Developing a Data Delivery Platform With Composite Information Server

A Technical Whitepaper

Author:
Rick F. van der Lans
Independent Business Intelligence Analyst
R20/Consultancy

June 21, 2010

Sponsored by

Table of Contents

1 Summary

The Data Delivery Platform *(DDP) is a modern architecture for developing business intelligence systems where data consumers, such as reporting and analytical tools, are decoupled from data stores. This whitepaper describes how to develop such an architecture using the federation server called* Composite Information Server. *The concepts and facilities of Composite Information Server are described in such a way that developers and BI specialists get a feeling of how this product works, what its features are, and what it would mean to develop a DDP-based business intelligence architecture using this federation server.*

The Data Delivery Platform is a business intelligence architecture that offers many advantages for developing business intelligence systems, including increased flexibility of the architecture, shareable transformation and reporting specifications, easy migration to other data store technologies, cost reduction due to simplification of the architecture, easy adoption of new technology, and transparent archiving of data. The DDP can co-exist with other more well-known architectures, such as the Data Warehouse Bus Architecture, and the Corporate Information Factory.

Composite Information Server is an *open federation server* capable of presenting a heterogeneous set of data stores as one logical database. This unified view can be used by almost any reporting and analytical tool, and in addition it can be accessed by applications through service-oriented interfaces. All those tools and applications will share the transformation specifications managed by Composite Information Server.

Unlike classic ETL tools that are based on *periodic transformation*, federation servers, such as Composite Information Server, deliver *on-demand transformation*. Periodic transformation means that the data sources accessed by the reporting tools are refreshed periodically. It also means that several derived data stores must be developed and maintained to store the periodically copied data. With on-demand transformation, when a reporting tool requests data, only then is data retrieved from the data stores and only then is the data transformed. The advantages are that users can work with more timely data, there is less need for creating and managing derived data stores, and report and transformation changes can be applied more quickly. These features make a federation server ideal for developing a Data Delivery Platform.

Views and *data services* are the core building blocks of Composite Information Server. Views are used by data consumers such as analytical and reporting tools to access data using relational methods. Data services are used by consumers such as applications and websites to access data using service-oriented methods. Beyond this consuming method distinction, Composite Information Server views and data services perform common functions. As a result and for the purpose of this paper, we will focus primarily on views.

Views are used as transformation steps. Each view can hold a number of transformation steps, such as join, selection, projection, and aggregation. Views can be stacked on top of each other. For different user(group)s different sets of views can be defined. Shareable transformation specifications can be placed in views that must be used by all users. Views can also be used to transform data stored in XML documents, sequential files, MDX cubes, SOAP-based services, and Java components, to relational tables. This makes it possible to seamlessly integrate non-relational with relational data.

Composite Information Server offers several mechanisms to optimize query performance, including advanced distributed join optimization, instant and scheduled caching, and push-down of query processing to the underlying database servers. In addition, developers can see how a query is being processed. The caching features of Composite Information Server make it possible to implement periodic transformation, if needed by the users.

To summarize, Composite Information Server is a mature and feature-rich open federation server. Its modular approach and its extensive optimization technologies make it very well suited for developing a business intelligence architecture based on the Data Delivery Platform.

2 What is the Data Delivery Platform?

The *Data Delivery Platform* (DDP) is a modern architecture for developing business intelligence systems where data consumers (such as reports developed with SAP BusinessObjects WebIntelligence, SAS Analytics, JasperReport, or Excel) are decoupled from data stores (such as data warehouses, data marts, and staging areas); see Figure 1. The DDP was introduced in a number of articles published at BeyeNETWORK.com, including *The Definition of the Data Delivery Platform*. The definition of the DDP is:

> *The Data Delivery Platform is a business intelligence architecture that delivers data and meta data to data consumers in support of decision-making, reporting, and data retrieval; whereby data and meta data stores are decoupled from the data consumers through a meta data driven layer to increase flexibility; and whereby data and meta data are presented in a subject-oriented, integrated, time-variant, and reproducible style.*

The primary goal of decoupling is to get a higher level of flexibility. For example, changes made to the data stores don't automatically mean that changes must be made to the data consumers as well, and vice versa. Or, replacing one data store technology by another is easier when that data store is 'hidden' behind the DDP.

Decoupling data consumers from data stores is based on the concept of *information hiding*. This concept was introduced by David L. Parnas[1] in the 70s and was adopted soon after by object oriented programming languages, component based development, and service oriented architectures. But until now, the concept of information hiding has only received limited interest in the world of data warehousing.

The DDP can be seen as a separate business intelligence architecture, but it can also co-exist with

[1] David L. Parnas, *'Software Fundamentals, Collected Papers by David L. Parnas'*, Addison-Wesley Professional, 2001.

more well-known architectures, such as Ralph Kimball's *Data Warehouse Bus Architecture*[2], Bill Inmon's *Corporate Information Factory*[3], and his more recent architecture called *Data Warehouse 2.0*[4]. In addition, some other generic architectures exist, such as the *Centralized Data Warehouse Architecture* and the *Federated Architecture*; see the article *Which Data Warehouse Architecture is Most Successful?* by T. Ariyachandra and H.J. Watson, published in 2006.

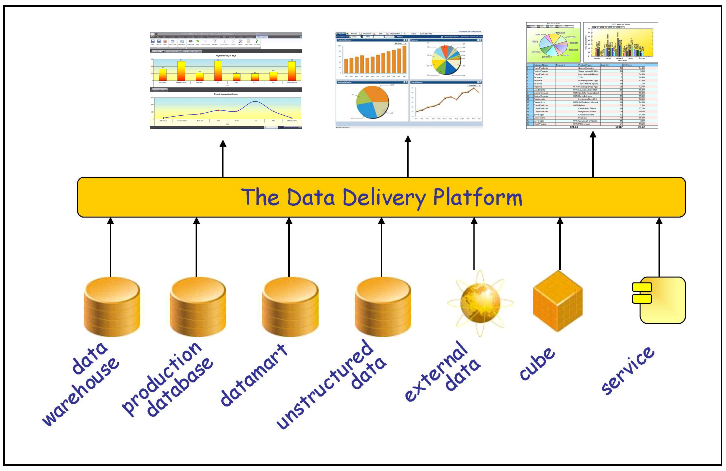


Figure 1 The Data Delivery Platform

3 Advantages of the Data Delivery Platform

Two principles are very fundamental to the Data Delivery Platform architecture: *shared specifications* and *decoupling* of data consumers and data stores. Both these principles lead to a number of advantages that are described in this section.

Most reporting and analytical tools require specifications to be entered before reports can be developed. Some of those specifications are *descriptive* and others are *transformative*. Examples of descriptive specifications are definitions of concepts; for example, a customer is someone who has bought at least one product, and the Northern region doesn't include the state Washington. But defining alternative names for tables and columns, and defining relationships between tables are also descriptive specifications. Examples of transformative specifications are 'how should country codes be replaced by country names', and 'how a set of tables should be transformed to one cube'. In the DDP those specifications are centrally managed and are shareable. The advantages resulting from shared specifications are:

Easier maintenance of specifications: Unfortunately, in most cases descriptive and transformative specifications can't be shared amongst reporting and analytical tools. So, if two users use different tools the specifications must be copied. The advantage of the DDP is that most

[2] R. Kimball et al., *The Data Warehouse Lifecycle Toolkit*, Second Edition, John Wiley and Sons, Inc. 2008.
[3] W.H. Inmon, C. Imhoff, and R. Sousa, *Corporate Information Factory*, Second Edition, John Wiley and Sons, Inc., 2001.
[4] W.H. Inmon, D. Strauss, and G. Neushloss, *DW 2.0, The Architecture for the Next Generation of Data Warehousing*, Morgan Kaufmann Publishers, 2008.

of those specifications can be defined once and can be used by all the tools. Therefore, maintaining existing and adding new specifications is easier.

More consistent reporting: If all reporting and analytical tools use the same specifications to determine results, the results will be consistent, even if the tools are from different vendors. This improves the perceived quality of and trust in the business intelligence environment.

Increased speed of report development: Because most specifications already exist within the DDP and can be re-used, it takes less time to develop a new report. Development can focus primarily on the use of the specifications.

As indicated, in a DDP data consumers are decoupled from the data stores. This means that the data consumers don't know which data stores are being accessed: a data warehouse, a data mart, or an operational data store. Nor do they know which data store technologies are being accessed, an Oracle or DB2 database or maybe Microsoft Analysis Service. The advantages resulting from this decoupling are:

Easier data store migration: Data store independency means that if a report that accesses a particular data store can easily be migrated to another data store. The report's queries can be redirected through the DDP to that other data store. For example, if a report is currently accessing a data mart, migrating it to the data warehouse doesn't require any changes in the report definition. The same applies if a need exists to migrate from a relational database to MDX-base technology, or if SQL Server has be replaced by Netezza. In most cases, these changes will have no impact on the reports. In short, if a DDP is in place, migration to another data store (technology) is easy. There are various reasons why an organization wants to migrate, for example, they may want to use technology that offers faster query performance, or data storage is outsourced and needs to be accessed differently.

Cost reduction due to simplification: If the DDP is installed in an existing business intelligence architecture, for example one based on the Corporate Information Factory architecture, the DDP makes it possible to simplify the architecture. Data marts and cubes can be removed and the existing reports must be redirected to another data store, which, as indicated, is easy to do with the DDP. The advantage of this simplification of the architecture is cost reduction.

Increased flexibility of the architecture: With less code and fewer specifications, it is easier to change a system. The DDP makes it possible to simplify the architecture and to work with shareable specifications. The effect is that new user requirements and demands can be implemented faster. In other words, the time to market for new reports is shortened.

Seamless adoption of new technology: New database and storage technology has appeared on the market, such as data warehouse appliances, analytical databases, columnar databases, and solid state disk technology. As indicated, because the DDP separates the data consumers from the data stores, replacing an existing data store technology with a new one is relatively easy and has no impact on the reports.

Transparent archiving of data: Eventually data warehouses become so big that 'older' data has to be archived. But if data is old, it doesn't always means that no one is interested in it anymore. The

DDP can hide where and how archived data is stored. Archiving data, meaning data is taken out of the original data store and moved to another, can be hidden for the data consumers. If users are still interested in all the data, the DDP can combine the non-archived data with the archived data store. The effect might be that the performance is slower, but reports don't have to be changed. Therefore, the DDP hides that some data has been archived.

4 Open versus Closed Federation Servers

But how should a Data Delivery Platform be developed? Currently, the best approach is to use a *federation server* (sometimes called an *enterprise information integration* tool and today a critical component in a *data virtualization* suite). A federation server, like the DDP, presents multiple heterogeneous data stores as one logical data store to the applications and tools. To them accessing the federation server is very similar to logging on to one large database. Without knowing it, reports join data coming from different data stores, and even data stores that use different storage models and concepts. Because a federation server presents this unified view of the data, it can act as the core building block of a DDP.

Various analytical and reporting tools implement a form of federation technology themselves. For example, Qlikview is more than capable of accessing a set of heterogeneous data stores, the same applies for the tools of SAP/Business Objects, IBM/Cognos, and many others. For example, the Universe concept in Business Objects can be seen as federation technology. However, all the specifications entered in these products are only usable by the tools themselves (or tools of the same vendor); see Figure 2. These are non-sharable specifications. Therefore, these federation technologies are called *closed federation servers*.

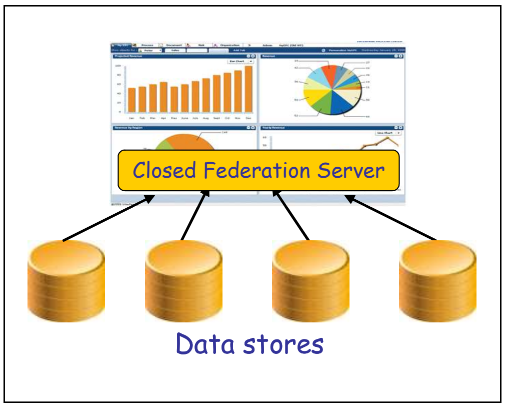


Figure 2 A Closed Federation Server with only one Reporting Tool

Like a closed federation server, an *open federation server* can access many different data stores, but in addition it also allows access for any BI tool and application; see Figure 3. The effect is that the specifications stored in the federation server become shareable. If, for example, we define that the Northern Region doesn't include the state Washington, each and every tool that accesses the federation server can make use of that same specification, whether it's Excel or SAS Analytics. This improves the maintainability of the environment, but it also minimizes the chance that users using different tools see inconsistent results.

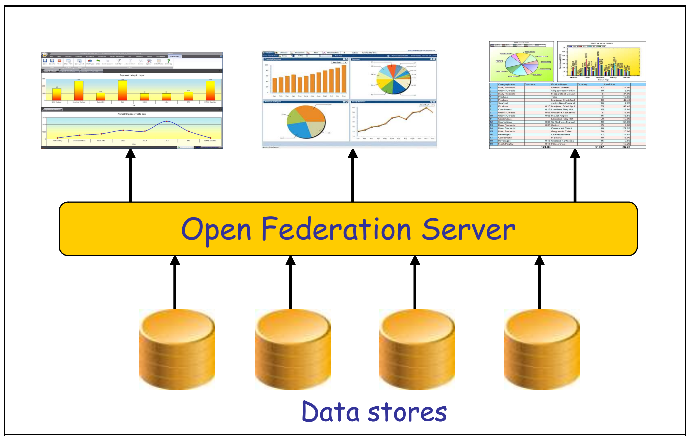


Figure 3 An Open Federation Server with Shareable Specifications

5 What is Composite Information Server?

A number of federation servers is available. This whitepaper describes how to develop the DDP with one of the more popular open federation servers called *Composite Information Server* from Composite Software.

Composite Software, which is based in San Mateo, California was founded in 2002. This was also the year the first version (1.0) of Composite Information Server was released. Currently, Version 5.1 of Composite Information Server is being shipped. The Composite Information Server forms the foundation of the Composite Data Virtualization Platform. On the OEM front, the product is also distributed by Netezza, IBM/Cognos, Informatica, Pitney Bowes, SAS, BMC, and many more.

