) ,
  product_id       varchar2(5) ,
  customer_id      varchar2(5) ,
  sales            number,
  units            number,
  extended_cost    number,
  forecast_sales   number,
  olap_calc        raw(32)
);
```

```
create type sales_type_table as table of sales_type_row;
```

In this example, the '_id' columns will represent the keys. The columns SALES, UNITS, EXTENDED_COST and FORECAST_SALES are the facts or measures. The OLAP_CALC column is very special – it is used to include

OLAP DML expressions in the select list (example to follow).

The fact view can be created as follows:

```
create or replace view sales_view as
  select *
  from table(OLAP_TABLE('global DURATION session',
    'sales_type_table',
    '',
    'DIMENSION time_id FROM time
    DIMENSION channel_id FROM channel
    DIMENSION product_id FROM product
    DIMENSION customer_id FROM customer
    MEASURE sales FROM sales
    MEASURE units FROM units
    MEASURE extended_cost FROM extended_cost
    MEASURE forecast_sales FROM fcast_sales
    ROW2CELL olap_calc'));
```

The DIMENSION keyword maps dimensions in the analytic workspace to columns that act as keys in the fact view. Measures, which can be any data in the analytic workspace that is dimensioned by at least one of the dimensions in the view, are mapped using the MEASURE keywords. ROW2CELL indicates that an OLAP DML expression can be used in the select list when selecting from this view.

SELECTING DIRECTLY FROM OLAP_TABLE

It is not necessary to create a view in order to select from an analytic workspace – applications can select directly from OLAP_TABLE (it is necessary to create the abstract data types before selecting from OLAP_TABLE). The following example illustrates a select statement that selects directly from OLAP_TABLE:

```
select  time_id,
        channel_id,
        product_id,
        customer_id,
        sales
  from table(OLAP_TABLE('global DURATION session',
    'sales_type_table',
    '',
    'DIMENSION time_id FROM time
    DIMENSION channel_id FROM channel
    DIMENSION product_id FROM product
    DIMENSION customer_id FROM customer
    MEASURE sales FROM sales'))
 where time_id = '2003'
    and channel_id = 'CATALOG'
    and product_id in ('GUNS', 'LIPSTICK')
    and customer_id = 'TEXAS';
```

USING OLAP DML EXPRESSIONS IN SQL BASED APPLICATIONS

The OLAP DML is used to define calculations in the analytic workspace. OLAP DML is a procedural language that can be used to:

- Define new objects in the analytic workspace and assign data to them.
- Define and execute ETL processes in the analytic workspaces (e.g., define data loading operations, aggregations, forecasts, allocations and other calculations).
- Define runtime calculations.

There are three ways in which OLAP DML can be used in a SQL application:

1. As an OLAP DML command argument to OLAP_TABLE. OLAP DML command arguments are often used to apply additional predicates or to perform some sort of calculations as part of the select statement, for example running a forecast.
2. As an OLAP expression included in the select list using the OLAP_EXPRESSION function. This can be to either select from an object (for example, a formula or variable) that is not listed in the OBJECT abstract data type or to define a new calculation that persists only for the duration of the query. Common examples would include functions for time series such as leads and lags, comparisons with prior periods and market shares.
3. As an OLAP DML command issued through PL/SQL. This allows the application to interact directly with the multidimensional engine outside the context of a select statement. There are many uses for this method including data loading, defining calculations that are persisted in the analytic workspace, updating and for applying predicates using the OLAP DML prior to selecting data with SQL.

USING AN OLAP DML COMMAND IN OLAP_TABLE

The example below illustrates the use of an OLAP DML command as an argument to OLAP_TABLE. This example calls a stored procedure that executes a statistical forecast. Because the OLAP DML command is executed after the application of the where clause, the forecast is run only for those products, customers and channels in the where clause.

```
select  time_id,
        channel_id,
        product_id,
        customer_id,
        sales
from table(OLAP_TABLE('global DURATION session',
                     'sales_type_table',
                     'forecast_sales',
                     'DIMENSION time_id FROM time
                     DIMENSION channel_id FROM channel
                     DIMENSION product_id FROM product
                     DIMENSION customer_id FROM customer
                     MEASURE sales FROM sales'))
where time_id = '2003'
      and channel_id = 'CATALOG'
      and product_id in ('GUNS', 'LIPSTICK')
      and customer_id = 'TEXAS';
```

Alternatively, a view could be created that includes the OLAP DML command. The following view and select statement would yield the same results as the preceding example.

```
create or replace view sales_view as
select *
from table(OLAP_TABLE('global DURATION session',
                     'sales_type_table',
                     'forecast_sales',
                     'DIMENSION time_id FROM time
                     DIMENSION channel_id FROM channel
                     DIMENSION product_id FROM product
                     DIMENSION customer_id FROM customer
                     MEASURE sales FROM sales
                     MEASURE units FROM units
                     MEASURE extended_cost FROM extended_cost
                     MEASURE forecast_sales FROM fcast_sales
                     ROW2CELL olap_calc'));

select  time_id,
        channel_id,
        product_id,
```

```

        customer_id,
        sales
from sales_view
where time_id = '2003'
      and channel_id = 'CATALOG'
      and product_id in ('GUNS','LIPSTICK')
      and customer_id = 'TEXAS';

```

Using the OLAP_EXPRESSION Function

When used in the select list, OLAP DML can be used as an expression that returns data for a cell (or row, as it is seen when returned through SQL) as shown in the following example:

```

select product_id,
       time_id,
       sales,
       olap_expression
         (olap_calc,'lagdif(sales,1,time,status)')
         as SALES_CHG_PRIOR_PRIOD,
       olap_expression
         (olap_calc,'sales/sales(product ''1'') * 100')
         as PRODUCT_SHARE
from sales_olap_view
where time_id = '2003'
      and channel_id = 'CATALOG'
      and product_id in ('GUNS','LIPSTICK')
      and customer_id = 'TEXAS';

```

In this example the OLAP expression is passed to the multidimensional engine, evaluated and returned as a column.

USING OLAP DML COMMANDS IN PL/SQL

Sometimes it is useful to be able to issue OLAP DML commands directly to the multidimensional engine prior to the execution of a SELECT statement. Common examples are the application of predicates to constrain the cube and commands that change data (that is, commands that update cells in a variable). The following example does both as part of an application that allows a user to specify assumptions for the effectiveness of a promotional campaign and project the resulting sales.

The example makes the assumption promotional sales will be 115% of the baseline sales projection for the selected periods. First, the cube is constrained to certain time periods, items that are children of the family 'LAPTOP', and certain members of the channel and customer dimensions. Next, the data for PROMOTIONAL_SALES are calculated.

```

execute dbms_aw.execute('limit TIME to ''SEP03'' ''OCT03'' ''NOV03''')
execute dbms_aw.execute('limit PRODUCT to descendants using PRODUCT_PARENT
'LAPTOP')
execute dbms_aw.execute('limit PRODUCT to PRODUCT_LEVEL ''ITEM''')
execute dbms_aw.execute('limit CHANNEL to ''INTERNET''')
execute dbms_aw.execute('limit CUSTOMER to ''AMERICAS''')
execute dbms_aw.execute('PROMOTIONAL_SALES = SALES * 1.15')

```

The next part of the example constrains the cube to the top 20% of customers at the account level based on the projected values of PROMOTIONAL_SALES.

```

limit CUSTOMER to descendants using CUSTOMER_PARENT
'AMERICAS'
limit CUSTOMER keep CUSTOMER_LEVEL 'ACCOUNT'
limit CUSTOMER keep top 20 percent based on
PROMOTIONAL_SALES

```

And then the data can be fetched using a SELECT statement.

```
select time_id,
       product_id,
       channel_id,
       customer_id
       PROMOTIONAL_SALES
From OLAP_SALES_VIEW;
```

Note that a WHERE clause was not required because the predicates were already applied using OLAP DML LIMIT commands.

DENORMALIZED FACT VIEWS

If disk storage and processing power were both infinite, people might design warehouse schema based on ease of query rather than on storage and processing efficiency. The easiest warehouse schema to query might be one that included all data – both dimension members and facts – in a single table. With such a design, hierarchical data such as child-parent-ancestor data, non-hierarchical attributes and text descriptors could be embedded within the fact view. Joins would be rarely needed. Hierarchical selections would be relatively simple.

In a relational data warehouse, such a design would be prohibitively expensive in terms of storage and processing. Although the physical model of the analytic workspace is fully normalized and thus a very efficient means of storing dimensional data, the OLAP option can very efficiently present the data in a normalized fact view. (The multidimensional engine features automatic referential integrity and transparent joins – including outer joins – that facilitate both efficient storage and denormalized views of the data).

An example of a normalized fact view follows. Note that the view contains dimension members (at all levels of summarization), attributes such as text labels, full hierarchical lineage (child-parent-ancestor relationship) and level attributes.

```
CREATE TYPE dnorm_fact_type_row AS object (
  time_id          varchar(5),      /* time member          */
  time_level       varchar(30),     /* time member's level  */
  time_desc        varchar(30),     /* time text label      */
  time_order       number,          /* attribute of time member */
  month            varchar(5),      /* month of the time member */
  quarter          varchar(5),      /* quarter of the time member */
  year             varchar(5),      /* year of the time member */
  customer_id      varchar(5),
  customer_level   varchar(30),
  customer_desc    varchar(30),
  ship_to          varchar(5),
  account          varchar(5),
  market_segment  varchar(5),
  total_market     varchar(5),
  warehouse        varchar(5),
  region           varchar(5),
  all_customers    varchar(5),
  product_level    varchar(30),
  product_desc     varchar(30),
  product_id       varchar(5),
  item             varchar(5),
  family           varchar(5),
  class            varchar(5),
  total_product    varchar(5),
  channel_id       varchar(5),
  channel_level    varchar(30),
  channel_desc     varchar(30),
  channel          varchar(5),
  all_channels     varchar(5),
```

```

sales                number,          /* sales fact    */
units                number,
extended_cost        number,
forecast_sales        number,
olap_calc             raw(32));

create type dnorm_fact_type_table as table of dnorm_fact_type_row;

create or replace view dnorm_fact_view as
select *
  from table(OLAP_TABLE('global DURATION session',
    'dnorm_fact_type_table',
    '',
    'DIMENSION time_id FROM time WITH
      HIERARCHY time_parentrel
        LEVELREL year, quarter, month
        FROM time_familyrel USING time_levellist
      ATTRIBUTE time_level FROM time_levelrel
    DIMENSION customer_id from customer WITH
      HIERARCHY customer_parentrel
        (customer_hierlist 'MARKET_ROLLUP')
        LEVELREL null,null,null,total_market,
          market_segment,account,null
        FROM customer_familyrel USING customer_levellist
      HIERARCHY customer_parentrel
        (customer_hierlist 'SHIPMENTS_ROLLUP')
        LEVELREL all_customers,region,warehouse,
          null,null,null,ship_to
        FROM customer_familyrel USING customer_levellist
      ATTRIBUTE customer_level FROM customer_levelrel
    DIMENSION product_id FROM product WITH
      HIERARCHY product_parentrel
        LEVELREL total_product, class, family, item
        FROM product_familyrel USING product_levellist
      ATTRIBUTE product_level FROM product_levelrel
    DIMENSION channel_id FROM channel WITH
      HIERARCHY channel_parentrel
        LEVELREL all_channels, channel
        FROM channel_familyrel USING channel_levellist
      ATTRIBUTE channel_level FROM channel_levelrel
    MEASURE time_desc FROM time_long_description
    MEASURE time_order FROM time_order
    MEASURE channel_desc FROM channel_long_description
    MEASURE customer_desc FROM customer_long_description
    MEASURE product_desc FROM product_long_description
    MEASURE sales FROM sales
    MEASURE units FROM units
    MEASURE extended_cost FROM extended_cost
    MEASURE forecast_sales FROM FCAST_SALES
    ROW2CELL olap_calc'
  ));

```

While this view might look complex, it's really not. Note that there are only three basic parts to the view: (1) the binding to the analytic workspace and abstract data type, (2) the dimension clauses, and (3) the measure clauses. Each part follows the same repeating pattern in each view. Once the basic construction is understood, creating them is relatively easy.

Querying such a view is very simple because joins are not needed and hierarchical attributes are provided. In the following example, applying predicates to level attribute columns specifies level selections.

```
select time_desc,
```

```

        product_desc,
        customer_desc,
        channel_desc,
        sales,
        forecast_sales
    from dnorm_fact_view
where time_level = 'YEAR'
        and product_level = 'TOTAL_PRODUCT'
        and channel_level = 'ALL_CHANNELS'
        and customer_level= 'ALL_CUSTOMERS'
    order by time_order;

```

In the next example, we see a select statement that selects the children of your '2003':

```

select time_desc,
        customer_desc,
        sales,
        forecast_sales
    from dnorm_fact_view
where year = '2003'
        and time_level = 'QUARTER'
        and product_level = 'TOTAL_PRODUCT'
        and channel_level = 'ALL_CHANNELS'
        and customer_level= 'TOTAL_MARKET'
    order by time_order;

```

'SOLVED CUBES'

Many of the preceding examples have shown how you can access the multidimensional engine to define calculations at runtime through SELECT statements. In most cases, however, many or most of the calculation rules are predefined and persisted in the analytic workspace. Examples include aggregations and allocations, models, forecasts and formulas.

To the application, the cube appears to be fully solved (materialized) regardless of whether data is calculated and stored or if it is calculated dynamically at runtime. This is because the multidimensional engine automatically calculates data points in the analytic workspace based on the predefined calculation rules. For example, all summary data are automatically calculated based on the rules in an aggregation map object and hierarchies. As a result, the application does not need to express aggregation rules using a GROUP BY.

Consider the following view. Note neither the rules for aggregations nor the product share measure are expressed in the view. Instead, the calculation rules for aggregations are persisted in the analytic workspaces in an aggregation map and product share is persisted as a formula.

