

# Understanding Business Intelligence: ETL and Data Mart Best Practices

---

Jeremy Dean, iQ4bis Software, Inc.

Andreas Schindler, Schindler IT-Solutions

Jean King-Sperlak, iQ4bis Software, Inc.

[www.iQ4bis.com](http://www.iQ4bis.com)

## **TABLE OF CONTENTS**

<a href="#"><u>Introduction</u></a>	<a href="#"><u>Page 3</u></a>
<a href="#"><u>ETL Basics</u></a>	<a href="#"><u>Page 3</u></a>
<a href="#"><u>Principles for a Successful ETL Project</u></a>	<a href="#"><u>Page 5</u></a>
<a href="#"><u>Principles for the Extraction</u></a>	<a href="#"><u>Page 7</u></a>
<a href="#"><u>Principles in Dimension Design</u></a>	<a href="#"><u>Page 8</u></a>
<a href="#"><u>Principles of Fact Design</u></a>	<a href="#"><u>Page 10</u></a>
<a href="#"><u>Principles for the Project Design</u></a>	<a href="#"><u>Page 11</u></a>
<a href="#"><u>Minimum Transformation Requirements</u></a>	<a href="#"><u>Page 12</u></a>
<a href="#"><u>Advanced ETL Topics</u></a>	<a href="#"><u>Page 15</u></a>
<a href="#"><u>Summary</u></a>	<a href="#"><u>Page 19</u></a>
<a href="#"><u>About iQ4bis</u></a>	<a href="#"><u>Page 20</u></a>

## **INTRODUCTION**

This paper is about building powerful data marts that require minimal administration and are simple to change. This may seem like an impossible goal to anyone who has been involved in the usual complexity of these projects, but there are a number of simple, practical concepts and methodologies that have been employed and tested over many years of successful data warehouse implementations that are repeatable and are easy to understand. These techniques have been used in projects around the world by the author both personally and as baked into software and methodology.

Data mart creation concepts are explained here; a basic understanding of Relational Database concepts including basic SQL is assumed.

## **ETL BASICS**

ETL is shorthand for **Extract**, **Transform** and **Load** and is essential for successful OLAP/multi-dimensional reporting. **Extract** pulls data from source systems, **Transform** changes it into a format suitable for analysis and reporting and **Load** moves it to a location accessible to users. An extension to the ETL process is to create the multi-dimensional data structures – cubes.

### **Sources**

For the purposes of data warehousing ETL is used to pull data from business systems into a database that is designed for analysis and reporting. Business systems that ‘feed’ an ETL process are typically ERP systems like Microsoft Dynamics AX, GP and NV, JD Edwards, SAP, Oracle Financials, MAS90/MAS200, MFGPro etc. but any data can be a source.

### **Destinations**

The destination database for the ETL process is a ‘Staging’ database. A Staging database is a working data repository where the raw data from source(s) is ‘transformed’ into a format suitable for reporting and analysis. Once transformed the data is moved (Loaded) to either a Warehouse database or directly from Staging to a multi-dimensional OLAP database containing cubes.

## Measures and Dimensions

When designing the system, these are the two things you are looking for: what are you counting (**Measures**) and how are you grouping the data (**Dimensions**). When the system starts getting complex it is always helpful to have a very clear, very solid understanding of what is a Dimension and what is a Measure.

## Single Version of the Truth

The data mart or data warehouse can be referred to as a 'Single Version of the Truth'. This process of designing the extraction of data and applying business rules to transform the data into a format suitable for analysis and reporting creates a data repository that can be relied on as the business' accepted and approved reporting data. A single definition of reporting data avoids different groups of an organization drawing different conclusions from the same base data.

## Use of a Staging Database

SQL Server 2000 allowed the creation of a cube directly from the source system, and for 'real time' databases in SQL Server 2005 this is still an option. However, to reduce the impact on the source transaction systems and to give the cube a solid foundation a 'Staging' database is essential. A Staging database ensures the data feeding a cube is in a format suitable for multi-dimension analysis and this format is very different transaction data.

