BI4 All

Evaluation Guide

Version 3.1.00
1st February 2012
Instant Business Intelligence
info@instantBI.com

Table of Contents

1. CHANGE CONTROL LOG 4

2. AUDIENCE 5

3. PURPOSE 5

4. INTRODUCTION TO BI4ALL 6
   4.1. What Went Wrong With Analytical Applications? 6
   4.2. What is BI4ALL? 8

5. BI4ALL DATA MODELS 9
   5.1. How are The BI4ALL Data Models Maintained? 9
   5.2. Design Features of the BI4ALL Data Models 16
      5.2.1. The Table Types in BI4ALL 16
      5.2.2. Naming Standards for BI4ALL Tables and Views 16
      5.2.3. Type 1 Dimension Tables 18
      5.2.4. Type 2 Dimension Tables 22
      5.2.5. Fact Tables That are Detailed Transactions 23
      5.2.6. Fact Tables That are Snapshot or Summary/Periodic Fact Tables 27
      5.2.7. Fact Tables That are Plan Tables 29
      5.2.8. Type 1 Association Tables 30
      5.2.9. Type 2 Association Tables 33
      5.2.10. Everything is Linked to Everything 33
   5.3. Coverage of the BI4ALL Data Models 35
      5.3.1. The Base Model BI4ALL Data Coverage 36
      5.3.2. BI4 Retail Data Coverage 42
      5.3.3. BI4 Web Data Coverage 46
      5.3.4. BI4 Media Data Coverage 49
      5.3.5. BI4 Telco Data Coverage 52

6. THE PROCESS OF BI4ALL BASED DATA MODELLING 57
   6.1. Summarising Because it is Important 60

7. THE DATA ACCESS LAYER 62
   7.1. Advantages of the Data Access Layer 63
   7.2. Disadvantages of Stored Procedures 65
   7.3. Features of the Data Access Layer 66
      7.3.1. Parameters for Reports Sourced Via SPs 66
      7.3.2. Accepting Strings as Parameters 67
      7.3.3. The “*” Character Is the Wild Card 67
      7.3.4. Redundant Tables and Redundant Joins 68
      7.3.5. SP Name linked to the Report Name 69
      7.3.6. Userid is Always Passed to the SP 69
      7.3.7. Partitioning Strategy for SQL Server 2005 EE 70

8. BI4ALL STANDARD REPORTS 71
   8.1. Connected Parameters 72
   8.2. Dashboard Level Reports 73

8.3. Detailed Level Reports 74
8.4. Report Services and Maps 76

9. APPENDIX 1 – SAMPLE STORED PROCEDURES 77
9.1. Drilling Down A Hierarchy 77
9.2. Plan Versus Actual Templates 79

10. APPENDIX 2 – THE VISUAL BASIC CUSTOM ASSEMBLY 92
10.1. Calculate a Percentage 93
10.2. Calculate a Variance Percentage 94
10.3. Calculate the Estimated Response for a Campaign 94
10.4. Calculate the Over/Under Variance Percentage 95
10.5. Calculate the Over/Under Variance Number 95
10.6. Colour the Over Under Bars and Lines By Variance 96
10.7. Formatting Currencies 97
10.8. Setting the Fonts and Colours of the Report 98
10.9. Reports By Text 110

1. CHANGE CONTROL LOG

#	Date	Name	Description
3.1.00	1/2/12	IBI Development	Initial version for publication with BI4 ALL

2. AUDIENCE

The intended audiences for the BI4ALL Evaluation Guide are:

- IT and Business staff from client organisations who are involved in the evaluation process for the selection of data models and analytical applications for their company.

3. PURPOSE

The purposes of the BI4ALL Evaluation Guide are:

- Explain clearly and concisely the wide variety of features of all the components of the BI4ALL product set.

- To provide enough input for client Business and Technology staff at a prospective client to determine whether it is worth further discussions with Instant Business Intelligence to determine level of fit of the BI4ALL product set.

  This document is primarily for the Technology People in the prospective client.

- It is expected that Technology People from the prospective client will review this document prior to detailed discussion with IBI Sales Staff as to the suitability of BI4ALL for their company.

  We have placed a lot of detailed information into this document for the purposes of review prior to investing our time into the sales cycle with prospective clients.

4. INTRODUCTION TO BI4ALL

BI4ALL is much more than just a product. BI4ALL is a suite of ideas that have been developed over a period of more than 20 years. BI4ALL is the culmination of all that we have learned about how to build data warehouses and analytical applications over the last 20 years. The techniques used in the BI4ALL data models are tried and tested.

Before we jump into BI4ALL it is instructive for the reader to review, very briefly, what the issues were with "Analytical Applications" when they were first released by various vendors in the early 00s.

4.1. What Went Wrong With Analytical Applications?

Those of us who have been around long enough can remember the hype of "Analytical Applications" that was trumpeted around the 00s. Everyone was building their brand of Analytical Applications. Informatica, Business Objects, Cognos, Oracle. They were all having a go at creating the "silver bullet".

Analysts like Wayne Eckerson said things like "this could be the final frontier for large software deals". It was thought that if someone could crack the "Analytical Applications" problem then they would be in the pole position for the follow on sales.

The "Analytical Applications" business died. There is no two ways to put that. The interesting questions are:

1. Why did "Analytical Applications" die?
2. Is there anything that can be done about that?

There were three main reasons "Analytical Applications" died.

1. The cost of connecting the data models to the source systems and thereby customising or writing the ETL subsystem was prohibitive. It was no cheaper to build the ETL from scratch than to customise existing ETL.

2. The cost of customising the data model was prohibitive.

3. The cost of customising the reports over the top was prohibitive.

In short. Once the client purchased the "Analytical Applications" the cost of customising them was so high that the return on investment simply was not there. Clients found they could build similar applications from scratch with about the same amount of money or even less.

In order to do something about this there would need to be six inventions made.

1. There would need to be an ETL tool that reduced the cost of development and maintenance of ETL by a factor of 10 over the productivity profile of the leading tools such as DataStage and Informatica.

   The ETL tool would have to be able to either alter existing ETL fast enough and at a low enough price that the customisation of the models would not dramatically increase the cost of customising or writing the ETL from scratch for each client.

   The biggest inhibitor to deploying "Analytical Applications" was getting the data from a set of unknown source systems at the time of the development of the applications into the data models that are part of the "Analytical Applications".

2. There would need to be a way to customise data models that is at least 5 times more cost effective than the best Data Modelling tools available like PowerDesigner and Erwin. The customisation time of the data models was seen to be excessive.

3. There would need to be a way to disconnect the data model presented to the applications and the end users from the physical implementation of the underlying database. DBAs have the right to refuse to support data models that do not conform to corporate naming standards. And for good reason. The DBAs need to be completely in control of the physical implementation of the data models.

4. There would need to be a way to disconnect the reports from the data model so that changes to the data model and changes to the reports did not cause large amounts of rework in the reports. Experience was showing that as data models became more complicated the ability to connect them to the reporting tool via some abstraction layer was becoming more and more strained.

5. There would need to be a way to customise reports at run time, or perhaps at design time, so that reports could be easily customised to conform to the look and feel that was already established as the standard in the client. This is particularly important around colours and logos.

6. There would need to be a way to translate data models and reports into the many different languages that the people who need to understand them can read. This translation would be required to be from the base language (presumably english) to any one of a number of target languages.

   There are many multi-national companies in the EU and other areas of the world today. It is no longer enough for analytical applications to only come in English. It is not enough to force the client to maintain multiple versions of data models and reports so that multiple languages can exist.

This list was first written up in mid 2002. Instant Business Intelligence consultants looked over the list and decided that if ever such a product could be invented, designed and built it would have a bright future.

In mid 2002 there were serious question marks as to whether this could even be done let alone how to do it.

As a technical person who has worked in BI for some years you will easily understand the problems that killed “Analytical Applications” just as easily as you will understand the solutions for those problems.

In this Evaluation Guide we will present the solutions to these problems for the data modelling and applications portion. We will go into detail as to exactly how these solutions work so that you can consider acquiring these solutions for your company.

The solution to the ETL tools problems is SeETL and detailed information about SeETL can be retrieved from our downloads page.

http://www.instantbi.com/Downloads/tabid/59/Default.aspx

4.2. What is BI4ALL?

BI4ALL is made up of many components which implement the latest and best ideas as to how an “Analytical Application” should be designed and built. BI4ALL is also a suite of ideas and techniques that have been invented and deployed to large clients over many years. These ideas and techniques are instantiated into the products but the reader is well advised to consider them as “separate” since they can be applied to any organisation.

The major components of BI4ALL are as follows:

1. A suite of data models that are stored in the SeETL DesignTime Workbook.

2. A suite of Stored Procedures that are stored as text and documented in the SeETL DesignTime Workbook.

   The current suite of stored procedures operate on SQL Server. We have developed techniques to be able to maintain Stored Procedures across multiple database platforms but we have not implemented the stored procedures across other databases at this time. We will implement the early versions of the applications on SQL Server.

   The current suite of Stored Procedures is 60,000 lines of SQL Server SP code.

3. A suite of 70+ SQL Server Report Services reports. These reports address various needs in Campaign Management, Sales and Customer Segmentation areas. They are not intended to be exhaustive, they are meant to be indicative as to how the client can implement reports across the organisation.

   The reports cover the most important areas.
   a. Campaign Management Analysis (CPA)
   b. Sales Analysis (SAA)
   c. Segmentation Analysis (SGM)

   Naturally it is perfectly possible to build an endless suite of reports. Development on reports was halted because it was seen that this was far easier to do at client implementation time. Further, penetration of Report Services by Microsoft has been less than was expected in 2005 when the decision to use Report Services was made. Report development effort has been redirected to development of data models and the underlying SeETL tools for some years now.

4. SeETL DesignTime comes with a metadata repository and a suite of Report Services Reports that expose all the components of BI4ALL to the technology support people as well as to the business users.

   This portion of SeETL DesignTime is recommended to be used with BI4ALL even if a third party ETL tool is used for the project.

5. A visual basic custom assembly that is invoked by the reports to perform run time customisation.

The major capabilities that are delivered with BI4ALL are as follows:

1. The ability to maintain the data model in up to 10 languages.

2. The ability to maintain the Report Services Reports in up to 10 languages.

3. The ability to deploy multi language data models and reports over the same base data warehouse if desired.

4. The ability to customise many of the features of the reports at run time using the visual basic custom assembly.

5. The maintenance of a “documentation” database that reflects the deployed production database which can be reverse engineered into any data modelling tool.

The companion product, SeETL, has solved the problem of reducing the development and maintenance costs of ETL by a factor of 10 as well as the data modelling work by a factor of about 5.

5. BI4ALL DATA MODELS

This section discusses the BI4ALL Data Models and how they are maintained. In the first instance this section addresses how the data models are maintained so that the reader gains an understanding of why we have selected Excel to be the tool of choice for maintaining data models.

The second section will go into the data models themselves.

5.1. How are The BI4ALL Data Models Maintained?

It is true that when we tell people that we maintain data models in Excel workbooks that we get a few raised eyebrows. We usually follow that up by saying that the early versions of BI4ALL were simple text documents that were create view statements. When people hear that we initially just created text documents the excel workbook sounds like a big upgrade!

Why are we using Excel?

We are experts in the usage of a wide variety of data modelling tools. We have been involved with Data Modelling tools since 1989 when Bachman first came onto the scene. Tools such as Erwin and Powerdesigner have their place in the world. And it is not as data warehouse design tools. We have used these tools on data warehouse projects and it is truly an exercise in “click yourself to death”. The maintenance of comments for fields is tedious in the extreme. Trying to maintain data models in different languages is not cost effective.

Further. Because constraints are turned off in the production database and most data modelling tools do not let you create constraints between views the whole issue of how to document joins is a problem.

When IBI consultants were discussing how to maintain the data models there was only one thing we were unanimous on. We did NOT want to use a data modelling tool. The time and effort of using it would destroy our ability to build a model sophisticated enough to be worth real money.

So the first releases of BI4ALL were text documents expressing views. The second release was the upgrade to Excel workbooks. The full documentation of the workbooks to maintain data models can be found on this link.

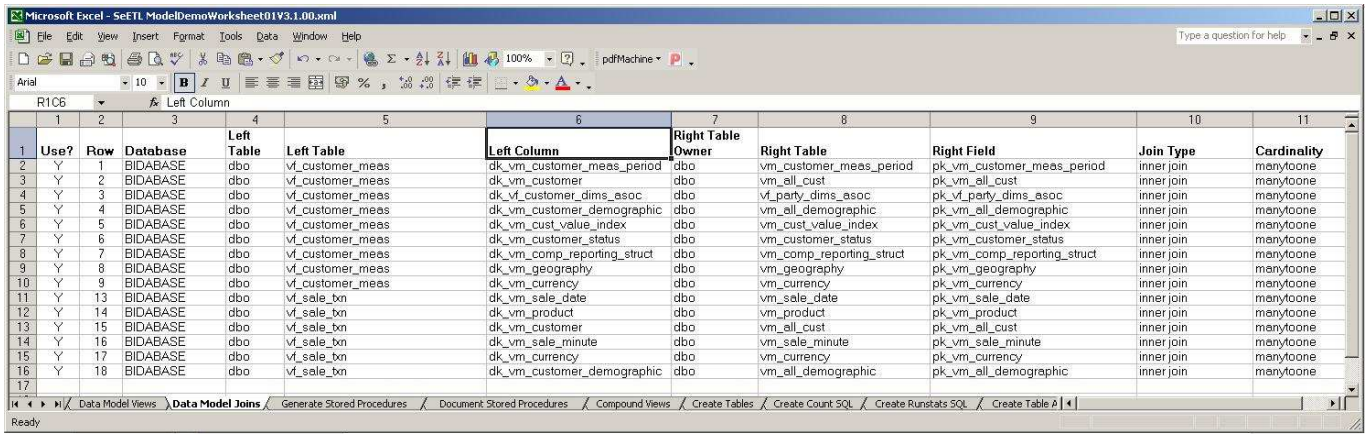
http://www.instantbi.com/LinkClick.aspx?link=SeETL+DesignTime+User+Guide.pdf&tabid=59&mid=442

A sample workbook with a snippet of a data model in it can be found on this link.

http://www.instantbi.com/LinkClick.aspx?link=SeETLDemoWorksheet01V3.1.00.zip&tabid=59&mid=448

Open the file SeETL DemoWorksheet01V3.1.00.xml in Excel and scroll across to the tabs “Data Model Views” and Data Model Joins.

To maintain joins in a data model you simply fill in one row per join. This will generate the create constraints statements in SQL Server to create the join inside the documentation database.



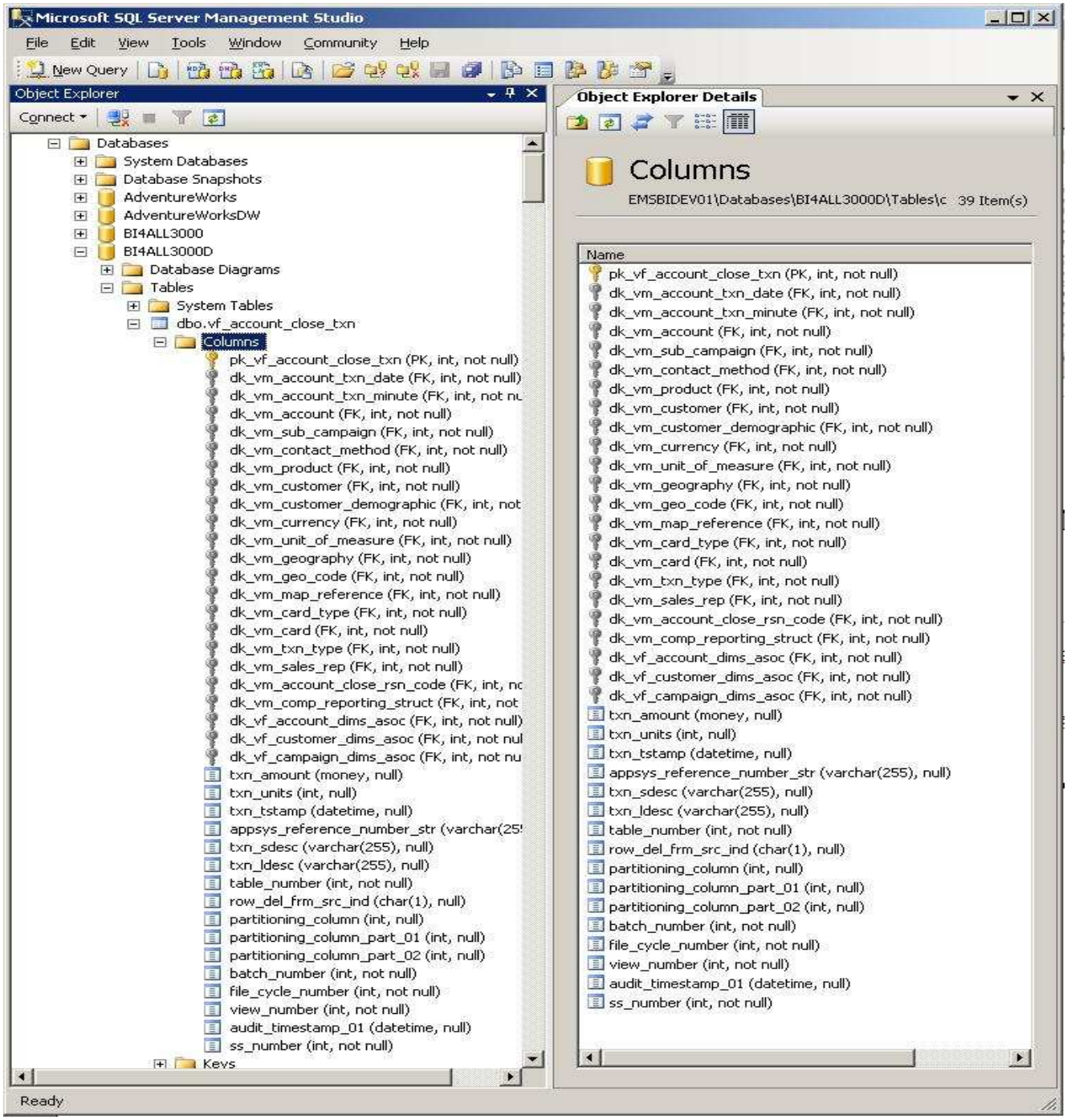
Use?	Row	Database	Left Table	Left Table	Left Column	Right Table Owner	Right Table	Right Field	Join Type	Cardinality
Y	1	BIDABASE	dbo	vf_customer_meas	dk_vm_customer_meas_period	dbo	vm_customer_meas_period	pk_vm_customer_meas_period	inner join	manytoone
Y	2	BIDABASE	dbo	vf_customer_meas	dk_vm_customer	dbo	vm_all_cust	pk_vm_all_cust	inner join	manytoone
Y	3	BIDABASE	dbo	vf_customer_meas	dk_vf_customer_dims_asoc	dbo	vf_party_dims_asoc	pk_vf_party_dims_asoc	inner join	manytoone
Y	4	BIDABASE	dbo	vf_customer_meas	dk_vm_customer_demographic	dbo	vm_all_demographic	pk_vm_all_demographic	inner join	manytoone
Y	5	BIDABASE	dbo	vf_customer_meas	dk_vm_cust_value_index	dbo	vm_cust_value_index	pk_vm_cust_value_index	inner join	manytoone
Y	6	BIDABASE	dbo	vf_customer_meas	dk_vm_customer_status	dbo	vm_customer_status	pk_vm_customer_status	inner join	manytoone
Y	7	BIDABASE	dbo	vf_customer_meas	dk_vm_comp_reporting_struct	dbo	vm_comp_reporting_struct	pk_vm_comp_reporting_struct	inner join	manytoone
Y	8	BIDABASE	dbo	vf_customer_meas	dk_vm_geography	dbo	vm_geography	pk_vm_geography	inner join	manytoone
Y	9	BIDABASE	dbo	vf_customer_meas	dk_vm_currency	dbo	vm_currency	pk_vm_currency	inner join	manytoone
Y	13	BIDABASE	dbo	vf_sale_txn	dk_vm_sale_date	dbo	vm_sale_date	pk_vm_sale_date	inner join	manytoone
Y	14	BIDABASE	dbo	vf_sale_txn	dk_vm_product	dbo	vm_product	pk_vm_product	inner join	manytoone
Y	15	BIDABASE	dbo	vf_sale_txn	dk_vm_customer	dbo	vm_all_cust	pk_vm_all_cust	inner join	manytoone
Y	16	BIDABASE	dbo	vf_sale_txn	dk_vm_sale_minute	dbo	vm_sale_minute	pk_vm_sale_minute	inner join	manytoone
Y	17	BIDABASE	dbo	vf_sale_txn	dk_vm_currency	dbo	vm_currency	pk_vm_currency	inner join	manytoone
Y	18	BIDABASE	dbo	vf_sale_txn	dk_vm_customer_demographic	dbo	vm_all_demographic	pk_vm_all_demographic	inner join	manytoone

This worksheet produced code similar to the following when processed by SeETL DesignTime.

```
alter table dbo.vf_ap_invoice_line
add constraint vf_ap_invoice_line_c1 foreign key (dk_vm_invoice_date)
references dbo.vm_invoice_date(pk_vm_invoice_date)
```

There is an obvious connection between each line of the “Data Model Joins” worksheet and the drop and create constraints sql that is generated.