```

create type sales_type_row as object (
    time_id          varchar2(5),
    channel_id       varchar2(5),
    product_id       varchar2(5),
    customer_id      varchar2(5),
    sales            number,
    units            number,
    product_share    number
);

create type sales_type_table as table of sales_type_row;

```

The fact view can be created as follows:

```

create or replace view sales_view as
select *
from table(OLAP_TABLE('global DURATION session',

```

```

        'sales_type_table',
        '',
'DIMENSION time_id FROM time
DIMENSION channel_id FROM channel
DIMENSION product_id FROM product
DIMENSION customer_id FROM customer
MEASURE sales FROM sales
MEASURE units FROM units
MEASURE product_share FROM product_share')));

```

The PRODUCT_SHARE measure is defined as a formula in the analytic workspace as follows:

```

DEFINE PRODUCT_SHARE FORMULA DECIMAL
  <CHANNEL CUSTOMER PRODUCT TIME>
EQ (sales/sales(product '1')) * 100

```

An application can then issue a select statement such as:

```

select  time_id,
        channel_id,
        product_id,
        customer_id,
        sales
        product_share
  from  sales_view
 where time_id = '2003'
    and channel_id = 'CATALOG'
    and product_id in ('GUNS', 'LIPSTICK')
    and customer_id = 'TEXAS';

```

The ability to present the cube as being fully solved allows applications that have absolutely no knowledge of the OLAP option to easily select all data from an analytic workspace. This is a key feature of the OLAP option since it, along with the SQL interface, allows many non-OLAP applications to be used as a tool to query data managed by the OLAP option.

SUMMARY OF THE SQL INTERFACE

The following points summarize features of the SQL interface:

- Applications can either be aware of the analytic workspace as a data source or the analytic workspace can be made transparent to the SQL application through the use of views.
- Both relational and multidimensional data can be returned within the same query. This allows a SQL based application to use both relational and multidimensional data within the application.
- All calculations that are predefined in the analytic workspace – aggregations, allocations, models, forecasts, formulas, etc. – appear to the application as if they were pre-materialized regardless of whether they are pre-materialized or are calculated dynamically. As a result, it is possible for a SQL based application to query the results of the calculation without needed to express or otherwise understand the calculation rule.
- Applications can interact with the multidimensional engine within the context of a select statement. Applications can issues OLAP DML commands in a select statement and they can include OLAP DML expressions in the select list.
- The ability to present the analytic workspace as a denormalized fact view with all dimensional and fact data can dramatically simplify the SQL that is used to query the analytic workspaces. This can eliminate the need for joins when making hierarchical or attribute based selections.

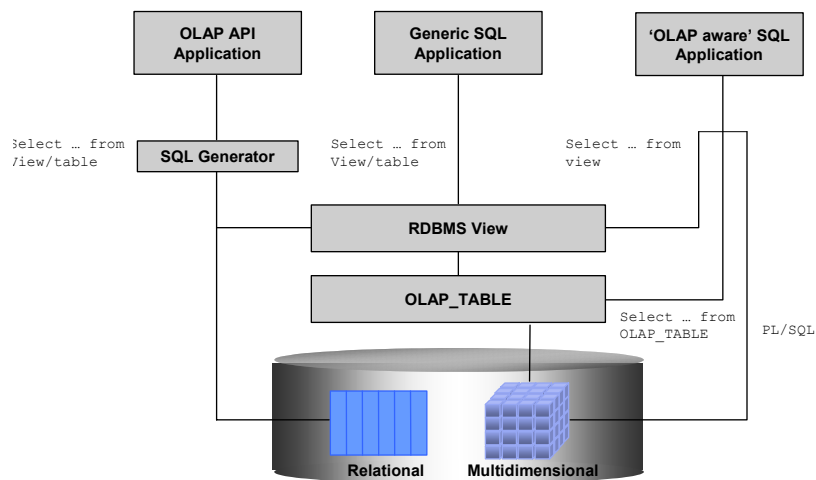
Combined, these features present a system that allows SQL based applications to be completely unaware of the analytic workspace yet still allow them to present the results of complex OLAP calculations. Or, SQL based applications can interact with the multidimensional engine to enhance their ability to support analysis.

THE OLAP API

The OLAP API is a Java, object oriented API that supports querying the database in a dimensional context. It provides an object model, dimensional data selection and navigation capabilities, and support for common OLAP calculations. The OLAP API includes a very sophisticated SQL generator that allows it to support both relational tables and analytic workspaces as data sources.

When used with relational tables, the OLAP option generates SQL directly against the tables; analytic SQL functions are used to perform calculations. When used with multidimensional data types, the OLAP API selects from relational views over multidimensional data types. In this case, the multidimensional engine will typically process calculations. The use of SQL to access both relational and multidimensional data types allows the OLAP API to leverage SQL's ability to join relational and multidimensional data within a single query and present a unified view of the data warehouse in an OLAP context.

The following diagram illustrates both the SQL and OLAP API query paths.



Query paths for the OLAP Option to the Oracle 10g Database

Fragmentation in Business Intelligence

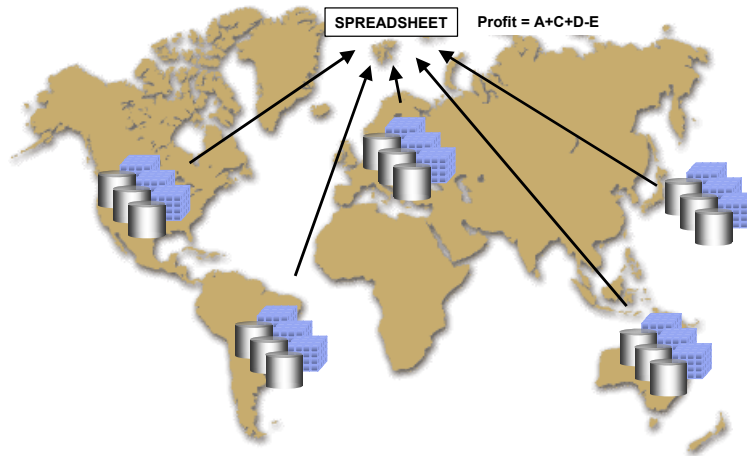
Historically, technology has forced organizations into highly fragmented business intelligence architectures. In the not too distant past, it was common for organizations to have tens or hundreds of data marts, each with subject specific data. Examples might include separate data marts for sales, marketing, human resources and finance subject areas. To make matters worse, there were often subject-specific data marts for each of an organization's geographic region operating areas.

Add separate multidimensional servers to the mix and the problem becomes more severe. If there was one OLAP database derived from each relational data mart the number of separate data repositories doubles. Consider a case where an organization has ten relational data marts, one multidimensional cube derived from each data mart, in each of five regions around the world - there would be 100 separate relational and multidimensional databases. This situation creates several problems:

- It is expensive to develop and maintain many different systems.
- It is extremely difficult to have visibility into the entire organization.
- Users must be trained to use different sets of query tools – one for relational data and other for multidimensional data.

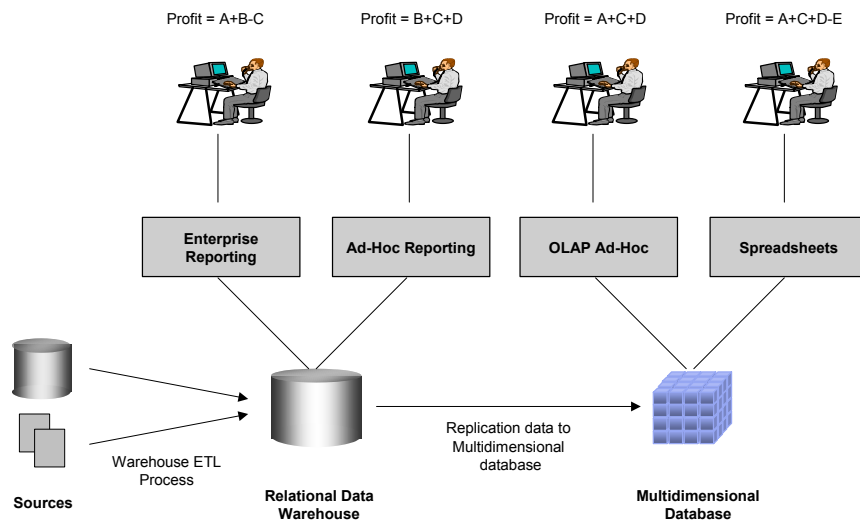
The expense of such an environment is intuitive. More systems result in higher hardware and administrative costs. Two sets of query tools are more expensive than one. It isn't hard to imagine how one system that combines subject

areas and geographies, and converges relational and multidimensional data, might be less expensive to maintain. Less obvious might be the problems with gaining full and consistent visibility into the enterprise. Fragmentation across subject areas and geographies makes it extremely difficult to view top-level results or effectively compare functional areas or geographies. Additional fragmentation across technologies makes the problem more acute.



Geographically and technologically fragmented data in the enterprise

As if fragmentation of the data across different geographic regions and across different database technologies isn't enough of a problem, additional information fragmentation can occur due to the differences in relational and multidimensional query tools. Business rules might be applied differently within the different tools. Or, the users' view of data might be restricted to one database technology due to the data access capabilities of the tool, thereby allowing them to see only part of the picture.



A fragmented business intelligence system allows multiple definitions of a calculation

DEFRAGMENTING THE BUSINESS INTELLIGENCE INFRASTRUCTURE

The Oracle Database is designed to allow you to eliminate information fragmentation within the enterprise. Defragmentation is imperative because a fragmented business intelligence infrastructure prevents the enterprise from

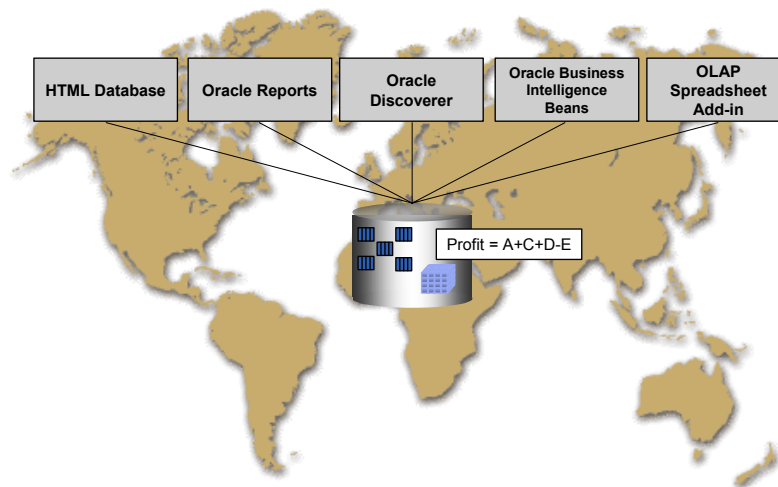
obtaining a clear and consistent view of operations. It is also more costly to maintain.

The Oracle Database has been developed to meet this challenge. First, with Oracle8i, the database became a platform for very large data warehouses. It was then possible to combine all historical data in an enterprise relational data warehouse.

With Oracle9i and Real Application Clusters – a highly scalable and reliable clustering technology – the capacity of the Database, both in terms of data and users, became virtually limitless. It was then possible to implement a single instance of the database in support of business intelligence systems that managed all of an enterprise's data.

While Oracle9i with Real Application Clusters facilitated the elimination of geographic fragmentation in relational data warehouses, the division between relational and multidimensional technologies still existed. With the availability of the OLAP option to the Oracle9i Release 2 Database and the introduction of the multidimensional engine and multidimensional data types to the Database, the technology divide between relational and multidimensional database was eliminated.

The remaining part of the information defragmentation process is at the data access layer – the business intelligence tools. The remainder of this paper will illustrate how leveraging the OLAP option's support for both SQL and OLAP API interfaces to both relational and multidimensional data types can solve this problem. You will see how the Oracle Database is can provide a single view your data with consistent business rules regardless of the data access method you choose.



A single instance of the Oracle Database servicing multiple business intelligence tools with both relational and multidimensional data. This system encourages a common definition of calculations to be shared among different types of applications.

DEMONSTRATION

This section of the paper describes a demonstration of several different types of business intelligence applications and describes how they are implemented with the OLAP option to the Oracle 10g Database.

REQUIREMENTS OF THE SYSTEM

The demonstration is centered on Global Computing Exchange, a fictional supplier of computer hardware, software and supplies. A business intelligence system must service a number of different users within Global Computing Exchange, Global's customers and Global's suppliers.

The system must meet three fundamental requirements:

1. It must present a single, integrated view of the business. Data and business rules must be consistent, regardless of the access method used.
2. It must meet a number of analytic requirements from simple measure calculations to predictive analysis.

3. It must offer a variety of access methods.

In order to serve different types of end users, Global Computing Exchange has chosen to use the following types of tools:

- A simple web based application that presents key performance indicators using HTML pages. This will be used to present snapshots of current period sales, shipments and profitability along with near term projections.
- A reporting system that is used to produce monthly summaries for customers. This system will also offer the customer the opportunity to receive discounts if they accept offers to reorder stock based on sales forecasts generated by the system.
- An ad-hoc reporting system that allow users to analyze data in both relational and multidimensional contexts.
- A statistical forecasting system that allows marketing and shipping departments to predict product demand, costs, pricing, revenue and profits.
- A spreadsheet interface for Global's finance department.

PRODUCTS USED IN THE DEMONSTRATION

This demonstration is designed to highlight both Oracle's business intelligence tools and the OLAP option's calculation capabilities and data access methods. To this end, Oracle tools will be used within the demonstration.

It should be understood that other tools could play the same roles as the Oracle tools. The OLAP option's OLAP API is public and can be used by third parties to develop dimensionally aware applications on the Database. The SQL interface allows other SQL based tools to access Oracle10g's multidimensional data types. These SQL based applications can either be unaware of the multidimensional data types or they can interact with the multidimensional data types and multidimensional engine..

Products used in the demonstration are listed in the following table.

Role	Product	Description
"Lite" query and reporting tool	HTML Database	A SQL based tool used to quickly design and deploy reports and graphs on the web.
Enterprise Reporting	Oracle Reports	An enterprise-reporting tool that can use both SQL and the OLAP API as data sources.
Ad-Hoc Reporting, both relational and multidimensional	The upcoming version of Oracle Discoverer	An add-hoc reporting tool that allows the user to explore the data. Provides interactive report and charts, and query and calculation builders. Works in either relational or multidimensional contexts.
Custom forecasting system	Oracle Business Intelligence Beans	Business Intelligence Beans are OLAP aware application components that can be used to build custom analytic applications based on the OLAP option. Uses the OLAP API.
Spreadsheet	Oracle OLAP Add-in for Microsoft Excel	Allows Microsoft Excel to be used as an OLAP interface to the Oracle Database.

DATABASE IMPLEMENTATION

ORACLE DATABASE

The Oracle10g Enterprise Edition Database is used for both the operational data source and for the data warehouse implementation. The OLAP option to the Oracle10g Database is used for advanced analytic capabilities.

DATA SOURCE

The source of the data is Global's shipments and general ledger systems that feed the following measures to the data warehouse: units shipped, units price and unit cost.

LOGICAL DATA MODEL

To support advanced analysis and to promote ease of use, a dimensional model was implemented. The model includes four dimensions, time, channel, product and customer, and about 40 measures. Each dimension has several levels of summarization within one or more hierarchies. A list of measures follows.

Actual units	Change in profit as compared to prior period	Forecast Units, moving average over three periods
Actual unit cost	Change in units as compared to prior period	Profit, moving average over three periods
Actual unit price	Change in sales as compared to prior period	Sales, moving average over three periods
Actual sales	Percent in cost as compared to prior period	Units, moving average over three periods
Actual extended cost	Percent change in profit as compared to prior period	Forecasted cost for the next period
Forecast of extended cost	Percent changes in units as compared to prior period	Forecasted profit for the next period
Forecast of units	Percent change in sales as compared to prior period	Forecasted sales for the next period
Forecast of unit price	Cost, moving average over three periods	Forecasted units for the next period
Forecast of unit cost	Forecast Cost, moving average over three periods	Share of sales for current channel as compared to All Channels
Forecast of sales	Forecast Profit, moving average over three periods	Share of sales for current customer as compared to All Customers
Actual Profit	Forecast Sales, moving average over three periods	Share of sales for current product as compared to All Products
Forecast Profit		
Margin, as percent of sales		
Cost for prior period		
Units for prior period		
Sales for prior period		
Profit for prior period		
Change in cost as compared to prior period		

PHYSICAL MODEL

Because Global Computing Exchange chose to use a dimensional model and there are significant analytic requirements, the physical model was implemented using multidimensional data types managed by the OLAP option to the Oracle10g database.

Since all of the measures were related to the same subject area (Global's sales data) and they shared logical dimensions, it was decided to include the entire data model in a single analytic workspace. An analytic workspace being a container in the Database for a collection of related multidimensional data types.

The analytic workspace was constructed according to the *database standard form* specification. This is the physical design of the analytic workspace that contains data structures and metadata that is understood by Oracle administrative and end user tools. Oracle administrative tools build analytic workspaces according to the database standard form specification.

Because not all of the tools used in the solution are OLAP aware – some are general purpose SQL based tools – it was decided to persist the definitions for all calculation in the analytic workspace. This not only made it easy to query the database with SQL based tools, but it also ensured that the calculation rules would be used consistently across all of the applications.

The measures could be categorized into three groups:

1. The source measures: units sold, unit price and unit cost. These were loaded into multidimensional variables and

a partial aggregation strategy was used to pre-materialize some of the data at summary levels to optimize query performance. Areas of the cube that were not pre-materialized are automatically calculated at runtime.

2. Some measures could be calculated from other measures only at leaf (detail) levels of the model (for example extended cost and forecast measures). These measures were stored in multidimensional variables and a partial aggregation strategy was used to pre-materialize some of the data at summary levels. Areas of the cube that were not pre-materialized are automatically calculated at runtime.
3. Measures that are always calculated at runtime using formulas (for example, the time series, profit and market share measures).

ETL PROCESS

Oracle Warehouse Builder was used to build the analytic workspace. Oracle Warehouse Builder can be used to map to source operational systems, define the logical model and deploy the model to an analytic workspace (as well as to traditional relational data warehouses).

Oracle Warehouse Builder uses the API provided by the AW_CREATE package, which is a component of the OLAP option, to create and manage the analytic workspace. Alternatively, either SQL scripts with AW_CREATE calls or the Analytic Workspace Manager GUI (which is also layered on the AW_CREATE package) could be used to build and manage the analytic workspace.

Because the SQL interface to multidimensional data types makes multidimensional data types accessible to SQL based applications, it was not necessary to replicate data in relational tables.

ORACLE HTML DATABASE DESCRIBED

Oracle10i HTML Database is representative of simple, easy to use report builders that can be used by developers of web pages. In this demonstration, it is used to build:

1. Database-centric interactive web pages.
2. A simple user interface.
3. Reports and charts using data sourced from an analytic workspace.

Specifically, Global Computing used HTML database to provide web-based access to key performance indicators such as profitability, sales, units shipped, trending and period-to-period comparisons and forecasts. It provides support for some interactive reporting, but it is not meant to be a fully interactive reporting system.

The documents generated by HTML database were designed to be very easy for the end user to manipulate – they require only limited knowledge of the logical data model and no knowledge of the physical model.

GLOBAL'S REPORTING IN HTML DATABASE

The types of reports made available using HTML database include:

- Channel performance snapshot – sales, % change sales, units, % change units, cost, % change cost, profit, % change profit, profit margin and next month forecasts for sales, units, cost and profit by distribution channel for the current month.
- Product performance snapshot – sales, units, cost, profit, profit margin and next month forecast by family for the current month.
- Region performance snapshot – sales, units, cost, profit, profit margin and next month forecast by Region for the current month.
- Top 10 customers based on sales, year to date 2003.
- Top 10 customers based on sales, current period.
- Top 10 customers based on profit, year to date 2003.
- Top 10 customers based on profit, current period.
- Top 10 customers based on profit, year to date 2003.

- Top 10 customers based on percent change sales compared to the prior year.
- Top 10 customers based on percent change in profit compared to the prior year.
- Product family margin report – product families ranked by margin.
- Item margin report – items ranked by margin.
- Sales, cost and profit trends and forecast, last 24 months and next 12 months (report and graph)
- Margin trend (report and graph), last 24 months and next 12 months.
- Three month moving averages of sales, units, cost and profit.
- Product share based on profit for YTD 2003
- Channel trend and forecast- actual and forecast sales.

Examples of reports created using HTML DB follow.

Customer Analysis | **Product Analysis** | **Channel Analysis** | **Sales Analysis**

Reports

- Region Performance Snapshot
- Top 10, Sales, YTD
- Top 10, Sales, CP
- Top 10, Profit, YTD
- Top 10, Profit, CP
- Highest Sales Share
- Highest Forecast Sales
- Top 10, Sales %Change
- Customer Sales

Region Performance Snapshot

Region	Sales	Units	Cost	Profit	% Profit Margin	Forecast Sales
North America	\$5,814,511.91	27,254	\$15,434.76	\$459,828.77	.079	\$5,816,688.97
Asia Pacific	\$4,254,783.96	11,747	\$15,434.76	\$253,265.04	.06	\$4,285,903.44
Europe	\$1,303,940.35	7,988	\$15,434.76	\$116,102.83	.089	\$1,289,552.32

row(s) 1 - 3 of 3

Region Performance Snapshot report

SQL for this report follows