Besides being able to access almost any relational database server, including DB2, Microsoft SQL Server, MySQL, Netezza, and Oracle, Composite Information Server makes it possible to access XML documents, MDX databases, flat files, spreadsheets, and other non-relational data stores. Composite Information Server will 'flatten' these non-relational stores to tables. This makes it possible that, for example, an Excel spreadsheet can join data stored in an SAP InfoCube with an XML document, or that a Cognos report combines data in an Oracle database with data from Microsoft Analysis Service.

Together with their other product *Composite Application Data Services*, even modules inside applications, such as those of Oracle, Salesforce.com, and SAP, can be turned into tables that can be queried using any type of tool. In fact, any Java component can be queried as if it's a table. So again, this means that, for example, a report can combine data from an external source by using SOAP with internal data stored in a relational database.

Reporting tools can use any of the popular API's, including JDBC, ODBC, and ADO.NET, to access data. In addition, Composite Information Server can present data as services through SOAP, REST, and XQuery. In addition, applications can access the data through JMS as well.

Figure 4 contains a diagram that shows the architecture of Composite Information Server and includes most of the data stores that can be accessed and most of the API's being supported.

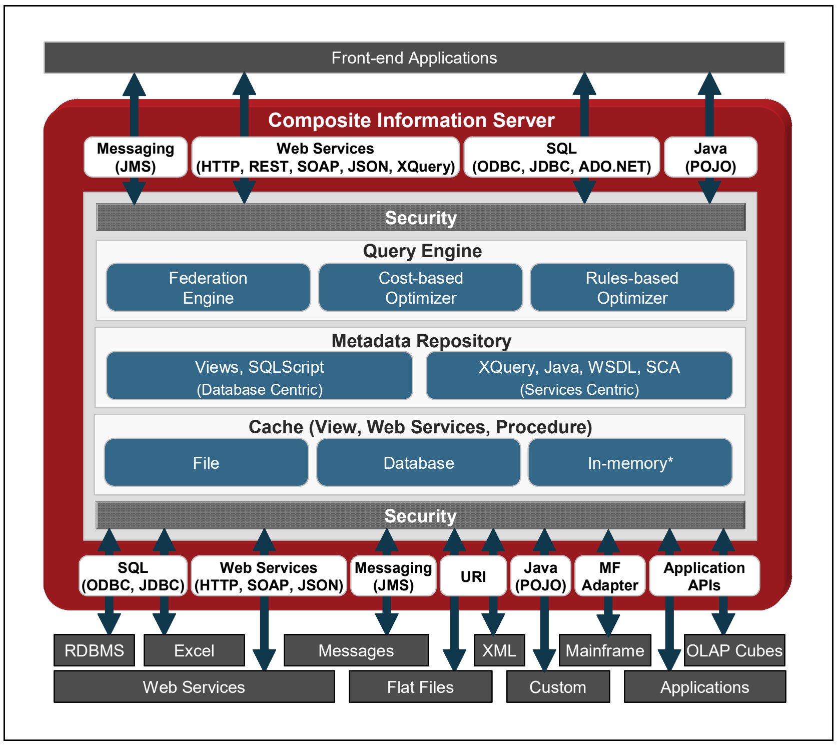


Figure 4 An Overview of the Architecture of Composite Information Server (source Composite Software)

What this whitepaper shows is that all the specifications relating to the data are shared by all tools and applications accessing Composite Information Server.

6 On-Demand Transformation versus Periodic Transformation

In most current business intelligence architectures the format and the contents of the data stored in the source systems is quite different from how the users want to see the data in their reporting and analytical tools. Here are some examples: in the source systems customer data might be spread out over multiple databases while users want to have an integrated view, data is the source systems might be heavily coded while users want to see meaningful values, historical data might be missing from the source systems while users need it for trend analysis, or the values of data elements in source systems might be incorrect (defective data) while users want to work with correct data, and so on. To summarize, source data has to be 'massaged' before users can use it. This whole process is sometimes referred to as *data transformation*.

In classic business intelligence architectures data transformations are executed by ETL jobs (Extract Transform Load). In most cases all the data transformation is not done in one step, but in multiple where intermediate results are stored in various data stores, such as staging areas, operational data stores, and data marts. Additionally, the ETL jobs are scheduled to run periodically, for example, once a week or every midnight. So all the data transformation takes place before users run the reports. In other words, the transformation of data is done in advance and periodically. In this whitepaper we will refer to this form of transformation as *periodic transformation*; see Figure 5. In most cases, ETL tools are used for data transformations.

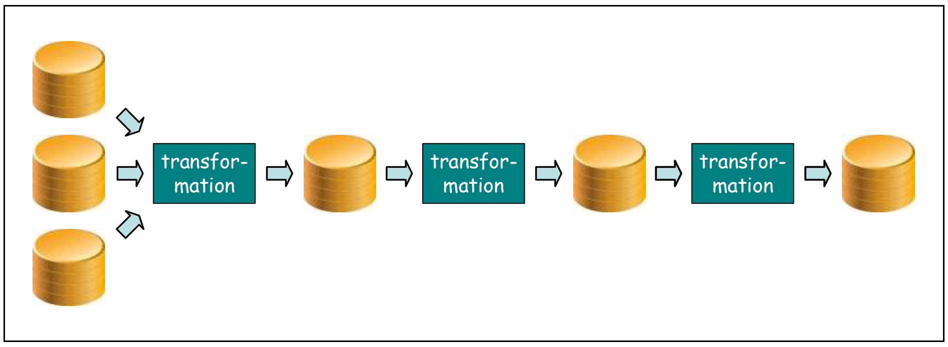


Figure 5 Periodic Transformation Requires Data Stores

Federation servers offer *on-demand transformation*. When the user executes his report or runs his analysis, only then is data queried and integrated. It's almost as when the report is executed, the data flows from the databases through the federation server to the reports. The federation server will do the necessary transformations. It's comparable to buying a sandwich created right in front of you (on-demand), versus one that has been prepared in a factory, wrapped in plastic, and sent to a store (periodic).

On-demand transformation offers the following advantages to the users:

- Users can work with more timely data.
- Less need exists for creating derived data stores, which will reduce costs.
- With less data stores, the overall architecture will be simpler, and that means the whole architecture is more flexible.
- The time-to-market for new reports is better; creating a new report might only require a few hours work before the required data is available.
- Sometimes reports must be developed that will run only once, on occasion these are called throw-away reports. If a report is used only once on-demand transformation fits better.

But some disadvantages of on-demand transformation must be considered as well:

- Transformations are done repeatedly. Every time a report accesses data, that data will go through the same transformations (unless a cache is used, see Section 16). Compare this to periodic transformation, where data is only transformed once or a few times, in fact only when data is added or changed. Note that this is only true for those solutions where only new data is copied; in some organizations all the data is copied every time. In that case, the same transformations are also done repeatedly.
- The transformation might be so complex that it takes too long.

- Some production systems overwrite old data when new data is entered. If this historical data is needed, it must be copied from the source for retention in a special data store (for example a staging area or operational data store) using a periodic transformation approach.

Whether data is transformed on-demand or periodically, each tool involved in data transformation should at least support the following types of operations:

- Transformation: Values in columns are changed by applying, for example, concatenations, code transformations, calculations, and string manipulations.
- Join: Data in two or more tables are combined into one table.
- Aggregation: Rows are grouped based on equal values in certain columns.
- Selection: Rows are selected based on specified conditions.
- Projection: Columns are selected, and others are removed.

One more consideration: Most organizations want to cleans defective data before it is used by the reporting and analytical tools. Special tools exist on the market to help out with this. These tools can, for example, help with the spelling of company names, or can find the correct zip code. Some of the simple forms of cleansing can be implemented with an on-demand transformation solution. But beyond these simple cases, it's recommended to perform the more complex quality functions using periodic transformation, or by cleansing the data stores accessed by the federation server.

7 How does Composite Information Server Work?

When are we allowed to call a solution a Data Delivery Platform? To which requirements should it adhere? In the article The Requirements of the Data Delivery Platform the requirements are listed for an implementation of a Data Delivery Platform-based system. An example of such a requirement is that a DDP-based system should support access to a large array of heterogeneous data store technologies and systems, including relational database technology, MDX-based database technology, and XML-storage technology, and web services. Another requirement is that a DDP-based system should support all the common types of integration, including name changes, joins, selects, aggregations, splits, projections, and cleansing.

The next sections describe how particular features and concepts are implemented in Composite Information Server (CIS). This should give readers a feeling of how CIS meets the requirements of the Data Delivery Platform.

The following features are described:

- Accessing tables stored in relational database servers.
- Integrating data stored in databases managed by different database servers.
- Presenting different table structures to different reports.
- Making XML documents and web services accessible as tables.
- Making MDX cubes accessible as tables.
- Flattening repeating groups and accessing procedures as tables.
- Tracking all the relationships between views and tables (lineage analysis).
- Making relational tables be accessible as data services.
- Caching data for performance reasons and how CIS supports periodic transformation.
- Optimization techniques for running queries on foreign databases.
- Securing the access to data.

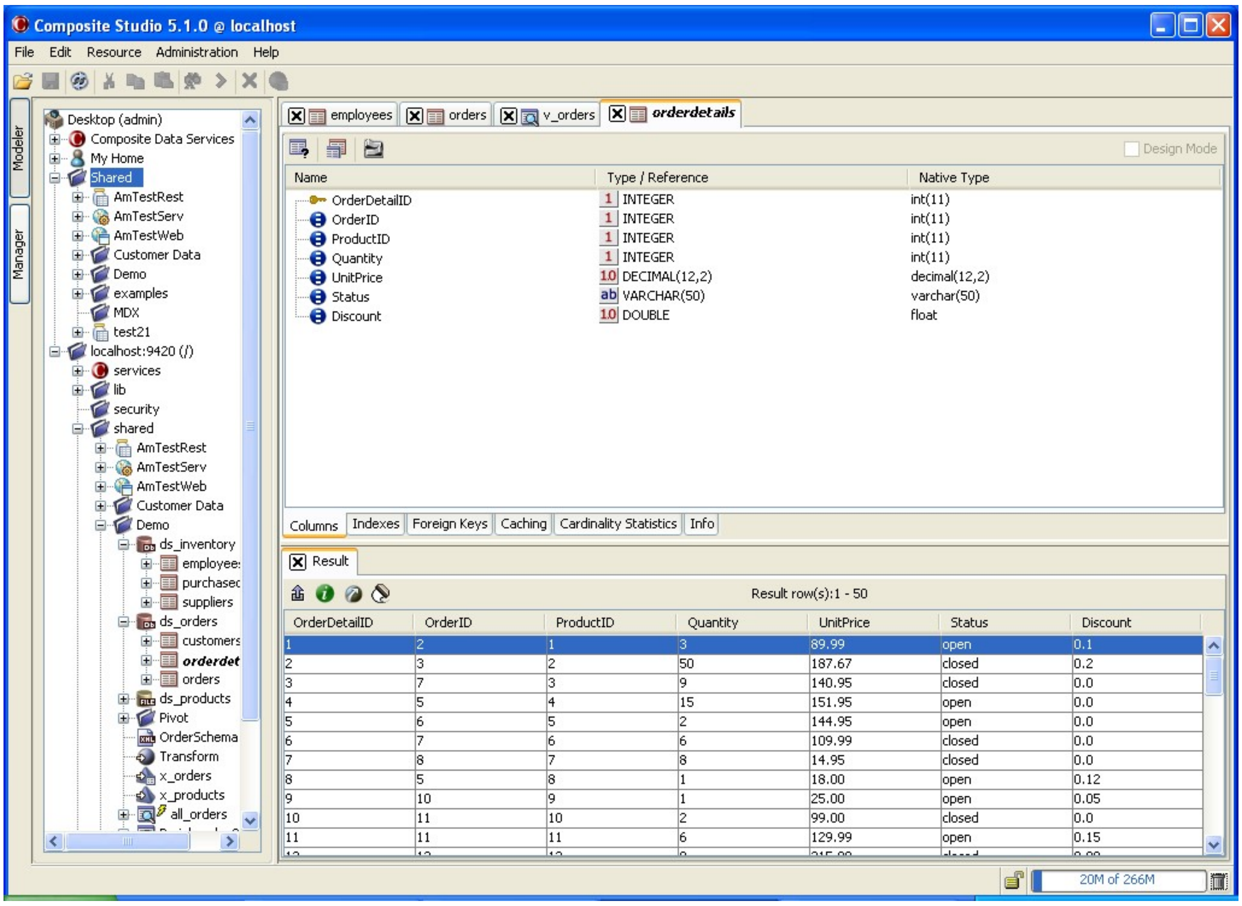
8 Accessing Data Stored in Foreign Tables

A Data Delivery Platform has to give the data consumers (the reporting tools) access to data stored in relational databases, such as Oracle11*g*, Microsoft SQL Server, and MySQL. This is also one of the easiest things to do in Composite Information Server. In this whitepaper we refer to such a table with the term *foreign table*.

Each foreign table that needs to be accessed has to be imported. This import process is quite simple. CIS will create a connection with the database server, reads data from the catalog, such as column names, data types, null specifications, and population data. This meta data is stored in the CIS catalog. Once the table is known to CIS, it becomes a *data source* that can be used within views and data services.

If the data type of a specific column is not according to that of a standard SQL data type a conversion is made automatically. For example, in Figure 6 the foreign table called `orderdetails` has some columns with a data type called `int(11)`. This data type is transformed to the standard integer data type. The transformation to standard data types is quite important when columns from different database servers are joined and their respective data types are not 100% identical.

When a data source has been created for a foreign table, it's not yet accessible by the reporting tools. First, a *view* (or *data service*) has to be created for it. Only then does the foreign table become accessible for all the tools; see Figure 7. A view in CIS is very much comparable to the view concept in SQL database servers. Basically, it's a query definition with a name and a set of columns. The contents of the view is virtual and is derived from underlying tables and views. Later in this whitepaper we will discuss the notion of cached views, meaning the contents of the view is not virtual, but is stored (temporarily) in a cache.