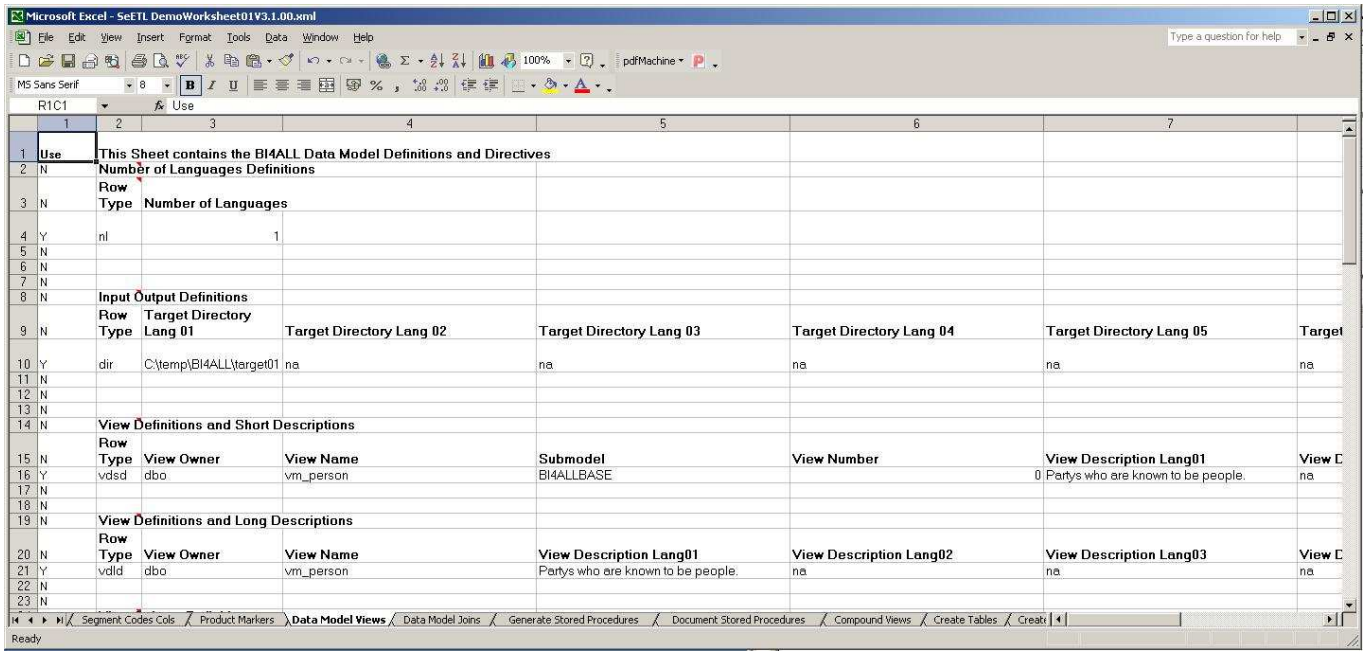
Here is an example transaction table from the documentation database that has the primary key and RI constraints defined into SQL Server. Clearly the creation of the constraints works as no one would bother creating that many create constraints clauses by hand. There are over 5,000 constraints in the BI4ALL data models. The generated create constraints code is over 12,000 lines of SQL.



When defining “tables” in the data modelling worksheet there are a number of aspects to be defined for the generation to take place.

1. The number of languages to be supported.
2. The output directories for the generated SQL for each of the languages.
3. The short and long descriptions of each “table” being defined.
4. The table definitions themselves.

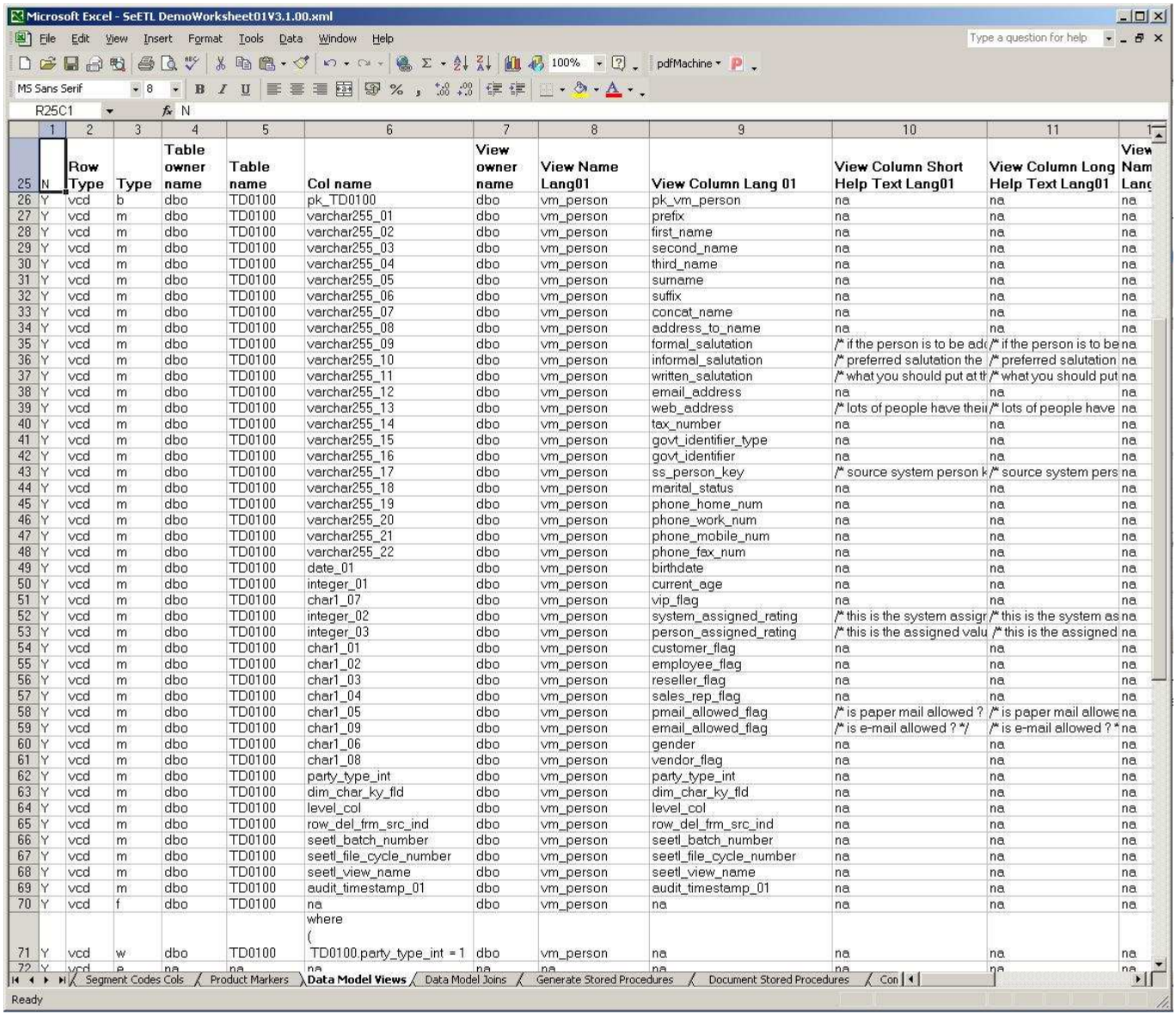
Here is a screen shot of how the first three things are defined in the workbook. It is very simple.



In the screen shot you can see how there are numerous target directories and there are numerous short and long description fields. This is the support for up to 10 languages.

The “table” that is being defined is “vm_person”. This is a view that presents those parties that are known to be persons.

The screen shot below shows how an individual view is created. The SQL will then also be reproduced below. For people who have used data modelling tools this screen shot will be self evident. The view vm_person is being created over the top of the table TD0100.



Microsoft Excel - SeETL DemoWorksheet01V3.1.00.xml

	1	2	3	4	5	6	7	8	9	10	11	12
25	N	Row Type	Type	Table owner name	Table name	Col name	View owner name	View Name Lang01	View Column Lang 01	View Column Short Help Text Lang01	View Column Long Help Text Lang01	View Nam Lang
26	Y	vcd	b	dbo	TD0100	pk_TD0100	dbo	vm_person	pk_vm_person	na	na	na
27	Y	vcd	m	dbo	TD0100	varchar255_01	dbo	vm_person	prefix	na	na	na
28	Y	vcd	m	dbo	TD0100	varchar255_02	dbo	vm_person	first_name	na	na	na
29	Y	vcd	m	dbo	TD0100	varchar255_03	dbo	vm_person	second_name	na	na	na
30	Y	vcd	m	dbo	TD0100	varchar255_04	dbo	vm_person	third_name	na	na	na
31	Y	vcd	m	dbo	TD0100	varchar255_05	dbo	vm_person	surname	na	na	na
32	Y	vcd	m	dbo	TD0100	varchar255_06	dbo	vm_person	suffix	na	na	na
33	Y	vcd	m	dbo	TD0100	varchar255_07	dbo	vm_person	concat_name	na	na	na
34	Y	vcd	m	dbo	TD0100	varchar255_08	dbo	vm_person	address_to_name	na	na	na
35	Y	vcd	m	dbo	TD0100	varchar255_09	dbo	vm_person	formal_salutation	/* if the person is to be ad	/* if the person is to be	na
36	Y	vcd	m	dbo	TD0100	varchar255_10	dbo	vm_person	informal_salutation	/* preferred salutation the	/* preferred salutation	na
37	Y	vcd	m	dbo	TD0100	varchar255_11	dbo	vm_person	written_salutation	/* what you should put at th	/* what you should put	na
38	Y	vcd	m	dbo	TD0100	varchar255_12	dbo	vm_person	email_address	na	na	na
39	Y	vcd	m	dbo	TD0100	varchar255_13	dbo	vm_person	web_address	/* lots of people have thei	/* lots of people have	na
40	Y	vcd	m	dbo	TD0100	varchar255_14	dbo	vm_person	tax_number	na	na	na
41	Y	vcd	m	dbo	TD0100	varchar255_15	dbo	vm_person	govt_identifier_type	na	na	na
42	Y	vcd	m	dbo	TD0100	varchar255_16	dbo	vm_person	govt_identifier	na	na	na
43	Y	vcd	m	dbo	TD0100	varchar255_17	dbo	vm_person	ss_person_key	/* source system person k	/* source system pers	na
44	Y	vcd	m	dbo	TD0100	varchar255_18	dbo	vm_person	marital_status	na	na	na
45	Y	vcd	m	dbo	TD0100	varchar255_19	dbo	vm_person	phone_home_num	na	na	na
46	Y	vcd	m	dbo	TD0100	varchar255_20	dbo	vm_person	phone_work_num	na	na	na
47	Y	vcd	m	dbo	TD0100	varchar255_21	dbo	vm_person	phone_mobile_num	na	na	na
48	Y	vcd	m	dbo	TD0100	varchar255_22	dbo	vm_person	phone_fax_num	na	na	na
49	Y	vcd	m	dbo	TD0100	date_01	dbo	vm_person	birthdate	na	na	na
50	Y	vcd	m	dbo	TD0100	integer_01	dbo	vm_person	current_age	na	na	na
51	Y	vcd	m	dbo	TD0100	char1_07	dbo	vm_person	vip_flag	na	na	na
52	Y	vcd	m	dbo	TD0100	integer_02	dbo	vm_person	system_assigned_rating	/* this is the system assigr	/* this is the system as	na
53	Y	vcd	m	dbo	TD0100	integer_03	dbo	vm_person	person_assigned_rating	/* this is the assigned valu	/* this is the assigned	na
54	Y	vcd	m	dbo	TD0100	char1_01	dbo	vm_person	customer_flag	na	na	na
55	Y	vcd	m	dbo	TD0100	char1_02	dbo	vm_person	employee_flag	na	na	na
56	Y	vcd	m	dbo	TD0100	char1_03	dbo	vm_person	reseller_flag	na	na	na
57	Y	vcd	m	dbo	TD0100	char1_04	dbo	vm_person	sales_rep_flag	na	na	na
58	Y	vcd	m	dbo	TD0100	char1_05	dbo	vm_person	pmail_allowed_flag	/* is paper mail allowed ?	/* is paper mail allowe	na
59	Y	vcd	m	dbo	TD0100	char1_09	dbo	vm_person	email_allowed_flag	/* is e-mail allowed ? */	/* is e-mail allowed ? *	na
60	Y	vcd	m	dbo	TD0100	char1_06	dbo	vm_person	gender	na	na	na
61	Y	vcd	m	dbo	TD0100	char1_08	dbo	vm_person	vendor_flag	na	na	na
62	Y	vcd	m	dbo	TD0100	party_type_int	dbo	vm_person	party_type_int	na	na	na
63	Y	vcd	m	dbo	TD0100	dim_char_ky_fld	dbo	vm_person	dim_char_ky_fld	na	na	na
64	Y	vcd	m	dbo	TD0100	level_col	dbo	vm_person	level_col	na	na	na
65	Y	vcd	m	dbo	TD0100	row_del_frm_src_ind	dbo	vm_person	row_del_frm_src_ind	na	na	na
66	Y	vcd	m	dbo	TD0100	seetl_batch_number	dbo	vm_person	seetl_batch_number	na	na	na
67	Y	vcd	m	dbo	TD0100	seetl_file_cycle_number	dbo	vm_person	seetl_file_cycle_number	na	na	na
68	Y	vcd	m	dbo	TD0100	seetl_view_name	dbo	vm_person	seetl_view_name	na	na	na
69	Y	vcd	m	dbo	TD0100	audit_timestamp_01	dbo	vm_person	audit_timestamp_01	na	na	na
70	Y	vcd	f	dbo	TD0100	na	dbo	vm_person	na	na	na	na
71	Y	vcd	w	dbo	TD0100	where (TD0100.party_type_int = 1	dbo	vm_person	na	na	na	na
72	Y	vcd	e	na	na	na	na	na	na	na	na	na

Segment Codes Cols | Product Markers | Data Model Views | Data Model Joins | Generate Stored Procedures | Document Stored Procedures | Con

Ready

The sql that is generated from this input is as follows.

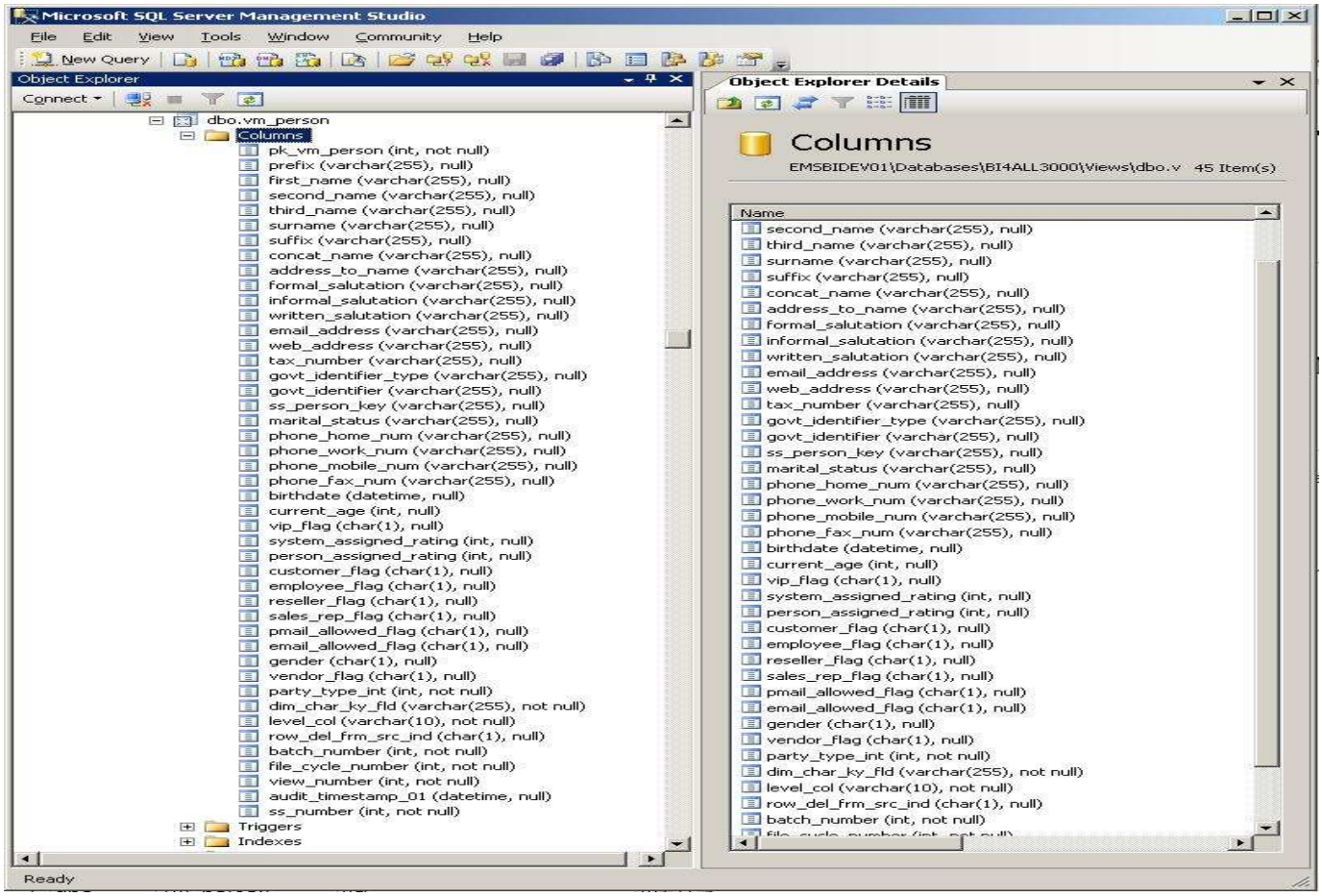
```
create view dbo.vm_person as select
 TD0100.pk_TD0100                                pk_vm_person
,TD0100.varchar255_01                            prefix
,TD0100.varchar255_02                            first_name
,TD0100.varchar255_03                            second_name
,TD0100.varchar255_04                            third_name
,TD0100.varchar255_05                            surname
,TD0100.varchar255_06                            suffix
,TD0100.varchar255_07                            concat_name
,TD0100.varchar255_08                            address_to_name
,TD0100.varchar255_09                            formal_salutation
,TD0100.varchar255_10                            informal_salutation
,TD0100.varchar255_11                            written_salutation
,TD0100.varchar255_12                            email_address
,TD0100.varchar255_13                            web_address
,TD0100.varchar255_14                            tax_number
,TD0100.varchar255_15                            govt_identifier_type
,TD0100.varchar255_16                            govt_identifier
,TD0100.varchar255_17                            ss_person_key
,TD0100.varchar255_18                            marital_status
,TD0100.varchar255_19                            phone_home_num
,TD0100.varchar255_20                            phone_work_num
,TD0100.varchar255_21                            phone_mobile_num
,TD0100.varchar255_22                            phone_fax_num
,TD0100.date_01                                  birthdate
,TD0100.integer_01                               current_age
,TD0100.char1_07                                 vip_flag
,TD0100.integer_02                               system_assigned_rating
,TD0100.integer_03                               person_assigned_rating
,TD0100.char1_01                                 customer_flag
,TD0100.char1_02                                 employee_flag
,TD0100.char1_03                                 reseller_flag
,TD0100.char1_04                                 sales_rep_flag
,TD0100.char1_05                                 pmail_allowed_flag
,TD0100.char1_09                                 email_allowed_flag
,TD0100.char1_06                                 gender
,TD0100.char1_08                                 vendor_flag
,TD0100.party_type_int                           party_type_int
,TD0100.dim_char_ky_fld                          dim_char_ky_fld
,TD0100.level_col                                level_col
,TD0100.row_del_frm_src_ind                      row_del_frm_src_ind
,TD0100.batch_number                             batch_number
,TD0100.file_cycle_number                        file_cycle_number
,TD0100.view_number                              view_number
,TD0100.audit_timestamp_01                       audit_timestamp_01
,TD0100.ss_number                                ss_number
from dbo.TD0100
where
(
 TD0100.party_type_int  = 1  /* party type 1 = person */
and TD0100.level_col  = 'detail'
and TD0100.char1_04  = 'Y'
)
or
TD0100.pk_TD0100 = 0
```