```

SELECT customer_desc,
       sales,
       units,
       unit_cost,
       profit,
       ROUND(olap_expression(olap_calc, 'PCT_MARGIN'), 3)
         as Profit_Margin,
       OLAP_EXPRESSION (olap_calc, 'FCAST_SALES_NEXT_PERIOD')
         as Forecast_next_preiod
FROM   dnorm_fact_view_all
WHERE  time_desc = TO_CHAR(ADD_MONTHS(SYSDATE, -3), 'Mon-YY')
       AND product_level = 'TOTAL_PRODUCT'
       AND channel_level = 'ALL_CHANNELS'
       AND customer_level= 'REGION'
ORDER BY sales DESC

```

Note that in the HTML DB examples, OLAP_EXPRESSION is often used to select measures from the analytic workspace. These measures could have been included in the view over the analytic workspace and selected as columns from the view or they could be selected using OLAP_EXPRESSION. Since there are many measures in this analytic workspace, OLAP_EXPRESSION is used to avoid the need to include every measure in the view. In your application, you can do either – whichever you find most convenient.

Reports

- Act, Fcst Sales Trend
- Sales, Cost, profit, Fcst

Sales, Cost, Profit Trends and Forecast

Period	Sales	Cost	Profit	Forecast Sales	Forecast Cost	Forecast Profit
Jan-03	\$8,400,439.78	\$15,490.75	\$613,280.30			
Feb-03	\$8,953,826.84	\$15,418.69	\$625,120.47			
Mar-03	\$9,592,144.35	\$15,433.85	\$725,663.94			
Apr-03	\$10,457,164.90	\$15,440.73	\$781,607.71			
May-03	\$11,373,236.22	\$15,434.76	\$829,196.64			
Jun-03				\$11,392,144.73	\$15,369.48	\$865,715.40
Jul-03				\$11,529,069.05	\$15,352.20	\$876,841.16
Aug-03				\$11,642,964.70	\$15,337.44	\$885,189.22
Sep-03				\$11,761,283.16	\$15,324.39	\$894,706.34
Oct-03				\$11,883,189.92	\$15,312.60	\$905,124.31
Nov-03				\$12,008,534.30	\$15,301.75	\$916,309.98
Dec-03				\$12,137,337.23	\$15,291.63	\$928,176.79

Actual and Forecast Sales, Cost and Profit report

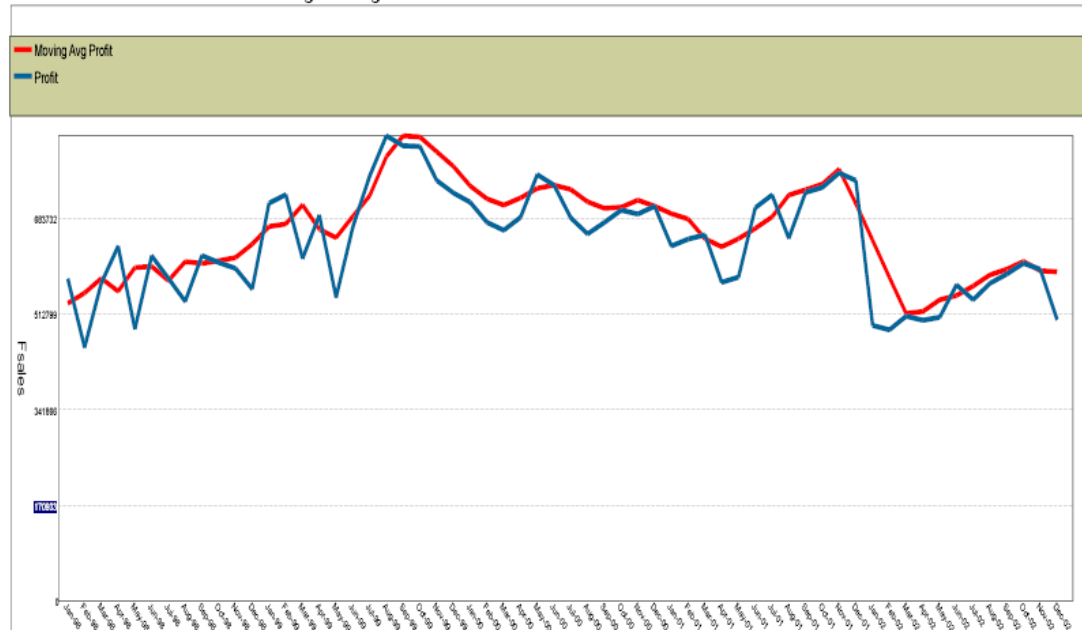
The SQL used to generate this report in Oracle HTML DB follows.

```

select time_desc,
       SALES Sales,
       EXTENDED_COST Cost,
       olap_expression(olap_calc,'PROFIT') Profit,
       olap_expression(olap_calc,'FCAST_SALES') Fsales,
       olap_expression(olap_calc,'FCAST_UNIT_COST') Fcost,
       olap_expression(olap_calc,'FCAST_PROFIT') Fprofit
from dnorm_fact_view_all
where time_level = 'MONTH'
      and year = 102
      and product_level = 'TOTAL_PRODUCT'
      and customer_level= 'ALL_CUSTOMERS'
      and channel_level = 'ALL_CHANNELS'
      and customer_desc is not null
order by month
    
```

Profit Trend

Profit - Actual and 3 Months Moving Average



Profit Trend – Actual Profit and Three Month Averaging

Two SQL statements are used for this graph, one for each series.