Name	Type / Reference	Native Type
OrderDetailID	INTEGER	int(11)
OrderID	INTEGER	int(11)
ProductID	INTEGER	int(11)
Quantity	INTEGER	int(11)
UnitPrice	DECIMAL(12,2)	decimal(12,2)
Status	VARCHAR(50)	varchar(50)
Discount	DOUBLE	float

Result row(s):1 - 50

OrderDetailID	OrderID	ProductID	Quantity	UnitPrice	Status	Discount
1	2	1	3	89.99	open	0.1
2	3	2	50	187.67	closed	0.2
3	7	3	9	140.95	closed	0.0
4	5	4	15	151.95	open	0.0
5	6	5	2	144.95	open	0.0
6	7	6	6	109.99	closed	0.0
7	8	7	8	14.95	closed	0.0
8	5	8	1	18.00	open	0.12
9	10	9	1	25.00	open	0.05
10	11	10	2	99.00	closed	0.0
11	11	11	6	129.99	open	0.15

Figure 6 Transforming Product Dependent Data Types to the Standard Data Types

Accessing a view is as simple as accessing a table in an ordinary database. For example, if a tool uses SQL and an API, such as ODBC, JDBC, and OLE DB, to log on to CIS, the latter will show the view as a table. The reporting tool won't see the difference between CIS and a classic SQL-based database server.

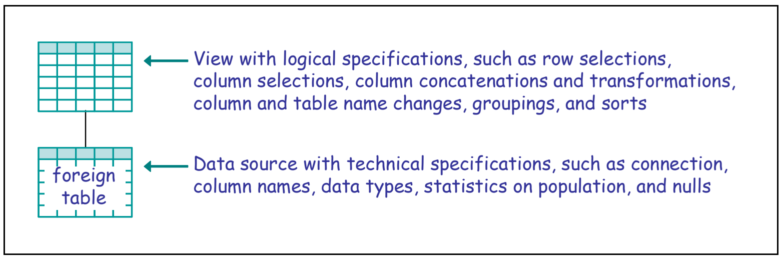

View with logical specifications, such as row selections, column selections, column concatenations and transformations, column and table name changes, groupings, and sorts
foreign table
Data source with technical specifications, such as connection, column names, data types, statistics on population, and nulls

Figure 7 A View on a Foreign Table

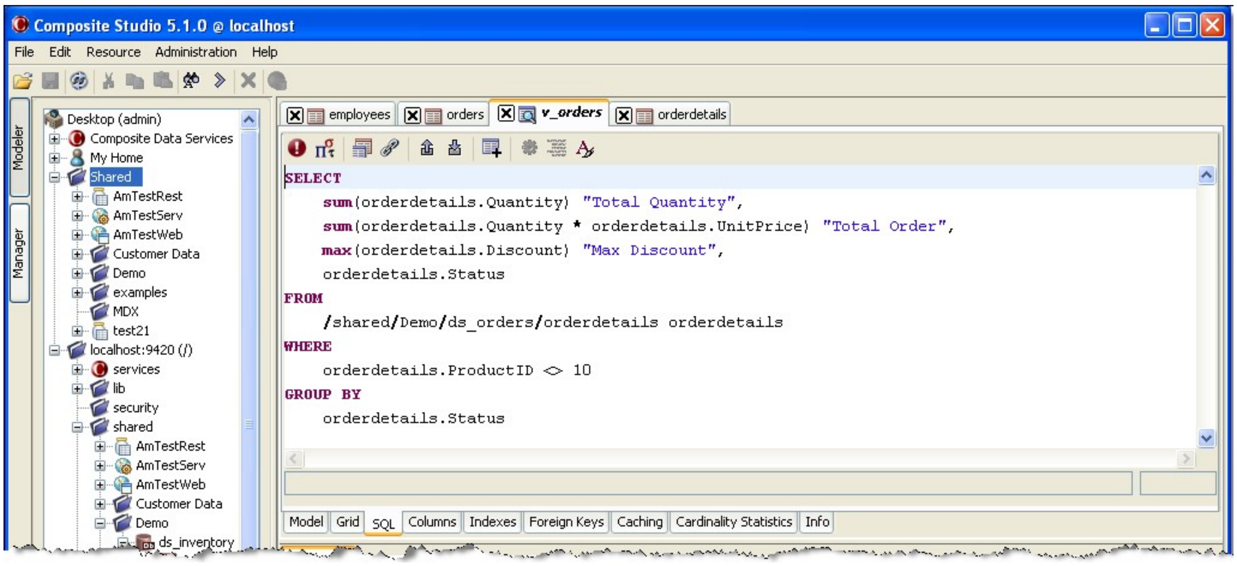
A view may be defined in such a way that it contains exactly the same contents as the underlying foreign table: all the rows and all the columns. But if required, the view contents can be adapted, or in other words, transformation operations can be specified. Here are some of the supported transformation operations:

- Conditions can be specified to select a subset of all the rows from the foreign table.

- Columns in the foreign table can be removed from the view.
- Columns in the foreign table can be concatenated.
- Names of the columns in the foreign table and the name itself can be changed.
- New virtual and derivable columns can be added.
- Group-by operations can be specified to aggregate data.
- Statistical functions can be specified.

Almost all the features of SQL are available to transform the structure and contents of the underlying foreign table to a view. Composite also provides several procedural language options to perform additional functions that are beyond the limits of standard SQL.

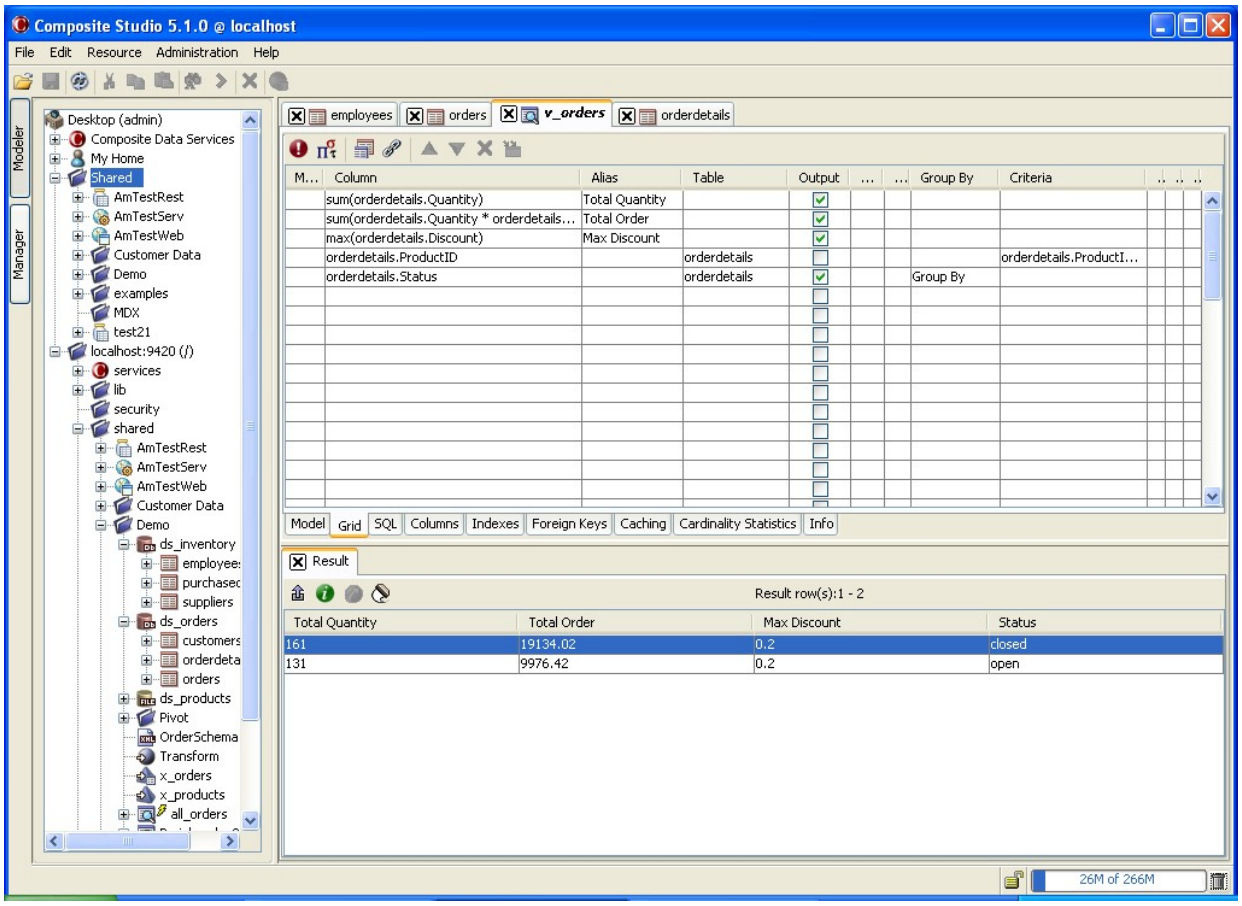
All these transformations can be defined in two ways. One, by writing SQL directly, or two, by filling in a grid with specifications from which SQL is generated automatically; see Figures 8 and 9. The first figure contains the SQL statement and the second the grid. If one is changed by the developer, the other is automatically changed accordingly. The tabs are just different views of the same definition.



```
SELECT
    sum(orderdetails.Quantity) "Total Quantity",
    sum(orderdetails.Quantity * orderdetails.UnitPrice) "Total Order",
    max(orderdetails.Discount) "Max Discount",
    orderdetails.Status
FROM
    /shared/Demo/ds_orders/orderdetails orderdetails
WHERE
    orderdetails.ProductID <> 10
GROUP BY
    orderdetails.Status
```

Figure 8 **Defining a View Using SQL**

Views can be defined on top of other views. This stacking of views can have different purposes. For example, it could be useful if two user(group)s have common transformation needs plus both have specific transformation needs. The common specifications can be placed in an intermediate view and the group specific transformations can be specified in separate views; see Figure 10. Another example why it would make sense to stack views on top of each other is when different reports or different user(groups) want to see the data in a specific foreign table differently. For example, one group might want to see the data in a slightly aggregated way, whereas another group is only interested in a subset of all the rows from the table. In this case different views are defined on the same data source.


Composite Studio 5.1.0 @ localhost
File Edit Resource Administration Help
employees
orders
v_orders
orderdetails
Model Grid SQL Columns Indexes Foreign Keys Caching Cardinality Statistics Info
Result
26M of 266M

M...	Column	Alias	Table	Output	...	...	Group By	Criteria
	sum(orderdetails.Quantity)	Total Quantity		✓				
	sum(orderdetails.Quantity * orderdetails...	Total Order		✓				
	max(orderdetails.Discount)	Max Discount		✓				
	orderdetails.ProductID		orderdetails					orderdetails.ProductI...
	orderdetails.Status		orderdetails	✓			Group By	

Result row(s):1 - 2

Total Quantity	Total Order	Max Discount	Status
161	19134.02	0.2	closed
131	9976.42	0.2	open

Figure 9 Defining a View Using a Grid

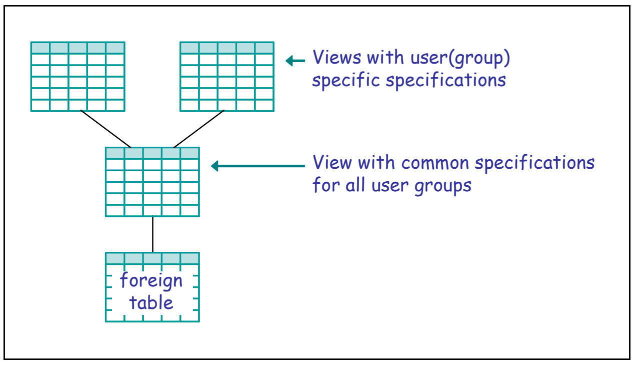

Views with user(group) specific specifications
View with common specifications for all user groups
foreign table

Figure 10 View with Shareable Specifications

9 Integrating Data from Different Databases

Reports normally require that data in different tables and even tables in different systems are merged together. In CIS this is done by creating a view that has as basis the join of two or more foreign tables. Those tables can be part of the same database, or from separate databases. They can even be tables stored in databases of different database servers. For example, Figure 11 shows a join of tables stored in respectively an Oracle, DB2, and Netezza database. Specifying such joins is quite simple. If foreign tables have been imported, views can be defined on them.

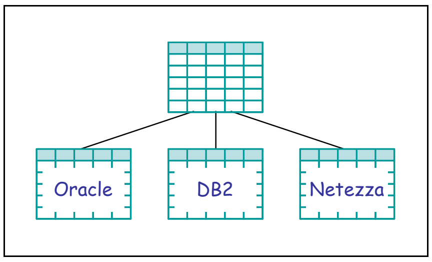


Figure 11 Join over Different Database Servers

CIS offers various ways to create joins. For example, it can be done graphically; see Figure 12 where five tables are joined. When a join is created graphically, the query that contains the join is created as well. At the bottom of that same figure a set of tabs is shown. The tab called `SQL` contains the real query, and the tab called `Grid` holds the virtual contents of the view. To create a join developers can also write the query by hand.