All data in the Staging area should be temporary and easily replaced by data from the source system.

## Use of a Warehouse Database

The creation of a 'Warehouse' database is optional with many business intelligence providers. The primary purpose of a Warehouse database is to provide a clean set of data for reporting and analysis – and to feed cubes. The Warehouse database can also be used to store data that does not exist in the source systems like history or legacy data, surrogate key information and slowly changing dimension data (covered later).

This document will use the term 'data mart' to refer to either the staging or warehouse database unless a distinction between them needs to be made. A data mart contains data from a particular business area and multiple data-marts can form a data warehouse.

### **Star Schema**

The logical database format of both the data mart and the cube is a 'Star Schema'. A Star Schema is a relational database structure in a particular shape:

- One or more 'Transaction' tables connected to many 'Dimension' (describing) tables.
- Each Dimension table is connected directly to the Transaction table(s) but there are *no connections between the Dimension tables.*

The Star schema is the basis for multi-dimensional analysis. The 'Transactions' (facts) can be aggregated by any of the Dimensions independently or a combination of Dimensions. The process of Transformation in the ETL process converts source data into a Star Schema design.

## **PRINCIPLES FOR A SUCCESSFUL ETL PROJECT**

The following principles are best practice design concepts for building any data mart. All are simply principles of good design that help ensure a successful data mart project.

### **Keep it Simple**

A key factor in a rapid and easily maintained system is simplicity of design. The time spent reverse engineering complex logic should not be underestimated and can run to hours or days – either for other staff working on your project or for you a couple of weeks/years later trying to remember what you'd done. A good ETL tool should promote an easily understood design by separating the process into logical areas and hiding much of the complexity of the ETL from the user but clear, well documented design is still important. A simple practice to keep the system as clear and easy to read as possible is:

- ***Use self-documenting names*** – self-documenting names are descriptions that describe the purpose of the object without needing any further investigation so are meaningful to the user.

Other tips for keeping a project simple:

- **Remove or clearly mark business logic that is no longer used.** Much time can be spent following business logic to a dead end.
- **Use Views rather than SSIS packages where appropriate.** SQL Server views are usually much easier to maintain, edit and debug than SQL Server Integration Services packages. SQL Server views can be queried and analyzed using simple select statements.
- **Keep a consistent naming convention throughout the project.** In particular for someone who maintains and extends the project after the initial configuration. Consistent naming conventions save enormous amounts investigation time so the design can be easily traced from source through staging and to the cubes.

## Super-Rapid Initial Configuration

Nothing helps the prospect of a successful and popular business intelligence situation as much as an early 'quick win' – the fulfillment of at least part of the promised delivery early in the project. With current software and hardware capabilities it is entirely possible to get an initial impressive result in the first full day of implementation. Two techniques help this:

1. **Get the prerequisites and software installed remotely before the project starts and test an initial extract**

The hardware and software environment, prerequisites and software installation process can appear overly complex and uninteresting to most people who will be involved in the ETL configuration. Get this time-consuming distraction out of the way before the real project actually starts, including an initial extraction so you know everything works.

2. **Make a concerted effort to build a very simple system end-to-end before the end of the first day**

With the software in place and working it should be easy enough to perform a simple extract of a single dimension table and a transaction table. Configure a simple Dimension with absolute minimum complexity, connect it to the Fact table, create a basic cube and display the result in your analysis interface. Seeing the process from beginning to end in a short time effectively reinforces the architecture and concepts of data warehousing and gives the project sponsors a great deal of confidence that the remainder of the project will go well.

## Configure in Iterations

Closely related to the Super-Rapid Implementation is to design the configuration process in repeating steps or iterations. By progressively adding to the design you avoid two bad situations:

1. **Paralysis by Analysis** – where the project starts to seem too daunting to ever get into production.
2. **Big Bang Failure** – where the project is carefully constructed in isolation from users and when they eventually get to see it, it is not what they want.