```
select null, time_Desc,  
ROUND(OLAP_EXPRESSION(olap_calc, 'PROFIT_MOVING_AVERAGE'),2) fsales  
  from dnorm_fact_view_all v2  
  where time_level = 'MONTH'  
        and product_level = 'TOTAL_PRODUCT'  
        and customer_level= 'ALL_CUSTOMERS'  
        and channel_level = 'ALL_CHANNELS'  
        and customer_desc is not null  
 order by olap_expression (olap_calc, 'TIME_END_DATE')
```

```
select null, time_Desc Month,  
       profit  
  from dnorm_fact_view_all  
  where time_level = 'MONTH'  
        and product_level = 'TOTAL_PRODUCT'  
        and customer_level= 'ALL_CUSTOMERS'  
        and channel_level = 'ALL_CHANNELS'  
        and customer_desc is not null
```

How HTML Database Accesses Oracle10g OLAP

HTML Database uses SQL to access data sources. SQL can be generated using the application wizard or the designer can provide a hand coded select statement. HTML can select from any style schema – dimensional or other.

Because HTML database can select from any schema, the denormalized views of the analytic workspace was used. This allowed the SQL generated to be emitted by HTML database to be very simple – no joins are required and the hierarchical structure of the model could be used to simplify where clauses. Also note that the SQL does not need to express the calculation rules or aggregations since the rules are embedded in the analytic workspace (again, the concept of the 'solved cube').

The view that HTML database selects from is described below.