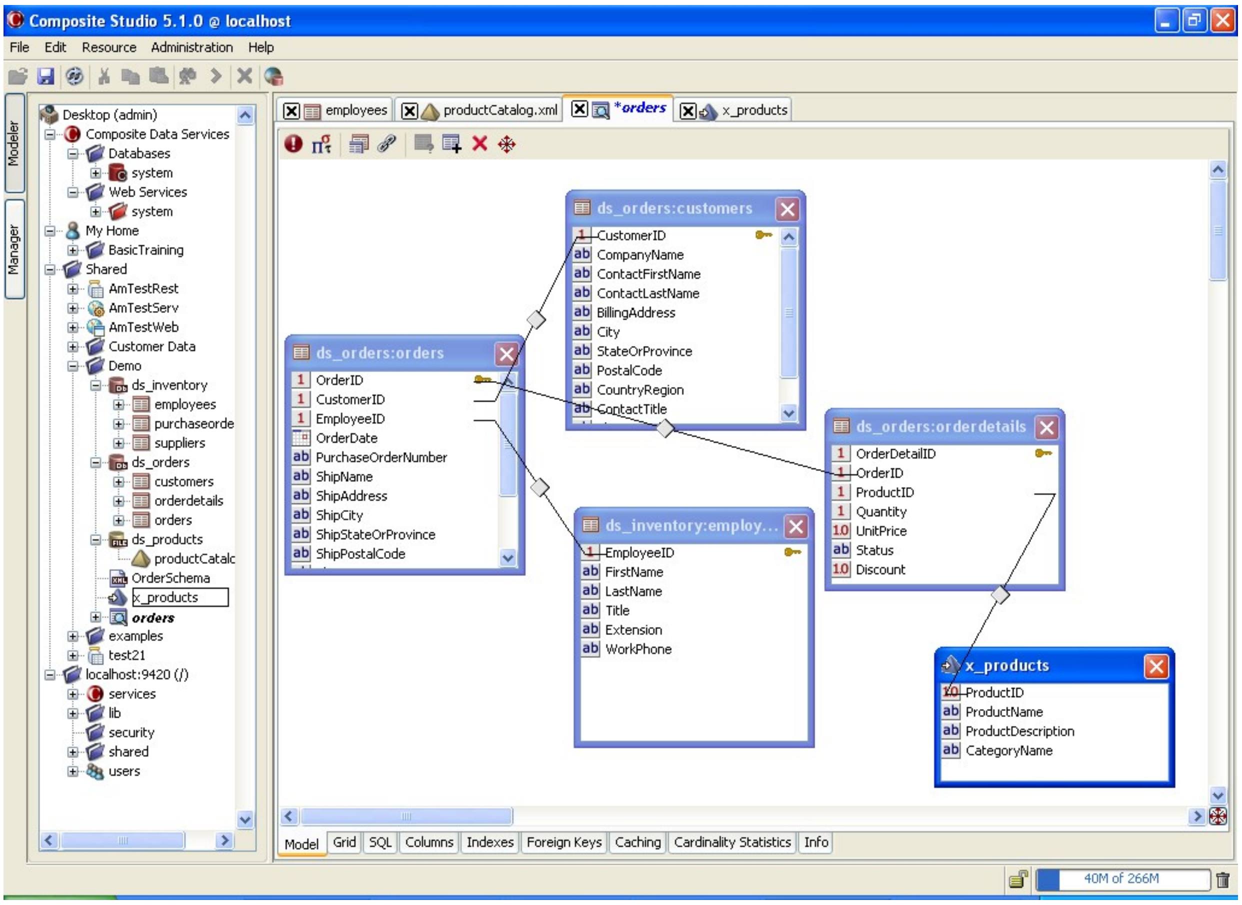
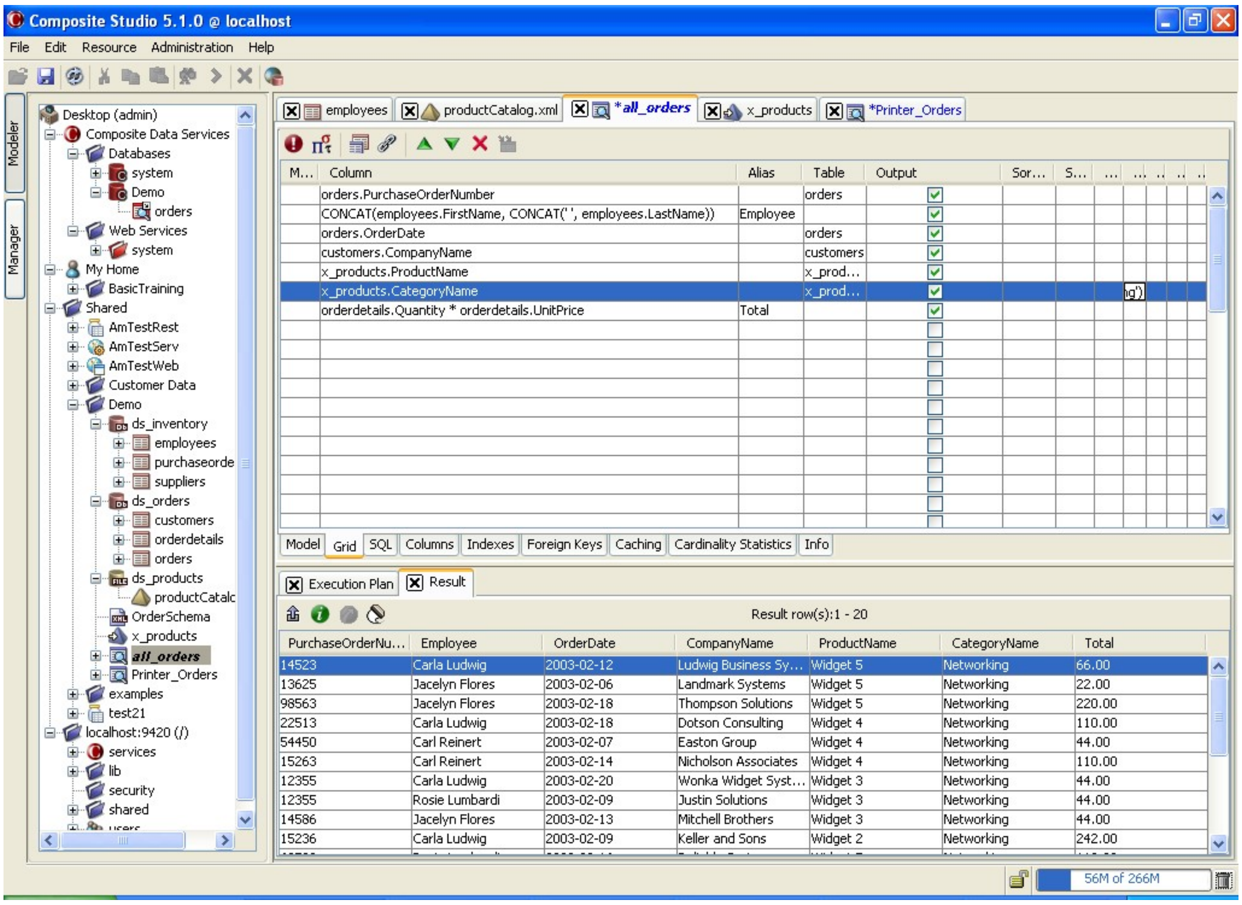


Figure 12 Graphical Style of Entering a Join

The view that holds the join, can have extra conditions, aggregations, and so on. Figure 13 contains a column called `Table` in which the tables are listed that are being joined together. The column called `Column` contains all the columns that must be presented, including concatenations of columns. At the bottom of that same figure, the result of the join with all the extra conditions and transformations are shown.



M...	Column	Alias	Table	Output	Sor...	S...
	orders.PurchaseOrderNumber		orders	✓		
	CONCAT(employees.FirstName, CONCAT(' ', employees.LastName))	Employee		✓		
	orders.OrderDate		orders	✓		
	customers.CompanyName		customers	✓		
	x_products.ProductName		x_prod...	✓		
	x_products.CategoryName		x_prod...	✓		
	orderdetails.Quantity * orderdetails.UnitPrice	Total		✓		

PurchaseOrderNu...	Employee	OrderDate	CompanyName	ProductName	CategoryName	Total
14523	Carla Ludwig	2003-02-12	Ludwig Business Sy...	Widget 5	Networking	66.00
13625	Jacelyn Flores	2003-02-06	Landmark Systems	Widget 5	Networking	22.00
98563	Jacelyn Flores	2003-02-18	Thompson Solutions	Widget 5	Networking	220.00
22513	Carla Ludwig	2003-02-18	Dotson Consulting	Widget 4	Networking	110.00
54450	Carl Reinert	2003-02-07	Easton Group	Widget 4	Networking	44.00
15263	Carl Reinert	2003-02-14	Nicholson Associates	Widget 4	Networking	110.00
12355	Carla Ludwig	2003-02-20	Wonka Widget Syst...	Widget 3	Networking	44.00
12355	Rosie Lumbardi	2003-02-09	Justin Solutions	Widget 3	Networking	44.00
14586	Jacelyn Flores	2003-02-13	Mitchell Brothers	Widget 3	Networking	44.00
15236	Carla Ludwig	2003-02-09	Keller and Sons	Widget 2	Networking	242.00

Figure 13 **Extending a Join with Extra Conditions and Column Transformations**

So, basically, developing a join of foreign tables is as simple as creating a number of data sources and then writing a query with a join.

10 Presenting Different Table Structures to Different Reports

Foreign tables have structures typically organized for the applications and tools accessing them directly. But usually they don't have the right structure for reports and other uses. For example, some tools prefer the tables to have a *star schema* arrangement, others prefer a *snowflake schema* arrangement, and a third group likes to see the data fully normalized[5]. By creating multiple levels of views in CIS, for each tool the ideal table structures can be defined.

Let's illustrate this with an example. Imagine that a set of foreign tables doesn't have the right normalized structure. A set of views can be defined in CIS that transforms the tables to the required normalized form; see Layer 2 in Figure 14. Next, on top of those views a set of views is defined that contains transformations that apply to all the users; this would be Layer 3 in that same figure. Subsequently, for each tool a separate set of views is defined that presents the data in a form suitable for that tool; Layer 4. This could mean that one set of tables has a star schema arrangement, and the other a snowflake schema arrangement, and the third a classic normalized arrangement. This might all be dependent on the tools used by the users. In other words, Layer 4 will contain the specifications that are specific to a user or tool, while Layers 2 and 3 contain shareable specifications that apply to all the users.

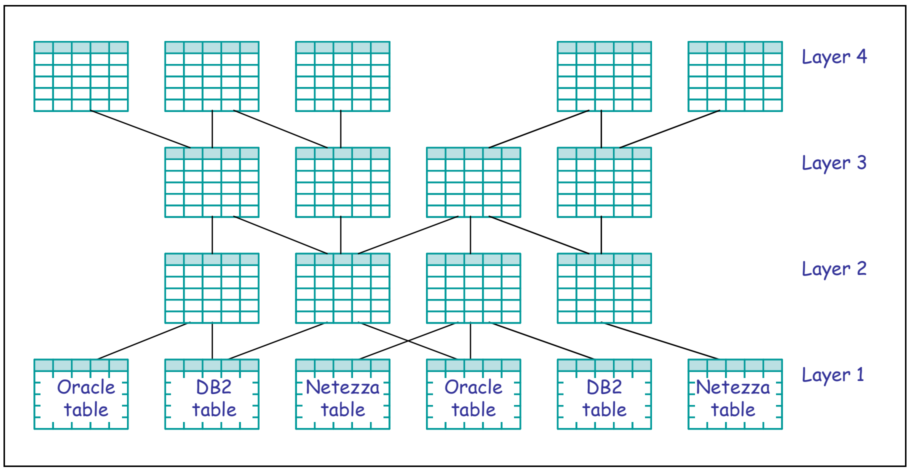


Figure 14 Levels of Views

Another option could be that the views on Layer 2 are modeled according to the design principles of *Data Vault*, and that Layer 3 contains the more user specific specifications.

Because so many specifications can be entered in the definitions of the views, there is far less need to have those specifications inside the reporting and analytical tools themselves. This frees those tools to focus on their strengths, such as analytics, visualization, reporting, and drill-downs. So, even if these tools support light federation capabilities themselves, with this shared specifications approach, these limited functions won't be needed.

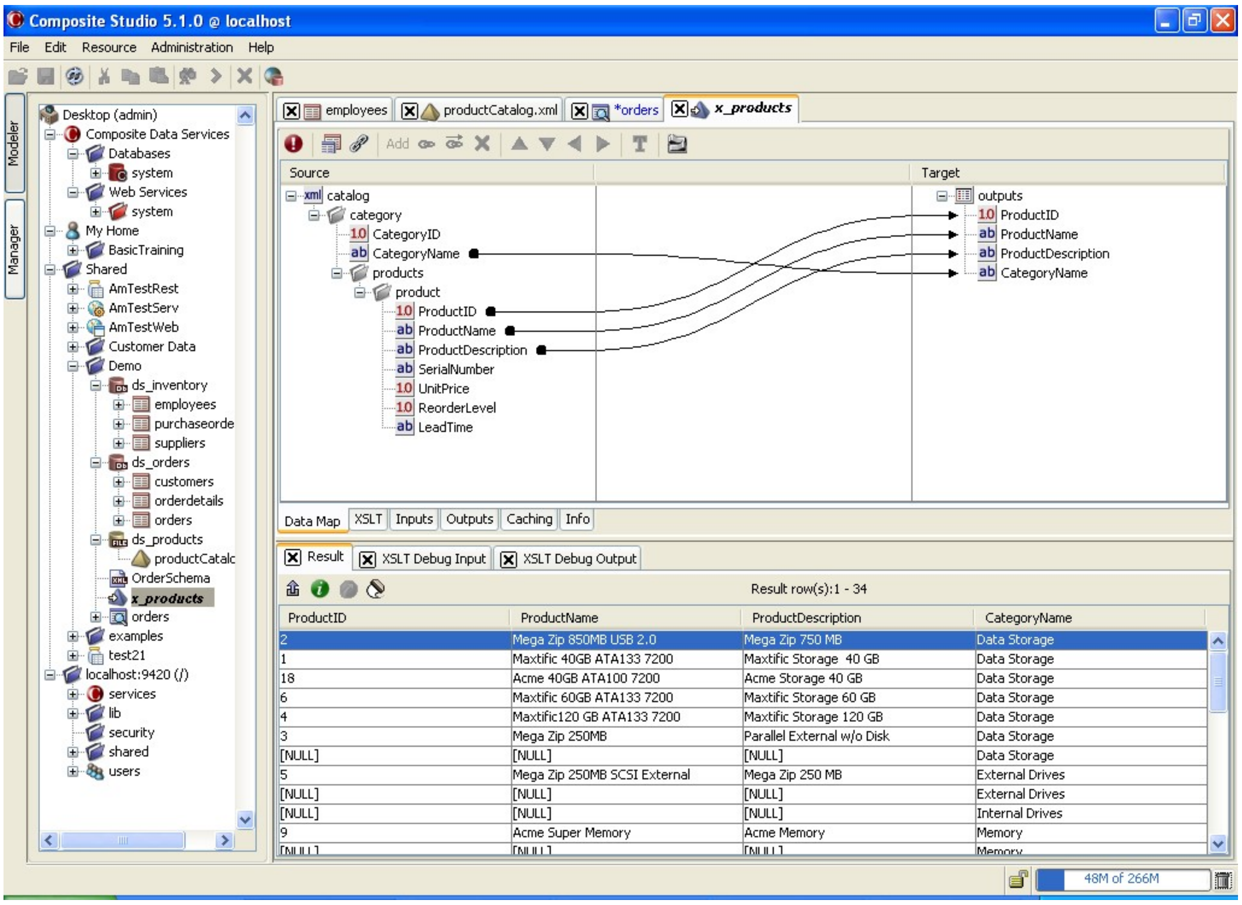
11 Transforming XML Documents and Web Services to Tables

Not all data needed for analytics and reporting is stored in relational tables. It can be stored in all kinds of data stores and formats. This section shows how data stored in XML documents and how

[5] M. Golfarelli and S. Rizzi, *Data Warehouse Design – Modern Principles and Methodologies*, McGraw-Hill, 2009.

data only accessible through SOAP web services, can be imported into CIS and becomes accessible as tables.