If the initial end-to-end build can be achieved on the first Monday of the project then the initial presentation to users should be possible by day 4 (Thursday). This is the presentation where the theory gives way to the practical and feedback on what is actually useful becomes very clear.

## PRINCIPLES FOR THE EXTRACTION

### Minimize the Impact on the Data Source

One of the key reasons for a data mart is to separate the reporting and analysis system from the transaction system to ensure that the processing of reports does not impact the transaction system in any way. To make the impact on source systems as light as possible:

- Set any source connections to 'Read Only' to ensure the ETL process cannot change the source system in any way.
- Keep the extract as simple as possible. Get in and get out quickly – perform additional transformation and manipulation of the data once it is in the Staging database and not during the initial extract
- Only extract what is required for reporting and analysis. **Select only the tables and columns required for analysis and do this on the first pass.** Don't be tempted to select all fields with the assumption that you will remove unused fields later because removing fields once the design is set risks destabilizing the design. A well designed data mart will typically be a small fraction of the size of the source data.

## Divide the Data into Manageable Chunks

Building data marts usually involves processing large volumes of data. During the initial design process when the design is changing rapidly, limit the data being processed down to a small subset to speed processing time. There is no point in waiting for large volumes of data to process through the system when a small subset would have been an adequate test. ***Limiting test data to a small subset can save hours and often days of time.***

## Traps in Filtering the Data

As above, limiting the source data extracted can save large amounts of processing and configuration time for data that you never want to see in the end result – or to limit the data volume for the initial configuration.

But be cautious of filtering data out of the system based on apparent business rules identified early in the project. Often the power of analysis comes from exposing data as information for questions that users did not know to ask. A good example is the common request to exclude Freight Cost from Sales analysis. This request may come from a section of the business with a vested interest in making Sales margin look as good as possible but it may also be very valid to show the business what the Freight cost is. It is easy enough to retain all information and filter it out in the final validation process rather than hiding it entirely.

## PRINCIPLES IN DIMENSION DESIGN

### Use Hierarchies to Manage Dimension Volume

A very large cube with millions of transactions that are well aggregated into Dimension Attributes can perform better than a smaller cube where the Dimensions are relatively flat. By grouping Dimension Attributes into Hierarchies the data returned to the user is sent in relatively small, manageable sets that don't strain the server, network or client.

Grouping the data by attributes makes it more meaningful for analysis, since if the user had wanted to see only the detail they could go directly to the source.

### Don't Skimp on Dimensions or Attributes

Some data warehouse thinking recommends limiting the number of Dimensions and Attributes in the design for performance reasons.

With current hardware and software you can easily include as many Dimensions and Attributes as are available in your source data without noticing a particular performance hit. The primary strength of multi-dimensional analysis is the fact that it is multi-dimensional.

### **Use Natural Hierarchies**

One of the key benefits of multi-dimensional analysis is that the end result, if presented by a good analysis tool, makes intuitive sense to users and matches the way they think about their business. Using clearly understood natural hierarchies supports this intuition.

Natural Hierarchies are Attributes grouped in an order that matches the data structure. A good example of a non-natural Hierarchy is the common request from users to drill from Customer down to Product. In a Star Schema Customer and Product should be entirely separate Dimensions so they can be analyzed both separately and together so forcing them together into a Hierarchy is an unnatural act and likely to cause confusion and complexity in the end result.

### **Dimension Members should only have One Parent**

One of the characteristics of Dimension members is that they should only belong to one level above them. For example, one particular town of Springfield can only exist in California. Since the data may contain many Springfields this repeating value must be made unique and this is easily done by combining it with the parent to make a unique member: Springfield-California, Springfield-Missouri etc.