```

SQL> desc dnorm_fact_view;
Name                                         Null?      Type
-----
TIME_ID                                     VARCHAR2 (5)
TIME_LEVEL                                  VARCHAR2 (30)
TIME_DESC                                   VARCHAR2 (30)
TIME_ORDER                                  NUMBER
MONTH                                       VARCHAR2 (5)
QUARTER                                     VARCHAR2 (5)
YEAR                                        VARCHAR2 (5)
CUSTOMER_ID                                 VARCHAR2 (5)
CUSTOMER_LEVEL                              VARCHAR2 (30)
CUSTOMER_DESC                               VARCHAR2 (30)
SHIP_TO                                     VARCHAR2 (5)
ACCOUNT                                     VARCHAR2 (5)
MARKET_SEGMENT                             VARCHAR2 (5)
TOTAL_MARKET                               VARCHAR2 (5)
WAREHOUSE                                  VARCHAR2 (5)
REGION                                      VARCHAR2 (5)
ALL_CUSTOMERS                              VARCHAR2 (5)
PRODUCT_LEVEL                              VARCHAR2 (30)
PRODUCT_DESC                               VARCHAR2 (30)
PRODUCT_ID                                  VARCHAR2 (5)
ITEM                                        VARCHAR2 (5)
FAMILY                                     VARCHAR2 (5)
CLASS                                       VARCHAR2 (5)
TOTAL_PRODUCT                              VARCHAR2 (5)
CHANNEL_ID                                  VARCHAR2 (5)
CHANNEL_LEVEL                              VARCHAR2 (30)
CHANNEL_DESC                               VARCHAR2 (30)
CHANNEL                                     VARCHAR2 (5)
ALL_CHANNELS                               VARCHAR2 (5)
SALES                                       NUMBER
UNITS                                       NUMBER
EXTENDED_COST                              NUMBER
FORECAST_SALES                             NUMBER
OLAP_CALC                                  RAW (32)

```

Note that not all of the required measures are included as columns in the view. Since HTML database allows the report designer to provide their own SQL for the report, the report designer can take advantage of the OLAP_EXPRESSION feature to select objects the analytic workspace that are not explicitly listed as columns in the view. In this case, OLAP_EXPRESSION is used to select from formulas containing various calculations.

SQL for the three sample reports listed about follows. In each, note:

- The use of OLAP_EXPRESSION to include measures in the analytic workspace that are not predefined in the view. This technique simplifies the definition of the view, but requires that the application allow the user to use any SQL function in the SQL. Either method – including the measure in a view or using OLAP_EXPRESS – is perfectly valid.
- The ability to mix OLAP_EXRESS and standard SQL functions. This illustrates the high degrees of integration between the SQL and multidimensional engines in the database.
- The use of hierarchical elements in the WHERE clause. This is a good example of how the denormalized fact view allows hierarchical selections without the use of joins to dimension tables.

It would have been possible for the SQL to contain the actual calculation rule in OLAP_EXPRESSION rather than having to embed the calculation rule in the analytic workspace. Either method is effective from a technical point of view, however Global Computing wants to use these calculations in a variety of applications and requires the calculations be consistent in each applications. Therefore, the calculations where installed into the analytic workspace.

The flexibility of OLAP_EXPRESSION, however, guarantees that every application can access the full power of the analytic workspace even if calculations have not been predefined by the DBA.

ORACLE DISCOVERER

DESCRIBED

Oracle Discoverer is an intuitive ad-hoc query, reporting, analysis, and Web-publishing tool that provides the user with the opportunity to explore data using a variety of data selection and navigation methods. It falls into the general category of query and reporting tools that provides reports with navigation features such as drilling and pivoting, graphs, and query builders that hide the complexity of the physical data model and provide the ability to define various types of calculations.

Oracle Discoverer will allow report builders and analysts to create, modify and execute ad hoc queries and reports. More casual users can view and navigate through pre-defined reports and graphs.

Discoverer is highly integrated with the Oracle10g database, enabling end users to leverage underlying functionality including SQL OLAP functions and analytic workspaces.

Discoverer provides a business view to hide the complexity of the underlying data structure. This data structure can be relational as well as multidimensional.

The version of Discoverer used in this demonstration is a preview of an upcoming release, which can access data either through a relational context or a multidimensional context.

When operating in a relational context, Discoverer always views data through relational data sources (tables and views). It is extremely flexible since it makes no assumptions regarding the data model in this context – the relationships between data are described in a semantic layer known as the End User Layer. This flexibility allows it to do some types of reporting that wouldn't be available in a dimensional context, for example joining data from a dimensionally modeled system such as a star schema, with data in a relational or network model as might be found in a transactional system. In addition, when run in a relational context Discoverer can make the most effective use of functions in the SQL language.

When operating in a dimensional context, Discoverer leverages the power of the dimensional model with dimensionally oriented query and calculation builders. These dimensionally aware components make it extremely easy for end users to construct very complicated queries and calculations purely from a dimensional view of the data. When models fit within the dimensional context, users typically find using Discoverer in this context easy to use and most powerful from an analytic perspective.

DISCOVERER AS AN EXAMPLE

Some ad-hoc query and reporting tools operate purely in a relational context. Others operate purely in a dimensional context. When viewing the upcoming examples in Discoverer, keep in mind that similar techniques can be used by these specialized tools to access data managed by the OLAP option in analytic workspaces.

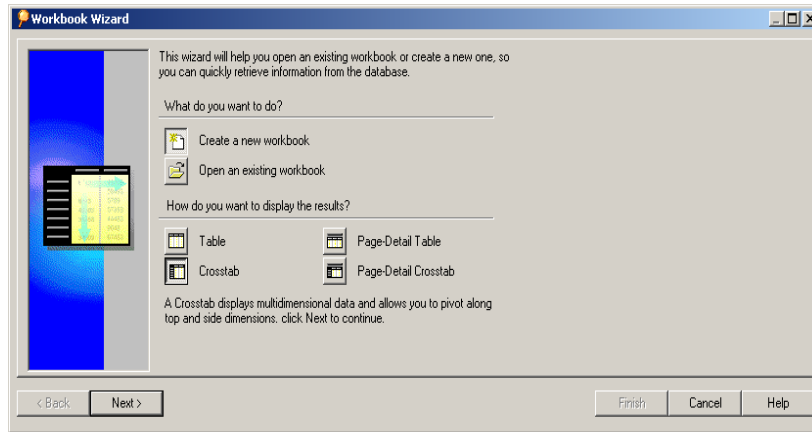
GLOBAL'S REPORTING IN ORACLE DISCOVERER

While HTML Database could deliver the results of sophisticated calculations produced by the OLAP option, opportunities to interactively explore and analyze the data are relatively limited. Discoverer allows users to start with the types of analysis that were available in HTML database and expand upon it by making their own data selections using query builders and data navigation techniques such as drilling and pivoting.

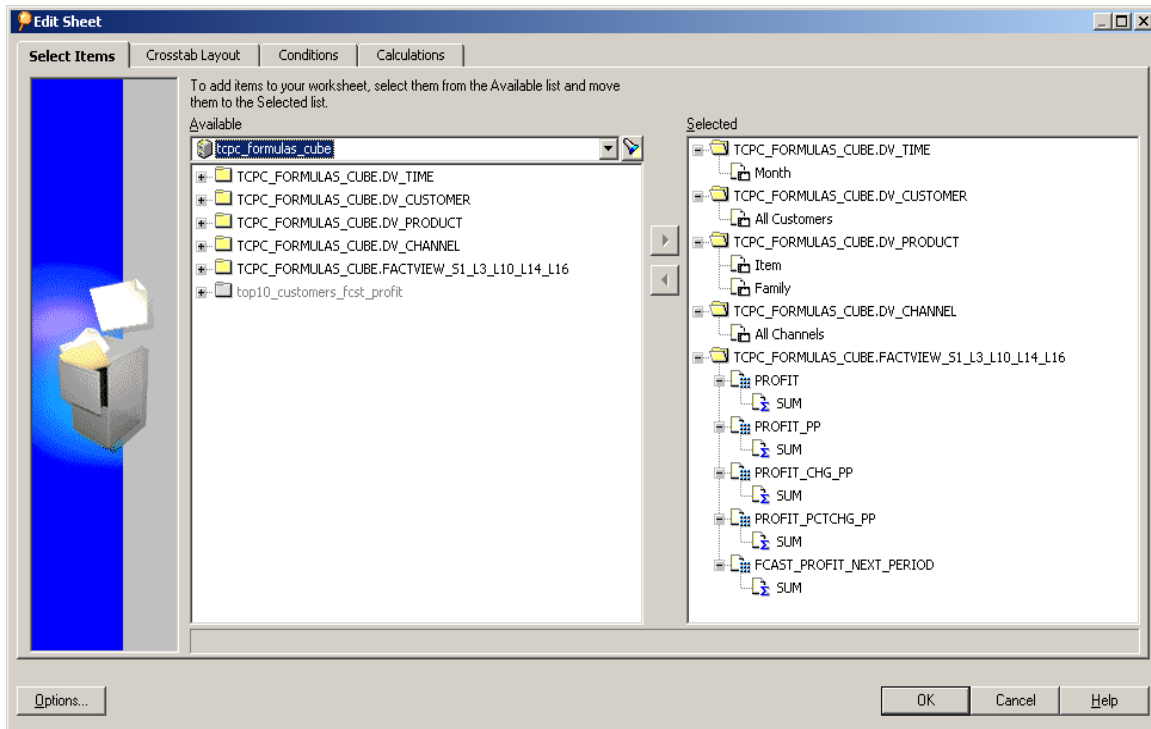
The user experience in Discoverer will vary somewhat depending on whether they are using it in the relational or dimensional context. If they are using it in the relational context, they will be exposed to the relational concepts such as SQL functions. If they are using it in a dimensional context, the query builder will be based purely on the dimensional logical model.

DISCOVERER IN A RELATIONAL CONTEXT

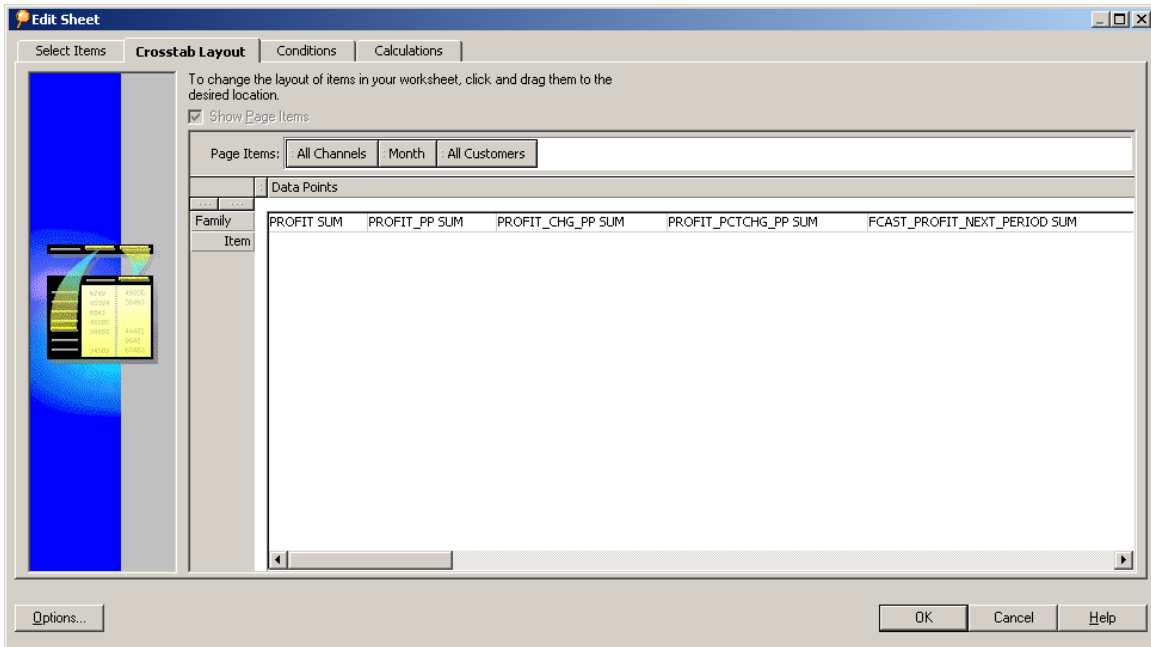
The following illustrations in which a report with several time series functions is built show how Discoverer might look to the user in a relational context. Note that the objects that the user manipulates are relational and operators are based on SQL (for example, SUM).



Choosing a document type



Selecting data in Discoverer



Selecting the cross tab layout

	Profit	Profit Prior Period	Profit Change From Prior Period	Profit % Change from Prior Period	Forecast Profit For Next Month
▶ Accessories	\$103,934.38	\$94,374.87	\$9,559.51	10	\$104,352.92
▶ CD-ROM	\$264,622.74	\$228,886.93	\$35,735.81	16	\$271,567.24
▶ Envoy External 6X CD-ROM	\$1,660.75	\$1,629.28	\$31.47	2	\$1,576.41
▶ Envoy External 8X CD-ROM	\$34,085.01	\$33,122.04	\$962.97	3	\$34,459.52
▶ External 6X CD-ROM	\$4,553.22	\$4,807.40	-\$254.18	-5	\$5,066.34
▶ External 8X CD-ROM	\$177,740.16	\$143,607.35	\$34,132.81	24	\$183,156.46
▶ Internal 6X CD-ROM	\$2,276.54	\$2,443.00	-\$166.46	-7	\$2,437.82
▶ Internal 8X CD-ROM	\$44,307.06	\$43,277.86	\$1,029.20	2	\$44,870.70
▶ Desktop PCs	\$139,716.77	\$152,076.92	-\$12,360.15	-8	\$172,340.10
▶ Sentinel Financial	\$63,698.91	\$98,029.08	-\$34,330.17	-35	\$113,594.67
▶ Sentinel Multimedia	\$9,613.80	-\$10,641.96	\$20,255.76	-190	-\$4,263.69
▶ Sentinel Standard	\$66,404.06	\$64,689.80	\$1,714.26	3	\$63,009.12
▶ Documentation	\$32,867.65	\$31,935.86	\$931.79	3	\$33,402.16
▶ Memory	\$56,916.18	\$54,715.82	\$2,200.36	4	\$57,655.35
▶ Modems/Fax	\$72,387.84	\$68,603.20	\$3,784.64	6	\$68,685.22
▶ Monitors	\$51,452.59	\$49,033.44	\$2,419.15	5	\$52,184.05
▶ Operating Systems	\$83,540.20	\$73,527.47	\$10,012.73	14	\$84,021.81
▶ Portable PCs	\$23,758.29	\$28,453.20	-\$4,694.91	-17	\$21,506.57

Product profitability report

In the following example, Discoverer uses SQL to display a report of forecasted sales and the customer's share of forecasted sales.

	Forecast Sales	Customer Share of the Total Sales
Computer Services Tokyo	17,071,599.46	22.64
Business World New York	11,173,138.84	15.35
Computer Warehouse Singapore	4,412,115.30	6.71
Business World San Jose	3,431,255.77	5.79
KOSH Entrpr New York	2,655,901.87	5.01
Computer Warehouse London	2,594,147.64	4.59
Computer Warehouse San Jose	2,581,365.99	4.58
KOSH Entrpr Tokyo	1,937,100.48	3.49
Computer Services Toronto	1,529,656.31	2.57
Computer Wiz Tempe	1,286,568.72	2.33
Computer Warehouse San Diego	1,274,649.45	2.39
Business World Hamburg	1,186,009.20	2.23
IBS Computers London	1,118,976.29	1.89
KOSH Entrpr Boston	885,222.19	1.82
Computer Warehouse Atlanta	872,155.75	1.57

Forecast Sales and Customer Share report in Discoverer using SQL to access analytic workspace

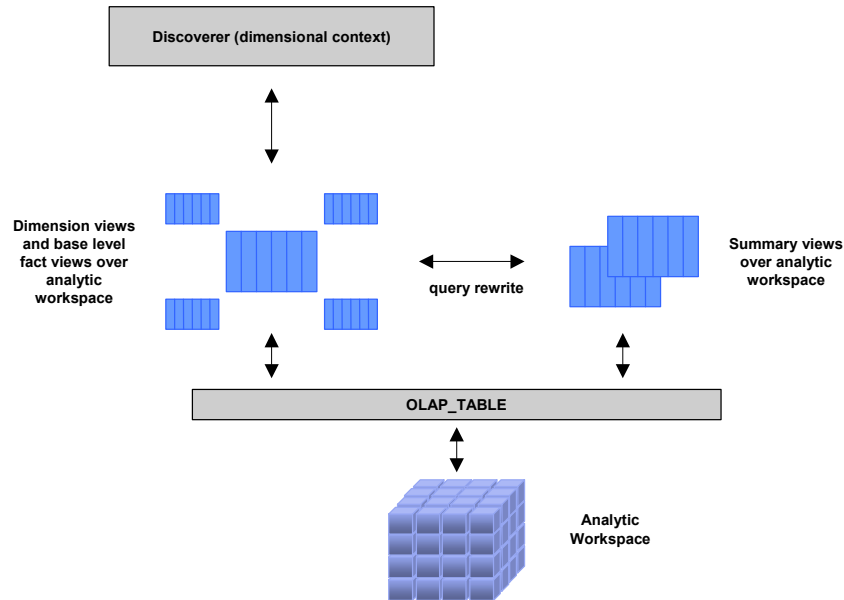
It is important to note that even though Discoverer, when running in its relational context, has no specific knowledge of the OLAP option or how it has calculated any of the data. The user, however, has full access to the results of the calculations provided through SQL by the OLAP option. This is an excellent example of how the OLAP option can be used to extend the analytic capacity of SQL based query and reporting tools.

HOW DISCOVERER ACCESSES ANALYTIC WORKSPACE IN A RELATIONAL CONTEXT

In its relational context, Discoverer uses SQL to access data in the Database. Access to data in analytic workspaces is through SQL written to relational views over the analytic workspace. In this context, Discoverer is unaware that the source of the data is an analytic workspace.