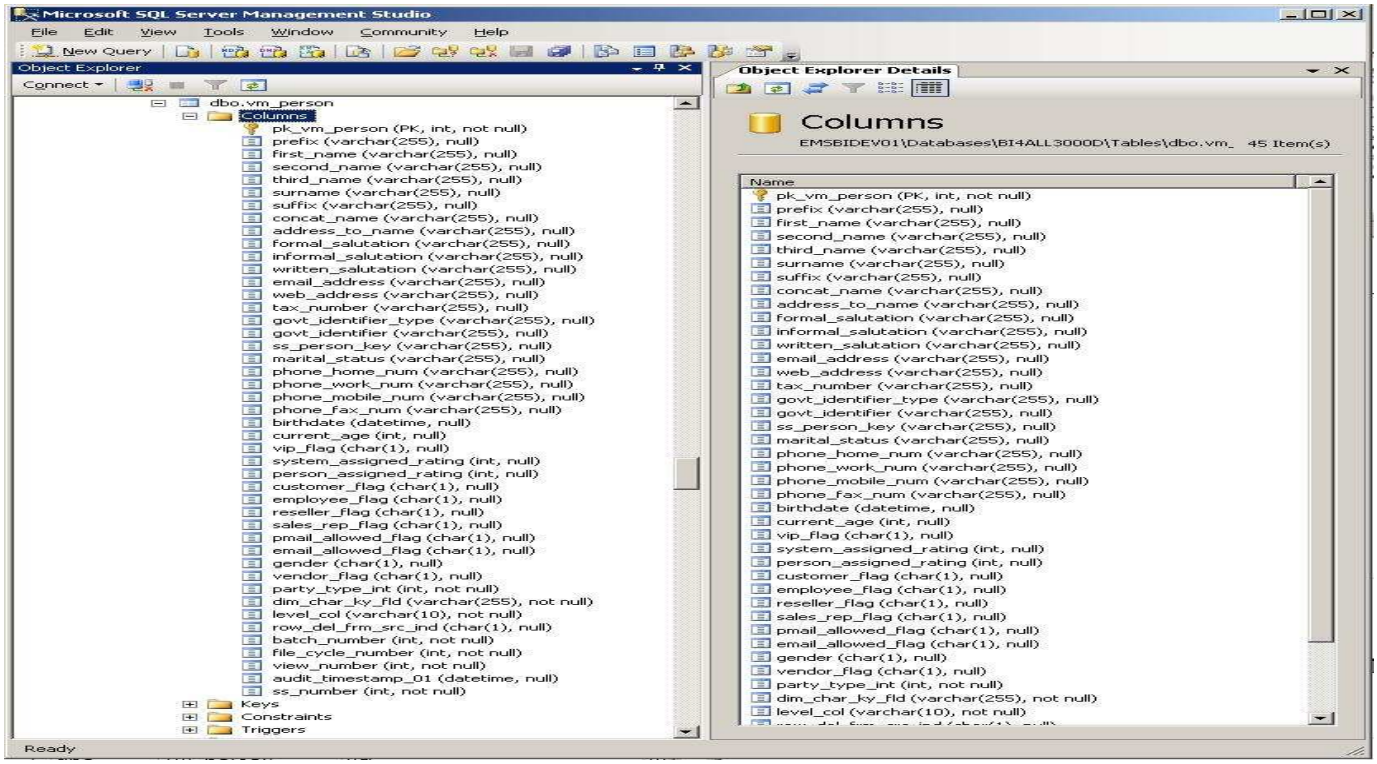
The correlation between the input in the workbook and the generated view statement is obvious.

Just as obviously, if the DBAs deem that it is necessary to change the table name or the real columns of the tables due to corporate standards it can be seen that this is a relatively easy job. The underlying names are changed and the exposed names remain the same.

It is also obvious that any data model can be maintained this way. The tool is a generalised tool that can maintain data models and generated the sql code to generate the views and the tables in the documentation database from the workbooks.

Just to prove that the view is created as well as the table being created in the target databases here are the screen shots for these. The first is the view and the second is the table in the documentation database. They have been compressed a little merely to put them onto the same page.





The reader will notice that the views are created over the top of meaningless table names and in the standard BI4ALL product as delivered the views co-habit tables. Each “table” (which is really a view) is given a “table number” and this is used to constrain the selection of data out of the underlying table.

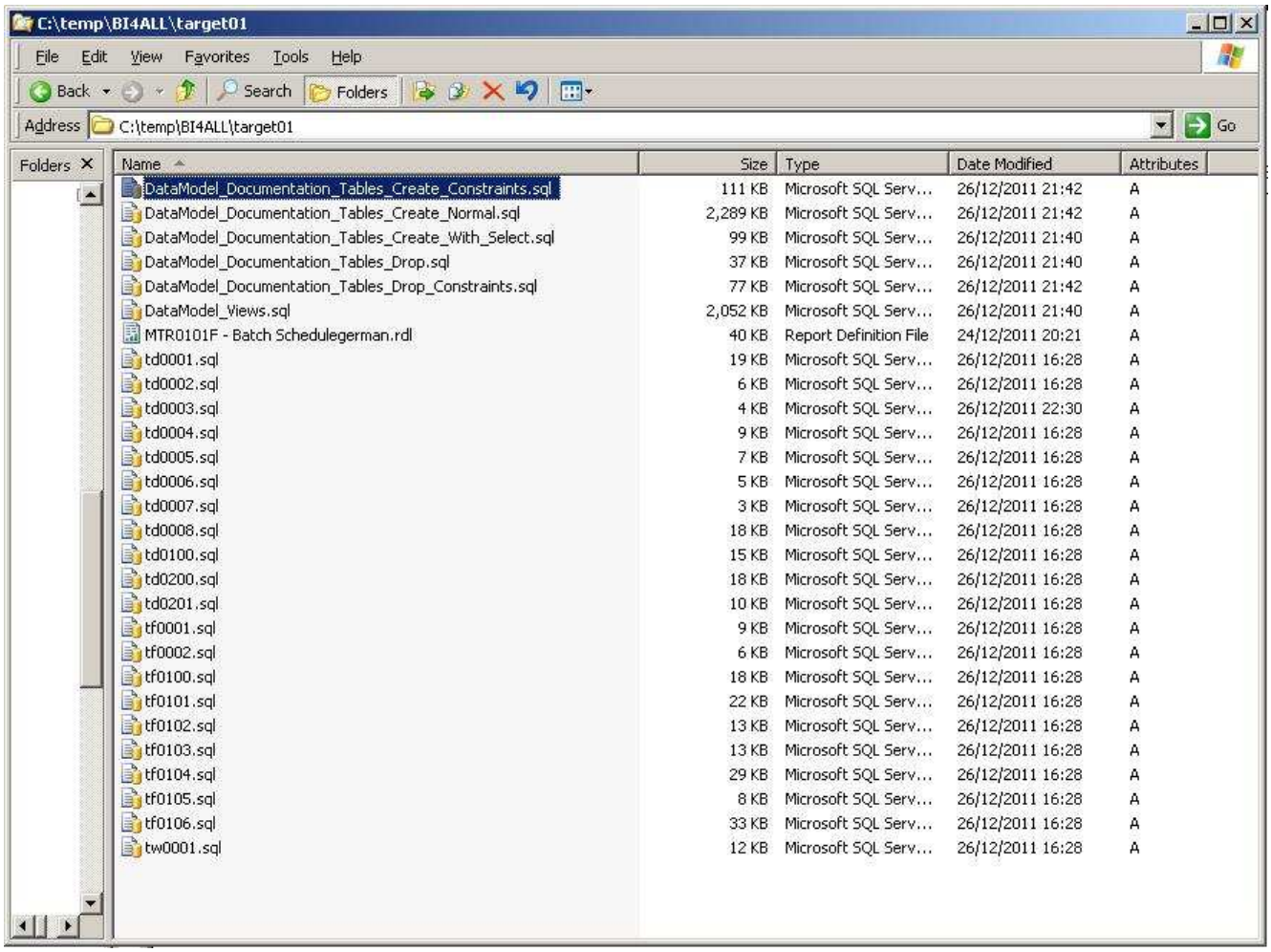
This technique allows the DBA to have total control over the co-habitation of data in single physical tables. In databases like Sybase IQ and Netezza twin fin there is practically no overhead for having “tables” co-habiting physical tables. Therefore the DBA is free to create fewer database objects for the data and manage the presentation of that data out through the views.

As the system is deployed and performance stats are gathered the DBA has the option of moving data out of a co-habited table and into its own table.

On databases like SQL Server and Oracle this kind of co-habiting has the ability to vastly reduce the number of objects that need to be maintained by the DBAs. Always a welcome thing.

It is important to stress that the documentation database makes it possible for the DBAs to use an ER tool should they choose. IBI uses Enterprise Architect from Sparx Systems as a useful data model documentation tool. Though, to be honest, it is again a lot of work to present data models in such a tool.

The SeETL DesignTime tool reads all these definitions and places the generated SQL out into a directory for the DBAs to review prior to running it against the database. Here is a screen shot of the output folder for the generation of the BI4ALL models. The file “Data_Model_Views.sql” is 2MB and contains 34,800+ lines of generated SQL for all the views.



Now that we have shown how the data model is maintained let us move on to the data models themselves.

5.2. Design Features of the BI4ALL Data Models

This section discusses the design features of the BI4ALL Data Models. It has been largely reproduced from the BI4ALL Data Models User Guide. This User Guide is a confidential document and so only portions of the materials have been reproduced into this document.

5.2.1. The Table Types in BI4ALL

There are a number of 'types' of tables in BI4ALL. They are as follows:

1. Type 1 dimension tables.
2. Type 2 dimension tables.
3. Fact tables that are detailed transactions.
4. Fact tables that are snapshot or summary/periodic fact tables.
5. Type 1 association tables.
6. Type 2 association tables.

Each of these will be described in some detail below. Examples will also be given.

5.2.2. Naming Standards for BI4ALL Tables and Views

The BI4ALL Data Models have a set of naming standards that the reader is obliged to learn and understand to be able to read the data models. They are as follows:

TDNNNN = '**T**'able '**D**'imension with a sequence number for the table number.
TFNNNN = '**T**'able '**F**'act with a sequence number for the table number.

VM_descriptive_name = '**V**'iew di'**M**'ension with a descriptive name for the view.
VF_descriptive_name = '**V**'iew '**F**'act with a descriptive name for the view.

The suffix of dimension tables can tell you something about what is in them. The common suffixes for dimension tables are as follows.

Suffix	Description
_status	The dimension table contains some form of status information about objects. This could be things like bank account statuses, delivery statuses etc.
_code	The dimension table contains some form of code information about objects. This could be things like bank account codes, delivery codes etc.
_type	The dimension table contains some form of type information about objects. This could be things like bank account types, delivery type, building types etc.
_inds	The dimension table contains some form of indicator information about objects. This could be things like bank account indicators, contract indicators etc.
_method	The dimension table contains some form of method information about objects. This could be things like customer contact methods, followup methods, payment methods etc.
_terms	The dimension table contains some form of terms information about contracts in most cases. For example payment terms for contracts, freight on board terms for shipping, subscription terms for a media subscription and the like.
_date	The dimension table is a view over the date table that shows the corporate calendar in days.
_period	The dimension table is a view over the date table that shows the periods of the corporation. They may be monthly, and are set to monthly 'out of the box', but can also be other periods as the client might wish to construct.
_week	The dimension table is a view over the date table that exposes weeks as according to the clients definition of a week.
_minute	The dimension table is a view over the minute table to present the minutes of a 24 hour day as 1440 records.

The suffix of fact tables can tell you something about what is in them. The common suffixes for fact tables are as follows.

Suffix	Description
_txn	The fact table contains individual transactions generated out of some operational system or derived by the ETL system that indicates some sort of change to some sort of object that generated some sort of transaction. The examples are endless. Sales transactions, bank account transactions, phone calls, short messages, invoice transactions. You name it. Operational computer systems capture LOTS of transactions now.
_header	The fact table contains header information about some object. The typical header information is things like invoice headers, General Ledger headers, order headers and the like.
_line	The fact table contains line information about some object. The typical line information is things like invoice lines, order line and the like.
_meas	The fact table contains sets of measures calculated on a periodic basis that are used for trending and reporting. This is typically called a 'snapshot' table. The transactions are sifted and calculated to calculate the important measures that are stored away once a period. That period can be such things as daily, weekly, monthly or every quarterly and others.
_plan	The fact table contains a list of important measures that are the plans that go out into the future for the set of measures. It will also contain the historical plans that were calculated. Typically plans are not calculated from transactions as they are calculated out in to the future. There are many ways plans get created. Most usually they are created on spreadsheets based on prior years performance with adjustments made to those prior years performance. The plan tables are essential for performing 'plan vs actual' comparisons.
_asoc	The fact table associates two or more dimension tables together to provide the relationships between the dimension tables. The class association table is the party_address_association. This associates a party with an address for a reason for a period. For example a party might have a home address, a work address, a post office box. They might be held for a purpose and for a period. They will inevitably change over time. There are two types of association table. The first only associates two dimensions and the reason for association is obvious. This is the case for the party to address association. It is also the case with the party to party association that contains such things as 'husband and wife' associations or 'company owns company' associations etc. The less intuitive association until discussed is the type of association that binds many dimension tables to one association record. This is the case for such association tables as party dimensions association.

5.2.3. Type 1 Dimension Tables

It is expected that all readers know what a Type 1 dimension table is. It is a table where you have an integer key, the string key being translated to the integer key, and then fields that are associated with that string key from the source system.

In BI4ALL the string key to be used to find the record in a dimension table is known as the dim_char_ky_fld. This field is limited to varchar 255. The data warehouse designer is expected to create a key that is unique at any given time for a dimension table considering the passage of time and the fact that source data may come from many source systems.

There are a number of different ways to construct the dim_char_ky_fld in ***SeETL*** and they are explained in the relevant documentation.

In a Type 1 dimension table the row in the table is over-written when a change is detected.

A typical Type 1 Dimension table is the vm_party view.

Here is part of the SQL for the vm_party view. This is readily readable by anyone who has done any level of work in Business Intelligence

```
create view dbo.vm_party as select
TD0100.pk_TD0100                                pk_vm_party
,TD0100.varchar255_01                           prefix
,TD0100.varchar255_02                           first_name
,TD0100.varchar255_03                           second_name
,TD0100.varchar255_04                           third_name
,TD0100.varchar255_05                           surname
,TD0100.varchar255_06                           suffix
,TD0100.varchar255_07                           concat_name
,TD0100.varchar255_08                           address_to_name
,TD0100.varchar255_09                           formal_salutation
,TD0100.varchar255_10                           informal_salutation
…
,TD0100.date_01                                 birthdate
,TD0100.integer_01                              current_age
,TD0100.char1_07                                vip_flag
,TD0100.integer_02                              system_assigned_rating
,TD0100.integer_03                              person_assigned_rating
,TD0100.char1_01                                customer_flag
,TD0100.char1_02                                employee_flag
,TD0100.char1_09                                email_allowed_flag
,TD0100.char1_06                                gender
…..
,TD0100.party_type_int                          party_type_int
,TD0100.party_type_sdesc                        cust_type_sdesc
,TD0100.party_type_ldesc                        cust_type_ldesc
,TD0100.char1_01                                char1_01
,TD0100.dim_char_ky_fld                         dim_char_ky_fld
,TD0100.level_col                               level_col
,TD0100.row_del_frm_src_ind                     row_del_frm_src_ind
,TD0100.batch_number                            batch_number
,TD0100.file_cycle_number                       file_cycle_number
,TD0100.view_number                             view_number
,TD0100.audit_timestamp_01                      audit_timestamp_01
,TD0100.ss_number                               ss_number
from dbo.TD0100
where
(
 TD0100.level_col  = 'detail'
)
or
 TD0100.pk_TD0100 = 0
   ;
```

Let us explain the various component of this view such that the reader can understand how to read other views.

```
create view dbo.vm_party as select
```

All BI4ALL Data Model ‘tables’ are actually views presented over tables. The underlying table names are NOT fixed and can be easily changed by the DBA. The DBA has FULL CONTROL over the naming standards of the tables while the views may not be changed. Most DBAs do not have a concern about the view names but nearly ALL DBAs have physical naming standards for objects that they would like to follow.

IBI has had many instances of prospective clients asking us to alter the naming standards of the physical tables. One of the biggest changes from previous versions of the data models is that we have completely moved away from having meaningful names for the underlying tables.

```
TD0100.pk_TD0100                                pk_vm_party
```

Every table has a primary key which is an integer and the naming standard is pk_<table_name>

```
,TD0100.varchar255_01                           prefix
,TD0100.varchar255_02                           first_name
,TD0100.varchar255_03                           second_name
,TD0100.varchar255_04                           third_name
```

This is a typical set of fields that you will find in any party table. They are the elements of the name.

In the ***BI4ALL Data Models*** as they come out of the box these fields are varchar 255 as indicated by the name of the underlying columns.

In this example it can be seen that the underlying field names have been aliased to expose them as real names that have fairly understandable values such as first, second and third names.

```
…
,TD0100.date_01                                 birthdate
,TD0100.integer_01                              current_age
,TD0100.char1_07                                vip_flag
,TD0100.integer_02                              system_assigned_rating
,TD0100.integer_03                              person_assigned_rating
,TD0100.char1_01                                customer_flag
,TD0100.char1_02                                employee_flag
,TD0100.char1_09                                email_allowed_flag
,TD0100.char1_06                                gender
…..
```

In this example it can be seen that the underlying fields can have different data types and that the data types are pretty obvious from the field names. Fields that are ‘date_’, ‘integer_’, ‘char1_’ etc have obvious data types. Or at least we hope they are obvious data types.

You can notice other rules such that ‘flag’ is usually char1 but it is sometimes an integer as well. Fields that are indicators are almost always integers.

```
,TD0100.party_type_int                          party_type_int
,TD0100.party_type_sdesc                        cust_type_sdesc
,TD0100.party_type_ldesc                        cust_type_ldesc
```

The party table has a specific field that is the party type which is set to 1, 2 or 3 depending on whether it is a person, a company or a household.

We would draw attention to the common use of sdesc and ldesc for short descriptions and long descriptions.

```
,TD0100.dim_char_ky_fld                    dim_char_ky_fld
,TD0100.level_col                          level_col
```

All dimension tables have these two columns. The dim char key field is the string that uniquely identifies the row in the table at a point in time. In a type 1 dimension table it is across all time.

The level column is a column that is set to 'detail' for dimension tables that are at the detail level. The field can then have 'level1' through to 'level9' for the 10 levels of a dimension table. There are 10 levels of detail possible in any dimension table in the ***BI4ALL Data Models***. The ***BI4ALL Data Models*** support the multi-level summary ideas that were made popular by Ralph Kimball in the 80s when he was working at Metaphor. However this knowledge about how to build multi-level data models has disappeared. The SeETL Runtime C++ Engine was built with full multi-level data model support built right into it. This is because the Consultant that built it was trained at Metaphor. However we have seen very few clients actually use this feature.

```
,TD0100.row_del_frm_src_ind                row_del_frm_src_ind
```

The vast majority of detailed level tables have this indicator on them. This is to signify that this row has been deleted from the operational system but that it lives on in the data warehouse. Because data warehouses are supposed to be no-volatile the records that are deleted from the operational systems should not be deleted from the data warehouse. However it can be very handy to know that the record has been deleted from the operational system.

```
,TD0100.batch_number                       batch_number
```

This is the batch number in which this record was processed. It is placed on all rows in the data warehouse to give the support staff a fighting chance to remedy problems for the 'latest batch' should there be a significant problem in the ETL processing. A good example of a very difficult to resolve problem without a restore is 'double processing' of transactions in a single batch. This happens. Because ever row in every fact table will carry the latest batch number if it was affected by that batch it is possible to delete the rows processed in the data warehouse and then reprocess the batch. This is always much faster than a restore in the sizes of the data warehouses we are seeing now.

```
,TD0100.file_cycle_number                  file_cycle_number
```

Every time a file is processed in a batch the file cycle number is forwarded. It is then placed on all rows in the data warehouse to give the support staff a fighting chance to remedy problems for the 'latest batch' should there be a significant problem in the ETL processing. A good example of a very difficult to resolve problem without a restore is 'double processing' of a set of transactions. With the file cycle number in place it is possible to delete all the rows that were inserted or updated in the last round of processing. This is not 'automatically' what should be done. The support team must do some work to find out what exactly needs to be done. But the file cycle number and batch number give them a better chance of sorting this out.

```
,TD0100.view_number                        view_number
```

An innovation that we saw that we liked was to place the 'mapping name' onto each row that is processed so it can be easily determined what mapping inserted or last updated some row. Therefore if there is a problem on the row the culprit mapping can be more easily determined. In many cases more than one mapping inserts or updates rows in the one table. Some times many mappings update the one row as is often the case with vm_party.