Sometimes the hierarchy may not be so clear and it can be tempting to force attributes into hierarchies when they shouldn't be. Before defining hierarchies analyze the data to check for good natural hierarchies. If an item at one level has more than one parent then perhaps the two attributes should not be forced into a hierarchy and the user would be better served by being able to analyze the attributes independently of each other.

For example, the attributes 'Product Color' and 'Product Type'. It may be possible to drill down from Type to Color but is there really a relationship between Type and Color? The user may be better served analyzing all the Colors of a certain Type and all the Types of a certain Color rather than forcing Colors to be a sub-set of Type.

## **Use Drill-Through to Keep Detail out of the Cube**

Where the number of members in a Dimension is close to the number of members in the Fact table consider using the drill-through function of Analysis Services. Good examples of this are Invoice numbers. There is no particular analysis benefit of slicing and dicing Invoice numbers in the cube but they are useful for detail. Keep such information (termed 'Degenerate Dimensions') in the Fact table and make it available by drill-through but don't include it in the Cube.

## **PRINCIPLES OF FACT DESIGN**

### **Minimize Processing of the Transaction/Fact Tables**

The transaction tables are typically the largest in the data mart with hundreds of thousands or millions of rows. Keep the processing of these large tables to a minimum and avoid storing transaction data multiple times by using views as necessary.

### **Cubes Naturally Aggregate**

When selecting the numbers to count for a Cube (the Measures) be aware that the Cube is designed to aggregate or sum all Measures. Any number defined as a Measure in a cube will be summed so measures cannot usually be averages or Units. Where Units are required as Measures they must always be multiplied by Row Quantity. For example:  $\text{UnitCost} \times \text{Quantity} = \text{ExtendedCost}$

### **Divide the Components of the Project into Logical Groupings**

Grouping the components of a project gives it a clear structure, makes it easier to read and understand and allows more precise control of operations within the project. For example, divide the extract of dimension data and transaction data into two separate steps. This clear definition allows you to quickly run a dimension extract in isolation from the much larger transaction extract.

Clearly separated stages within a project also help the debugging process by isolating logic for testing.

## **PRINCIPLES FOR THE PROJECT DESIGN**

### **Use Stored Procedures to Avoid Duplicating Logic**

A single stored procedure can be maintained in one location but used in multiple instances. Use Stored Procedures for the Transformation processes so all applicable business logic is applied consistently every time the process is run.

Using other components for transformation and manipulation of data – like SSIS or DTS packages, means each process is stored in isolation within the package. You can reuse a dataflow but only if the columns are identical. This duplicates effort, increases the chance of error and multiplies administration overhead.

### **Ability to Re-Run the Process Whenever You Want**

The system should have a simple, clear workflow that allows the process to be run in full at any time. The process should be tolerant of changes to the state of the source data and timing so that if something goes wrong the process can be immediately re-run with little or no additional effort or re-working of data.

This is particularly important with incremental updates. The process should be forgiving enough to re-run an incremental update that partially completes.

### **Handle All Foreseeable Data Issues**

As a general rule all foreseeable issues with data should be handled by the ETL process without causing the system to fail in day-to-day operation. This means 'typical' data errors like null keys, missing relations and even incorrect data types should be handled by the process and included but tagged in the end result. The cost of maintaining a system is drastically increased if the ETL process has to be nursed to conclusion each time it is run.

### **Ensure Easy Error Handling and Notification**

Any error in the process that does cause the system to fail must be immediately identifiable and traceable. Expose each detailed step in the process to the designer and if a step fails the location of the step, what was attempted and the full SQL error message should be made clearly available for debugging purposes.

## **Manifest Views as Tables if they are Affecting Performance**

While SQL Server Views are simple to maintain and build, Views that are used to manipulate large volumes of data, multiple layers of Views or Views that perform very complex or resource-intensive operations can perform considerably faster if they are manifested as tables. Just remember to include the statement to drop the table in each operation. Since Stored procedures will treat Views and Tables equally it is easy to test the performance difference by switching a View for a Table without changing its name.