In Oracle10g, Discoverer is mapped to views over the analytic workspace representing dimension tables and a detail level fact table. That is, a view that includes only the lowest level of summarization for data in the analytic workspace.

Summary level fact views are also used, however they are not mapped to Discoverer's End User Layer. Instead, the Database's new *query equivalency* feature is used to allow the database to automatically rewrite SQL with aggregation operators (for example, SUM) and GROUP BY from the detail level view to a summary level view over the analytic workspace. As compared to methods used with previous versions of the database, this significantly reduces the complexity of the End User Layer.



Discoverer accessing Oracle10g OLAP in relational context

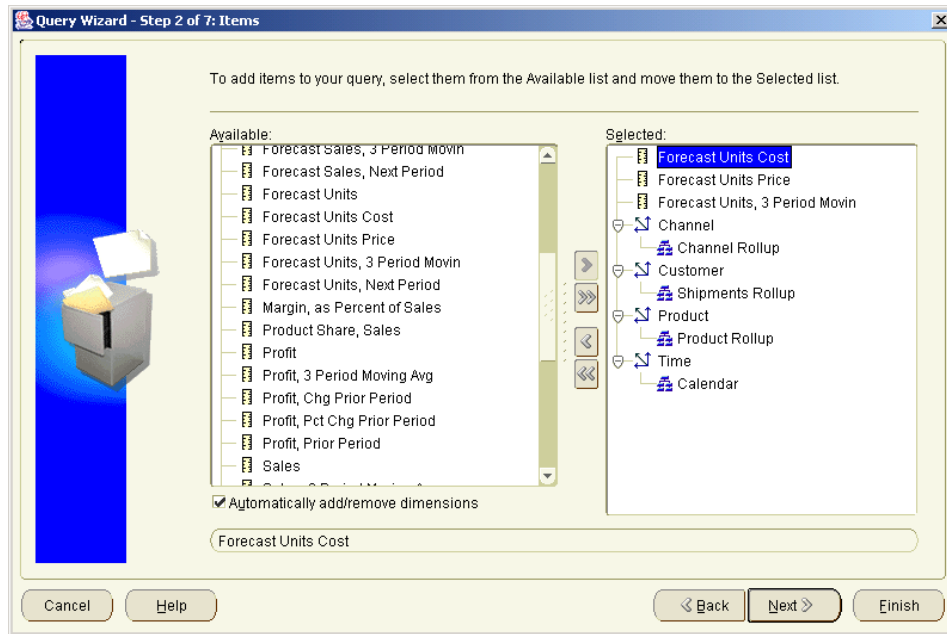
Since Discoverer generates SQL for the end user (as compared with HTML Database, where it is common for the report designer to write their own SQL by hand) all measures are included in the views. This allows the end user to use any of the measures in the analytic workspace without having to use OLAP_EXPRESSION in SQL. It is, however, possible for Discoverer to use OLAP_EXPRESSION.

DISCOVERER IN A DIMENSIONAL CONTEXT

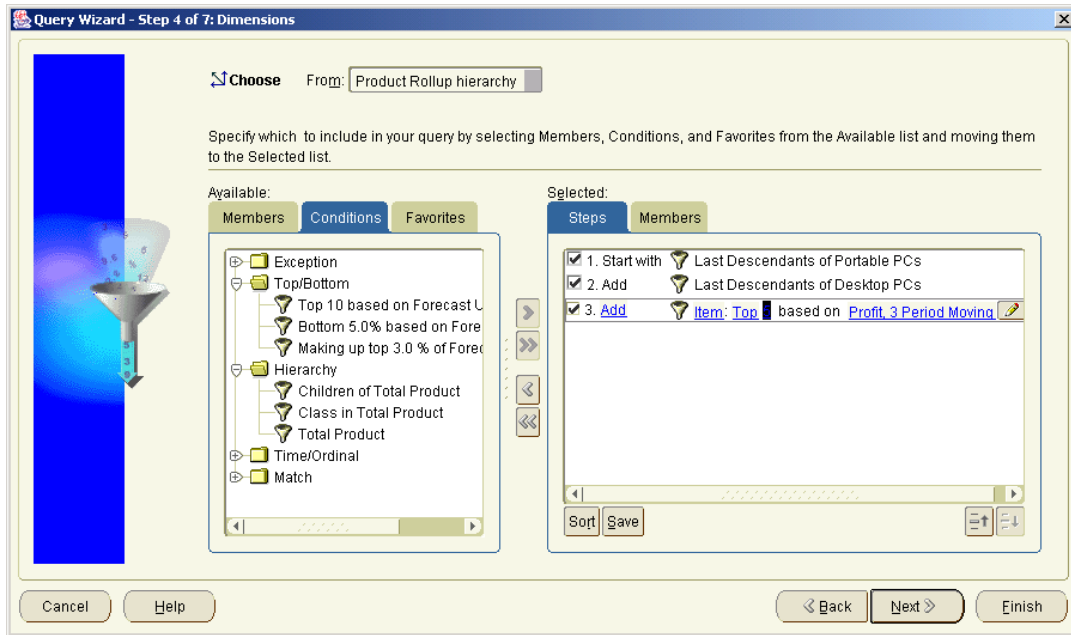
When in a dimensional context, Discoverer makes full use of the dimensional model by providing a dimensionally aware query and calculation builders. These simplify the tasks of defining queries and calculations by leveraging the models' structure and the capabilities of the OLAP API.

There is also the added benefit that the logical model is completely abstracted from the physical model and thus the physical data store is completely transparent to the end user. Although there are differences in query building and calculation definition tools, it should be noted that the look and feel of objects such as reports and graphs is fully consistent with Discoverer in the relational context.

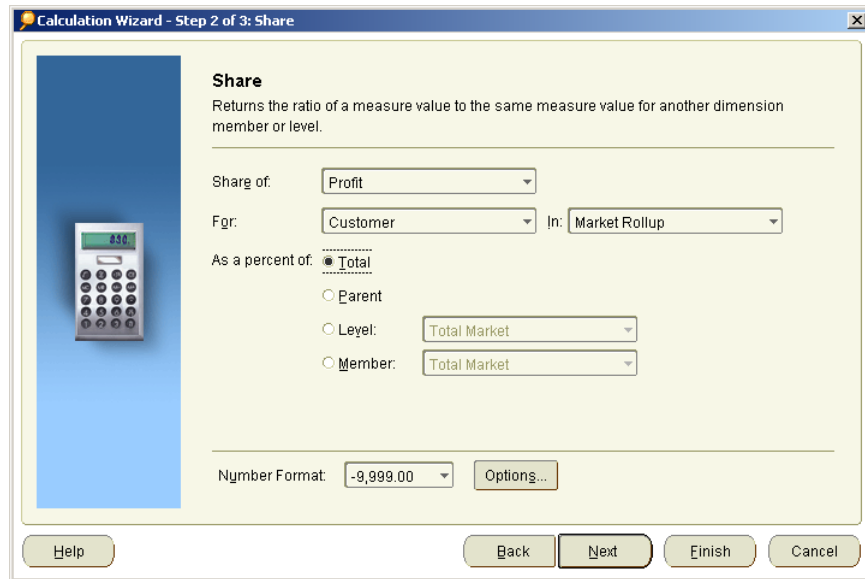
From the end user's perspective, the main differences between the relational and dimensional contexts are how queries and calculations are defined. The following illustrations show how queries and calculations are defined in a dimensional context.



Choosing items in the dimensional context. Note how all selections are based on the logical model.



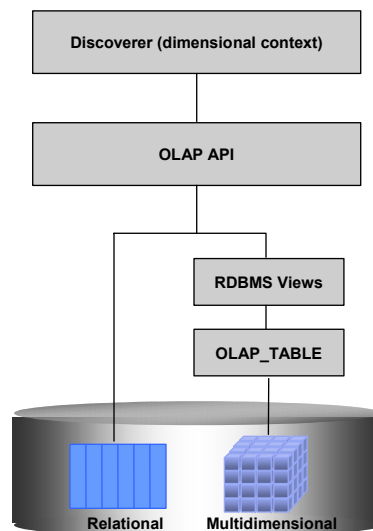
Selecting the top 10 items based on three period moving averaging where items are descendants of Portable PCs and Desktop PCs. Note the support for multi-step, hierarchically aware data selections.



Defining a profit share calculation in Discoverer's calculation builder

HOW DIMENSIONAL DISCOVERER ACCESSES ORACLE10G OLAP

Discover, in the dimensional context, accesses data using the Oracle OLAP API. It is important to note that the OLAP API can access data in both relational and multidimensional data types. As a result, it is possible to query a star schema in the dimensional context if the user desires the benefits of the dimensional model.



Discoverer accessing Oracle10g OLAP in dimensional context

Since Discoverer is simply accessing data through the OLAP API, the query processing occurs as described in previous sections of this document that describe the OLAP API.

ORACLE REPORTS

DESCRIBED

Oracle Reports is a powerful reporting tool that is used for enterprise reporting. Reports specializes in providing highly formatted output in a variety of document types including textual documents, business forms or tabular output.

Oracle Reports can access a variety of data sources and supports many different output mechanisms. Data access methods or languages include SQL, PL/SQL, JDBC, XML and the Database's OLAP API. Output formats include PDF, RTF, HTML, text, XML, Postscript and PCL. Distribution methods include file, email, printer, cache (for direct output in the browser) and Oracle Portal.

One access method Oracle Reports can use is the OLAP Pluggable Data Source. The OLAP Pluggable Data Source is based on the Oracle Business Intelligence Beans and the OLAP API and allows Reports access to both star schema and analytic workspaces in a dimensional context through. In a dimension context, Reports allows the Business Intelligence Beans' Query Builder to be used to define data selections.

GLOBAL'S REPORTING USING ORACLE REPORTS

Rather than simply sending their customers a monthly invoice, Global Computing Exchange wanted to improve their relationship with customers by providing them with analysis of their purchasing from Global. Global also wanted to encourage customers to promptly reorder and to do so through a low cost channel – their Internet web site.

To facilitate reordering through their web site, they began a program to encourage their customer to automatically reorder based on forecasts generated by Global's business intelligence system. This could lower the customers' purchases costs, maintain Global's sales and decrease Global's cost of sales. Although customers can order product throughout the month, this system would encourage customers to order in bulk once a month and reduce shipping costs.

The following document, produced by Oracle Reports, selects data from both relational and multidimensional data sources. Note the following:

- Customer information and billing amounts are selected from relational tables in a transaction system.
- In the text portion of the document information related to the purchase history, projections and savings estimates are selected from an analytic workspace.
- The chart with forecasted units, prices and discount projections are selected from the analytic workspace.



Global Exchange Corporation

Customer: Computer Services Tokyo
 10-5, Akasaka 1-chome
 Tokyo
 Japan
Billing Period: May-03
Billing Date: Aug. 14, 2003

4453-665-78763

Page 1 of 2

Advance Purchase Order Services: *A special service for our Platinum level customers.*

Your total invoice amount for this period was \$2,937,416.17. Based on your purchase history with us we have estimated that your invoice amount for the next period will be \$3,012,650.72, an increase of 2.56%, for total number of 5,577 units.

For your conveniec we have created your purchase order for the next period. Pleas note that this is not an invoice and will not be charged to your account untill you make the purchases. This is just an estimation. Please visit our web site www.globalxchange.com and enter code FCSTINV8782 to make required modifications and purchases.

If you choose this option you will be entitled for a 5% discount on each and every item purchased on or before 03-Sep-2003 for a total saving of upto \$150,632.54. The discounted unit price and sales amounts are included in this estimation.

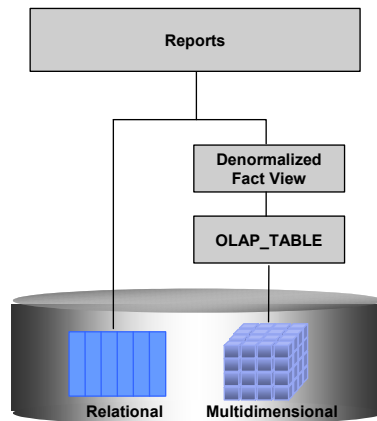
Product	Forecast Units	List Price	List Amount	Discount Price	Discount Amount	Saving Amount
SIMM- 8MB PCMCIAII card	103	\$275.91	\$28,413.42	\$262.11	\$26,992.75	\$1,420.67
SIMM- 16MB PCMCIAII card	30	\$546.39	\$16,617.59	\$519.07	\$15,786.71	\$830.88
External 6X CD-ROM	21	\$127.80	\$2,707.78	\$121.41	\$2,572.39	\$135.39

Advance Purchase Order created in Oracle Reports using both relational and multidimensional data types

HOW REPORTS ACCESSES ORACLE10G OLAP

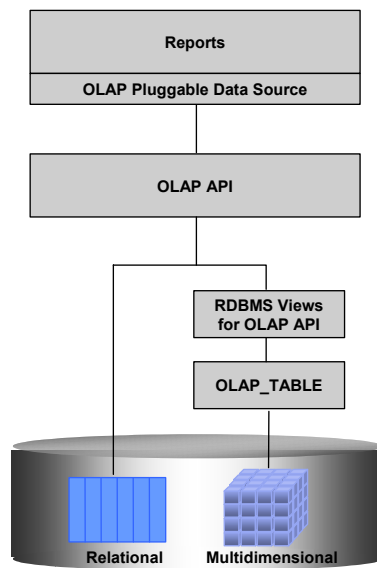
Access methods are either SQL or the Oracle Reports Pluggable Data Source for Oracle OLAP.

When using SQL, data are accessed using a view over the analytic workspace. The same denormalized fact view that is used with HTML Database is also used when Oracle Reports accesses analytic workspaces using SQL. The denormalized fact view includes all data for dimension members – member ids, descriptions, hierarchies, attributes – and facts.



Oracle Reports using SQL to access analytic workspace

When the Oracle Reports Pluggable Data Source (OLAP PDS) is used, access occurs through the OLAP API. As a result, Oracle Reports shares the same infrastructure (the views for the OLAP API and the OLAP catalog metadata) that is used by Discoverer in the dimensional context and the Business Intelligence Beans.



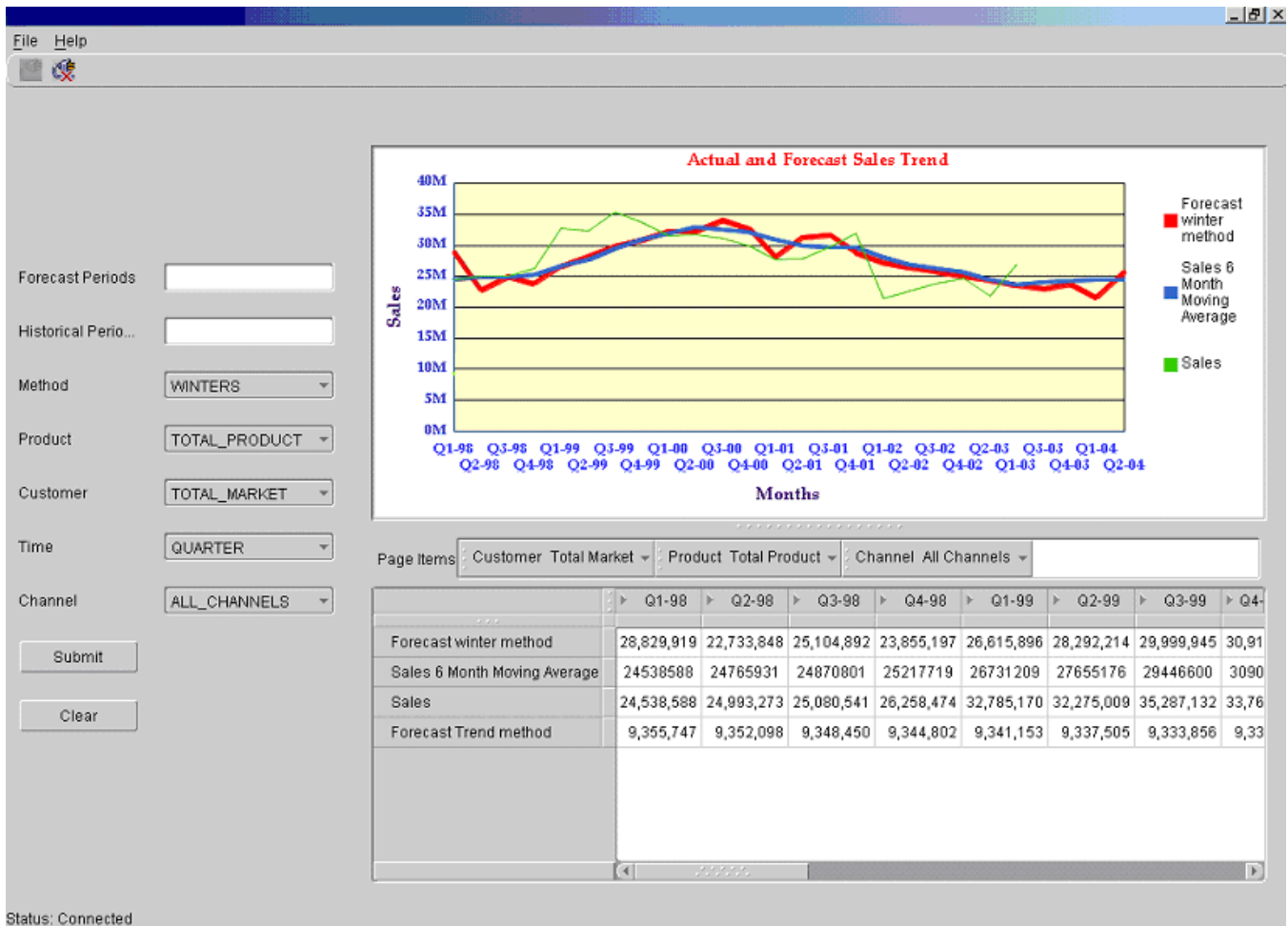
Oracle Reports using SQL to access analytic workspace

FORECASTING APPLICATION

DESCRIBED

Forecasting is an important part of Global's business intelligence system. Reports accessed from HTML Database, Discoverer and Reports all utilize forecast measures. The job of creating the forecasts falls to demand planning experts at Global. To facilitate this process, a forecasting application was created using the Oracle Business Intelligence Beans.

This application provides an interface that allows the demand planning experts to choose forecast types and set a variety of parameters such as the historical periods, the forecast periods and data selections for the channel, product and customer dimensions. The interface presented to the demand planning experts was designed to be very simple and entirely focused on the task of generating forecasts. The main interface is shown below.



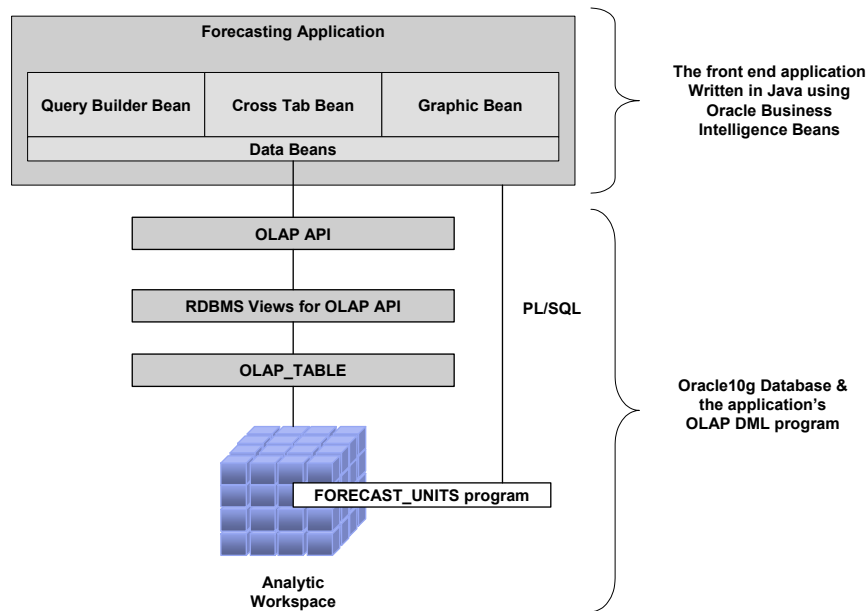
Forecasting application based on Oracle OLAP

HOW THE FORECASTING APPLICATION IS BUILT

The forecasting application's interface was built in Java using the Oracle Business Intelligence Beans. The Business Intelligence Beans provide the data presentation components (cross tabs and graphs) and high level interfaces to the database. These high level interfaces insulate the application developer from the lower level OLAPI API and provide services such as query generation.

Since the forecasting application is built using the Oracle Business Intelligence Beans, the data access path is similar to the other applications that are based on the OLAP API. What is really different about this application is the interaction with the multidimensional engine for the generation of the forecast data.

The following diagram describes the overall architecture of the forecasting program.



Architecture of the forecasting application

The forecast rules are defined within a program in the analytic workspace using the OLAP DML. The forecast program is designed to allow the user to specify a forecast method, the levels within the data model that should be forecasted, the number of historical periods and the periodicity of the data. See APPENDIX A for a listing of the OLAP DML code used for the forecasts.

OLAP SPREADSHEET ADD-IN FOR EXCEL

DESCRIBED

The OLAP spreadsheet add-in for Microsoft Excel allows Excel to be used as a front end to data managed by the OLAP option. For some users, a spreadsheet interface provides several advantages:

- User might already be familiar with the use of a spreadsheet and thus little training is required.
- This method allows users to join OLAP data with preexisting spreadsheet data.