Over a few implementations it became obvious that the text of the mapping name was a bit lengthy and that it should be replaced with a number that points to the name. So the view number is now an allocated number which points to the name of the mapping. Whether the client bothers to place that data into the database as well is up to the client.

```
,TD0100.audit_timestamp_01                    audit_timestamp_01
```

It is VERY highly recommended that the timestamp of the extraction of the record out of the staging area is recorded onto every row going into the data warehouse. This gives the support personnel a MUCH better chance to know what happened when mistakes occur. Therefore this field is on all views in the model and we would urge clients to consider that it is mandatory..

```
,TD0100.ss_number                             ss_number
```

The Source System Number is an integer allocated to tell the data warehouse which source system this row came from. It is a field that can be used to find data from difference source systems more easily in a single table. There are cases when a single row has data from many source systems, of course, and then the client is advised to create some other number series to indicate the combination of data or to set it to the default of 0 to say that it is not possible to know exactly which source system the row came from.

```
from dbo.TD0100
```

Notice that tables are made up of meaningless names in the model as it comes 'out of the box'.

```
where
(
 TD0100.level_col  = 'detail'
)
or
 TD0100.pk_TD0100 = 0
   ;
```

Dimension Tables all contain a field called 'level_col' as described above. Out of the box this is constrained to 'detail' on the assumption that should levels be later placed into the model that current views will need no change.

If the DBA wishes to remove this constraint then he is free to do so. We would recommend commenting it out rather that deleting it.

The default for SeETL in attribution processing when a row is not found is to set the dimension table key to zero. Therefore a 'zero row' must exist in all dimension tables to be able to be connected to any fact table where the lookup failed.

Zero keys can then be searched for to determine broken referential integrity in fact tables.

Another field commonly found across MOST tables is the table number. The vm_party view is one of the FEW that do not have this field exposed. It is similar to this.

```
where TD0001.table_number=11021
```

Many tables have table numbers that allow for data to be co-hosted in underlying tables. Data can be migrated between tables at the expressed desire of the DBA for the data warehouse. All physical placement of data is the sole responsibility of the DBA and the ***BI4ALL Data Models*** are 'agnostic' as to which physical tables data is stored in and on what drives they are stored. It is expected the DBA will be responsible for such decisions and the BI4ALL Data Models give him the flexibility he needs to build the database according to the parameters of his company.

5.2.4. Type 2 Dimension Tables

The best table to talk about as a type 2 dimension table is the vm_product table. Every data warehouse must have some form of 'product' that is being delivered via what ever means.

A portion of the vm_product view is as follows.

```
create view dbo.vm_product  as select
 TD0200.pk_TD0200                                pk_vm_product
,TD0200.varchar255_01                            sku_number
,TD0200.varchar255_02                            barcode_num
,TD0200.varchar255_03                            prod_name
,TD0200.varchar255_04                            prod_sub_group
,TD0200.varchar255_05                            prod_group
,TD0200.varchar255_06                            prod_category
,TD0200.integer_01                               shelf_life_days
,TD0200.money_01                                 unit_alloc_cost
,TD0200.money_02                                 unit_abc_cost
,TD0200.money_03                                 list_price_per_unit
,TD0200.varchar255_22                            unit_of_measure_code
,TD0200.varchar255_23                            unit_of_measure_sdesc
,TD0200.char1_01                                 discount_allowed
,TD0200.decimal_01                               discount_rate_1
,TD0200.decimal_02                               discount_rate_2
,TD0200.decimal_03                               discount_rate_3
,TD0200.decimal_04                               discount_rate_4
,TD0200.decimal_05                               discount_rate_5
,TD0200.varchar255_12                            discount_sdesc_1
,TD0200.varchar255_13                            discount_ldesc_1
,TD0200.date_from                                date_from
,TD0200.date_to                                  date_to
,TD0200.current_flag                             current_flag
,TD0200.level_col                                level_col
,TD0200.dim_char_ky_fld                          dim_char_ky_fld
,TD0200.table_number                             table_number
,TD0200.row_del_frm_src_ind                      row_del_frm_src_ind
,TD0200.batch_number                             batch_number
,TD0200.file_cycle_number                        file_cycle_number
,TD0200.view_number                              view_number
,TD0200.audit_timestamp_01                       audit_timestamp_01
,TD0200.ss_number                                ss_number
from dbo.TD0200
where ( TD0200.level_col = 'detail' and TD0200.table_number = 146 ) or TD0200.pk_TD0200 = 0
```

The pieces that are different are as follows.

```
,TD0200.date_from                                date_from
,TD0200.date_to                                  date_to
```

All type 2 dimensions have the starting date and the ending date of the record. If this is the current record then the ending date is ctl_month_control.high_date.

The prior record is closed out at ctl_batch_control.batch_date minus one day and the current record is inserted with ctl_batch_control.batch_date as the current date_from.

```
,TD0200.current_flag                             current_flag
```

The current flag is set to 0 for rows which are no longer current and 1 to the current flag. The key for the record is the dim_char_ky_fld plus the level_col plus the date_from in most cases. However, if multiple rows are processing on the ONE batch_processing_date it is possible to have duplicates of these fields in multiple rows. Therefore the one and only key that can be trusted as a key is the pk_ field for the type 2 dimension.

For lookups only the current row will ever be looked up. There is no ability to look up by date range.

5.2.5. Fact Tables That are Detailed Transactions

Here is a typical fact table that is a transaction table. It is the simplest fact table of them all, the sales fact table. Virtually all commercial corporate data warehouses have something similar to a sales fact table. The following pages will discuss the components of a standard fact table that is a transaction fact table.

```
create view dbo.vf_sale_txn as select
 TF0101.pk_TF0101                              pk_vf_sale_txn
,TF0101.dk_dim_01                              dk_vm_sale_date
,TF0101.dk_dim_02                              dk_vm_sale_minute
,TF0101.dk_dim_03                              dk_vm_sub_campaign
,TF0101.dk_dim_04                              dk_vm_product
,TF0101.dk_dim_05                              dk_vm_customer
,TF0101.dk_dim_06                              dk_vf_customer_dims_asoc
,TF0101.dk_dim_07                              dk_vm_account
,TF0101.dk_dim_08                              dk_vf_account_dims_asoc
,TF0101.dk_dim_09                              dk_vm_customer_demographic
,TF0101.dk_dim_11                              dk_vm_customer_geography
,TF0101.dk_dim_12                              dk_vm_geo_code
,TF0101.dk_dim_13                              dk_vm_map_reference
,TF0101.dk_dim_14                              dk_vm_response_type
,TF0101.dk_dim_15                              dk_vm_currency
,TF0101.dk_dim_16                              dk_vm_sales_rep
,TF0101.dk_dim_17                              dk_vm_unit_of_measure
,TF0101.dk_dim_20                              dk_vm_txn_type
,TF0101.dk_dim_21                              dk_vm_card
,TF0101.dk_dim_22                              dk_vm_card_type
,TF0101.dk_dim_23                              dk_vm_comp_reporting_struct
,TF0101.dk_dim_24                              dk_vm_account_status
,TF0101.dk_dim_25                              dk_vm_account_type
,TF0101.dk_dim_26                              dk_vf_campaign_dims_asoc
,TF0101.money_01                               sale_unit_amount
,TF0101.money_02                               sale_extended_amount
,TF0101.money_03                               cost_unit_amount
,TF0101.money_04                               cost_extended_amount
,TF0101.money_05                               tax1_unit_amount
,TF0101.money_06                               tax1_extended_amount
,TF0101.money_07                               tax2_unit_amount
,TF0101.money_08                               tax2_extended_amount
,TF0101.integer_01                             sale_units
,TF0101.varchar255_01                          appsys_reference_number_str
,TF0101.varchar255_02                          sale_sdesc
,TF0101.varchar255_03                          sale_ldesc
,TF0101.timestamp_01                           sale_tstamp
,TF0101.integer_02                             dk_vf_solicitation_txn
,TF0101.integer_03                             number_sales
,TF0101.char1_01                               sale_txn_returned_flag
,TF0101.char1_02                               return_period_expired_flag
,TF0101.decimal_01                             return_probability_pct
,TF0101.table_number                           table_number
,TF0101.row_del_frm_src_ind                    row_del_frm_src_ind
,TF0101.partitioning_column                    partitioning_column
,TF0101.partitioning_column_part_01            partitioning_column_part_01
,TF0101.partitioning_column_part_02            partitioning_column_part_02
,TF0101.batch_number                           batch_number
,TF0101.file_cycle_number                      file_cycle_number
,TF0101.view_number                            view_number
,TF0101.audit_timestamp_01                     audit_timestamp_01
,TF0101.ss_number                              ss_number
from dbo.TF0101
where TF0101.table_number = 532
```

The features of the standard fact table that are necessary to understand are as follows:

```
create view dbo.vf_sale_txn as select
```

The fact views are created over the top of underlying tables as per the same as dimension tables and dimension views.

```
 TF0101.pk_TF0101                                pk_vf_sale_txn
```

There is always a primary key on a fact table and it is unique. We do a lot of fact to fact table joins and these primary keys make it much easier and faster for the database to join two fact tables together. These fact to fact table joins often happen in the stored procedures.

```
,TF0101.dk_dim_01                                dk_vm_sale_date
,TF0101.dk_dim_02                                dk_vm_sale_minute
,TF0101.dk_dim_03                                dk_vm_sub_campaign
….
```

The dimension and fact tables that the fact tables are linked to are specified as the next set of keys They are always of the naming standard dk_<table name>. Note that fact tables are often linked to fact tables and an example follows.

On most fact tables there is a date or period field that is placed first as most queries are constrained by some element of time. Some tables then have a ‘minute’ field to be able to break the transactions up into minutes across the day.

```
,TF0101.dk_dim_21                                dk_vm_card
,TF0101.dk_dim_22                                dk_vm_card_type
,TF0101.dk_dim_23                                dk_vm_comp_reporting_struct
,TF0101.dk_dim_24                                dk_vm_account_status
,TF0101.dk_dim_25                                dk_vm_account_type
,TF0101.dk_dim_26                                dk_vf_campaign_dims_asoc
```

As can be seen from the **vf_campaign_dims_asoc** field there are fact to fact table joins in the model. Also note how many dimension tables are commonly joined in the model. In this case there are 26+ tables joined to the sales transaction table in the model.

```
,TF0101.money_01                                 sale_unit_amount
,TF0101.money_02                                 sale_extended_amount
,TF0101.money_03                                 cost_unit_amount
,TF0101.money_04                                 cost_extended_amount
,TF0101.money_05                                 tax1_unit_amount
,TF0101.money_06                                 tax1_extended_amount
,TF0101.money_07                                 tax2_unit_amount
,TF0101.money_08                                 tax2_extended_amount
```

A fact table may have many money fields such as the unit and extended amounts as well as the unit tax and extended tax amounts for items being sold on a sales transaction.

```
,TF0101.integer_01                               sale_units
,TF0101.integer_03                               number_sales
```

A fact table may have many money integer files such as the number of units that this sales record refers to. transaction. The second field called number_sales is set to ‘1’. It is a field that can be summed to count the number of sales that occurred and it is safer than a count(*). The reason for this is as follows.

Some items are sold in groups of products and the group of product is counted as one sale. A good example of this is telephone handsets. The handset, the cover, the sim are counted as one sale where all three occur together. In this case there would be three sales records because three things were sold but only one would have a ‘number sales’ set to 1. It becomes impossible to filter on the data inside the row at query time because, for example, if someone comes in and buys a cover for an existing handset because the old cover was damaged this too is one sale.

There are often issues around counting the ‘number of sales’ or, indeed, the number of anything. The data models suggests that this is always managed by a stored integer and to not use count (*).

,TF0101.varchar255_01 **appsys_reference_number_str**

The **appsys_reference_number_str** deserves a special mention. Many operational systems assign keys to transaction records, indeed MANY records, that are unique to that source system. These keys can be very useful to have in the data warehouse to be able to go back and connect the record to the original from the source system.

Values of key pieces of data can be verified etc. Virtually every dimension table that is a 'data' table and nearly every transaction fact table have the field **appsys_reference_number_str** so that the IT support can find their way back to the key in the operational system and link the data back to the operational system.

,TF0101.varchar255_02 **sale_sdesc**
,TF0101.varchar255_03 **sale_ldesc**

Short and long descriptions of data is also common on the BI4ALL tables. This would have been quite frowned on in years gone past. But disk is getting cheaper all the time now.

,TF0101.timestamp_01 **sale_tstamp**

Timestamps are quite common on transaction fact tables. This is the timestamp of the sales transaction and not the timestamp of the placement of the row into the data warehouse of course.

,TF0101.integer_02 **dk_vf_solicitation_txn**

This field is a good example of how a fact to fact table link works. In this example if this sale resulted from a specific solicitation that can be tracked then a link is made to the solicitation record at the time the ETL occurs. The link is via the unique integer key as this is the fastest link that can be made. In this way it can be seen how well the solicitations are driving sales and how many sales are happening without being necessarily linked to any solicitation.

,TF0101.char1_01 **sale_txn_returned_flag**
,TF0101.char1_02 **return_period_expired_flag**
,TF0101.decimal_01 **return_probability_pct**

Here are three good examples of how flags are now much more useful on fact tables. Many sales transactions can have their sale reversed by a return of the product at a later date. This is particularly a problem for the catalogue retail businesses. They have very generous returns policies because this is required to encourage people to buy from catalogues with confidence.

Therefore when counting sales it's good to know if the product can no longer be returned, or if indeed it was returned. That is what the first two flags are for.

Also in retail the likelihood that the product will be returned degrades over time until the return period expires. And even then sometimes returns are accepted. So by creating a 'return probability' the client can calculate the estimated actual sales taking into account the expected returns. All very useful.

,TF0101.table_number **table_number**

Every fact table gets a number that is used to tag it's rows so that it can co-habit with other fact view rows in the one physical table. This example has the sales fact table tagged as table number 532.

,TF0101.row_del_frm_src_ind **row_del_frm_src_ind**

Every fact transaction record has the row deleted from source flag to indicate if the row has been deleted from the source system. This is very useful for auditing, fraud prevention and security purposes. If people are made aware that all changes in the operational system is logged into the data warehouse they will be less likely to attempt fraud.

```
,TF0101.partitioning_column                    partitioning_column
,TF0101.partitioning_column_part_01            partitioning_column_part_01
,TF0101.partitioning_column_part_02            partitioning_column_part_02
```

These three fields were introduced for SQL Server 2005. SQL Server 2005 introduced partitioning for tables. However there was only one value that was allowed to be used as input to the partitioning strategy. This was quite limiting. We developed a solution where there were two parts to the partitioning column. The first part was the element of time be it month of day etc. The second part was the corporate reporting structure. This meant that in large multi-national implementations, or even just large organisational implementations, the data could be physically partitioned by time and by country (or organisational unit). By creating this two part partitioning key along the same lines as the data was queried we were able to improve performance in those accounts where we implemented this type of solution.

This partitioning strategy does not relate to other databases that have more sophisticated partitioning strategies.

```
,TF0101.batch_number                           batch_number
,TF0101.file_cycle_number                      file_cycle_number
,TF0101.view_number                            view_number
,TF0101.audit_timestamp_01                     audit_timestamp_01
,TF0101.ss_number                              ss_number
from dbo.TF0101
where TF0101.table_number = 532
```

And in the same way as dimension tables all fact tables have the audit fields added to them to give the IT Support a good chance to recover any damage to the database.

5.2.6. Fact Tables That are Snapshot or Summary/Periodic Fact Tables

The next type of fact table is one that rolls up transactions into some set of measures tables that can trend over time the facts in the transactions. It is very common for some more complex calculations to be performed on the transactions to calculate the values in the fact tables. This can be such things as cost allocations etc.

It is important to note that because summary or snapshot fact tables are summarised up from the detailed transactions it is less common for them to retain string data that is descriptive and fields that link back to operational systems make no sense.

```
create view dbo.vf_sale_meas as select
 TF0101.pk_TF0101                              pk_vf_sale_meas
,TF0101.dk_dim_01                              dk_vm_sale_meas_period
,TF0101.dk_dim_02                              dk_vm_sub_campaign
,TF0101.dk_dim_03                              dk_vm_product
,TF0101.dk_dim_04                              dk_vm_customer_demographic
,TF0101.dk_dim_05                              dk_vm_customer_geography
,TF0101.dk_dim_06                              dk_vm_currency
,TF0101.dk_dim_07                              dk_vm_unit_of_measure
,TF0101.dk_dim_08                              dk_vm_comp_reporting_struct
,TF0101.dk_dim_09                              dk_vf_campaign_dims_asoc
,TF0101.money_01                               sale_unit_amount
,TF0101.money_02                               sale_extended_amount
,TF0101.money_03                               cost_unit_amount
,TF0101.money_04                               cost_extended_amount
,TF0101.money_05                               tax1_unit_amount
,TF0101.money_06                               tax1_extended_amount
,TF0101.money_07                               tax2_unit_amount
,TF0101.money_08                               tax2_extended_amount
,TF0101.integer_01                             sale_units
,TF0101.table_number                           table_number
,TF0101.row_del_frm_src_ind                    row_del_frm_src_ind
,TF0101.partitioning_column                    partitioning_column
,TF0101.partitioning_column_part_01            partitioning_column_part_01
,TF0101.partitioning_column_part_02            partitioning_column_part_02
,TF0101.batch_number                           batch_number
,TF0101.file_cycle_number                      file_cycle_number
,TF0101.view_number                            view_number
,TF0101.audit_timestamp_01                     audit_timestamp_01
,TF0101.ss_number                              ss_number
from dbo.TF0101
where TF0101.table_number = 531
```

```
,TF0101.dk_dim_01                              dk_vm_sale_meas_period
```

Notice that the element of time is listed as ‘period’ and not necessarily monthly or weekly as the client might select a different period.

```
,TF0101.dk_dim_02                              dk_vm_sub_campaign
,TF0101.dk_dim_03                              dk_vm_product
,TF0101.dk_dim_04                              dk_vm_customer_demographic
,TF0101.dk_dim_05                              dk_vm_customer_geography
,TF0101.dk_dim_06                              dk_vm_currency
,TF0101.dk_dim_07                              dk_vm_unit_of_measure
,TF0101.dk_dim_08                              dk_vm_comp_reporting_struct
,TF0101.dk_dim_09                              dk_vf_campaign_dims_asoc
```

Notice that there are far fewer keys once the data is aggregated. The client may choose to put more keys back onto the table but these are the minimum keys for this table. Notice that in this example there is no link to customer at all, just the customer demographics.

```
,TF0101.money_01                    sale_unit_amount
,TF0101.money_02                    sale_extended_amount
,TF0101.money_03                    cost_unit_amount
,TF0101.money_04                    cost_extended_amount
,TF0101.money_05                    tax1_unit_amount
,TF0101.money_06                    tax1_extended_amount
,TF0101.money_07                    tax2_unit_amount
,TF0101.money_08                    tax2_extended_amount
,TF0101.integer_01                  sale_units
```

Notice quite a limited number of fact fields on the table. These are the key measures to be used to measure summaries of the business.

```
,TF0101.table_number                table_number
,TF0101.row_del_frm_src_ind         row_del_frm_src_ind
,TF0101.partitioning_column         partitioning_column
,TF0101.partitioning_column_part_01 partitioning_column_part_01
,TF0101.partitioning_column_part_02 partitioning_column_part_02
,TF0101.batch_number                batch_number
,TF0101.file_cycle_number           file_cycle_number
,TF0101.view_number                 view_number
,TF0101.audit_timestamp_01          audit_timestamp_01
,TF0101.ss_number                   ss_number
from dbo.TF0101
where TF0101.table_number = 531
```

The normal audit fields are retained. The row deleted from source has been mostly redundantly retained.

5.2.7. Fact Tables That are Plan Tables

The next type of fact table to be familiar with is the plan table. These are not heavily populated in most clients as most clients have not built plans that they manage the business to as effectively as might be done. They often do not place them into the data warehouse.

Plan tables are always almost exactly the same as the ‘measures’ fact tables. This is because the numbers that are planned are very frequently a subset of the measures calculated. If it is important enough to plan a number over time then it is usually important enough to calculate it from business performance after the event from the transactions generated.