Before data in an XML document can be used, it has to be *flattened*, or in other words, the hierarchical structure of the XML document has to be turned into a flat relational table. In CIS, the mechanism to do this is by defining a data source for the document. A special module of CIS is designed for developing data sources that flatten XML documents; see Figure 15.



Result row(s):1 - 34

ProductID	ProductName	ProductDescription	CategoryName
2	Mega Zip 850MB USB 2.0	Mega Zip 750 MB	Data Storage
1	Maxtific 40GB ATA133 7200	Maxtific Storage 40 GB	Data Storage
18	Acme 40GB ATA100 7200	Acme Storage 40 GB	Data Storage
6	Maxtific 60GB ATA133 7200	Maxtific Storage 60 GB	Data Storage
4	Maxtific120 GB ATA133 7200	Maxtific Storage 120 GB	Data Storage
3	Mega Zip 250MB	Parallel External w/o Disk	Data Storage
[NULL]	[NULL]	[NULL]	Data Storage
5	Mega Zip 250MB SCSI External	Mega Zip 250 MB	External Drives
[NULL]	[NULL]	[NULL]	External Drives
[NULL]	[NULL]	[NULL]	Internal Drives
9	Acme Super Memory	Acme Memory	Memory
[NULL]	[NULL]	[NULL]	Memory

Figure 15 Transforming a Hierarchical XML Structure to a Flat Table

In this figure, the column called `Source` contains the structure of the XML document, and the column called `Target` contains the required table structure. A transformation is specified by linking elements from the XML document to columns in the data source (slightly curly arrows). The arrows indicate how the hierarchical structure of the XML document should be mapped to a table with columns. This XML document contains a number of categories, and each category might consist of a set of products. Each product has a product id, name, description, serial number, and some other characteristics. On the right hand side of that figure you see the flat relational structure. This data source contains a row for each product, and for each product a few columns of which one is the category. At the bottom of Figure 15 you can see the contents of the data source.

As with the foreign tables, to make data available for access, a view has to be defined on top of the data source; see Figure 16. If certain rows of the document must be left out, or if certain transformations must be applied, they can be defined in this view.

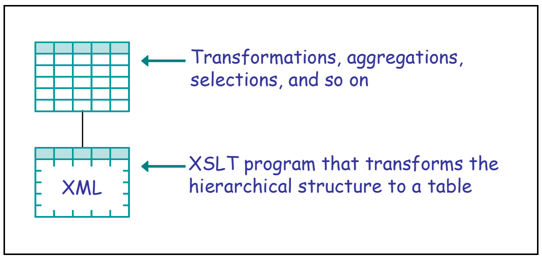

Transformations, aggregations, selections, and so on
XSLT program that transforms the hierarchical structure to a table
XML

Figure 16 Relational View on XML Document

This view on top of a data source can be used like any other view, so, for example, data stored in XML documents can be joined with data stored in relational tables; see Figure 17.

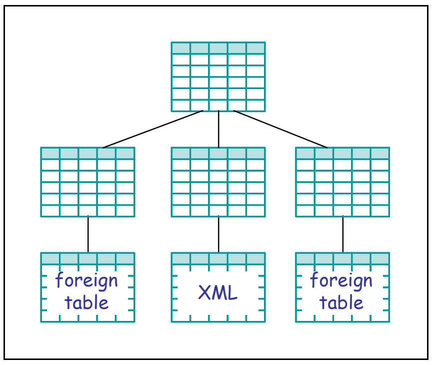


Figure 17 Joining Relational with XML Data

Behind the scenes, the language used for this transformation is the standardized language called XSLT; see one of the tabs at the bottom of Figure 15. This language is used for executing the transformation. The graphical presentation of XSLT offers only limited functionality. If developers want, they can use the full power by coding XSLT by hand.

In the same way that XML documents can accessed as relational tables, so can web services with SOAP interfaces be accessed as relational tables; see Figure 18. The result of a SOAP service is an XML document. The only difference is that to call a service, input parameters must be specified, and the result is known when the web service execution is finished. But besides those differences, developing a data source for a SOAP service is comparable to developing one for an XML document.

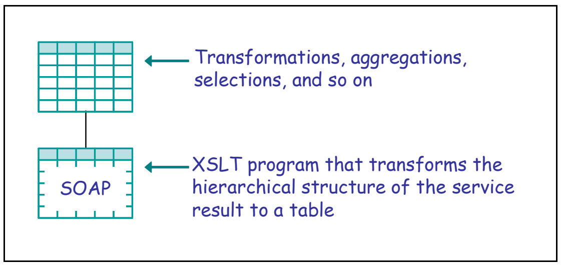

Transformations, aggregations, selections, and so on
XSLT program that transforms the hierarchical structure of the service result to a table
SOAP

Figure 18 Relational View on SOAP Services

Why would we be interested in joining relational tables with the result of a web service? There are several reasons. A service might give access to an external website, one that delivers external data, such as demographic data, weather-related data, or price information of competitive products. It might also be that data stored inside a packaged application can only be accessed through a web service, one that has been pre-defined by the vendor. More and more internal and external information is accessible through web services. Therefore, CIS makes it possible to integrate all these data sources and make them accessible as if they are flat tables.

Note: Besides SOAP, CIS also supports service-oriented data source interfaces based on JSON and HTTP.

12 Making Data Stored in MDX Cubes Available as Relational Data

In data warehouse environments, data is sometimes stored in MDX cubes. This is a storage technology not based on tables and columns, but on *dimensions* and *hierarchies*. To be able to access data in MDX cubes the language MDX must be used.

Microsoft's MDX (MultiDimensional Expressions) was first introduced as part of their OLE DB for OLAP specification in 1997 and has quite some support in the market. The specification was quickly followed by a commercial release of Microsoft OLAP Services 7.0 in 1998 and later by Microsoft Analysis Services. Many vendors have currently implemented the language, including Oracle/Hyperion (TM1), Microstrategy, SAS, and SAP, and vendors on the client side, such as IBM/Cognos, SAP/Business Objects, and Microsoft.

MDX is a powerful language for querying MDX cubes, but there a few limitations. For example, it can't join relational tables with cubes, nor can data in different cubes be joined. CIS allows cubes to be wrapped as data sources. Afterwards, if a view is defined on the data source the data cube can be integrated with any other table, cube, or XML document.

Creating a data source on MDX is comparable to creating one on an XML document, except that XSLT is not used internally but MDX. The MDX query can be defined using a graphical user interface or by writing MDX directly

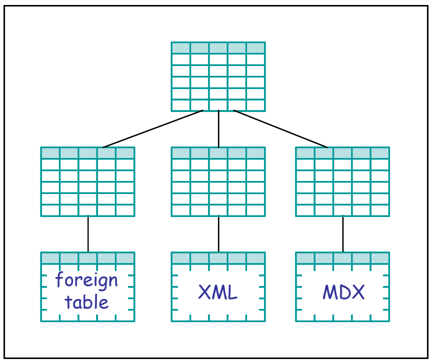


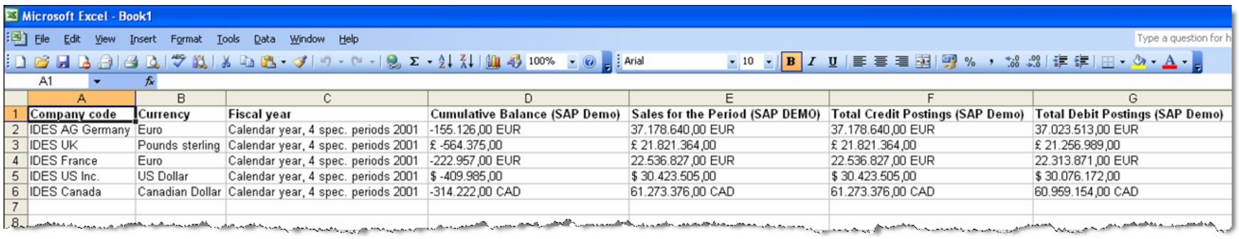
Figure 19 Joining MDX Cubes with Relational Data and XML Documents

Being able to access MDX cubes as tables offers several practical advantages:

- Data stored in MDX cubes can be joined with data stored in tables, XML documents, web services, and other cubes.

- If data is stored in MDX cubes, it can only be accessed by tools supporting MDX. By creating a data source and view on top, it can be accessed using SQL, or in other words, it can be accessed by almost all tools available on the market. For example, data stored in an MDX cube can be accessed by Excel; the result might look like Figure 20.

- Migration to MDX becomes easier. If certain SQL queries on a set of foreign tables are slow, and if we expect that running similar queries on an MDX implementation will be faster, the data can be moved to a cube, and CIS will hide the fact that MDX is be used; see Figure 21. No rewrite of the report is required.



	A	B	C	D	E	F	G
1	Company code	Currency	Fiscal year	Cumulative Balance (SAP Demo)	Sales for the Period (SAP DEMO)	Total Credit Postings (SAP Demo)	Total Debit Postings (SAP Demo)
2	IDES AG Germany	Euro	Calendar year, 4 spec. periods 2001	-155.126,00 EUR	37.178.640,00 EUR	37.178.640,00 EUR	37.023.513,00 EUR
3	IDES UK	Pounds sterling	Calendar year, 4 spec. periods 2001	£ -564.375,00	£ 21.821.364,00	£ 21.821.364,00	£ 21.256.989,00
4	IDES France	Euro	Calendar year, 4 spec. periods 2001	-222.957,00 EUR	22.536.827,00 EUR	22.536.827,00 EUR	22.313.871,00 EUR
5	IDES US Inc.	US Dollar	Calendar year, 4 spec. periods 2001	$ -409.985,00	$ 30.423.505,00	$ 30.423.505,00	$ 30.076.172,00
6	IDES Canada	Canadian Dollar	Calendar year, 4 spec. periods 2001	-314.222,00 CAD	61.273.376,00 CAD	61.273.376,00 CAD	60.959.154,00 CAD
7							
8							

Figure 20 The Flattened View of an MDX Cube

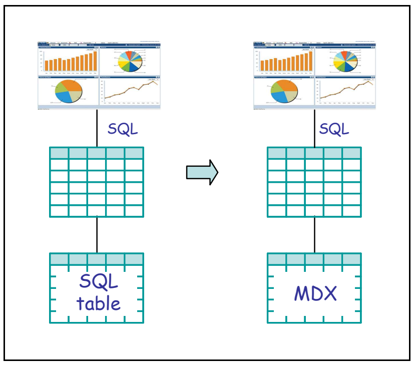
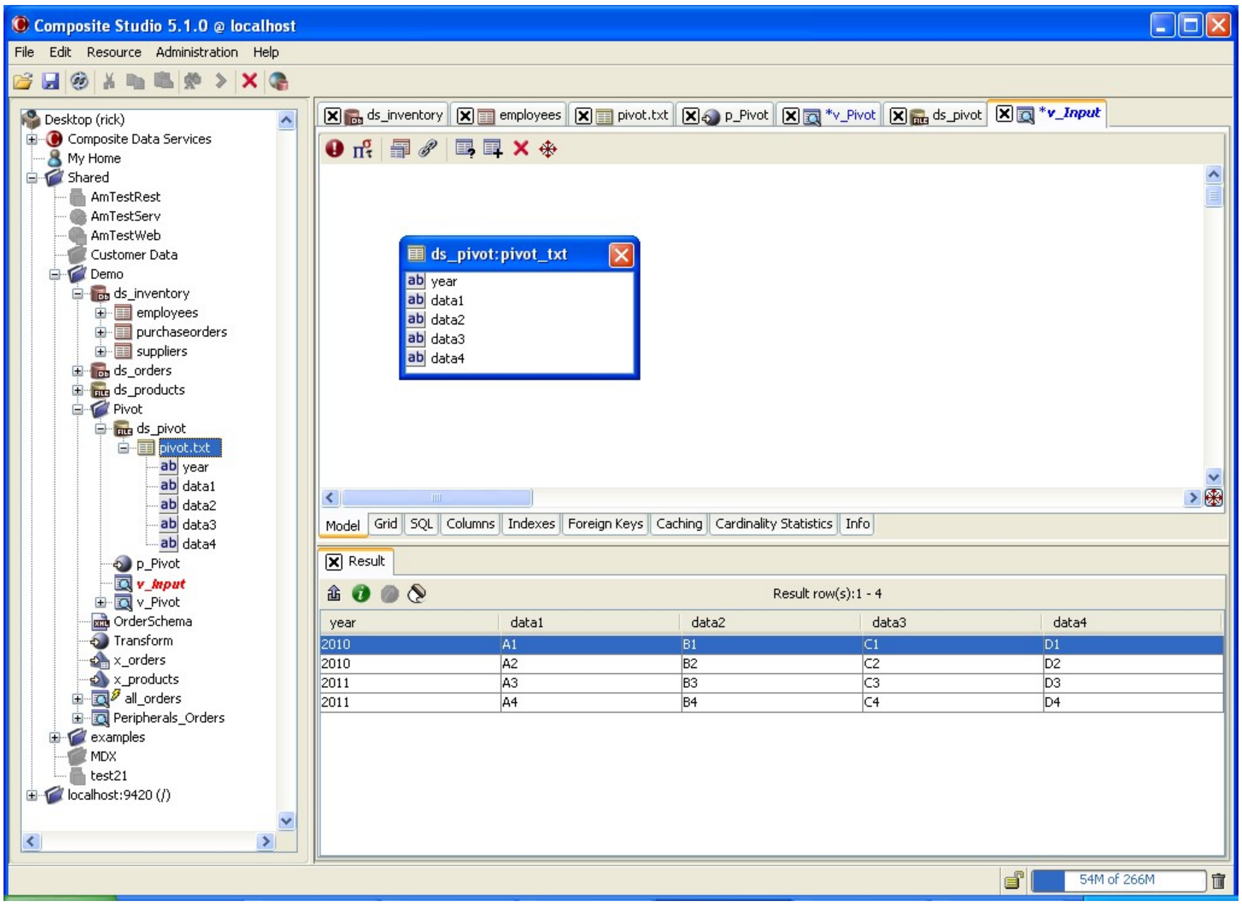


Figure 21 Seamlessly Migrating From SQL to MDX

13 Dealing with Repeating Groups and Procedures

In some older systems, data might be stored as *repeating groups*, meaning a set of comparable values is stored ‘next to each other’ instead of ‘underneath each other’. For example, if we would store in a table the total sales values for each quarter in one row, those four columns would form a repeating group. In Figure 22 a simple table contains four such columns (data1 up to data4). Most reporting tools have difficulty in processing those repeating groups, so they must be transformed in such a way that they are presented underneath each other in four different rows. This transformation process is sometimes called *pivoting*. Pivoting such a table into a table where those four values are stored underneath each other, as separate rows, can be done in CIS by using *procedures*.