- Since Excel interfaces with other office productivity applications, it provides the means for these applications to access data managed by Oracle OLAP.

The OLAP spreadsheet add-in for Excel allows the user to connect to an instance of the Oracle database, choose data using the Business Intelligence Beans Query Builder, define new calculations using the Business Intelligence Beans Calculation Builder and display the results in Excel. Once the data are in Excel, other cells, worksheets or workbooks can refer it to.

GLOBAL'S USE OF THE OLAP SPREADSHEET ADD-IN

Like finance departments in most organizations, Global Computing Exchange's finance department uses Excel for many day-to-day tasks. Because the users are very familiar with Excel, they will use it to query the OLAP option to the Oracle10g database. They will also use Excel as a means of creating presentations that include data from Oracle's multidimensional data types.

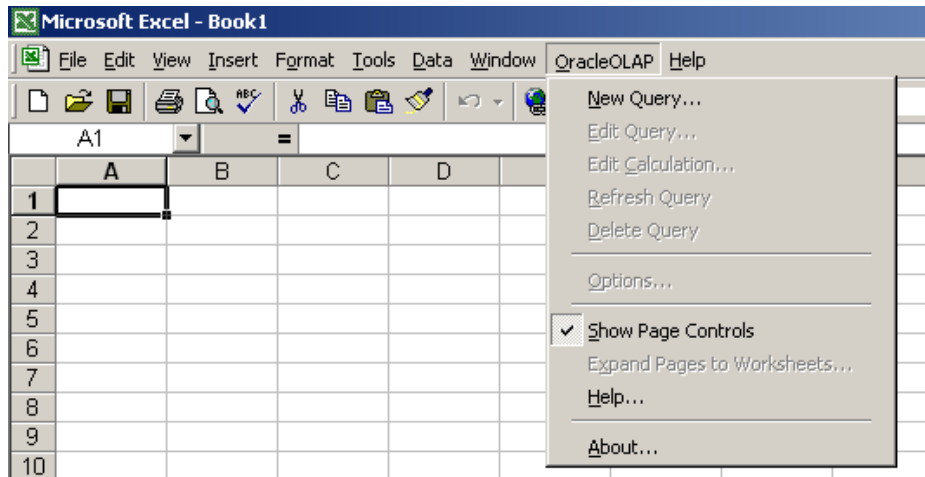
HOW THE OLAP SPREADSHEET ADD-IN ACCESSES ORACLE10G OLAP

The OLAP spreadsheet add-in is based largely on the Oracle Business Intelligence Beans. Like the OLAP Pluggable Data Source for Oracle Reports, the spreadsheet add-in uses the Business Intelligence Beans Query Builder, Calculation Builder and Data Beans to define queries and access data.

From the end users perspective, the process of accessing data managed by Oracle OLAP is shown in the following

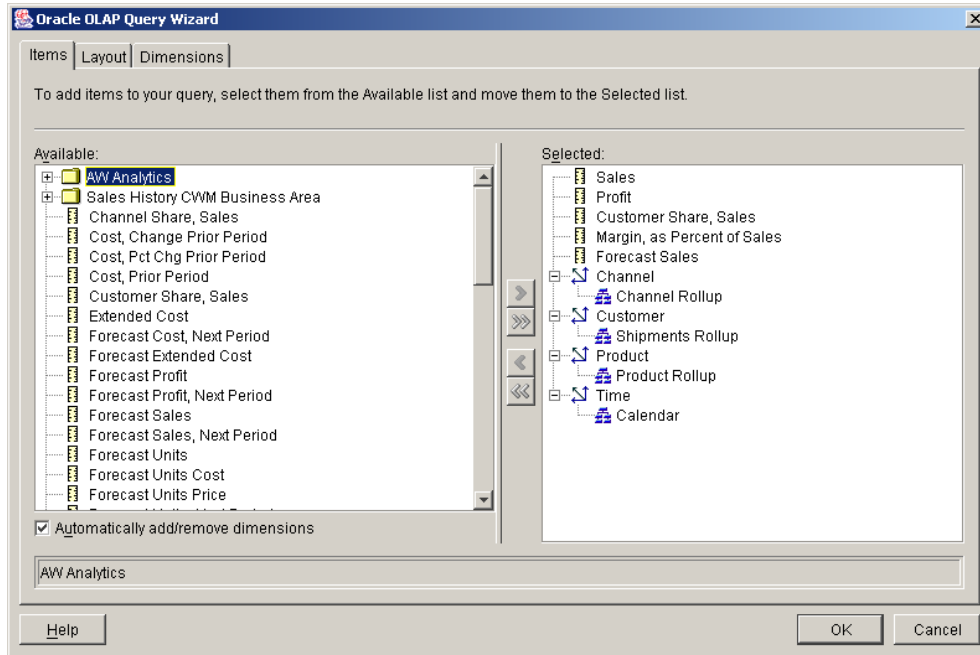
illustrations.

The Oracle OLAP add-in is accessed through a menu command in Excel. The first step for accessing data through the OLAP option is to create a new query.



Connecting to Oracle10g OLAP as a data source

The following dialog is part of the Query Builder. Dimensions and measures are selected in this step. Selection conditions are made on the Dimensions tab (not shown).



Defining a query

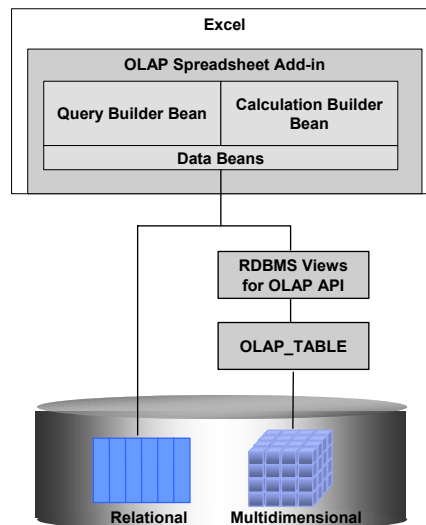
The report in Excel is shown below. Note that dimension members for the product, time and channel dimensions can be selected using the drop down list boxes.

Customers	Sales	Profit	Forecast Sales	Forecast Units	Forecast Profit
Computer Services Tokyo	\$11,043,613.94	\$460,874.41	\$21,759,315.56	39,945.80	\$932,530.60
Business World New York	\$7,485,094.37	\$488,671.17	\$14,292,268.45	46,707.39	\$994,275.57
Computer Warehouse Singapore	\$3,273,676.09	\$385,838.83	\$5,773,883.56	31,403.86	\$849,284.94
Business World San Jose	\$2,826,413.96	\$235,706.69	\$4,522,591.89	21,983.04	\$401,595.55
KOSH Entrpr New York	\$2,445,730.64	\$217,077.87	\$3,546,300.99	21,876.95	\$356,160.59
Computer Warehouse London	\$2,237,061.58	\$194,785.38	\$3,487,085.62	20,099.64	\$318,246.51
KOSH Entrpr San Jose	\$2,231,549.55	\$188,230.83	\$3,446,013.01	16,579.83	\$304,912.48
KOSH Entrpr Tokyo	\$1,701,222.29	\$112,232.20	\$2,612,578.63	8,570.44	\$158,989.32
Computer Services Toronto	\$1,254,660.76	\$106,705.67	\$2,030,501.37	13,597.89	\$182,557.58
Computer Warehouse San Diego	\$1,264,847.77	\$94,067.75	\$1,760,290.21	7,496.77	\$128,734.17

OLAP data in an Excel workbook

Since the spreadsheet add-in is based on the Oracle Business Intelligence Beans, the OLAP API is used to access data through the OLAP option. Therefore, the infrastructure – the RDBMS views over the analytic workspace and the OLAP catalog metadata – is common to all other applications that are based on the OLAP API.

The architecture of the spreadsheet add-in and data access path is shown below.



Architecture of the forecasting application

AN ALTERNATIVE METHOD OF USING EXCEL WITH ORACLE10G OLAP

As an alternative, Excel can use SQL to access data from an analytic workspace. In this case, SQL would be used to access data via a view over an analytic workspace. This is functional, however it doesn't offer any of the advantages (for example, the Query Builder) that are offered by the OLAP Spreadsheet add-in.

ALTERNATIVE METHOD OF SHARING CALCULATIONS ACROSS APPLICATIONS

The primary method of sharing calculations across different applications that has been discussed in this paper is to define the calculation as part of the analytic workspace. This is very effective because it works across all applications,

regardless of the access method (that is, SQL, PL/SQL or OLAP API). It also offers the most analytic power because it allows the construction of custom functions using the OLAP DML.

An alternative method is to define the calculation using the Business Intelligence Beans' Query Builder and to persist it in a publicly accessible repository. In this case, the calculations are accessible to any application that uses either the Business Intelligence Beans to interface with the OLAP API or the Business Intelligence Beans API to discover the definition of the calculation.

As has been noted when discussing Oracle Discoverer, Oracle Reports and the Business Intelligence Beans, these applications all provide access to a calculation builder that is based on the Business Intelligence Beans and the OLAP API. As a result, calculations defined in one application using the calculation builder can be used by the other applications.

The ability to define, persist and reuse calculations that are managed using the Business Intelligence Beans calculation specification API provides an alternative means for defining calculation rules in a common repository and sharing them across any number of tools. This strategy for avoiding fragmentation of calculation rules would be practical if all applications that needed to use the calculation were based on the Business Intelligence Beans or accessed the calculation definition using the Business Intelligence Beans API.

CONCLUSION

This demonstration has been designed to show how several of the most significant problems with traditional business intelligence architectures can be solved using the Oracle10g Database and the Oracle OLAP option to the Oracle10g Database.

In summary, these problems include:

- Fragmentation of data across multiple database instances. This fragmentation occurs both across subject areas and geographies.
- Fragmentation of data across relational and multidimensional databases. This fragmentation is driven by the fact that, historically, relational and multidimensional technologies have existed in separate databases.
- Fragmentation of business rules. This is a side effect of geographic, subject area and technological fragmentation.
- High costs associated with purchasing and maintaining parallel systems, both for database and access tools.

The end result of this fragmentation is the inability to gain a full, clear and consistent view of the enterprise. It is simply impossible to have full visibility into the organization when data are distributed over multiple systems across subject areas, geographies and technologies.

The Oracle10g Database solves these programs by providing a database that:

- Its suitable for combining data from all subject areas and all geographies into a single instance of the database.
- That combines the relational and multidimensional engines and data types into the same database instance.
- Facilitates the central definition business rules.
- Allows wide variety of tools and application to access both relational and OLAP data sources.

Oracle provides a complete business intelligence infrastructure, from Database to business intelligence tools, to support this ability to gain complete visibility into the enterprise. While Oracle tools were used as examples, access to the database it through public APIs – similar solutions could be implemented using SQL based third party tools or by tools written to the OLAP API.

APPENDIX A – LOGICAL DATA MODEL USED IN DEMONSTRATION

The logical data model presented to the end user consists of four dimensions: time, channel, customer and product and 40 measures. The dimensional structure of the model follows.

Time dimension

Calendar Hierarchy

Year level

Quarter level

Month level

Time span attribute, applies to all levels

End date attribute, applies to all levels

Short description attribute, applies to all levels

Long description attribute, applies to all levels

Channel dimension

Channel Rollup hierarchy

All Channels level

Channel level

Product dimension

Product Rollup hierarchy

All Products level

Class level

Family level

Item level

Package attribute, applied to item level only

Customer dimension

Market Segment hierarchy

Total Market level

Market Segment level

Account level

Ship To level

Shipments hierarchy

All Customers level

Region level

Warehouse level

Ship to level

Time span attribute, applies to all levels

End date attribute, applies to all levels

Short description attribute, applies to all levels

Long description attribute, applies to all levels

APPENDIX B – MEASURES IN THE ANALYTIC WORKSPACE

MEASURES IN THE ANALYTIC WORKSPACE

The measures in the analytic workspace can be placed in three categories:

- Measures that were loaded from the source system.
- Measures that were derived from source measures and stored in variables.
- Measures that were calculated dynamically as formulas.

MEASURES LOADED FROM SOURCE SYSTEM

Measures that were loaded from the source system were stored as variables in the analytic workspace. In this demonstration, there are only three measures loaded from the source system: units, unit price and unit cost. All other measures are derived. Units can be aggregated up hierarchies using SUM. Unit price and unit cost could be aggregated using a hierarchical weighted average weighted by units sold.

As is typical in an analytic workspace, a partial aggregation strategy is used to minimize the time needed to prepare the data for query. In this case, summary data are pre-aggregated within some regions of the cube. Other summary data are calculated at runtime as needed. With this database, data are pre-aggregated to every other level of each hierarchy for the product and customer dimensions. There is no pre-aggregation within either the time or channel dimension because their hierarchies are relatively shallow (few levels; few child per parent in each level).

When measures are aggregated dynamically, the OLAP options AGGREGATE function is used to define the aggregation rule. The AGGREGATION function takes the form AGGREGATE(*measure* USING *aggregation map*) where the *measure* is typically the variable storing the measure's data and the *aggregation map* is an object in the analytic workspace that defines the aggregation rules.

A common method of defining such a calculation is to define a formula whose equation is the call to the AGGREGATE function. If the data point is stored, the AGGREGATE function will use the stored value. If the data point is not stored, the AGGREGATE function will calculate it according to the rules in the aggregation map.

The formulas representing actual units, actual unit price and actual unit cost are show below. (See the "LD", or long description, for the description of the measure.