It is possible to have more measures on the measures tables where there are simply more things calculated that were not considered worth the effort to calculate into the plan. But it is rare to plan a number for performance monitoring and then not have it be part of the measures table.

```
,TF0101.dk_dim_21                                dk_vm_plan
```

The only significant difference between the above sales measures table and this plan measures table is the field **dk_vm_plan.** It points to the plan table that looks as follows.

```
create view dbo.vm_plan as select
 TD0001.pk_TD0001                                pk_vm_plan
,TD0001.varchar255_01                            plan_code
,TD0001.varchar255_02                            plan_sdesc
,TD0001.varchar255_03                            plan_ldesc
,TD0001.date_01                                  plan_start_date
,TD0001.date_02                                  plan_end_date
,TD0001.varchar255_04                            plan_hier_01_code
,TD0001.varchar255_05                            plan_hier_01_sdesc
,TD0001.varchar255_06                            plan_hier_01_ldesc
,TD0001.varchar255_07                            plan_hier_02_code
,TD0001.varchar255_08                            plan_hier_02_sdesc
,TD0001.varchar255_09                            plan_hier_02_ldesc
,TD0001.varchar255_10                            plan_hier_03_code
,TD0001.varchar255_11                            plan_hier_03_sdesc
,TD0001.varchar255_12                            plan_hier_03_ldesc
,TD0001.varchar255_13                            plan_hier_04_code
,TD0001.varchar255_14                            plan_hier_04_sdesc
,TD0001.varchar255_15                            plan_hier_04_ldesc
,TD0001.varchar255_16                            plan_hier_05_code
,TD0001.varchar255_17                            plan_hier_05_sdesc
,TD0001.varchar255_18                            plan_hier_05_ldesc
,TD0001.integer_01                               plan_version
,TD0001.char1_01                                 current_plan_flag
,TD0001.level_col                                level_col
,TD0001.dim_char_ky_fld                          dim_char_ky_fld
,TD0001.table_number                             table_number
,TD0001.row_del_frm_src_ind                      row_del_frm_src_ind
,TD0001.batch_number                             batch_number
,TD0001.file_cycle_number                        file_cycle_number
,TD0001.view_number                              view_number
,TD0001.audit_timestamp_01                       audit_timestamp_01
,TD0001.ss_number                                ss_number
from dbo.TD0001
where ( TD0001.level_col = 'detail' and TD0001.table_number = 144) or TD0001.pk_TD0001 = 0
```

As can be seen form the SQL the plan table is a type 2 dimension table that records versions of plans over time. So the client can create many plans and then compare the plan versus actual by the specific plan that is linked to the plan table.

This means that clients can build plans with many versions calculated different ways and then compare and contrast with plan calculation method was ‘most correct’ after the event so as to improve the planning process into the future.

5.2.8. Type 1 Association Tables

The Type 1 association tables simply relate two pieces of dimensional data to each other. The simplest and best example of this is associating a party with an address. The BI4ALL Data Models define addressed to be independent of parties. This means that an address has an existence even if no-one lives there or no-one owns it.

Further. It is very common for parties to have multiple addresses such as home address, work address and a post office box address as well.

This view gives us a good example of what a type 1 association table looks like.

```
create view dbo.vf_party_address_asoc as select
 TF0002.pk_TF0002                                pk_vf_party_address_asoc
,TF0002.asoc_key_TF0002                          asoc_key_vf_party_address_asoc
,TF0002.dk_dim_01                                dk_vm_party
,TF0002.dk_dim_02                                dk_vm_address
,TF0002.dk_asoc_code                             dk_vm_asoc_code
,TF0002.dk_asoc_role                             dk_vm_asoc_role
,TF0002.dk_date_from                             dk_vm_date_from
,TF0002.dk_date_to                               dk_vm_date_to
,TF0002.date_from                                date_from
,TF0002.date_to                                  date_to
,TF0002.current_flag                             current_flag
,TF0002.asoc_code                                asoc_code
,TF0002.asoc_role                                asoc_role
,TF0002.asoc_amount1                             asoc_amount1
,TF0002.asoc_amount2                             asoc_amount2
,TF0002.integer_01                               party_type
,TF0002.table_number                             table_number
,TF0002.row_del_frm_src_ind                      row_del_frm_src_ind
,TF0002.partitioning_column                      partitioning_column
,TF0002.partitioning_column_part_01              partitioning_column_part_01
,TF0002.partitioning_column_part_02              partitioning_column_part_02
,TF0002.batch_number                             batch_number
,TF0002.file_cycle_number                        file_cycle_number
,TF0002.view_number                              view_number
,TF0002.audit_timestamp_01                       audit_timestamp_01
,TF0002.ss_number                                ss_number
from dbo.TF0002
where ( TF0002.table_number = 309 ) or TF0002.pk_TF0002 = 0
```

```
TF0002.pk_TF0002                                pk_vf_party_address_asoc
```

As per standards there is an integer key at the front.

```
,TF0002.asoc_key_TF0002                          asoc_key_vf_party_address_asoc
```

The ‘asoc key’ is a string that is the key to the association table at any point in time. Therefore, given this is the party to address association key this string field must contain some form of key to uniquely identify an address and a party at that address.

```
,TF0002.dk_dim_01                                dk_vm_party
```

Next comes the field that is the key to the first element of association. Obviously in the party address association table it is the party key.

```
,TF0002.dk_dim_02                                dk_vm_address
```

And, of course, the next key must be the second key that is being associated to the first key. Obviously in the party address association table it is the address key.

```
,TF0002.dk_asoc_code                    dk_vm_asoc_code
```

The association between two dimension is given a code by the client and this code can be anything that the client would like it to be. In this case there in not really a ‘code’ to have. But in other cases such as the party to party association there are codes to have. The most obvious code is ‘marriage’ or ‘married’. That is two parties are married to each other and the code for the relationship is married or spouses or something similar that client would like to be able to constrain on.

```
,TF0002.dk_asoc_role                    dk_vm_asoc_role
```

Entities can be related for a ‘role’. The obvious roles for the party address association table are such things as home address, work address, post office box.

For other association tables such as the party to party association table the roles are things like ‘wife’ or ‘husband’ for those relationships that are about ‘marriage’ or ‘spouse’.

There can be other relationships such as ‘son of’ or ‘father of’ or ‘mother of’ or ;daughter of’.

```
,TF0002.dk_date_from                    dk_vm_date_from
,TF0002.dk_date_to                      dk_vm_date_to
,TF0002.date_from                       date_from
,TF0002.date_to                         date_to
,TF0002.current_flag                    current_flag
```

Entities can be related for a period of time. The relationship can be current or not current. In this way the data of the association is stored rather like a type 2 dimension as it has from date and to date as well as a current flag on the record. Associations are associated only at the day level and the time does not go down to seconds or fractions of second.

```
,TF0002.asoc_code                       asoc_code
,TF0002.asoc_role                       asoc_role
```

The dimension tables that the association code and role point to have some of the text denormalised onto the association record as well just to make it easier to see what the association is for support developers.

```
,TF0002.asoc_amount1                    asoc_amount1
,TF0002.asoc_amount2                    asoc_amount2
```

Associations can be for an amount. For example a party (man) might own 100 shares in a company. In this case the first amount might be 100. He might have bought these 100 shares for EUR5,000 so the second amount might be 5,000. Other associations might be things like reporting structures where a person might report to another person for 50% of their time and someone else another 50% of their time.

The amount that is stored is very much up to the client. For the party address association no amount would be stored as there is not a sensible ‘amount’ field involved in the definition.

```
,TF0002.integer_01                      party_type
```

In the party address association the party type is redundantly stored so that it can be over typed for other associations such as person address association to list addresses of people or company address association to list addresses of companies.

```
,TF0002.table_number                    table_number
```

Just like every other fact table association tables have table numbers applied to them.

```
,TF0002.row_del_frm_src_ind             row_del_frm_src_ind
```

If the record of the association can be deleted in the operational system and this can be detected the following field can be used to note the deletion.

```
,TF0002.partitioning_column                    partitioning_column
,TF0002.partitioning_column_part_01            partitioning_column_part_01
,TF0002.partitioning_column_part_02            partitioning_column_part_02
```

The partitioning columns for fact tables are also applied to association tables as they too can grow to be quite large. This is SQL Server 2005 only partitioning. We have not yet checked SQL Server 2008 to see if it needs similar partitioning.

```
,TF0002.batch_number                           batch_number
,TF0002.file_cycle_number                      file_cycle_number
,TF0002.view_number                            view_number
,TF0002.audit_timestamp_01                     audit_timestamp_01
,TF0002.ss_number                              ss_number
from dbo.TF0002
where ( TF0002.table_number = 309 ) or TF0002.pk_TF0002 = 0
```

The normal audit fields are also applied. Further, it should be noted that because association tables are often linked to fact tables a zero record must be exposed. This is a record where the key is 0 so that joins to other fact tables do not fail with an RI constraint failure.

The ETL system is responsible for ensuring RI as RI constraints are turned off inside the database for any data warehouse.

5.2.9. Type 2 Association Tables

Type 2 associations are not discussed in detail in this document. It is sufficient to say that they allow us to store historical information inside the dimensional modelling paradigm. IBI has detailed documentation and presentations on type 2 associations which we would be pleased to share in confidence.

5.2.10.Everything is Linked to Everything

Every Data Warehouse Data Modeller has faced the eternal question in his designs of data warehouses.

"Which dimension tables do I link to this fact table?"

This has given many of us sleepless nights. The reason being that the more links made the more disk and processing used to put the key there, however, when a key is left off and it turns out that it was needed? The change request to change the fact table can be quite expensive.

The linking of dimension tables to fact tables has been one of the 'pain points' in data warehouse development for all our professional data warehouse modelling lives.

On one project that a colleague who will simply be referred to as 'Sam' and this author were doing together we were in the very unusual position that there was no wall space near us to paste the data models onto the walls. Our usual practice was to print large diagrams and put the data models onto the walls so that we could learn them and understand them visually. This is 'normal'.

However, at this client we were in the middle of an open plan office and there was no close wall space. It also seemed too rude to plaster data model diagrams next to other peoples work spaces and discuss them so close to other people. This was for a media organisation and there were lots and lots of links we were wanting to put into the fact tables. The number of links was exploding.

We went to dinner one night and we discussed this problem over dinner. We joked that the models were getting more and more complex and there was no 'expansion slot' on our heads. We wondered aloud if there might not be a way of designing databases where it would no longer be necessary to remember what was linked to what and it was no longer necessary to draw diagrams.

It seemed like an amazing thing to wonder if it might be possible. And we both went to our respective apartments joking about 'getting older' and 'remembering less' as happens to us humans.

Amazingly, that night, this author came up with the idea to make this possible!

This author discussed this with 'Sam' in the morning who immediately 'got' the idea and approved strongly that we should try it out. Indeed Sam congratulated this author on beating him to this idea because, as with most good ideas, it was both simple and elegant.

The idea was to place an integer key onto all the association tables, both type 1 and type 2, and to link the type 2 association tables freely to the fact tables.

There is another really important fact to consider with this data modelling technique. Marketing departments are always buying external data and trying to get it integrated into the data warehouse. Of course, this data is not known at data warehouse design time. It can't be known. But after they buy this data they want to link it to the fact tables along with customer data and they want it linked to as many fact tables as possible.

This brings up a very difficult question. Does the designer put the data onto the customer record? Perhaps in a new table and try and use hot swappable dimension keys for the new data and the customer data so he does not have to place new keys on to fact tables? Or does he create new keys on fact tables and link the new data to the new keys on fact tables.

We have all faced this question. At least those of us who have been around long enough.

Suffice to say we usually put the data onto the customer dimension table as a compromise.

The type 2 associations tables give us a new choice.

The new data can be introduced as a separate dimension table that is dedicated to this new source of data, as it should be. Then the new dimension data is attached to the party dimensions association table. Only the ONE table needs to be changed and only the ONE mapping needs to be changed. Because the key for the party dimensions association table is already on all fact tables the new dimensional data will be immediately available to be joined to the fact tables.

Indeed? If the ETL developers can determine the validity of this data into the past then it is possible to add the key to the new dimension table to the party dimensions association table for historical records. This is going to then make the new data immediately available to the historical records that the ETL team can properly link to the party dimensions association table.

This is what is meant by the box on the slide above called ‘any customer data’. Any customer data can be added to the data model far more easily and for more elegantly than ever before.

One more important point to note is also this. There is no rule that says the dimension tables linked to the party dims association can not be directly linked to a fact table like a sales fact table. For example? The demographics table is linked to BOTH the sales fact table and the party dimension association table.

This redundant join is at practically no cost.

Now. The OTHER really important thing to note is this.

By placing MANY type 2 association tables onto ALL fact tables we create a kind of ‘mesh’ of joins for all the tables. Practically everything is joined to everything. And by linking so many tables to so many tables via the type 2 associations it then becomes quite feasible to not need to remember what tables are linked to what tables. There are too many links to remember in any case!

The designer can go forward based on the idea

“Everything is linked to everything.”

That, or course, is the sales pitch. The truth is still a very powerful:

“Everything that should be linked is linked, and if it happens to not be linked then it is only a very small amount of work to link it.”

Indeed? In the C++ version of ***SeETL*** it has a feature to automatically determine if there are new keys added to a fact table that are on the association tables and it links these tables in the ETL with no ETL changes necessary are all. All that is necessary is to expose the new columns to the ETL subsystem and it will ‘auto-detect’ the change and alter the ETL as appropriate. Quite remarkable.

By using these techniques it is now possible for a data warehouse database designer to design and build a data warehouse without the need for diagrams of the data warehouse. Indeed. It is possible to build the data warehouse without remembering the joins are involved in the mind of the designer.

5.3. Coverage of the BI4ALL Data Models

This section discusses, as openly as possible, the coverage of the BI4ALL data models including the submodels of Retail, Web, Media and Telco.

BI4ALL can be acquired in a number of ways. Consulting houses might acquire all models so as to be able to refer across models for ideas. Clients might simply ask for the data models to be used to deliver their own customised data model and wish to never know what is “under the covers”. In these days of outsourcing all that is not essential to the core business IBI sees that there will be many clients who simply give us the job of building their data warehouse for them.

The base model, BI4ALL, is required for any implementation. This is where key base information such as parties, addresses, contracts, sales, campaigns etc are located.

The vertical models over the top that are available today are specific to industry segments. They are:

- Retail
- Web
- Media
- Telco.
  Please Note. IBI retains a close working relationship with Sean Kelly and Associates.
  It is anticipated that most Telco projects would be a join project between SKA and IBI.

It is obvious that IBI has table names, table descriptions, column names and column descriptions as part of its confidential documentation of the BI4ALL Data Models. There are reports that print all these materials for our clients who are licensed to use BI4ALL Data Models.

It is just as obvious that to release that kind of documentation is to run the risk of the ideas of the data model being pilfered by other people. So this section of the document will describe the areas that are covered by the data models.

Should the reader wish to know more about the data models this will require a video conference for IBI Staff to present further details.

The final decision as to whether to use BI4ALL or not usually requires an on site visit where full details of the data models can be disclosed to a group of qualified staff who are able to participate in some form of decision making process. The usual process is for there to be a voting or balloting system where qualified staff cast their ballot/vote to acquire the BI4ALL data models or not.

This is a 6 hour session that is quite intense. Business people are invited for the first hour or so.

The remainder of the day is technical detail that only data modellers and DBAs are going to understand.

5.3.1. The Base Model BI4ALL Data Coverage

The base model covers the following areas. They are listed in the order of the table names and not particularly grouped for the purposes of this table.

Firstly we will present the dimensional type of information.

Business Area	Description
Time information	Time is slices into Minutes, hours, days, weeks, months, quarters and years. There are also financial quarters etc.
Codes	There is a very wide range of code tables created and it is easy to extend the codes tables. Each codes table is allocated a codes name and codes are stored with the code name typing the record. Examples of codes include but are not limited to: • Account close reason code. • Contract end reason code. • Association reason code. • GL account code reason code. • Product return reason code. • Geography geo code. • Customer loyalty code. • Segment code for the segmentation engine. • Segmentation type code to explain what sort of segmentation is being done. • Stats type code to explain what sort of stats is being reported in the segmentation engine. • Credit limit breach code. • Customer contact code. • Customer debt treatment code. • Customer save effort code. • File code to explain what type of file this is. • File transfer code to explain how the file is transferred. • File transfer failure reason code. • Fraud close code to explain how the fraud activity was closed down. • Fraud method code to explain how a fraud was performed. • Fraud type code to explain the type of fraud. • Fraud reason code • Dispute code. • Payment error Code. • Payment error type code. • Taxation reason code. • Account limit type code. • Customer type code. • Credit classification code. • Credit breach limit action code. • Schedule type code to explain the type of schedule under consideration. • Task type code for the task in the schedule. • Source file code to explain the source file in more detail. • Target file code to explain the target file in more detail.
Statuses	There is a very wide range of statuses implemented Each status table is allocated a status name and statuses are stored with the status name typing the record. Examples of statuses include but are not limited to: • Account status • Accounts payable account status. • Accounts payable invoice status. • Contract status • Customer Contact status.

	• Customer status. • Delivery status. • GL account status. • GL Header Status. • Order Status. • Transaction Status. • Credit limit breach status. • Customer debt treatment status. • Customer save effort status. • File status. • File transfer status. • Fraud status. • Dispute status. • Payment error status. • Schedule status.
Types	There is a very wide range of Types implemented Each type table is allocated a type name and types are stored with the type name typing the record. Examples of types include but are not limited to: • Adjustment types to explain account adjustments. • Blacklist type to explain the type of blacklisting for the party. • Document type to explain the type of document being dealt with. • Exchange rate type to explain the type of exchange rate being dealt with. • Campaign type to explain the type of campaign being dealt with. • Account type. • Accounts payable account type. • Building type. • Card type. For example credit, debit, amex, visa. • Contract type. • GL account type. • GL Transaction type. • Response type for campaigns. • Segmentation type to explain the type of segmentation being dealt with. • Solicitation type such as mail, catalog, email, phone, meeting. • Statistics type to explain the type of stats being dealt with.
Parties	Most data warehouses have the idea of Party. These are people, companies or households. They can also be sales reps, primary sales reps, vendors, customers etc. There is a wide variety of information about Parties stored and this is usually enhanced by collecting as much information as possible from operational systems. There is also a party archive.
Addresses	All the addresses that parties have had are able to be recorded. There is also an address archive.
Accounts	There are all sorts of businesses that maintain accounts. There is a generic account model to support accounts.
Accounts Payable	There are all sorts of businesses that maintain accounts payable. There is a generic account model to support accounts payable.
Behaviour Scores Financial Scores Credit ratings Customer value indexes Credit Breaches Fraud	Many companies score customers on behaviours, especially financial behaviours. There is a scoring model for these areas. Companies that extend credit also have to deal with credit ratings, credit breaches, fraud and many other aspects of offering credit.
Campaign Management Solicitation types Response Types	In BI4ALL the word Campaign means any initiative to influence the performance of the business, not just marketing campaigns. Marketing campaigns are a subset of the BI4ALL campaigns submodel.

Campaign Exclusions Opt in-out.	There are three levels of campaigns, group, campaign, sub campaign. This has been very robust over a long period of time.
Contracts	Many companies write contracts with their customers. This contract information is critical information to the organisation and is widely used in many companies.
Customer Contacts and Followups	Virtually all companies have contacts with their customers. The customers have contact preferences both in mechanisms and in time of day. And follow-ups are best dealt with via the customers stated preference. This entire area is also modelled in BI4ALL.
Company Reporting Structures	All companies have at least one reporting structure. Many have more than one structure overlayed over the top of the base reporting structure. Further, companies alter their reporting structures on a regular basis. The company reporting structure table is a type 2 table that records all changes to all reporting structure elements over time and is connected to all fact tables where appropriate. It is one of the most important tables after the party table.
Delivery and building information for deliveries	Many companies deliver products to buildings or warehouses or residentially. In these cases all deliveries need a rich set of dimensional attributes. Further, for companies like UPS or newspapers they need to know if they can deliver their product to an address if no prior appointment is made. A lot of time and money can be wasted delivering to addresses where no delivery is safely possible.
GL Accounts, Headers, Books, periods, Transaction types	Virtually all companies have General Ledgers. All sizable companies need to analyse their general ledgers. There is a generalised GL submodel that allows basic analysis of a standardised General Ledger.
Ethnicity, Language, Profession, psychographics and other personal attributes	In many industries the personal attributes of the customer can be indicative or even predictive of their preferences. Many companies collect this information to better understand their clients.
Products and Product cross holdings	Virtually all companies sell products. BI4ALL contains a base product dimension that can be extended to define the products of the company. The Product is linked to virtually every fact table. Knowing the level of product cross holding and the products cross held is important to many customers. There is a submodel that allows the storage of the understanding of the product cross holding position of a customer.
Segmentation dimensions Product Markers Relationship Segmentation	BI4ALL comes with a two specific segmentation models already implemented with the ability to store any number of segmentation models. The first is the standard strength and value of relationship segmentation model that many companies implement. The second is newer and much more innovative. IBI calls it “Product Catalog Segmentation”. Early and less sophisticated forms of this have been in place for years. This is the “diapers and beer” example. Product Catalog Segmentation is all about observing what people buy and deducing something about the buyer from what is purchased. In many cases the product purchased tells you a lot about a customer. This idea was pioneered in a very larger pharmacy chain that had loyalty cards. The idea is that each product on the product list tells you something about the customer. The standard PCS engine provides 64 segments (chess board) and every product purchased scores a variable number of points to zero or more PCS segments. The scores for sales are the summed to place customers into PCS segments. PCS also has recency, frequency and monetary value segments. More details on the Product Catalog Segmentation Engine are available from IBI.
File Transfers Job Schedules Assorted maintenance	All companies need to keep track of the movement of data around the company between different systems. Indeed, transfer of data to the data warehouse is just one of the many sets of file transfers that have t o be tracked. There is a submodel that maintains rich dimensional information about the movement of data in files into, out of, and around the company.
Currencies	Many companies operate in multiple currencies. In these cases the various currencies need to be catered for and the exchange rates between these

	currencies need to be taken into account. In general BI4ALL takes the approach that where the same data is to be presented in multiple currencies then two records for the same amount are entered into the fact tables. The exception to this is the telco mode. The reason being that volume in telco are so large that to carry the base reporting currency on the same record is more efficient than carrying a second record.