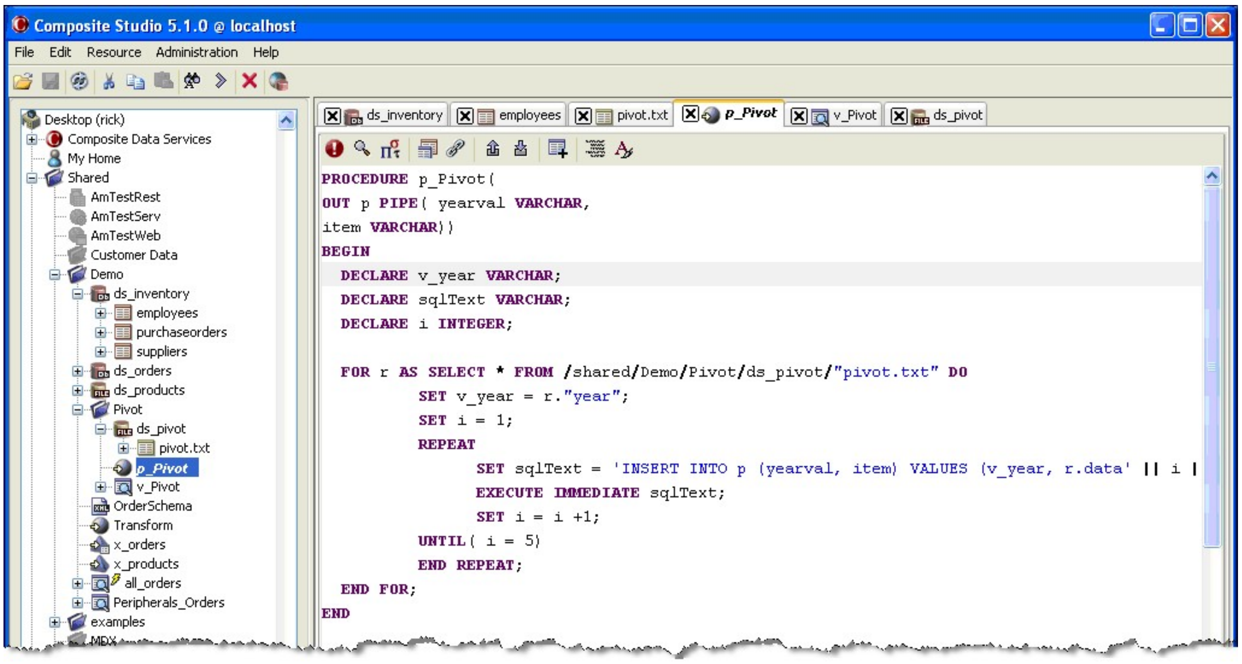

Composite Studio 5.1.0 @ localhost
File Edit Resource Administration Help
ds_pivot:pivot_txt
year
data1
data2
data3
data4
Model Grid SQL Columns Indexes Foreign Keys Caching Cardinality Statistics Info
Result
Result row(s):1 - 4
54M of 266M

year	data1	data2	data3	data4
2010	A1	B1	C1	D1
2010	A2	B2	C2	D2
2011	A3	B3	C3	D3
2011	A4	B4	C4	D4

Figure 22 **A Table with a Repeating Group**

A procedure is a piece of code that returns a set of rows. The code is written in a procedural language that resembles PL/SQL of Oracle and Transact-SQL of Microsoft SQL Server. Because a procedure in CIS returns a set of rows, a view can be defined on top of it. The effect is that the procedure with the view on top can be treated like any other view.

Figure 23 shows the procedure that pivots the table. The code is short and not too complex. This particular procedure returns a table of rows where each row contains one value. Procedures can also be used for very complex transformation for which procedural code is the only solution. Procedures are allowed to access views.



```
PROCEDURE p_Pivot(
OUT p PIPE( yearval VARCHAR,
item VARCHAR))
BEGIN
  DECLARE v_year VARCHAR;
  DECLARE sqlText VARCHAR;
  DECLARE i INTEGER;

  FOR r AS SELECT * FROM /shared/Demo/Pivot/ds_pivot/"pivot.txt" DO
          SET v_year = r."year";
          SET i = 1;
          REPEAT
                SET sqlText = 'INSERT INTO p (yearval, item) VALUES (v_year, r.data' || i |
                EXECUTE IMMEDIATE sqlText;
                SET i = i +1;
          UNTIL( i = 5)
          END REPEAT;
  END FOR;
END
```

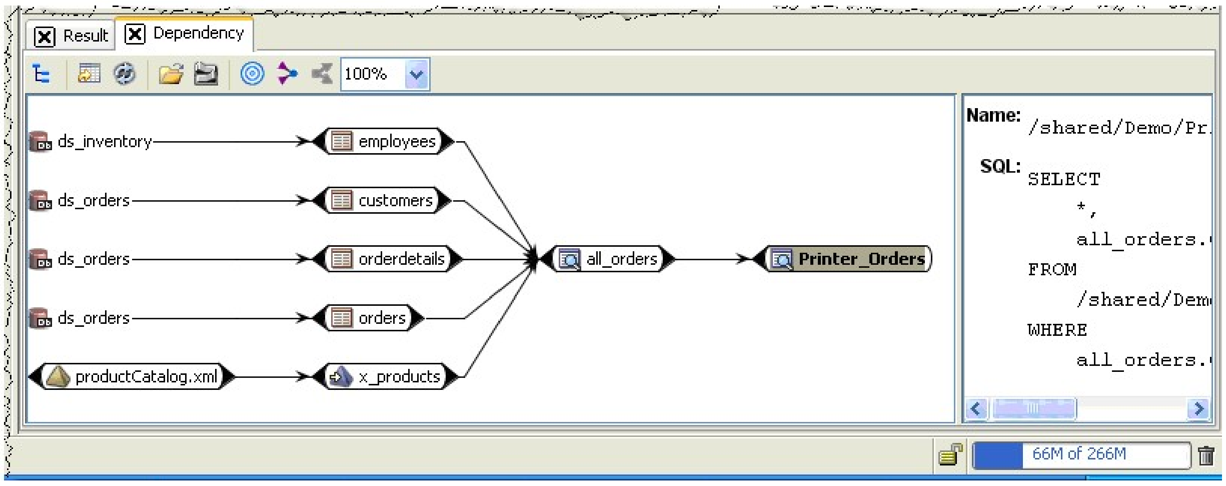
Figure 23 **A Procedure Returns a Set of Rows**

14 Keeping Track of All Relationships

A large environment may end up with many views, and many relationships between the views and the data sources. It's important that developers and administrators can quite easily see all those relationships. It's important for example for determining what the effect will be if the structure of a foreign table or view changes; which other views have to be changed as well?

CIS stores all the definitions of data sources, views, procedures, and so on, in one central repository. This makes it easy for CIS to show all the dependencies between those objects. For example, Figure 24 shows a dependency diagram that presents all the views and data sources that directly and indirectly make up a view called PRINTER_ORDERS (on the left hand side of the diagram). This is sometimes called a *lineage diagram*. Such a diagram allows us to do impact analysis. If the structure of a foreign table or view changes, we can see on which other views it might have an impact.

When objects such as views and tables are interrelated, what will happen if the underlying foreign table is changed? For example, what happens if a column is added to a foreign table, if one is removed, if the data type of a column is changed, or if a column is renamed? The solution is *introspection*. CIS can introspect a foreign table, determine that something has changed, and then it can then indicate which changes have been applied to the foreign table. This by itself is already very useful. But CIS can also automatically make comparable changes to the views and data sources that are directly or indirectly dependent on the changed foreign table. Imagine that a column is dropped from a foreign table, CIS will notice that and it can remove that column from each view that uses that column, and it will change the queries accordingly.



```
Name: /shared/Demo/Pr
SQL:  SELECT
          *,
          all_orders.
      FROM
          /shared/Dem
      WHERE
          all_orders.
```

Figure 24 A lineage diagram that shows the inter-dependencies between Views and Data sources

15 Exposing Views as Data Services

Most views in CIS will be accessed by reporting tools using SQL. But CIS does also allow views to be accessed as data services. In other words, the views are exposed as SOAP- or REST-based services. This can be done for every view defined in CIS, including the ones defined on XML documents, MDX cubes, and sequential files. The advantage is that non-BI consumers, such as an internet application or a more classic data entry application, can also use the data made available to the BI consumers; see Figure 25. They will share the same specifications, and data will be consistent across the BI and other environments.

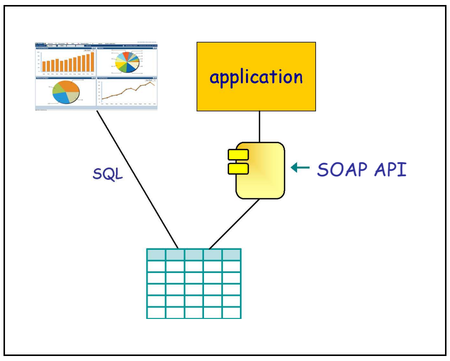
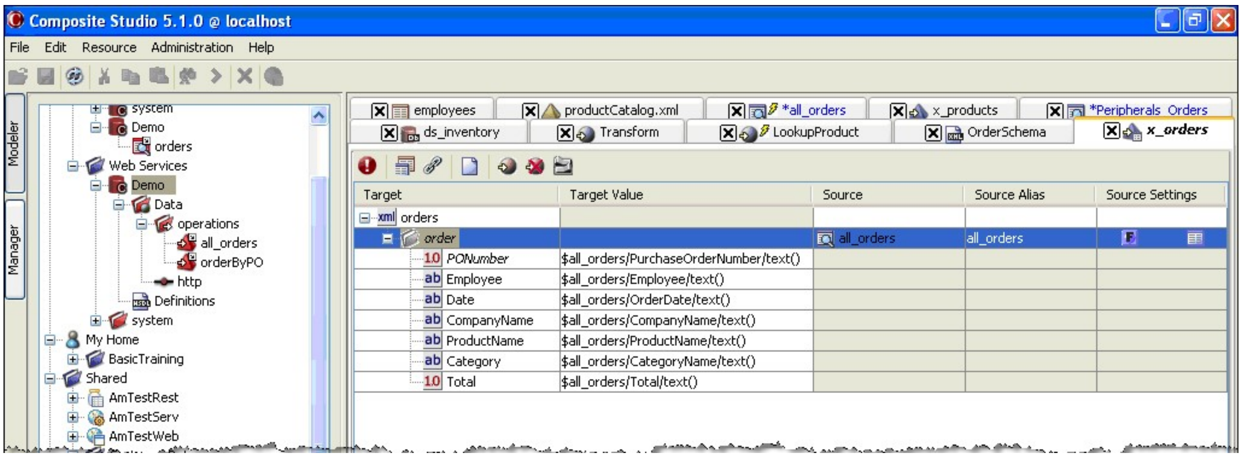


Figure 25 Reports and Applications Share the same View Definitions

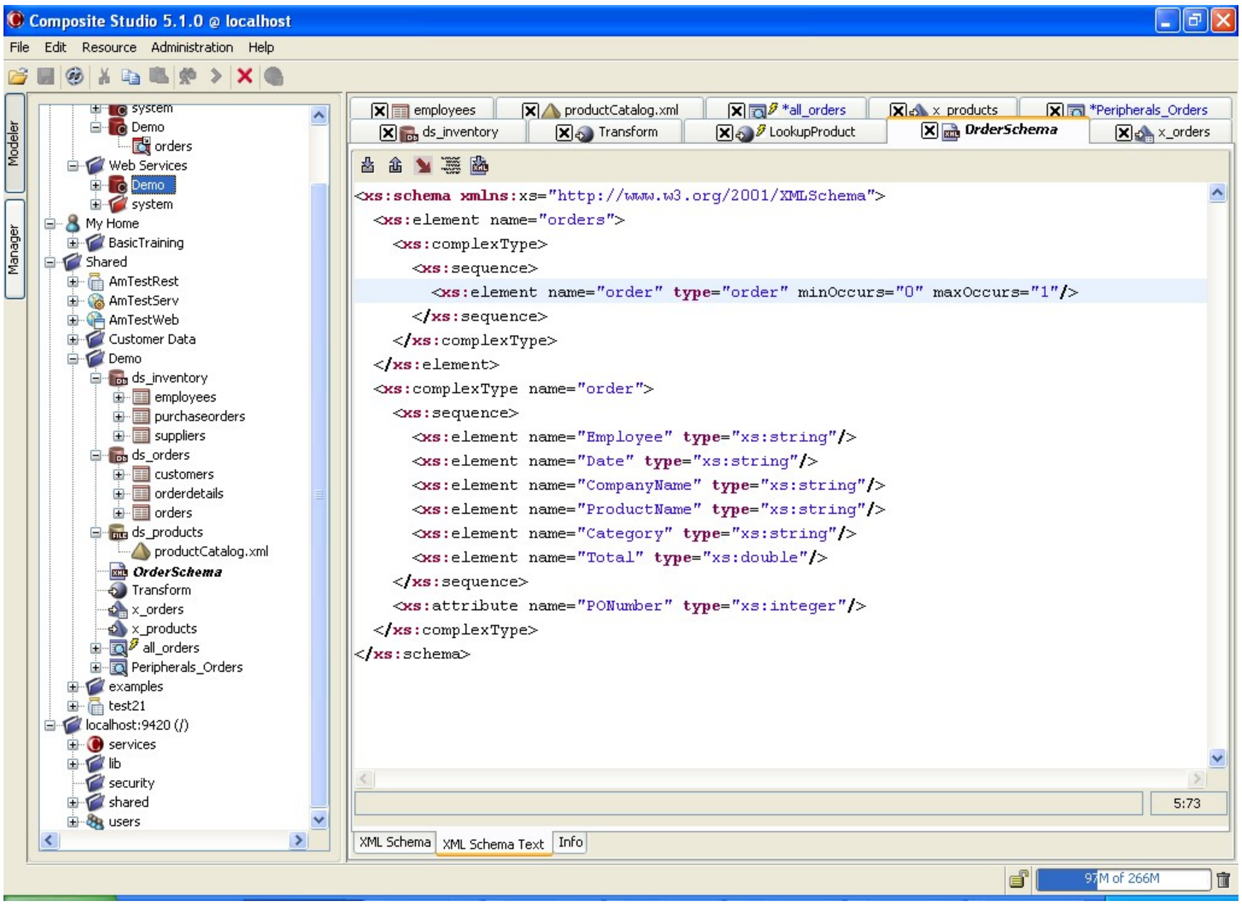
To expose a view as a service, we have to go through a number of relatively simple steps. First, we must define how the column structure of the view has to be transformed to an XML structure; see Figure 26. Note that defining this transformation is not mandatory. If we want to create a service that has a flat XML structure, then that flat XML structure can be derived automatically from the view structure. However, if we want to bring some hierarchy to the structure and make it look like a real XML structure, we must define a transformation.