## **Be Able to Rebuild or Replicate the Entire System Easily**

In case of hardware or software failure the entire ETL system must be easily recoverable on the same or different server. Even for basic testing and quality control the system should be simple to transfer from machine to machine with minimum changes to configuration.

In the case of Slowly Changing Dimensions or other data that exists only in the Warehouse database, a simple backup process and clear, fully tested recovery plan are very important.

## **MINIMUM TRANSFORMATION REQUIREMENTS**

The basic concepts of the ETL process are very simple. Use the following as a baseline and carefully consider if further complexity has a real business advantage:

### **Collect Data from All Sources**

This is the Extract stage. Remember to keep the load on the source system as light as possible.

### **Match Business Keys**

Adjust the data that has been Extracted into the Staging database to have matching keys and data types. This step prepares the data for the joins that will be made between the tables that make up the Dimensions and the connections between those Dimensions and the Transaction tables.

Ensure that the keys and foreign keys that will be used for the joins:

- Have the same or compatible data types
- Are checked for Nulls if Nulls are considered a possibility

## **Enrich the Data**

This step depends on how closely the source data conforms to the requirements for Analysis and Reporting. To achieve a useful environment according to the business rules defined by the sponsors of the system the source data will need some manipulation including: lookup of descriptions, adjusting decimal places, joining tables, filtering data and any other data change to achieve the 'Single Version of the Truth' mentioned earlier.

## **Dimension Processing**

The most simple and easiest to maintain structure is a single table per Dimension. There will be redundant data in this method but the relatively small size of Dimension tables (relative to Transaction tables) and performance benefit from a minimal use of joins makes this 'Star Schema' structure efficient in terms of both performance and maintenance.

## **Nulls**

Most ERP data sources that are likely to be used as Dimension keys are unlikely to have nulls but it is good practice to check for them anyway.

To check for and correct Nulls use the isnull() function.

## **Elements that Exist in the Fact Table but not Dimension Table**

Even with ERP source data it is likely that some Transactions will not have a corresponding Dimension member. SQL Server 2005 includes the cube option to ignore this error and convert the missing members to 'unknown members'.

However by ignoring missing Dimension member elements there is the risk of not identifying more serious issues like the wrong data type or mismatched keys. It is preferable to get the distinct business keys from the Fact table into a temporary table and join this to the Dimension data. This makes the joining data very clear and if there are any missing members they can easily be identified.

Relying on Analysis Services to return an error but ignoring them makes bug tracking difficult because the Dimension members and the corresponding Facts will be different with no clear method of determining why. Always manage missing Dimension members as part of the ETL process and don't rely on Analysis Services to handle and ignore the error.

### **Dimension Elements that have No Facts**

A very useful optimization practice is to remove Dimension members that have no Transactions. The typical example for this is a Customer table that includes 400,000 Customers and a Sales Transaction table with transactions for the past two years. There may only be 100,000 Customers with Transactions over this two-year period. By removing the inactive Customers with no Transactions the Dimension is reduced from 400,000 to 100,000 members.

An easy way to achieve this is to use the business key from the Fact joined to the Dimension table using an Inner Join. The resulting table will contain only active Customers. Use the Fact table as the primary key to get all current Customers and all missing members by using a Full Outer Join instead of an Inner Join. This will ensure you get all current Customers and all missing keys.

### **Fact Processing**

In all cases any processing of the Fact tables should be as simple and minimal as possible because Fact tables are relatively large compared to the rest of the database. Use an Incremental Update (covered later) if the Fact extraction and processing is taking too much time but where the process time is acceptable it is preferable to reduce complexity and chance of error by performing a full extract with each process.

### **Simple Facts**

Fully processing the Fact table each time gives you the freedom to change the design without considering data in existing tables stored in the data mart. The use of Incremental tables, Slowly Changing Dimensions and Surrogate keys creates additional levels of complexity that must be maintained when the design changes and significantly add to the time needed to administer and change the design.