Secondly let us document the areas that are covered by fact tables in the base model.

Business Area	Description
Addresses to map references	Linking addresses to map references is common. Google does this on the fly. Many companies want to code this into the data warehouse.
Account close transactions	This table contains the transactions that record account closures.
Accounts transactions, measures and plans	This set of tables contains the transactions for account transactions, plans and measures.
Adjustment measures and transactions	This records account adjustments as transactions and measures.
Accounts Payable transactions, headers, lines, measures.	These tables record the accounts payable invoices in detail.
Contract Measures and transactions	These tables record the contract transactions and measures.
Contact Measures and transactions	These tables record the customer contact measures and transactions.
Delivery Measures and transactions	These tables record the delivery transactions and measures.
GL Account transactions GL Account balances GL account headers	This set of tables stores the various facts about GL transactions.
Invoice transactions Invoice headers Invoice lines Invoice measures	The invoice is the most important item in a company. It is the highest quality information in a company as well. So all the information that is available about invoices is able to be stored.
Order transactions Order headers Order lines Order measures	The orders that a company takes that turns into invoices is about the second most important information in a company. These turn into deliveries and invoices. So storing the order transactions is also very well worthwhile.
Product Measures and Plans	These tables store facts about the products as planned and measured.
Sales Plans, measures and transactions	This set of tables stores facts about sales transactions, measures and plans.
Solicitations transactions and measures	This set of tables stores facts about solicitations transactions and measures.
Response transactions and measures	This set of tables stores facts about responses from the outbound solicitations where the link can be made to a level of certainty or simple responses where the link to solicitation can not be determined with a reasonably level of certainty.
Customer debt treatment transactions, plans and measures	This set of tables stores facts about customer debt treatments as transactions, plans and measures.
Customer save efforts as transactions, measures and plans	Many companies run specific efforts to save customers who are predicted to leave or have actually expressed a desire to leave. These tables record the efforts to save customers from leaving the company as customers.
Fraud transactions, plans and measures	Most companies that extend credit suffer some level of fraud. There are usually people who are tasked with finding and stopping fraud. These tables store transactions, plans and measures for transactions that are considered or suspected of being fraudulent.
Credit Limit Breaches transactions, plans and measures.	When customers order products or use products that are in breach of their credit limits the situation needs to be handled delicately and carefully. There is no point being negative to the customer as the company wants to keep them as a customer but the credit breach must be dealt with. So there are tables to take specific and details account of the important business process.
File Transfer Run Logs transactions, plans and	Many organisations transfer files into the organisation, out of the organisation, and between systems inside the organisation. For example into the data warehouse.

measures	This set of tables records the file transfers that are planned, measured, and the transactions from those file transfers.
Segmentation	BI4ALL has a series of segmentation engines that are able to be used to segment customers to provide differentiated service as well as differentiated offers. There is a suite of tables that data from the segmentation engine is placed into for ease of analysis and for historical analysis.

5.3.2. BI4 Retail Data Coverage

BI4ALL was very much originally invented and then developed for the retail industry. There was a special emphasis around loyalty cards in retail as IBI was closely collaborating with Sean Kelly and Associates on a major loyalty card program. It seemed to just “make sense” to build retail data models around loyalty cards.

When IBI consultants were discussing the target market for such a product in 2005 it was clear retail was the market place to go after. This is because there are so many retailers and their level of BI usage is generally very low. Their volumes are generally relatively low (compared to telco) and so SQL Server was an appropriate platform for retail.

IBIs largest client is Key-Work Consulting GmbH (www.key-work.de). Key Work specialises in Retail (though they do have other clients) and can make a very solid claim to be some of the most knowledgeable Retail BI people in Germany if not all of Europe.

Key-Work acquired BI4ALL in 2007 and the development work in the retail and web space has been continuing ever since. IBI has also performed other Retail projects which has further enhanced the capabilities of the retail data model. The segmentation engines were first developed for the retail business by linking product purchases to the loyalty cards in the loyalty card program IBI was involved in.

As a small “advertisement” for Key-Work. Key-Work performs fully hosted BI outsourcing for their clients. They have their own data centre in Karlsruhe, the technology heart of Germany. All staff are able to work in both German and English. They are very well trained and very competent in the use of all IBI products and services.

IBI has absolute confidence that any company in the EU that wishes to fully outsource and host their BI data can do so with the team at Key Work. You can sleep well at night if Key Work are taking care of your BI solutions. The data WILL be there in the morning!

Because we have been involved in a number of Retail data warehouses the coverage of Retail and web is very extensive. There is not much that the model does not cover. It does not cover the merchandising process and the estimation process that the merchandisers go through prior to placing the purchase acquisition with the vendor upstream.

The starting point of the data flows for BI4 Retail is the Purchase Acquisition which is a non binding estimate of what will be ordered from the vendor. It is the “first guess” at what will be ordered and can be quite volatile.

Firstly we will present the dimensional type of information.

Business Area	Description
Time information	All the normal time information is included. There is also a set of dates around planned dates, firm dates, and other dates where the status of an order goes from uncommitted to committed. Often this happens automatically based on dates and replenishment programs. Retailers want to know what their commitments are as wll as their likely commitments.
Codes	There is a very wide range of code tables created and it is easy to extend the codes tables. Each codes table is allocated a codes name and codes are stored with the code name typing the record. Examples of codes include but are not limited to: • Complaint code to describe the complaint. • Freight on board code to describe FOB for this movement.
Statuses	Each status table is allocated a status name and statuses are stored with the status name typing the record. Examples of statuses include but are not limited to: • Outlet Status
Types	Each type table is allocated a type name and types are stored with the type name typing the record. Examples of types include but are not limited to: • Outlet types to describe the various outlets

	• Inventory type • Loyalty card type • Catalog type • Package type • Purchase order header type • Purchase order line type
Person Preparer	The person who prepared purchase requisitions and purchase orders is a party. However, in retailers these people play such key roles that they have been given separate dimensional attributes for analysis purposes.
Buyer	The person who is a buyer for a suite of products is a party. However, in retailers these people play such key roles that they have been given separate dimensional attributes for analysis purposes.
Outlets	Retailers sell their products via outlets. Usually these outlets are stored but more and more retailers are also selling products via the web and home delivery. In Germany DHL has taken the interesting step of putting “pack stations” in many major locations like town centres and train stations. So you can buy something from amazon and have it delivered to the pack station and you will be notified by email or sms when the package is placed into the pack station. This has taken away the big issue of where to deliver a package to that is reliable and safe. IBI expects to see this sort of thing happen in more countries around the world. Outlets are also scored for product performance as well as financials.
Inventory Organisations and Locations	In retail one of the biggest issues is inventory. Retailers have inventory locations that may or may not be under their control. The purchased product flows from the manufacturer to some inventory location. These locations are organised into a hierarchy in most cases. The inventory locations needs to be fully described and every movement of every product inside the inventory location as well as in and out of the inventory location needs to be recorded to be able to gain a good view of the contents of an inventory location at any given time. There are also the issues of measuring the velocity of the inventory movement to ensure that the minimum inventory is kept while minimising sell outs of inventory items.
Purchase Acquisitions Bulk Purchase Agreements, Purchase orders Contracts etc	Retailers buy a lot of goods and services from vendors. Where they are buying a lot from one vendor they enter into bulk purchase agreements which are quite complex contracts. They then buy off this BPAs over the life of the contract. When there is no BPA in place a retailer will create purchase acquisitions, purchase orders, contracts and similar. All this information needs to be captured to be able to analyse and understand the relationship with the vendor when the next round of price negotiations gets underway.
Loyalty Cards and Programs	There is extensive information recorded about loyalty cards and programs so as to be able to determine changes in loyalty of the customer.
Catalogs	Catalog Retailing is very big business in Germany. Much more so than in the US or UK. It is also done very differently than in the US/UK as far as IBI experience goes. This is why detailed information about catalogs has been included in the data models. The data models capture the entire structure of each catalog that is created and then links the sales that can be attributed to a catalog to the item in the catalog. The purpose of all this is to perform analysis on catalog performance and to improve the performance of catalogs over time.
Packages	In catalog retailing an outbound package does not just contain the catalog any more. In places like Germany the outbound package contains a catalog, perhaps a letter, perhaps vouchers, perhaps samples. The outbound catalog mailing is used to carry along other items that have been determined to perhaps be of interest to the customer. All this is recorded and can be analysed.
Point of Sale device	Where a sale is recorded by a point of sale device this device is also recorded. The usual POS device is the cash register but more and more self scanned cash registers are being introduced. Tesco in the UK is a good example of self scanned

	cash registers.
Carriers	There is information stored about carriers of inventory and the freight on board codes and records.

Secondly let us document the areas that are covered by fact tables in the base model. Although this list seems rather small most of the data from retail that is rolled up is stored in the base model tables.

Business Area	Description
Sales Transactions	This set of tables records sales transactions at POS, web, catalog or other outlet. The sales summaries and plans are intended to be support by the base model. These sales transaction records are different enough that it was decided that the vf_sale_txn table was not sufficient.
Purchase Requisition and Order Headers and lines	These tables store every PR and PO header and line that is generated by the operational systems.
Inventory Location and Organisation receipt and exit transactions	These tables capture every movement of every piece of inventory in and out of an inventory location or inventory organisation.
Outlet Product Measures	These tables store the product sales by outlet as well as the total sales by outlet.
Product and catalog associations	This set of tables associate products with catalogs so as to store the relationships between products and catalogs for analysis against the sales performance of the product from the catalog.
Solicitation package transactions	It was decided that the base vf_soliticitation_txn table was not sufficient to store what was desired for the retail catalog companies. Therefore this data was added to the retail sub model.

5.3.3. BI4 Web Data Coverage

Many retailers also have web sites now. The trend is very definitely to put more and more offers onto the web. Therefore web sites and the information they collect are becoming more and more important to retailers. That said. Other companies also create web sites and offer various services over the webs sites. These include organisations like telco, media, banking etc. All these industries offer services over the web.

Firstly we will present the dimensional type of information.

Business Area	**Description**
Time information	All the normal time information is included.
Codes	There is a very wide range of code tables created and it is easy to extend the codes tables. Each codes table is allocated a codes name and codes are stored with the code name typing the record. Examples of codes include but are not limited to: • Web basket close reason code. • Http status code. • Web site subscription stop reason code. • Subscriber loyalty code. • Wishlist reason code meaning a code to describe why the subscriber put the item into his/her wishlist.
Statuses	Each status table is allocated a status name and statuses are stored with the status name typing the record. Examples of statuses include but are not limited to: • Basket status. • Download item status. • Download status. • Session status. • Subscription status. • Subscriber status. • Wishlist item status. • Wishlist status.
Types	Each type table is allocated a type name and types are stored with the type name typing the record. Examples of types include but are not limited to: • Content type. • Device type. • Download item type. • Session type. • Subscription type. • Web transaction type.
Subscribers	Subscribers are parties. Many web sites are looking to create subscribers who stay subscribed to the site for a considerable period. Facebook and Yahoo are good examples of this. In these cases these sites often like to call customers subscribers. Other web sites are attempting to get customers to subscribe for a fee as well.
Subscriptions	On those sites where the user is described as a subscriber then the relationship is called a subscription. So IBI has created the subscriptions naming in the web models. Subscriptions and subscribers have tenure bands associated with them.
Loyalty and Programs to promote loyalty	There is extensive information recorded about loyalty and programs so as to be able to determine changes in loyalty of the customer.
Descriptions and definitions of web sites	Web sites consist of pages and content. There is an extensive model to document the pages and content on the web site so that it can be linked to what the user

	clicks on while on the site.
Browers and devices along with IP addresses etc	Web site owners like to know about the devices people are using to come to the site, the browsers they are using as well as where the users are coming from. This is particularly important for sites which are selling goods or services.
Entry, exit and referring pages.	To go with the click transactions there are view presenting the entry and exit pages as well as the referring pages which are usually outside the site.
Session durations	There is a submodel around session statuses, types, duration bands so that users sessions on the web site can be analysed to work on improving the user experience on the site and determining if that had a positive effect.
Baskets	Items on a web site are placed into baskets. The tables around basket allow for the capturing of everything that went into a basket or was taken out of a basket. It also shows what baskets were abandoned etc.
Download Items	A lot of web sites provide downloads to provide extra information for a user. These include such things as PDFs of detailed specifications of a product, to video demonstrations etc. In some cases downloads are chargeable in which case there will also be a record in the sales transaction table.
Wishlists	Most retail web sites have wishlists. These can be very informative in terms of future possible purchases. Amazon has a very good wishlist feature. The data model contains all normal things around wishlists.

Secondly let us document the areas that are covered by fact tables in the base model. Although this list seems rather small most of the data from retail that is rolled up is stored in the base model tables.

Business Area	Description
Page to content associations	This area of the model links individual pieces of content to the pages they occur on. This is needed as it is possible a piece of content appears on many pages.
Basket Transactions	Every item placed into a basket or removed from a basket is recorded. Abandoned baskets are also recorded. When a basket goes via a checkout the items in the closed basket are recorded in the closed basket items fact table.
Download Item Transactions	This table stores a transaction record for every item that is downloaded.
Order Sessions	This table stores all sessions that lead to orders being placed.
Page Hit Transactions	This table stores transactions for every page that is displayed.
Sales Transactions	It was decided to add a sales transaction table over and above the base sales transaction table.
Session Transactions	This table stores details of completed sessions even if they do not lead to orders.
Site hit transactions	This table stores details of each site hit. It is a summary of all the transactions on the site for one or more sessions.
Subscription Transactions	This table stores a transaction each time a new subscriber subscribes to the site.
Subscriber plans and measures	The client might wish to plan and measure numbers of subscribers and some key facts about those subscribers. These set of tables provides for this.
Subscriber to Wishlist associations and Wishlist Items	These tables record ever in and out transaction when a subscriber puts and item into the wishlist or removes it. In some cases items expire out of the wishlist and this too would be recorded.

5.3.4. BI4 Media Data Coverage

The media data models cover such things as magazine and newspaper publishing as well as paid media content subscriptions like cable TV or satellite TV. It does not yet contain portions for models such as modelling the shows and the ads presented on TV and this sort of thing. That is likely to come when we build models for our first cable/satellite TV provider.

The focus in the magazine and newspaper industry is circulation and ads. Magazine and newspapers get paid for Ads by the number of copies in audited circulation. So for Media companies it is critically important to closely manage the circulation numbers for both home delivery and outlet sales. This puts a great emphasis on the estimation of returns from the outlets and all Media companies work very hard to calculate the predicted number of returns issue by issue.

Firstly we will present the dimensional type of information.

Business Area	**Description**
Time information	There are some added elements of time such as number of weeks the subscription is active by putting in start and stop weeks. This is important for Media companies because the subscription can only be counted as active for full weeks and the media company will have some day nominated as the start of the week. It is not the number of days active divided by 7 and rounded down.
Codes	There is a very wide range of code tables created and it is easy to extend the codes tables. Each codes table is allocated a codes name and codes are stored with the code name typing the record. Examples of codes include but are not limited to: • Ad classification code. • Ad type code • Ad category code • Media drop point code to describe the nature of the drop point. • Outlet loyalty code to describe the loyalty of the outlet. • Subscription stop reason code to describe the reason for a stop. • Subscriber delivery completion code. • Subscriber loyalty code to describe the loyalty of the subscriber.
Statuses	Each status table is allocated a status name and statuses are stored with the status name typing the record. Examples of statuses include but are not limited to: • Subscription status. • Subscriber status.
Types	Each type table is allocated a type name and types are stored with the type name typing the record. Examples of types include but are not limited to: • Outlet types to describe the various outlets • Subscription type. • Media transaction type.
Subscribers	Subscribers are parties. However due to the fact that virtually all media companies call their customers subscribers the terminology of “subscriber” and “subscription” has been used in the media models. There is a lot of dimensional information stored about subscribers such as the tenure band, status, loyalty as well as all the base model information.
Subscriptions	Subscribers sign up contracts which are generally known as subscriptions. A subscriber may have one or more subscriptions. A subscriber is active is any of his/her subscriptions are active. There is a lot of dimensional information stored about subscriptions such as the terms, the status, the initial offer to get the subscription in the first place and more.
Outlets	Magazines and newspapers are sold via outlets as well as via home delivery. The

	outlet sales are generally anonymous meaning that it is not known who is the eventual buyer of the magazine or newspaper. Some demographic information can be implied by the location of the outlet and the likelihood of what the people “look like” in terms of demographics around that location. A good example of this are the small shops that sell newspapers and magazines on suburban train stations. The drawing area of the train station will indicate the demographics of the people buying the magazines and newspapers. There is a lot of dimensional information around outlets. For example the reporting structure into the media company, the performance band of the outlet, the loyalty of the outlet, the value indexes of the outlet, the start and stop dates for each publication the outlet takes from the media company.
Drop Points	In many cases the magazine or newspaper is not delivered to an outlet. It is delivered to a home or an office or a mailbox or similar. These also have the name “drop points”. The terminology came about because the article is “dropped” and is not handed over to any particular person who can vouch for the safe delivery of the newspaper or magazine. It is normal for some newspapers to not be delivered properly by the carrier who is as often as not a young lad on a pushbike. The “paper boy” was a job many of us had as a lad!
Agency	A great deal of the advertising placed into magazines, newspapers and TV are placed by agencies. Agencies aggregate up the demands of their clients and then bargain with the publishers of content for better deals. WPP is a good example of an agency that has massive bargaining power when it comes to getting the best prices for placing ads in the print and TV media.
Advertiser	An advertiser is also a party. However in the media business the term advertiser is universal. The people and companies that place ads are called advertisers and so this name have been used in the model.
Carrier	A carrier is also a party. A carrier is the party that transports the magazine or newspaper from its place of manufacture to its place of sale. There may be many carries involved in the transportation of one magazine or newspaper. The usual system is that the carrier is a third party that also carries other products to the outlet or home. In one project in Australia the carrier passed 7,000 outlets each weekend and some outlets were passed twice in a week. Obviously, because the carrier was already passing 7,000 outlets other companies wanting their product to be placed into some of these outlets could purchase that service far more cheaply from this carrier than to try and purchase from someone else. Distribution of products to outlets like this is a very big business. Obviously there needs to be other information like the rate structures the carriers are using etc. When the carrier uses a truck the truck information is also kept. This includes such things as an identifier, capacity for weight, volume and pallets so that the media company or distributor can make sure the truck gets fully loaded to minimise cost of distribution.
Circulation Zone	Newspapers in particular zone their coverage areas and many try to change the ads that they place for each zone. An example from the US is that no one buys a car from dealer more than 20 miles from their home. It does not happen. So the car dealers do not want to pay for ads in newspapers that are further away than 20 miles. But MANY newspapers have distributions that are well over 20 miles across. Virtually ALL big city newspapers in the US have distribution zones larger than that. So the newspapers zone the distribution so that the same physical place on the page can carry one ad in one zone and another ad in another zone. This allows the newspaper to charge less for the ad in the one zone than for the ad in all the zones. It allows the sale of the advertising space to be more closely tied to the locations of the advertisers.

Secondly let us document the areas that are covered by fact tables in the base model.

Business Area	Description
Advertising transactions, plans and measures	This set of tables maintains advertising transactions, plans and measures. The advertising transaction table is capable of going down to the individual level in the case of the home delivery customers. It is expected that in the future that each ad that is displayed on the TV will be recorded into data warehouses like BI4ALL.
Campaign measures and plans	There are specific Media campaign measures and plans to cater for the uniqueness of media companies.
Carrier transactions, measures and plans.	The carriers form a critical component particularly in the newspaper business. Even though newspapers are a shrinking business they are still very large businesses and the carriage of newspapers to homes, offices and outlets every day is still a very large business. The fact is that people like to read newspapers and getting the newspaper into their hands every morning is good business for companies that sell newspapers. It is not quite so important for the magazine business as most magazines have at least a weekly or monthly release cycle. It does not matter if the magazine is not available first thing in the morning. The customer can always buy it the next day if they really want it.
Delivery transactions, measures and plans.	Because delivery of newspapers and magazines is so important there is an extensive submodel around this area. It splits the delivery measures by outlet and home delivery as well as providing a combined view of these things. These are all available at publication level in case the carrier is carrying products from multiple publishers.
Outlet plans and measures	There is a submodel for the performance measurement of outlets for both plans and measures.
Subscriptions Plans and Measures	In Media companies it is very important to keep subscription numbers relatively smooth. So forecasting periods when the number of cancels or service suspends goes up is important so that the “gap” can be filled with new subscribers. The best example of this is the newspaper industry and “summer holidays”. When people leave their homes for a summer holiday they suspend the delivery of the newspaper. This can then no longer be counted for advertising revenues. So the newspaper is going to lose advertising revenue unless they can get someone else to buy that newspaper during those summer months.
Subscriber plans and measures	When the publisher sells more than one publication by subscription the subscriber plans and measures become more important. When there is only one publication subscribers and subscriptions are relatively interchangeable.