Target	Target Value	Source	Source Alias	Source Settings
orders				
order		all_orders	all_orders	
PONumber	$all_orders/PurchaseOrderNumber/text()			
Employee	$all_orders/Employee/text()			
Date	$all_orders/OrderDate/text()			
CompanyName	$all_orders/CompanyName/text()			
ProductName	$all_orders/ProductName/text()			
Category	$all_orders/CategoryName/text()			
Total	$all_orders/Total/text()			

Figure 26 Mapping the View to an XML Structure

Next, an XML schema must be generated for this service; see Figure 27. In addition, the WSDL document must be generated. When this step is completed, the service is ready to be accessed. Any application that can access a service can access a CIS data service. CIS itself will process the service requests.



```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="orders">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="order" type="order" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="order">
    <xs:sequence>
      <xs:element name="Employee" type="xs:string"/>
      <xs:element name="Date" type="xs:string"/>
      <xs:element name="CompanyName" type="xs:string"/>
      <xs:element name="ProductName" type="xs:string"/>
      <xs:element name="Category" type="xs:string"/>
      <xs:element name="Total" type="xs:double"/>
    </xs:sequence>
    <xs:attribute name="PONumber" type="xs:integer"/>
  </xs:complexType>
</xs:schema>
```

Figure 27 The XML Schema of the Service

16 Caching Views

Regardless of how efficient a federation server is, it's an extra layer of software that sits between the reporting tools and the data stores, so it will consume cpu cycles and it will increase the response time of queries. Although the performance of a query is determined by the amount of time used by the federation server plus the time used by the underlying database server(s), the latter will consume most of the processing time, and the former only a small fraction. Still, it's important that a federation server optimizes and improves the performance of queries as much as possible. In this and the coming sections a few of the techniques deployed by CIS are described. This section explains *caching* of views.

CIS offers an extensive and flexible caching mechanism. In CIS caching means that the contents of a view is retrieved from the underlying tables and stored in a file or table. The effect is that when the view is queried the underlying view, table, service, or XML document is not accessed, but the data in the cache is. Accessing a cache can seriously improve the performance of a query on a view.

For every view a cache can be defined. Defining a cache involves nothing more than switching it on for a view. Technically it means that the query that makes up the definition of the view is executed and the result is stored in stead of passed to an application. The next time this view is queried, the data is retrieved from the cache; see Figure 28. Administrators can determine whether a cache should be defined and how the cache of the view should be refreshed: once or periodically. If periodically, we can specify at what time. Refreshing of the cache can be scheduled or it can be done manually. A cached view is sometimes referred to as a *materialized view*. Defining a cache for a view is simple; see Figure 29.

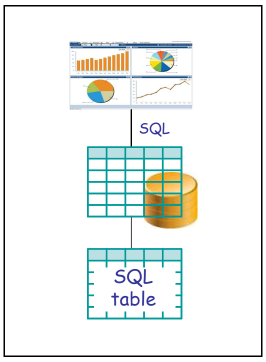
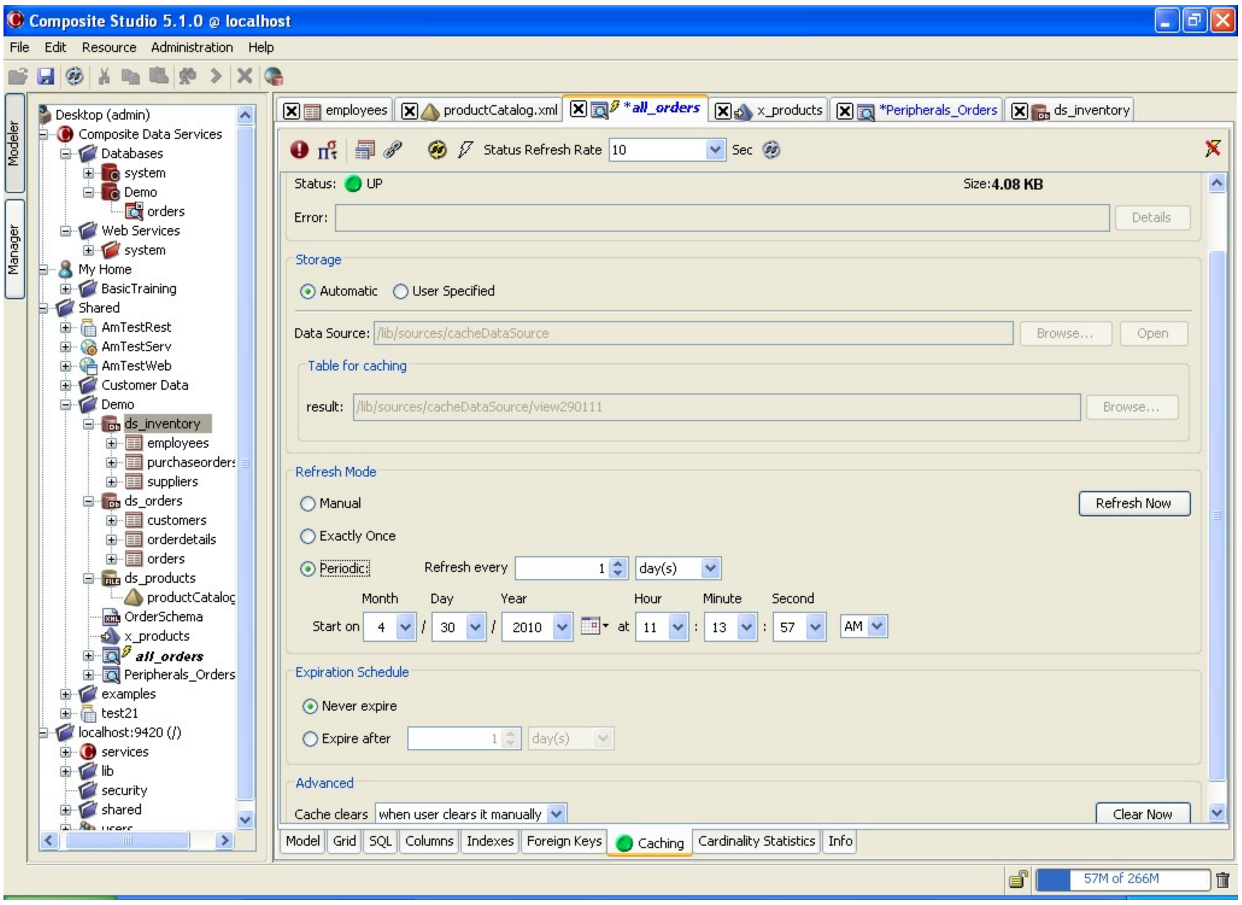


Figure 28 View with a Cache

As indicated, caches can be stored in files or tables. Retrieving cached data stored in files is normally very fast, especially if large subsets of all the records must be retrieved. If a query on a cached view only needs a small set of rows or if it contains a group-by operation, an index might be useful to speed up that query. In that case, a table is recommended because it can be indexed.


Composite Studio 5.1.0 @ localhost
File Edit Resource Administration Help
Modeler
Manager
employees
productCatalog.xml
*all_orders
x_products
*Peripherals_Orders
ds_inventory
Details
Browse...
Open
Refresh Now
Clear Now
Model Grid SQL Columns Indexes Foreign Keys Caching Cardinality Statistics Info
57M of 266M

Status Refresh Rate 10 Sec

Status: UP Size: 4.08 KB

Error:

Storage

(✓ Automatic) User Specified

Data Source: /lib/sources/cacheDataSource

Table for caching

result: /lib/sources/cacheDataSource/view290111

Refresh Mode

Manual

Exactly Once

(✓ Periodic) Refresh every 1 day(s)

	Month		Day		Year		Hour		Minute		Second	
Start on	4	/	30	/	2010	at	11	:	13	:	57	AM

Expiration Schedule

(✓ Never expire)

Expire after 1 day(s)

Advanced

Cache clears when user clears it manually

Figure 29 Caching Data per View

There can be various reasons for defining a cache:

- Load optimization: A cache might be useful to minimize the load on the underlying system. It could be that a view is defined on tables in an old system that already has issues with performance. Additional queries might be too much for this system. By defining a cache, fewer queries will be executed on the old system.

- Consistent reporting: A cache could also be useful if a user wants to see the same report results if he runs a report several times for a specific period of time (a day, week, or month). This is typically true for users of reports. It can be quite confusing if the same report returns different results. In this case, a cache might be necessary if the contents of the underlying database are constantly being updated.

- Source availability: If the underlying system is not always available, a periodically refreshed cache might enable 7x24 operation.

- Complex transformations: The transformations to be applied to the data might be so complex that doing them on-demand might be too slow. Storing the transformed result in the cache and reusing the result several times, might be more efficient.

The side effect of caching is that the data returned when querying the view may no longer be 100% up-to-date. Plus, caching means that we switch from on-demand transformation to periodic transformation.

By defining several cached views on one and the same view that contains the required data, different users can access the same data even when they need different caching settings. For example, we could define two views V_1 and V_2 on top of view V_3. V_3 contains the right data for all users. V_1 shows all the data of V_3 and uses a cache that is refreshed once a day, whereas V_3 also shows all the data of V_3 but uses a cache that is refreshed once a month.

17 Optimization Techniques for Accessing Foreign Data

The previous section described caching as a mechanism for optimizing queries. This section describes some of the query optimization techniques CIS supports.

When accessing foreign tables, data is first retrieved by the database server from disk. Next it's transferred to the federation server, and finally the data is sent to the report. The database server itself is responsible for optimizing I/O, meaning it's responsible for the amount of data transferred from the disks to the database server. For a large part, the amount of data transferred determines the performance of a query. In addition, a federation server is responsible for optimizing data traffic between the database server and itself. Again, the amount of data sent over the network between the servers determines for a large part the performance of the queries.

This means that a federation server has to optimize the amount of data transferred between the data stores where those tables reside and itself. It's the module called the *optimizer* that is responsible for this. This module is very much comparable to the optimizer of a relational database server. The difference though is that optimizers of database servers try to optimize the amount of I/O, whereas the optimizer of a federation server tries to optimize the amount of data traffic between the data stores and itself. Below we explain the key features of Composite's optimizer including:

- Combining queries
- SQL pushdown
- Substitution
- Parallel processing
- Distributed joins
- And other advanced query optimization techniques.

First of all, if a query is executed on a view that is defined on a number of other views, which are also defined on other views, how does CIS execute these queries? What won't happen is that query after query is executed sequentially. Imagine that a query is executed on view V_1 which is defined on views V_2 and V_3, and those are defined on tables T_2 and T_3 respectively. What will not happen is that first V_2's query is executed on T_2, then V_3's query on T_3, next those results are combined and joined and the result is kept somewhere in memory, and finally the query on V_1 is executed on that intermediate result. Although this approach will return the correct result, it would be slow. The approach taken by CIS is that queries are combined into one query which is passed to the database server to be executed. So, in the example the queries belonging to the views V_1, V_2, and V_3 plus the query entered by the application are combined into one query which leads to a join of tables T_2 and T_3. So it will be the database server that's doing the join and not CIS. In short, all the layered queries are coalesced into one single comprehensive query and optimized accordingly.

As indicated, the goal of the CIS optimizer is to minimize the amount of data send from the foreign data stores to the federation server. So it will try to 'push' as much processing to the underlying database servers themselves. This means that selections (such as, get only the customers from London), projections (such as, get only the names and addresses of the customers), and group-by operations are pushed down. Evidently, this is not possible for every data store. For example, if the data store is a sequential file, an XML document, or a SOAP service, no optimization can be executed by the data store, the federation server has to do all the work. It will retrieve all the data (unless a cache exists because then the data is retrieved from that cache).

If the query is a one-table query and the data store is a database server the whole query is pushed down and only the result of the query is returned. If two tables are joined and both are stored in the same database, again a query with a join is pushed down to the database server. It's a little bit more complex if two tables are joined and if they are not managed by the same database server. An inefficient strategy would be to retrieve all the data from both database servers and let CIS do the join because it would involve a lot of data traffic. CIS supports a few techniques to process this more efficiently, which we will discuss next.

If one of the two tables is relatively small (approximately 100,000 rows or less), that table could be read first and kept in memory. Next, the large table is read, and row by row comparisons are made with the smaller table in memory.

Another approach is that the data from the small table is retrieved first. Then the query on the large table is extended with data coming from the smaller table. Let's illustrate this with an example. Imagine we want to join a large with a small table with the following query:

```
SELECT   TL.*
FROM     TABLE_LARGE AS TL INNER JOIN TABLE_SMALL AS TS
         ON TL.COL1 = TS.COL1
WHERE    TS.COL2 = 1
```

First, CIS will get all the rows from `TABLE_SMALL`:

```
SELECT   COL1
FROM     TABLE_SMALL
WHERE    COL2 = 1
```

Let's assume the result consists of the values 1 to 8. Next, it will merge the result with a query on the `TABLE_LARGE` table:

```
SELECT   *
FROM     TABLE_LARGE
WHERE    COL1 IN (1, 2, 3, 4, 5, 6, 7, 8)
```

The effect is that only relevant data from the large table is returned to CIS in stead of all the rows. This style of processing joins is called *distributed semi-join*. This same approach can be used when the small table is not a table at all, but a file or XML document.