There are certainly business benefits in the more complex structures mentioned but the likelihood of project success and user adoption is strongly linked to the length of time to get the system into production. A simple but effective system that is in production is vastly more likely to succeed than a complex, difficult to maintain and more error-prone system that is not in production. A general and very important rule is: ***Get the system into Production as soon as possible and add complexity later.***

A simple system has the added benefit of allowing you to be much more flexible and dynamic in your design. This flexibility is often appreciated by users more than the more esoteric benefits of Surrogate keys and Slowly Changing Dimensions.

## **Granularity**

If you have Fact tables of different granularity it is easier to keep them separate and use multiple Fact tables in SQL Server 2005. For example, if you have a Sales Transactions table at Product level and a Forecast table at Product Group level you can map them using different attributes rather than manipulating the tables to have a common granularity.

## **ADVANCED ETL TOPICS**

There is one guarantee in using the techniques below: they will make your project take longer and it will be more difficult to maintain. While Incremental Updates and Surrogate keys can provide performance benefits and Slowly Changing Dimensions can help your analysis better match real-world situations always keep the 80/20 rule in mind – you can deliver 80% of the functionality relatively quickly. The last 20% can easily take at least the same amount of time and will often take much more.

## **Surrogate Keys**

A surrogate key is a key for a Dimension Element created by the system and replacing the business key. The surrogate key is usually an auto incremented integer.

A surrogate key helps to:

- Handle multiple-field Business keys - you get a single field Surrogate Key
- Optimizes very large (Text) keys - Surrogate key is 4 byte Integer
- Allows for changes in the Business key (see Slowly Changing Dimensions below)

## Non-Meaningful Keys

The disadvantage of having only the Surrogate key is that it is no longer meaningful to the user and it must be maintained by the ETL process. For example, it is not possible to tell if Date\_ID 2365 is in 2007 or 2008. If you have the Business key '12.3.2008' the value is clear without looking in the details.

## Configuring

Generate Surrogate keys for the Dimension using an Identity column and replace the corresponding Business key in your Fact. Use a lookup transformation in SSIS or to do this.

## Best of Both Worlds

Using Surrogate keys makes your data less "readable" but can give a considerable performance increase where the natural or Business key is very large – in particular with GUIDS. Consider keeping the business key in the fact (if space allows) to help in debugging. To achieve the performance gain of Integer keys without the administration overhead of maintaining and storing unique Surrogate keys you can fully re-key the system using Surrogate keys with each ETL process.

## Storage Overhead

If you use a surrogate keys without storing the corresponding Business keys and you damage your Dimension table your Fact is lost forever unless you have a backup. Without the Dimension table you have no way to find the meaning of the surrogate keys so backups are a very important component of Surrogate key usage.

## Slowly Changing Dimensions

A Slowly Changing Dimension stores changes to the Dimension members that occur over time whereas a normal Dimension will only store the most recent state of the member. To use SCDs you must use a Surrogate key.

There are up to 6 different types of SCDs (0,1,2,3,4,6) but the two most common are covered here:

**Type 1:** Does not track any history and shows only the current state of the Dimension member. This is the typical and default type of Dimension.

**Type 2:** Keeps track of changes by creating a new Dimension row. There are several methods to distinguish the different versions of a Dimension element but the most usual is to add two Date columns to the Dimension table and have a Start and End date for every row. If you add a new record to the dimension table you have only a Start date and an empty End date. If this record is changed the old record gets an End date and the new record gets a Start date.

Example:

Supplier_key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date
001	FSC	ForSail Supply Company	CA	01-Jan-2004	21-Dec-2004
002	FSC	ForSail Supply Company	IL	22-Dec-2004	

For a stable handling of SCDs use surrogate keys (like Supplier\_key in the above example). The easiest Way to handle a SCD is to use the SCD Wizard of SSIS. Just create your dataflow as usual and add the SCD wizard at the end instead of your destination table. To set up a SCD first add the Startdate and Enddate Field to your destination table. Start the wizard and choose your Business key. The Business key is the key identifying a row in your dimension. For all other rows you can choose if there a no changes allowed or if you want to handle them as a Type1 or Type2 SCD. Now you can set the Start and End field and your SCD generation is set up.