5.3.5. BI4 Telco Data Coverage

IBI and SKA consultants have worked in many Telcos over the years. Telcos were one of the early adopters of Data Warehousing and BI because they had such massive volumes of data and could make excellent use of it. They also had the money to be able to build large data warehouses. Telcos have operating margins that other companies can only dream about.

Skype might put a bit of a damper on this situation but Telcos are still very good BI clients. The major issue with Telcos is that there are actually so few of them. If one is to become a Telco BI consultant then you have to be ready to travel. It is as simple as that. Telcos are spread far and wide and to gain experience and knowledge the consultant has to go where the Telco is.

IBI had not initially intended to develop a Telco data model. We were focussed on the Retail space. However, in 2006 Sean Kelly of Sean Kelly and Associates (SKA) brought an opportunity to the table for a Telco data warehouse. This later turned into a very successful project for SKA and IBI collaborating together.

In response to this opportunity and more on the way it was decided that a Telco data model be created specifically for SKA. SKA and IBI staff collaborated on the development of this Telco data model. It was subsequently sold to two Telcos as an SKA product.

BI4 Telco is available from both SKA and IBI. Both companies would be involved in any Telco clients that choose to acquire the BI4 Telco product.

Because of the vast array of experience IBI/SKA consultants have had around Telco the Telco model is very broad and contains pretty much everything a Telco would want to have apart from IP TV. This is likely to be developed in the Media model as IP TV is more of a media domain item than a Telco domain item.

Firstly we will present the dimensional type of information.

Business Area	**Description**
Time information	All the normal time information is included.
Codes	There is a very wide range of code tables created and it is easy to extend the codes tables. Each codes table is allocated a codes name and codes are stored with the code name typing the record. Examples of codes include but are not limited to: • Bill charge code to explain what the charge is for. • Customer type code. • Payment code. • Billed code. • Calling circle code to sub type the calling circle. • Credit limit breach code. • Customer debt treatment code. • Package migration code to describe the movement from one package to another. • Package migration offer code to describe the offer that was made to try and get a package migration. • Rate plan code to describe rate plans. • Service code to describe a particular service. • Tariff charge code to describe tariff charges. • Tariff code to describe tariffs.
Statuses	Each status table is allocated a status name and statuses are stored with the status name typing the record. Examples of statuses include but are not limited to: • Bill cycle status. • Bill cycle summary status. • Calling circle status. • Credit limit breach status. • Customer debt treatment status. • Package migration offer status.

	• Package migration status. • Provisioning command status. • Rate plan status. • Service status. • Tariff charge status. • Tariff status.
Types	Each type table is allocated a type name and types are stored with the type name typing the record. Examples of types include but are not limited to: • CDR type. • Voip call type. • Bill cycle type. • TLI prefix type.
Subscribers	Subscribers are parties. Many web sites are looking to create subscribers who stay subscribed to the site for a considerable period. Facebook and Yahoo are good examples of this. In these cases these sites often like to call customers subscribers. Other web sites are attempting to get customers to subscribe for a fee as well. There is a model for residential subscribers as well as corporate subscribers.
Subscriptions	On those sites where the user is described as a subscriber then the relationship is called a subscription. So IBI has created the subscriptions naming in the web models. Subscriptions and subscribers have tenure bands associated with them. There is a model for residential subscribers as well as corporate subscribers.
Loyalty and Programs to promote loyalty	There is extensive information recorded about loyalty and programs so as to be able to determine changes in loyalty of the customer.
Bill Cycle and related information	These tables store information about Bill Cycles, the actual runs of the Bill Cycles and statistics gathered from the Bill Cycle and the type of the Bill Cycle. The schedule for Bill Cycles and other Billing Systems activities are also included.
Products and packages and rate plans	Telcos sell products which are bundled into packages that have various rate plans and tariffs applied to them. All this information from the billing system is replicated across to the data warehouse. If the Telco does IP calls this includes IP billing rates and profiles.
Call duration banding	This table allows easier analysis of calls by their duration bands.
Calling circle	Information about calling circles is included.
Provisioning commands	There are a set of provisioning commands that are issued against the network and this set of commands is stored.
Call Providers	As a call is carried nationally or internationally there may be various providers who are carrying portions of the call. If they can be identified they can be connected to the call. The call providers tables are an effort to identify the different providers that carry portions of the call.
Switches	Calls go between switches so all known switches can be recorded and they are referred to as the to and from switches for individual calls.
GSM Cells	Calls go between GSM cells for mobile to mobile. So all GSM cells that are known are recorded. For individual calls to and from GSM phones these cells are known as gsm cell from and gsm cell to.
Call completion descriptions	This table describes the various call completions that the Telco is able to detect.
Handsets	This table records the type and features of the handset where possible.
Inventory Items	Because some Telcos also sell a limited amount of hardware such as phones and accessories it is necessary to have a simple inventory item model. If the Telco would like a more complex inventory management system then pieces of the retail data model can be leveraged.

Secondly let us document the areas that are covered by fact tables in the base model. This seems very much longer than all the other lists. This is mostly because the Telco model has now become a very mature model.

It should be noted that in the Telco model IBI does not provide standardised tables for switch records and pre mediation data. The reason for this is that we recommend that all data be taken from the switch in a non lossy process and the appropriate keys be added to the front of the record. Since the various switch vendors have different record formats and these formats change over time it is just as easy to build these tables at the time the data model is deployed rather than creating more tables in the data model as a product.

It is also noted that IBI has C++ code that can decode switch records. This was developed at a client because the vendor of the mediation software wanted a rather large licence fee to run their switch decoding software on another machine. The C++ code receives files into a folder and processes the files based on the file names which are discovered and then sends the decoded file forward. It is quite a nice piece of code.

Business Area	Description
CDRs	Most vendors still call these CDRs though some are talking about Event Data records now. A CDR for the purposes of the BI4 Telco model is usually a single record that describes a completed call. Most mediation systems remove uncompleted or non billable calls. CDRs are broken up between in and out CDRs. Many Telcos are now recording what they can from in CDRs to identify the callers coming in to their network. As well as in/out the following CDR tables exist. • GSM SMS • GSM MMS • GSM Voice • Data, landline or GSM • Landline Voice • VOIP • IP CDRS There is then also an in ALL and an out ALL table that combines some of the fields from each of the above so that a single table cane be used to analyse the common fields. Because of this breakout there are 14 CDR tables in the standard model. If the client would like to forecast expected volumes for each of these 14 tables there are also plan tables for each of these 14 tables making up 28 tables in all to perform plan vs actual analysis. Experience shows not many Telcos actually plan out their loads like this. There would be more tables created if the switch records are also put into the data warehouse. Namely two tables for each switch record type, one for in and one for out.
Bill lines, charges, transactions, measures and plans.	The CDRs may be rated or unrated. The BI4 Telco model suggests that the rating process set a flag on the record. It is fully understood that performing updates on so many records might stress the data warehouse machine beyond its capability due to the logging of such updates. The data warehouse architect is the person who needs to determine how the pre and post rated records should be represented in the data warehouse. Once rated the CDRs await the Bill Cycle to be collected up and placed onto an invoice. The header charges that do not become line items on the bill are placed into the bill charges transactions. The individual bill lines are placed into the bill line transactions. In these cases things like VAT or package discounts may be separated from the bill lines if that is what is desired.
Bill Payments and Payment allocations and Payment errors for transactions, plans and	Hopefully the customers pay their phone bills but some do not pay promptly. Therefore there is a need to record bill payments and when a payment might be distributed across a number of bills there is a payment allocations model that splits the payment across those bills that it covers.

measures.	
Disputes and Account adjustment transactions, measures and plans	Inevitably in a Telco there are errors in billing and disputes in billing. These disputes are recorded in transactions as well as plans and measures. Should the dispute lead to the need for an account adjustment then the adjustment also has transactions, plans and measures. The volume of disputes and account adjustments are a good indication of billing quality within a Telco. The more disputes and adjustments the more problematic the billing quality is.
Aged Debt, Bad Debt, measures and plans.	Of course if some customers do not pay their phone bills for some time then the debt becomes classified as bad debt and this too must be carefully managed. The service has been provided and all costs have been incurred so bills not paid are pretty much lost profit. There is also the issue that in many countries the land line is also the only sure link to emergency services and the line must stay provisioned and working even if only for emergency calls. So the whole area of aged debt and bad debt needs to be closely managed in a Telco.
Debt treatment transactions, measures and plans.	When a customer is in debt there are numerous treatment methods that are applied by Telcos. These tables store the transactions, measures and plans around the debt treatment actions.
Credit Breach transactions, measures and plans	Closely related to aged debt and bad debt are credit limit breaches which are highly correlated to the debt become aged and then bad. When a credit limit has been breached the Telcos usually take some action to see if the customer can bring the account back in under its credit limit.
Deposit transactions, measures and plans.	In some countries the Telcos require a deposit prior to connecting a post billed land line. This is because the expectation is that too many customers will not pay their bills if no deposit is asked for. The deposit is used to guarantee equipment and payment. If the bills are not paid or the equipment is damaged then the deposit can be used to compensate the company for the equipment damage or unpaid bills.
Fraud transactions, measures and plans.	Telcos are often the targets of fraud from a wide array of fraudsters. These tables provide the ability to record transactions as suspected fraud transactions as well as to roll them up for measures. The plan table is used to predict the level of fraud expected to be discovered.
Loyalty programs transactions, measures and plans.	Many Telcos participate in loyalty plans with other companies of have their own loyalty programs. This set of tables enables the Telco to capture transactions representing loyalty programs and roll them up to measures and compare them against plans.
Package Migration transactions, measures and plans.	Telcos are always wanting to upgrade customers to higher margin packages and customers are often wanting to downgrade packages. This whole area of package migration up and down or between packages is closely monitored by Telcos. These tables allow all these migrations to be captured at a transaction level and then measures to be compared against plans.
Customer Save Transactions, measures and plans.	When a customer is predicted to churn or has actually given notice he/she wants to churn many Telcos respond with a “customer save” effort. These efforts are recorded as transactions, measures and plans. This is an area of intense study in most Telcos.
Provisioning, Service Orders transactions, measures and plans.	In the landline area when a new customer comes to the Telco there is usually some form of provisioning process. In the case of new houses there might be new phone lines to run to the house or new activation at a local exchange. For customers just changing providers there is co-ordination with the other provider to perform. These tables store information about service orders that are carried out and the provisioning commands that are issued to the network to provision services for customers.
Measures and plans for Subscribers Subscriptions The organisation	There is a set of tables that provide detailed measures and plans across the portfolio of subscribers, subscriptions, the organisation, sales, voicemail etc. Many are stored both by month and by bill cycle. These measures are used to create high level reports as to the overall health of the Telco.
Inventory measures	For those Telcos who sell a modest amount of hardware there is a modest inventory measures table. If the Telco performs retail in a big way then the retail model should be leveraged.

Housekeeping. File transfers and processing	In Telcos there are thousands of files that are moving around the network. The switch records come into the network to be mediated and then rated. Every Telco could do with a decent set of tables that record the movement of files around the company, especially CDR files. So BI4 Telco contains specific file movement and processing tables to be able to report and analyse this workload. This is not an area where the business users are interested. Unless files are not being processing on time. Then they will be interested.

6. THE PROCESS OF BI4ALL BASED DATA MODELLING

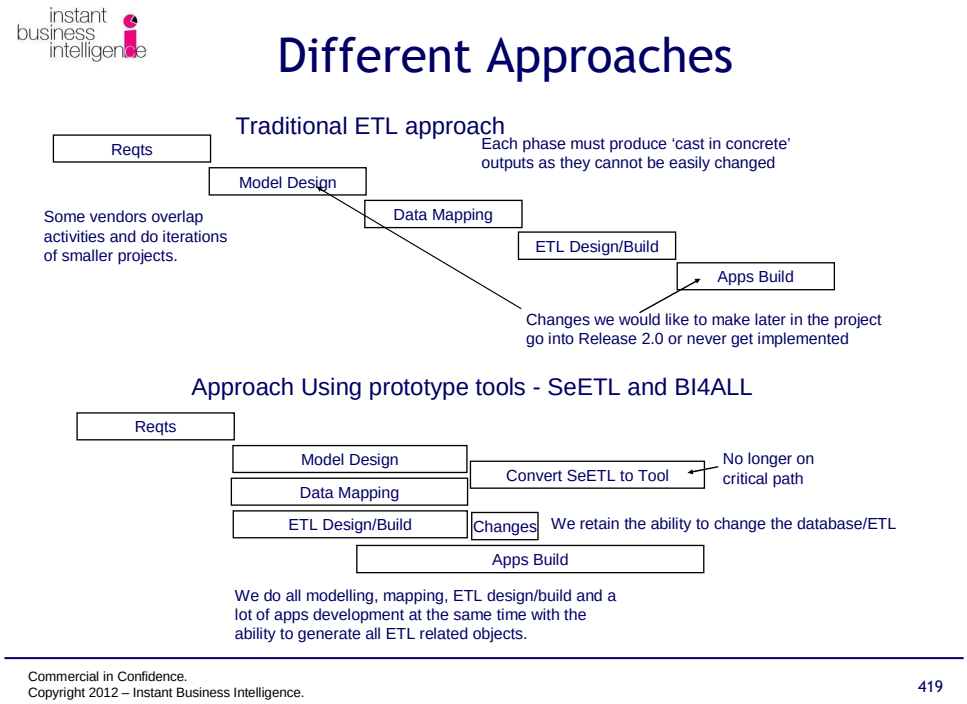
The BI4ALL Data Models are unique with the combination of the type 2 associations with integer keys attached to them and SeETL to rapidly populate the prototype of the data models. No other company can do this. We are well aware that no other company can do this which is why we offer our tools to selected resellers who wish to co-operate with IBI rather than to compete with us.

Another very important point that derives from these new tools is that IBI can build the data warehouse in about half the people cost of any other vendor. If our data models are used across both the ODS and Data Warehouse then the cost of the hardware and licenses is also decreased quite markedly as well.

One of the drivers of the dramatically reduced costs is our ability to build prototype data warehouses and to populate the tables with data as they the mappings are being documented and the tables are being designed.

The 'secret sauce' as it were is the fact that the ***SeETL DesignTime*** workbook acts as both the documentation tool for the mappings as well as the ***source code*** for the generation of the ETL subsystem. There is no intermediate 'translation' step from documentation to executables for the ETL subsystem. This, again, is unique. Other vendors are attempting to replicate this functionality but the code that is then generated is their own mapping code.

Informatica has tried to create just such a tool. It does generate Informatica mappings. But they are then executed as mappings inside informatica. In SeETL the ETL is then executed either as SQL or via the C++ engine. The obvious difference is the cost of tools like Informatica vs tools like ***SeETL.***



The process that nearly everyone else is using is the one at the top of the diagram. This is the old waterfall approach. Sure, they might build some prototypes with hand built ETL to populate the prototype just like we used to do before we had SeETL. But the vendor will NOT be able to build a full sized prototype to be studied for functionality and performance and then be able to adjust that prototype at will.

You are welcome to ask your prospective, or even incumbent vendor, whether they can do this. The answer will be no, thought it might come with a long explanation.

Today, with SeETL we can collapse the tasks of Model Design, Data Mapping, ETL Design/Build because these three activities all take place in one tool. The SeETL DesignTime Workbook and the vb.net application that reads it and processes it.

Further. The generated code can actually then directly populate the prototype data warehouse to the volume that the client provides hardware to support the development process. If a machine that is 'full size' can be deployed during development then the prototype can be a full sized prototype.

Our clients find it quite amazing that we can do this when they see it on our projects. Indeed they often make the comment that they think that our ETL tools are 'undersold'. We note to them that most people can not believe we can do what we say we can do, even when we show them how we do it. That is the nature of the ETL tool we have. So advanced no-one can believe it.

Because the ***SeETL*** ETL can be executed directly on translation out of the workbook the cycle of change, generate, execute, test is a very highly cyclical activity and portions of the data warehouse come online for testing and verification very early in the process.

It is important to note that this is true even when the client wishes to use some other ETL tool in production such as Informatica or DataStage. In the 'new development paradigm' the production ETL developed in some other tool is not developed until the end of the project.

It is also important to remember that other components like Business Objects Universes can be developed much earlier than would be normal because the data model is stabilised far sooner than normal. This again compresses the development time lines.

The process of being able to make changes to the data models, the mappings and generated ETL in very fast cycles also allows those changes that are desired due to learning new things about the data to be incorporated into the initial delivery project and not delayed to later projects.

The process of data modelling when using BI4ALL actually changes quite a bit as well.

- Associations dramatically change the process of modelling
- Makes the modellers life *MUCH* easier
- Eliminates some significant problems by design
- Dramatically enhances productivity
- Dramatically reduces development times

The process is now as follows. This is taken directly from the education materials provided.

For dimension tables:

- You identify the dimensional data
- Define and design dimension tables
  - Do not worry about what fact tables it will be linked to
  - Attempt to define unique discrete values only
  - If there are large combinations of values break the input data up into more smaller dimension tables to keep the number of rows in the dimension tables low
- Decide which dimension tables need to be type 2
  - There are usually very few, and they usually have very few rows in them, product, geography are two examples
- Determine 'real key' in the data
- Map to a BI4ALL table or define a new table

For fact tables:

- You identify the transaction/key measures data
- Define and design fact tables
  - Initially, only worry about linking dimensions that are unique to the fact row and dimensions obviously required for performance
  - Do not worry about linking 'low use' dimensions
  - Always put relevant association keys onto fact tables (party/account)
  - Assume that 'if it should be linked' it will be linked
  - Add extra keys for performance as the reports are designed and as analysis requirements become more clear
- Map to a BI4ALL table or define a new table

For historical data:

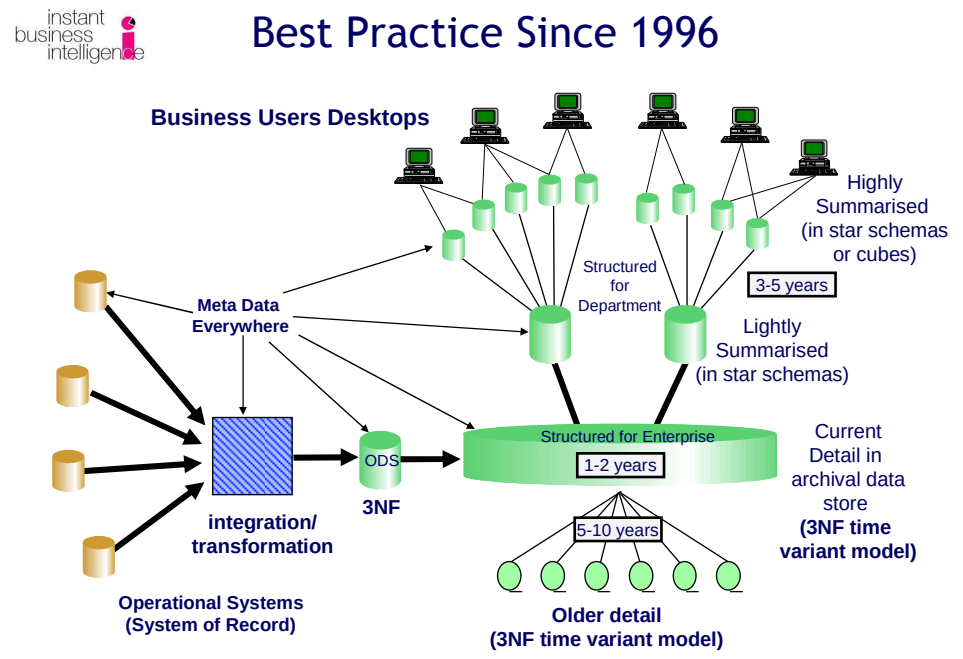
- Identify areas where you want to retain history
  - Entities/accounts already defined
  - You might add
    - End of month outstanding amounts
    - Daily outstanding balances
- Decide how you are going to retain the history
  - Type 2 dimension table?
  - Develop a new association?
  - Develop an event fact table with 'current/previous' amounts
    - Eg. Banking transactions with before/after withdrawal balance

The overall result is:

- BI4ALL Modelling is far faster and easier than other current methods of designing models
- It is also produces a faster/cheaper EDW
- No loss of functionality

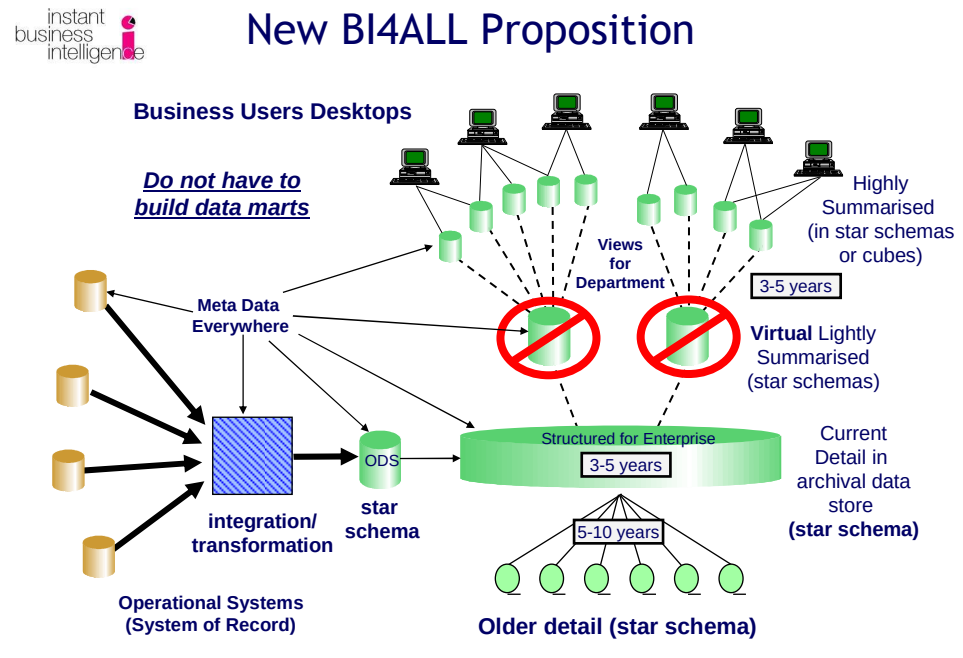
6.1. Summarising Because it is Important

In 1996 this author and another colleague, this authors mentor, invented a new way (for us that we had not see before) of designing data warehouses combining the best ideas of Bill Inmon and Ralph Kimball. The result can be presented as a diagram as follows:



BI4ALL Data Models replace the three different modelling techniques used in these types of data warehouses with one single modelling technique that is always dimensional. What has been done is a natural progression of solving all the issues needed to be solved inside the dimensional modelling paradigm.