When two large tables must be joined, CIS will use a so-called *sort-merge join*. In this case, both tables are retrieved from the two data stores and both have been sorted by the database server on the join columns. The effect is that the database servers perform the sorts, and CIS only has to do the merge. Let's take the following join as an example:

```
SELECT   TL1.COL1, TL1.COL2, TL2.COL1, TL2.COL2
FROM     TABLE_LARGE1 AS TL1 INNER JOIN TABLE_LARGE2 AS TL2
         ON TL1.COL1 = TL2.COL1
WHERE    TL1.COL3 > 1000
```

The following statement is send to the first data store:

```
SELECT   COL1, COL2
FROM     TABLE_LARGE1
WHERE    COL3 > 1000
ORDER BY COL1
```

And a comparable statement is send to the other:

```
SELECT   COL1, COL2
FROM     TABLE_LARGE2
ORDER BY COL1
```

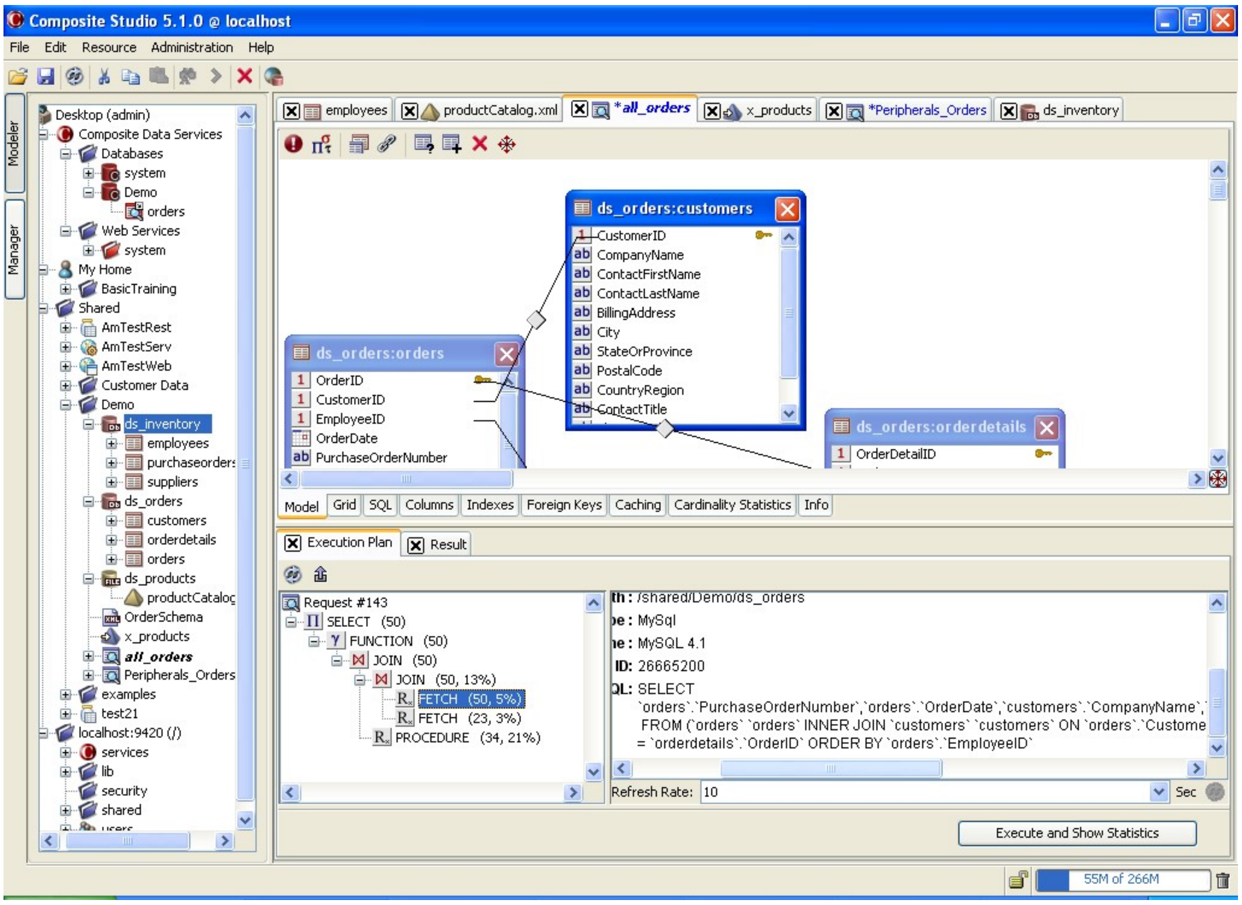
Merging the two results is straightforward for CIS. The advantage of this approach is that the work still to be done is the merge-step and that should not take that much processing time. Still, a lot of data is sent, but most of that processing is done in parallel.

When tables are joined and one or more are cached, it might be that the cache is used. Imagine that tables T_1, T_2, and T_3 are joined together and that a cached view is available on the join result of T_2 with T_3. In this case, CIS will perform a join of T_1 using the cache. If that cache is small, CIS might use the distributed semi-join approach again to join them.

Another form of optimization is that as much work is 'pushed down' to the underlying database server(s). For example, if a join has to be executed between tables T_1, T_2, T_3, and T_4, and if the last three of those tables are stored in the same database, a query that contains a join of those three tables is send to the database server. The result of this join is then joined with table T_1. Push down is an optimization mechanism that allows to do as much processing as close to the data itself. Plus, if more database servers are involved, processing can be done in parallel.

And if nothing else works, CIS performs a nested loop join, which is slow, but will return a correct result.

For most of the decisions the CIS optimizer has to make, it has to know the approximate size of the result set on each side of the join. CIS automatically gathers such data from the data stores. This type of data is sometimes called *statistical data*. There are of course sources that can't offer that type of data. For example, an XML documents and SOAP services can't answer questions concerning the number of rows they contain. In that case, statistical data can be entered by hand.



```
Request #143
SELECT  (50)
  FUNCTION  (50)
    JOIN  (50)
      JOIN  (50, 13%)
        FETCH  (50, 5%)
        FETCH  (23, 3%)
      PROCEDURE  (34, 21%)
```

```
th : /shared/Demo/ds_orders
pe : MySql
ne : MySQL 4.1
ID: 26665200
QL: SELECT
    `orders`.`PurchaseOrderNumber`,`orders`.`OrderDate`,`customers`.`CompanyName`,
    FROM (`orders` `orders` INNER JOIN `customers` `customers` ON `orders`.`Custome
    = `orderdetails`.`OrderID` ORDER BY `orders`.`EmployeeID`
```

Figure 30 Query Execution Plan

If developers have the feeling that the optimizer isn't coming up with the best possible strategy *commands* and *hints* can be specified. With commands the optimizer is forced to go for a specific strategy and with hints the optimizer can follow a strategy.

Another optimization form is that developers can see what CIS' processing strategy will be for a query. They can see the order in which tables are joined; see Figure 30. If they can come up with a better strategy, they can change the order in which particular operations take place.

To summarize, CIS supports many approaches and techniques to optimize the access to the data stores. Together, they give developers and administrators various instruments to optimize the performance of queries, and to influence when and how queries are executed. Research in this area will continue and has to continue because the size of databases keeps increasing.

18 Security Features

CIS offers a rich set of security features, including authentication, authorization, and encryption. All three are described in this section.

With respect to *authentication*, users who access CIS must present credentials (such as a user id and a password) to identify themselves; in other words, CIS checks whether they are really who they say they are. CIS can be configured in such a way that an external system, such as Kerberos, is used for authentication. In that case CIS will ask the external system to perform the authentication.

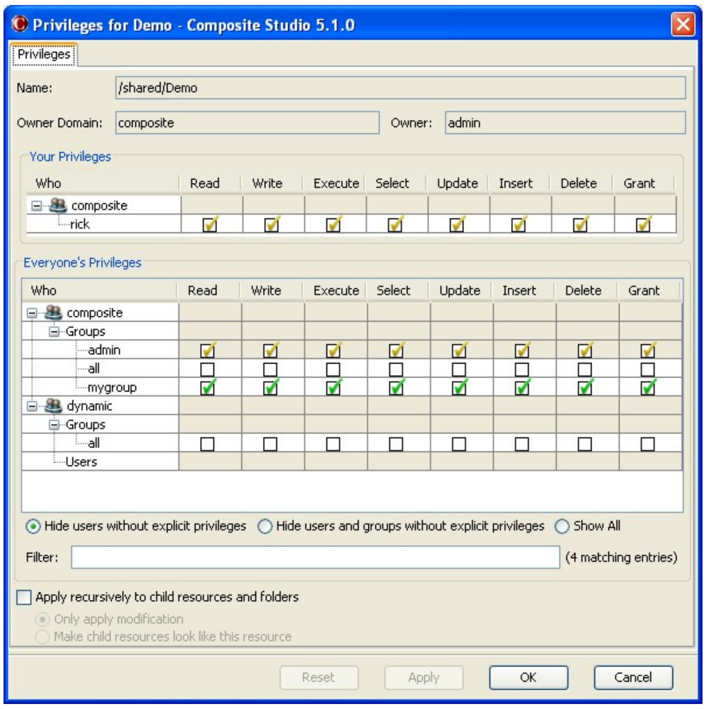
Users can be introduced and defined within CIS, but user definitions can also be stored outside CIS, for example in LDAP directories. Users can be grouped in domains and in user groups. Unfortunately, groups can't be nested.

It's not likely that every user should have access to all the data accessible through the federation server. Therefore, CIS offers features for *authorization* to control which user is allowed to access which data elements.

To each user group and individual user access privileges for views and data services can be assigned; see Figure 31. This is somewhat similar to assigning privileges to users with the GRANT statement in SQL. Figure 31 shows that the following types of privileges are supported: read, write, execute, select, update, insert, and grant.

Besides the privileges in CIS, the user has to have the privileges inside the underlying database servers to access certain tables.

It might be that two users may query the same view, but they are not allowed to see the same rows. For example, a manager might see the data on all the customers, but an account manager may only see the customers he is responsible for. Therefore, CIS supports *row-level security*. In the definition of a view we can add a condition that includes the id of the user so that only those rows are returned that the user is allowed to access.


Privileges for Demo - Composite Studio 5.1.0

Privileges

Name: /shared/Demo

Owner Domain: composite Owner: admin

Your Privileges

Who	Read	Write	Execute	Select	Update	Insert	Delete	Grant
composite								
rick	✓	✓	✓	✓	✓	✓	✓	✓

Everyone's Privileges

Who	Read	Write	Execute	Select	Update	Insert	Delete	Grant
composite								
Groups								
admin	✓	✓	✓	✓	✓	✓	✓	✓
all	☐	☐	☐	☐	☐	☐	☐	☐
mygroup	✓	✓	✓	✓	✓	✓	✓	✓
dynamic								
Groups								
all	☐	☐	☐	☐	☐	☐	☐	☐
Users								

[✓ Hide users without explicit privileges] Hide users and groups without explicit privileges Show All

Filter: (4 matching entries)

☐ Apply recursively to child resources and folders

Only apply modification

Make child resources look like this resource

Figure 31 **Security Specifications per User**

If applications call the views via a SOAP service interface, there might be a need that those calls are encrypted. CIS accomplishes *encryption* by using SSL as the connection mechanism.

19 Inserting, Updating, and Deleting Data

The Data Delivery Platform is primarily query focused. As a result, CIS functionality is as well. Nevertheless, CIS does support inserts, updates, and deletes within the underlying databases and files that are typically sources to the DDP. CIS also supports multi-site transactions by using the two-phase commit protocol.

20 Conclusion

Composite Information Server is a powerful and flexible open federation server that delivers on-demand transformation of data stored in various data stores. Using the extensive caching mechanism it can also offer periodic transformation. The product hides where and how data is stored to reporting and analytical tools and applications. Internally, it has a very modular structure based on views and data sources that allows developers to setup their federation solution the way they think is right. Internally, the popular language SQL is used to specify nearly all the required aggregations, transformations, and calculations. Because SQL is used, the tool is easy to learn for most developers.

In a nutshell, CIS offers the following features:

- Transparent query access to relational and non-relational data stores, including XML, MDX, services, files, and spreadsheets
- On-demand transformation of data coming from different data stores.
- Wide range of data transformation and data delivery capabilities.
- Management of shareable transformation specifications.
- Data access security rules.
- Optimization techniques for improving query performance.

All these features make CIS a product with which a Data Delivery Platform can be developed. CIS is capable of decoupling data consumers and data stores the way needed in a modern business intelligence architecture.

About the Author Rick F. van der Lans

Rick F. van der Lans is an independent analyst, consultant, author and lecturer specializing in data warehousing, business intelligence, service oriented architectures, and database technology. He works for R20/Consultancy, a consultancy company he founded in 1987.

Rick is chairman of the annual European Data Warehouse and Business Intelligence Conference (organized in London), chairman of the BI event in The Netherlands, and he writes for the B-eye-Network.

He introduced the Data Delivery Platform in 2009 in a number of articles that were published on BeyeNetwork.com.

He has written several books on SQL. His popular *Introduction to SQL* was the first English book on the market in 1987 devoted entirely to SQL. After more than twenty years, this book is still being sold, and has been translated in several languages, including Chinese, German, and Italian.

For more information please visit www.r20.nl, or email to rick@r20.nl.

About Composite Software

Composite Software, Inc. is the data virtualization gold standard at ten of the top 20 banks, six of the top ten pharmaceutical companies, four of the top five energy firms, major media and technology organizations; and multiple government agencies. These are among the hundreds of global organizations with disparate, complex information environments that count on the Composite to increase their data agility, cut costs and reduce risk. Backed by nearly a decade of pioneering R&D, Composite is the data virtualization performance leader, scaling from project to enterprise for data federation, data warehouse extension, enterprise data sharing, real-time and cloud computing data integration. Composite Software is a privately held, Silicon Valley-based corporation. For more information, please visit www.compositesw.com.