## Loss of Freedom

Without SCDs you could recreate the Dimension Table whenever you want. This allows you to make any changes to your project without considering historical data since it will be reloaded on the next processing. If you are using SCDs you cannot recreate the Dimension table. If you do so you will lose all your historical values and invalidate the fact table by losing the translation of the Surrogate key.

***When making changes to a system that stores SCDs and Surrogate keys you must always be aware of the existing data and alter the tables carefully. Changing the design requires much more care and much more time.***

## Incremental Extract

Use an Incremental Extract if the time to extract a large source table has an impact on system performance or the source system. However, unless the data volume is very large there are good reasons for loading all tables in full:

- A full extract guarantees that the data in the data mart matches the source data exactly
- A full extract is quicker and easier to set up and requires little knowledge of the table
- It does not rely on assumptions that historic data is stable data
- A full extract doesn't necessarily mean extracting all source data – a simple filter can also effectively limit the data and reduce the Extract cost

## First Pass

Working incrementally means that you import everything the first time and after this you read only the new rows. To configure the Incremental Update you need to know what identifies the 'new rows' and how far back into history to incrementally extract.

## Changing History

Identifying how far back into the data to extract is usually fraught with problems. On the surface most businesses would say that historic transactions do not change outside of the current period but in reality this is seldom the case. To ensure that the data mart continues to match the source system the 'buffer' or set of data that is incrementally extracted and added to the history data should be as generous as system resources allow.

The incremental component of data should also allow for disruptions in the extract. If the ETL process does not run for one day the increment should be able to recover the missing day without any problems. This is another reason for allowing the Increment of data that is extracted to be generous.

Lastly, disappearing transactions can cause the data mart to differ from the source system. While often considered theoretically impossible, a transaction that appears one day and not the next can remain in the data mart but not exist in the source system. A larger increment set and dropping and reloading the incremental time range will avoid this issue.

## **SUMMARY**

Building data marts and ETL processes involves large volumes of complex business data and the easiest outcome is complexity, lack of results and the use of considerably more resource than expected. By following some well-proven methodologies and principles, the most important of which is 'keep it simple', you can achieve a powerful result in a short amount of time that is useful to users and fulfills the core requirement of effective visibility into their complex business data.

## **ABOUT iQ4bis**

iQ4bis is an independent software provider with a well-earned international reputation for intuitive business analytics solutions for mid-sized companies. Recognized for its ease-of-use, flexibility and rapid implementation times, the iQ4bis solution is sold, implemented, and supported through offices in the United States, Europe, New Zealand, Australia, and Asia. Recently, growing demand for efficient and effective data warehousing for the midmarket has encouraged iQ4bis to increase efforts in the U.S. A *Microsoft Gold Certified Partner*, iQ4bis provides a data source and application neutral analysis, charting and reporting solution for business users including iQ4bis Analysis™ and iQ4bis DataServer™; solutions for a wide variety of ERP systems including Microsoft Dynamics AX, GP, NAV, CRM as well as JD Edwards and SharePoint Enterprise Portal integration. For more information, visit [www.iQ4bis.com](http://www.iQ4bis.com).

### **Contact iQ4bis:**

**iQ4bis USA:** (949) 872-2788

**iQ4bis New Zealand:** 011 064 9 529 3767

**iQ4bis Europe:** 011 049 911 252 4770

**iQ4bis Australia:** 61 03 9645 7377

Copyright © 2008 iQ4bis Software Incorporated. All rights reserved. iQ4bis and the iQ4bis logo are registered trademarks of iQ4bis Software Incorporated. All other trademarks and company names mentioned are the property of their respective owners.