```
DEFINE UNITS FORMULA NUMBER <TIME CUSTOMER PRODUCT CHANNEL>
LD 'Actual units'
EQ aggregate(GLOBAL_AW.GLOBAL!UNITS_VARIABLE using
GLOBAL_AW.GLOBAL!UNITS_CUBE_AGGMAP_UNITS_CUBE_AGGPLAN)
```

```
DEFINE UNIT_COST FORMULA NUMBER <TIME PRODUCT>
LD ' Actual unit cost'
EQ aggregate(GLOBAL_AW.GLOBAL!UNIT_COST_VARIABLE using
GLOBAL_AW.GLOBAL!PRICE_CUBE_AGGMAP_AWCREATEDDEFAULT_1)
```

```
DEFINE UNIT_PRICE FORMULA NUMBER <TIME PRODUCT>
LD ' Actual unit price'
EQ aggregate(GLOBAL_AW.GLOBAL!UNIT_PRICE_VARIABLE using
GLOBAL_AW.GLOBAL!PRICE_CUBE_AGGMAP_AWCREATEDDEFAULT_1)
```

An aggregation map is an object in the analytic workspace. The aggregation map used to define the summarization rules for Actual Units is shown below.

```
DEFINE UNITS_CUBE_AGGMAP_UNITS_CUBE_AGGPLAN AGGMAP
  <TIME UNITS_COMPOSITE <CUSTOMER PRODUCT CHANNEL>>
AGGMAP
RELATION GLOBAL_AW.GLOBAL!TIME_PARENTREL OPERATOR SUM
  PRECOMPUTE (GLOBAL_AW.GLOBAL!TIME_LEVELREL 'MONTH')
RELATION GLOBAL_AW.GLOBAL!CUSTOMER_PARENTREL OPERATOR SUM
  PRECOMPUTE (GLOBAL_AW.GLOBAL!CUSTOMER_LEVELREL 'SHIP_TO' 'REGION'
'MARKET_SEGMENT')
RELATION GLOBAL_AW.GLOBAL!PRODUCT_PARENTREL OPERATOR SUM
  PRECOMPUTE (GLOBAL_AW.GLOBAL!PRODUCT_LEVELREL 'ITEM' 'CLASS')
RELATION GLOBAL_AW.GLOBAL!CHANNEL_PARENTREL OPERATOR SUM
  PRECOMPUTE (GLOBAL_AW.GLOBAL!CHANNEL_LEVELREL 'CHANNEL')
AGGINDEX NO
END
```

MEASURE DERIVED AND STORED AS VARIABLES

Some of the measures were derived from the source measures and stored as variables. The choice to store the measures as variables, rather than to calculate them at runtime, was made based on performance and the type of calculation rule. The sales and extended cost measures are used very extensively and thus were persisted as variables. The forecasting engine requires that the results of a forecast calculation be stored in a variable, either permanently or temporarily during the session. The forecast sales and forecast extended cost measures were derived from the forecasts of units, unit price and unit costs.

The aggregation strategies for these measures are similar to those of the Actual Units measures (that is, a partial aggregation strategy using the AGGREGATE function).

The measures and their calculation rules follow.

ACTUAL SALES

The variable is defined as:

```
DEFINE SALES_VARIABLE VARIABLE NUMBER <TIME TCPC_STORED_CUBE_COMPOSITE <CUSTOMER  
PRODUCT CHANNEL>>
```

Data for the variable is calculated at the lowest level using the command:

```
set sales_variable = units * unit_price
```

The formula is defined as:

```
DEFINE SALES FORMULA NUMBER <TIME CUSTOMER PRODUCT CHANNEL>  
LD 'Actual sales'  
EQ aggregate( GLOBAL_AW.GLOBAL!SALES_VARIABLE using  
GLOBAL_AW.GLOBAL!TCPC_STORED_CUBE_AGGMAP_TCPC_STORED_CUBE_AGGPLAN)
```

ACTUAL EXTENDED COST

This follows the same pattern as Actual Sales, however the calculation rule is different. The variable is defined as:

```
DEFINE EXTENDED_COST_VARIABLE VARIABLE NUMBER <TIME TCPC_STORED_CUBE_COMPOSITE  
<CUSTOMER PRODUCT CHANNEL>>
```

Data for the variable is calculated at the lowest level using the command:

```
SET EXTENDED_COST_VARIABLE = UNITS * UNITS_PRICE
```

The formula is defined as:

```
DEFINE EXTENDED_COST FORMULA NUMBER <TIME CUSTOMER PRODUCT CHANNEL>  
LD 'Actual extended cost'  
EQ aggregate( GLOBAL_AW.GLOBAL!EXTENDED_COST_VARIABLE using  
GLOBAL_AW.GLOBAL!TCPC_STORED_CUBE_AGGMAP_TCPC_STORED_CUBE_AGGPLAN)
```

Some of the forecast measures are calculated using the OLAP option's forecast commands; others are derived from those measures. The measures calculated using the forecast commands are: Forecast Units, Forecast Unit Price and Forecast Unit Cost. Forecast Sales are derived and stored. The following OLAP DMP program calculates all of the stored forecast measures.

```
arg _histperiods      int  
arg _periodicity      int  
vrb _is_error         boolean
```

```

" Forecast at lowest levels
limit time to last 24
limit channel to channel_levelrel 'CHANNEL'
limit product to product_levelrel 'ITEM'
limit customer to customer_levelrel 'SHIP_TO'

" Forecast Units
FCAST_HANDLE_UNITS = fcopen('Units')
fcset FCAST_HANDLE_UNITS method 'automatic' histperiods _histperiods periodicity
_periodicity
fcexec FCAST_HANDLE_UNITS time time into FCAST_UNITS_VARIABLE seasonal
FCAST_SEAS_UNITS smseasonal FCAST_SMSEAS_units units
fcclose FCAST_HANDLE_UNITS

" Forecast Unit Price
FCAST_HANDLE_UNIT_PRICE = fcopen('unit price')
fcset FCAST_HANDLE_UNIT_PRICE method 'automatic' histperiods _histperiods
periodicity _periodicity
fcexec FCAST_HANDLE_UNIT_PRICE time time into FCAST_UNIT_PRICE_VARIABLE seasonal
FCAST_SEAS_UNIT_PRICE smseasonal FCAST_SMSEAS_UNIT_PRICE unit_price
fcclose FCAST_HANDLE_UNIT_PRICE

" Forecast Unit Cost
FCAST_HANDLE_UNIT_COST = fcopen('unit cost')
fcset FCAST_HANDLE_UNIT_COST method 'automatic' histperiods _histperiods periodicity
_periodicity
fcexec FCAST_HANDLE_UNIT_COST time time into FCAST_UNIT_COST_VARIABLE seasonal
FCAST_SEAS_UNIT_COST smseasonal FCAST_SMSEAS_UNIT_PRICE unit_cost
fcclose FCAST_HANDLE_UNIT_COST

"Forecast sales and cost
set FCAST_SALES_VARIABLE = FCAST_UNITS_VARIABLE * FCAST_UNIT_PRICE_VARIABLE'
set TIME TCPC_STORED_CUBE_COMPOSITE do 'FCAST_EXTENDED_COST_VARIABLE =
FCAST_UNITS_VARIABLE * FCAST_UNIT_COST_VARIABLE'

```

MEASURES DERIVED AS FORMULAS

The remaining measures are all calculated at runtime using formulas. The calculation rules are listed below, grouped by the type of measure.

SIMPLE MEASURE CALCULATIONS

```

DEFINE PROFIT FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Profit as defined by Sales less Cost
EQ sales - extended_cost

DEFINE FCAST_PROFIT FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Forecast Profit as defined by Forecast Sales less Forecast Cost
EQ fcast_sales - fcast_extended_cost

DEFINE PCT_MARGIN FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Margin, as percent of sales
EQ profit / sales

```

PRIOR PERIOD MEASURES

```

DEFINE COST_PP FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Cost for prior period
EQ lag(extended_cost,1,time,LEVELREL time_levelrel)

```

```

DEFINE UNITS_PP FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Units for prior period
EQ lag(units,1,time,LEVELREL time_levelrel)

DEFINE SALES_PP FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Sales for prior period
EQ lag(sales,1,time,LEVELREL time_levelrel)

DEFINE PROFIT_PP FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Profit for prior period
EQ lag(profit,1,time,LEVELREL time_levelrel)

```

CHANGE FROM PRIOR PERIOD MEASURES

```

DEFINE COST_CHG_PP FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Change in cost as compared to prior period
EQ lagdif(extended_cost,1,time,LEVELREL time_levelrel)

DEFINE PROFIT_CHG_PP FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Change in profit as compared to prior period
EQ lagdif(profit,1,time,LEVELREL time_levelrel)

DEFINE UNITS_CHG_PP FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Change in units as compared to prior period
EQ lagdif(units,1,time,LEVELREL time_levelrel)

DEFINE SALES_CHG_PP FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Change in sales as compared to prior period
EQ lagdif(sales,1,time,LEVELREL time_levelrel)

```

PERCENT CHANGE FROM PRIOR PERIOD

```

DEFINE COST_PCTCHG_PP FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Percent in cost as compared to prior period
EQ lagpct(extended_cost,1,time,LEVELREL time_levelrel)*100

DEFINE PROFIT_PCTCHG_PP FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Percent change in profit as compared to prior period
EQ lagpct(profit,1,time,LEVELREL time_levelrel)*100

DEFINE UNITS_PCTCHG_PP FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Percent change in profit as compared to prior period
EQ lagpct(units,1,time,LEVELREL time_levelrel)*100

DEFINE SALES_PCTCHG_PP FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Percent change in sales as compared to prior period
EQ lagpct(sales,1,time,LEVELREL time_levelrel)*100

```

MOVING AVERAGES

```

DEFINE COST_MOVING_AVG FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Cost, moving average over three periods
EQ movingaverage(extended_cost, -2, 1, 1, time, LEVELREL time_levelrel)

DEFINE FCAST_COST_MOVING_AVG FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Forecast Cost, moving average over three periods
EQ movingaverage(fcast_extended_cost, -2, 1, 1, time, LEVELREL time_levelrel)

```

```

DEFINE FCAST_PROFIT_MOVING_AVG FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Forecast Profit, moving average over three periods
EQ movingaverage(fcast_profit, -2, 1, 1, time, LEVELREL time_levelrel)

DEFINE FCAST_SALES_MOVING_AVG FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Forecast Sales, moving average over three periods
EQ movingaverage(fcast_sales, -2, 1, 1, time, LEVELREL time_levelrel)

DEFINE FCAST_UNITS_MOVING_AVG FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Forecast Units, moving average over three periods
EQ movingaverage(fcast_units, -2, 1, 1, time, LEVELREL time_levelrel)

DEFINE PROFIT_MOVING_AVG FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Profit, moving average over three periods
EQ movingaverage(profit, -2, 1, 1, time, LEVELREL time_levelrel)

DEFINE SALES_MOVING_AVG FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Sales, moving average over three periods
EQ movingaverage(sales, -2, 1, 1, time, LEVELREL time_levelrel)

DEFINE UNITS_MOVING_AVG FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Units, moving average over three periods
EQ movingaverage(units, -2, 1, 1, time, LEVELREL time_levelrel)

```

LEAD MEASURES

```

DEFINE FCAST_COST_NEXT_PERIOD FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Forecasted cost for the next period
EQ lead(fcast_extended_cost,1,time,LEVELREL time_levelrel)

DEFINE FCAST_PROFIT_NEXT_PERIOD FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Forecasted profit for the next period
EQ lead(fcast_profit,1,time,LEVELREL time_levelrel)

DEFINE FCAST_SALES_NEXT_PERIOD FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Forecasted sales for the next period
EQ lead(fcast_sales,1,time,LEVELREL time_levelrel)

DEFINE FCAST_UNITS_NEXT_PERIOD FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Forecasted units for the next period
EQ lead(fcast_units,1,time,LEVELREL time_levelrel)

```

SHARE MEASURES

```

DEFINE SHARE_SALES_CHAN FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Share of sales for current channel as compared to All Channels
EQ (sales/sales(channel '1')) * 100

DEFINE SHARE_SALES_CUST FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Share of sales for current customer as compared to All Customers
EQ (sales/sales(customer '1')) * 100

DEFINE SHARE_SALES_PROD FORMULA DECIMAL <CHANNEL CUSTOMER PRODUCT TIME>
LD 'Share of sales for current product as compared to All Products
EQ (sales/sales(product '1')) * 100

```