The result then looks like this.



The benefits to the client of using these kinds of data modelling techniques, be they BI4ALL Data Models or be they just the data modelling techniques, are as follows:

- Increased query performance
  - Attributes 'broken out' to type 1 dims, directly joined to fact tables
  - All joins through integer keys
  - Almost no type 2 dimensions
  - Same model can support ODS/DW query patterns

- You do not *have* to build data marts
  - Can answer all your queries directly from the ODS/DW
  - Approx 40% reduction in ETL Coding/Maintenance
  - HW/SW cost avoidance for data marts

- All data readily available to end user tools
  - Oracle Discoverer, Business Objects etc.
  - All star schema
    - Even the archival data store

These are very significant benefits and they are hard to ignore for any mid-sized to large corporation.

7. THE DATA ACCESS LAYER

In BI4ALL we are proposing a major innovation that may be used in a complementary fashion to data model abstraction tools like the Business Objects Universe and Cognos Catalog.

In the past virtually all tools have maintained a metadata layer that described the database to the tools. The first one that this author saw that was quite sophisticated was by Metaphor Computer Systems and was called “The Workstation Tools Data Dictionary”. The WTDD.

The WTDD maintained a copy of the data model, tables, columns, joins etc so that the Metaphor Tools could interact with databases of all types by only looking at the one data dictionary. The idea that Business Objects invented the abstraction layer is laughable. IBMs IMS had “Data Base Definitions” in the early 70s and this was an abstraction mechanism. This author wrote an sql like interpreter for IMS databases in 1989 that used a metadata layer to superimpose a relational style data model over the top of IMS databases. It allowed such things a select statements with where clauses to be executed directly against IMS databases. It was built as a prototype.

So the idea of some form of abstraction layer between tools, be they programming languages like Cobol or PLI or graphical tools like Metaphors DIS or Business Objects Reporting Tools, has been around for a long time.

But something interesting is happening as we build larger and more complicated data warehouses.

In 2002 IBI Consultants invented a new idea that allowed for data models to be made more complex without the need to retain mental pictures of the data model in ones mind. This invention spurred on the creation of significantly more complex data models that also had multiple paths between tables.

In subsequent projects the increased complexity caused more and more difficulties in presenting the data model to Business Objects Universes.

In 2005, as BI4ALL was being designed and the question about disconnecting the reports was discussed IBI Consultants investigated the soon to be released Reporting Services from Microsoft. We found Reporting Services to be a reasonably useful tool for base level reporting. More importantly, we were intrigued at the idea that Reporting Services did not have a data abstraction layer. Reporting Services allowed the connection of Stored Procedures directly to the report.

IBI tested Reporting Services extensively in this time and reviewed the possibility of using Stored Procedures as the Data Access Layer between reports and the data model.

The results of this testing left us with the opinion that in the future there would be a role for the creation of Stored Procedures as the “Data Access Layer”. We also found that on SQL Server stored procedures often performed considerably better than the same SQL passed to the database as SQL. The evidence indicated that the SQL Server optimiser was better able to optimise the stored procedure than it was able to optimise the query arriving from an external tool.

Based on all this work IBI Consultants set about developing a suite of templated Stored Procedures that would sit over the top of the data model and be able to turn the database into a “Question and Answer Machine”.

7.1. Advantages of the Data Access Layer

There are a number of advantages to using Stored Procedures in the Data Access Layer.

1. Written by people who understand the database better. Less prone to errors. Better performance.

   In most cases SQL written into SPs will be written by people who understand the data model better than the people who write reports in tools like Business Objects. The hand written SQL is generally better, faster, more efficient when written by hand. Naturally there is an increased cost in being written by hand.

2. Able to take advantage of the database and procedural language.

   Most query tools generate close to ANSI standard SQL. They have to because the SQL has to be transportable across databases. By using SPs it is possible to take advantage of database unique features as well as get access to the procedural language exposed by the database. Procedural languages are generally not available to the query tools because they vary across databases.

3. Able to completely disconnect the reports from the data model.

   By having reports source data from stored procedures and not via a data dictionary it becomes possible to completely disconnect the report from the data model. It does not matter how the SQL gets the answer. The report passes a parameter and accepts a data stream back from the SP. This means that when reports are customised the effort to customise them is vastly reduced over using an abstraction layer.

   Further. Because there is no abstraction layer like a Business Objects Universe the abstraction layer does not need to be maintained. Experience is showing us that maintenance of the data abstraction layer is growing larger and larger now that the data models are getting more complex.

4. Ability to invoke the same SP from many tools.

   Virtually ALL large companies have more than one reporting tool. Many companies also use Excel as a query tool to get at data or to chart data. By putting queries into SPs all the different query tools use the SAME SQL to get the same answer. There is no chance that errors in the construction of the abstraction layer will cause differences in the reports being produced.

5. Able to publish all SPs, input parameters and output fields.

   Once an SP is written SeETL provides the ability to document the SP and to publish all the relevant information about the SP. This information is also published via Report Services reports. This provides the ability to have a searchable set of documentation about all the SPs that have been implemented.

   Over time the library of the SPs will grow to provide the vast majority of the functionality needed. When someone wants a new report there will be a similar SP already available.

6. Data Security can be added to the SPs easily.

   All BI4ALL Stored Procedures accept a parameter of the userid of the person making the call to the Stored Procedure. It is therefore possible that all invocations of an SP can accept the userid and the security for reading data can be added directly into the SP thereby removing the need to put the security layer into the reporting tools.

7. Logging can be added to the SPs easily.

   If desired the ability to put logging statements into SP is vastly higher than the ability to put logging statements into query tools. Though, to be fair, the databases are now providing more and more logging and tracing facilities to be able to better tune the SQL that is being executed against the database.

8. Data Joining can be done inside the database at run time.

   One of the BIG issues with reporting is the plan vs actual report. This is one of the most common reports. The issue is that the plan data is stored at a higher level than the actual data. Therefore to join the two lots of data together they must be summarised to the same level of granularity and presented to the reporting tool.

   There have been three major ways of doing this.

   a. Join the data in ETL and store the plan next to the actual in a fact table.

   b. Join the data in the ETL and populate some form of cube product and place the plan vs actual data in the cube next to each other.

   c. Store the plan and the actual in different tables and perform the join at run time using the query tool to perform the join of the two data streams.

There is now a new way to do this. Inside the SP issue the select against the plan and sort sum, issue the select against the actual and sort sum, then perform a full outer join between the two data streams taking care to make sure that nulls are properly handled and presented back to the query tool a something it can understand.

By performing this join at run time in the SP the is no need for extra ETL to join these two data streams and there is no need for new tables or cubes to store the data output by this no longer needed ETL.

In short? Being able to perform joins between plan vs actual at run time in an SP makes the data modelling simpler and the ETL simpler. It reduces the number of tables as well.

This is a very big advantage of pushing the joins between plan vs actual into the database by using SPs. An example of this is provided in Appendix 1.

7.2. Disadvantages of Stored Procedures

Naturally there are disadvantages of Stored Procedures as well. If there were no disadvantages we would be using them more extensively.

1. SPs are database specific.

   The database vendors implemented different languages for SP and there is little commonality across them. For anyone building a product like BI4ALL this presents a major problem. How to develop SPs in the language for each database and how to maintain that over time?

   Mitigation: SeETL provides the ability to write one single text document that has the source code for all flavours of SPs that are to be supported. Tags in the source code indicate which pieces are common to all flavours of SPs and which portions are unique to a specific SP language.

   Other consulting or software houses that wish to have SPs deployed onto many database can use this tool to do the same.

2. SPs are code and therefore can not be scanned for data linearage.

   Data Linearage from cell on a report to the source data is something of a holy grail in the BI world. No one has really done that properly yet. SPs break the connection between the report and the field in the data model and therefore back to the source data.

   Mitigation: SeETL provides the ability to document the source for fields on reports. Simply put the link between the field on the report and the source of that field in the data model can be documented into the SeETL data dictionary. This is not far away from an “abstraction layer” and we all know how hard it is to get “optional extra” documentation maintained properly. Never the less the option is there.

3. SP are written by hand and not generated.

   The fact that SPs are written by hand is both an advantage and a disadvantage.

   The disadvantage is that it does take longer to write a stored procedure than to write a simple report from a data abstraction layer. However, by hand writing far more care can be taken with the development of the SQL.

4. SPs mean that there is a “wait time” for a new report to be written.

   One of the most repeated goals of BI is “the end user wants to write his/her own reports and does not want to be dependent on the IT staff”. As far as this author is aware there is only one product that has ever lived up to the billing of being able to do that. www.meta5.com. The major benefit of the Meta5 system was that the business users really could build their own applications and reporting packages.

   Tools like Microstrategy, Cognos, Business Objects very much require that there is some level of IT support to get a new report written.

   Given that the fact of the matter is that there still is IT involvement in creating new reports making this an explicit requirement via using SPs, though a disadvantage, is not as bad as it sounds.

7.3. Features of the Data Access Layer

This section documents specific features of the Stored Procedures developed for the Data Access Layer.

7.3.1. Parameters for Reports Sourced Via SPs

The first use of SPs for Report Services Reports is to use the SPs to retrieve the drop down lists that are used as filters on reports in the first place. Therefore, in the Data Access Layer there are many SPs that retrieve the selection list for a filter.

Further, Report Services allows a parameter to be dependent on a higher level parameter. Therefore the output of one SP may be put into a drop down list, one value selected on, and then that value passed to the next drop down list.

Here is an example of this.

```
create procedure dbo.SP0004_Select_Year_Name
 @UserID               varchar(255)
AS
select distinct vm_day.year_name from dbo.vm_day vm_day
where vm_day.pk_vm_day > 8036 and vm_day.pk_vm_day  < 9497
union
select '*'
order by 1
```

Notice how the Userid passed to the SP. Notice how there is a select distinct on the year_name from the vm_day table between two specific day keys. These are the years 2012-2016.

Notice the union select *. This is adding the * as the wild card to select all.

```
create procedure dbo.SP0004_Select_Month_Name
@year_name             varchar(255)
,@UserID               varchar(255)
 AS
select distinct vm_day.month_in_year ,  vm_day.month_name from dbo.vm_day vm_day
where 1=1
and     ((@year_name = vm_day.year_name) or  (@year_name = '*'))
and      (vm_day.pk_vm_day > 0)
UNION
Select 0 as month_in_year, '*' as month_name
order by vm_day.month_in_year
```

Now notice how the year_name is passed into this select month name stored procedure so that only those months in that year are selected. Now, since all years have the same months this is not an example where a different result would occur depending on the first parameter but you get the idea.

Also note how the month_in_year has to be returned and the output is sorted on this field so that the month names come up in the correct order and not in alphabetical order.

7.3.2. Accepting Strings as Parameters

A typical SP in BI4ALL has an opening declaration as follows:

```
create procedure dbo.SP_CPA0100A_001
 @campaign_group_name_short     varchar(255)
,@campaign_name_short           varchar(255)
,@sub_campaign_name_short       varchar(255)
,@campaign_priority             varchar(255)
,@campaign_type                 varchar(255)
,@contact_method_sdesc          varchar(255)
,@iso_currency_code             varchar(255)
,@UserID                        varchar(255)
AS
```

Frome this fragment it is clear that the procedure is accepting a set of strings as the parameters to be used in the query for the where clauses.

7.3.3. The “*” Character Is the Wild Card

The implementation of the wild card is done as follows.

```
and   (a.campaign_group_name_short      = @campaign_group_name_short     or @campaign_group_name_short = '*')
and   (a.campaign_name_short            = @campaign_name_short           or @campaign_name_short = '*')
and   (a.campaign_priority              = @campaign_priority             or @campaign_priority = '*')
and   (a.campaign_type                  = @campaign_type                 or @campaign_type = '*')
and   (a.sub_campaign_name_short        = @sub_campaign_name_short       or @sub_campaign_name_short = '*')

and   (a.contact_method_sdesc           = @contact_method_sdesc          or @contact_method_sdesc = '*')

and   (a.iso_currency_code              = @iso_currency_code             or  @iso_currency_code = '*')
and      a.pk_vm_sub_campaign           > 0
```

As can be seen the parameters are tested for being equal to the value on the database or being “*”. This does mean that only one value can be selected from the drop down lis and passed to the report as a parameter. There is no capability yet in BI4ALL reports to select multiple values in the drop down list.

In most databases if the parameter is “*” the optimiser will notice the or clause and remove the test from the query prior to sending it into the database to be evaluated so there is little overhead in this type of wild card processing.

7.3.4. Redundant Tables and Redundant Joins

There are many template SPs that are provided as starting points. It was intended that all fact tables released would have at least one template SP that joined all the dimension tables that were keyed onto the fact table. This has become excessive to develop and maintain and will only be delivered on demand for the portions of the models.

For those tables with template SPs there are redundant joins. Joins that are implemented in the SQL but no data from the joined tables is presented in the query. The joins are there to make it easier when data is needed from those tables. This appears in code as follows.

```
from

         dbo.vf_delivery_txn            a
        ,dbo.vm_delivery_date           vm_delivery_date
        ,dbo.vm_all_cust                vm_all_cust
        ,dbo.vm_all_demographic         vm_all_demographic
        ,dbo.vm_geography               vm_geography
        ,dbo.vm_currency                vm_currency
        ,dbo.vm_comp_reporting_struct   vm_comp_reporting_struct

/* #0001 redundant joins start tag */

        ,dbo.vm_delivery_minute         c
        ,dbo.vm_sub_campaign            d
        ,dbo.vm_product                 e
        ,dbo.vf_party_dims_asoc         g
        ,dbo.vm_account                 j
        ,dbo.vf_account_dims_asoc       k
        ,dbo.vm_geo_code                l
        ,dbo.vm_map_reference           m
        ,dbo.vm_delivery_instr_std      n
        ,dbo.vm_delivery_method         o
        ,dbo.vm_sales_rep               q
        ,dbo.vm_unit_of_measure         r
        ,dbo.vm_txn_type                s

/* #0002 redundant joins end tag */
```

In this case there is no data returned in the query from the tables below the comment.

The joins in the query are expressed like this.

```
/* #0001 redundant joins start tag */

and     a.dk_vm_delivery_minute         = c.pk_vm_delivery_minute
and     a.dk_vm_sub_campaign            = d.pk_vm_sub_campaign
and     a.dk_vm_product                 = e.pk_vm_product
and     a.dk_vf_customer_dims_asoc      = g.pk_vf_party_dims_asoc
and     a.dk_vm_account                 = j.pk_vm_account
and     a.dk_vf_account_dims_asoc       = k.pk_vf_account_dims_asoc
and     a.dk_vm_geo_code                = l.pk_vm_geo_code
and     a.dk_vm_map_reference           = m.pk_vm_map_reference
and     a.dk_vm_delivery_instr_std      = n.pk_vm_delivery_instr_std
and     a.dk_vm_delivery_method         = o.pk_vm_delivery_method
and     a.dk_vm_sales_rep               = q.pk_vm_sales_rep
and     a.dk_vm_unit_of_measure         = r.pk_vm_unit_of_measure
and     a.dk_vm_txn_type                = s.pk_vm_txn_type

/* #0002 redundant joins end tag */
```

The purpose of the comments is to be able to turn off and on the joins with an edit command in the file.

7.3.5. SP Name linked to the Report Name

All reports have a standardised name and an associated number. This is so that reports can be referred to by this shorthand. All SPs that are tied directly to a specific report are numbered like this. For example.

```
drop procedure dbo.SP_ASAA0101_001 ;
drop procedure dbo.SP_ASAA0101_002 ;
drop procedure dbo.SP_ASAA0101_003 ;
drop procedure dbo.SP_ASAA0101_004 ;
drop procedure dbo.SP_ASAA0101_005 ;
drop procedure dbo.SP_ASAA0101_006 ;
```

These six stored procedures are tied to the report. ASAA0101.

7.3.6. Userid is Always Passed to the SP

As noted in the first example stored procedure the Userid is always passed. There has been no security coded into the stored procedures so far. However, when this is requested it will be added.

```
create procedure dbo.SP0004_Select_Year_Name
 @UserID               varchar(255)
AS
select distinct vm_day.year_name from dbo.vm_day vm_day
where vm_day.pk_vm_day > 8036 and vm_day.pk_vm_day  < 9497
union
select '*'
order by 1
```

7.3.7. Partitioning Strategy for SQL Server 2005 EE

One of the big issues on SQL Server 2005, which was the first database that BI4ALL was implemented on, was that there was only one field available for partitioning. However this was simply not enough. We needed at least two fields. We needed partitions based on time and one based on the company reporting structure. The first major customer we were implementing BI4ALL for had clients who had customers over many European countries. The query load was typically by time by country. So creating partitions by time by country was the natural partitioning of the data.

This was also needed as SQL Server 2005 EE did not do a very good job of optimising queries. It tended to scan the whole partition from which it was possible to find data rather than scan the indexes and then find the records that matched as Oracle would do via bit mapped indexes. It came as no surprise to IBI that Microsoft bought Datalegro and decided to release a parallel database. SQL Server 2005 EE was clearly not up to the task of large data warehouses no matter how many times Bill Gates said so.

In order to implement two field partitioning when only one field was available we created a partitioning column on each fact table that was the company reporting structure portion of the partitioning column and the day portion of the partitioning column. It is expressed like this in queries.

```
/* #003 SQL Server 2005 Partitioning Strategy Joins start tag */

and    vm_solicitation_date.dummy_join_col          = vm_comp_reporting_struct.dummy_join_col
and    a.partitioning_column                        = vm_comp_reporting_struct.partitioning_column_part_01 +
vm_solicitation_date.pk_vm_solicitation_date

/* #004 SQL Server 2005 Partitioning Strategy Joins end tag */
```

The dummy join column was needed to force the optimiser to work properly. The two part partitioning column will allow the database to scan ONLY those partitions that can contain data that might satisfy the constraints of the query. In MOST queries the query was for only one country for a period of time.

The partitions we implemented were by country by month and each partition only had a few million rows in it. Even so we saw that SQL Server 2005 EE always scanned the full partition and then joined the dimension tables and then filtered out the rows that did not meet the constraints rather than in the reverse order.

This is not a very efficient plan. We could find no way to influence the optimiser to do anything different and we could find no one else who knew how to do this better either.

For more information on Stored Procedures please see Appendix 1.

8. BI4ALL STANDARD REPORTS

BI4ALL comes with a set of 70+ standard reports. The documentation for the reports can be downloaded from the following links.

05 – Example BI4ALL Reports
http://www.instantbi.com/LinkClick.aspx?link=BI4ALL+Example+Reports.zip&tabid=59&mid=499

06 – BI4ALL Report Descriptions
http://www.instantbi.com/LinkClick.aspx?link=BI4ALL+Report+Descriptions.pdf&tabid=59&mid=499

In this section we will explain some of the features of the reports rather than the content of the reports. The content of the reports is self evident by looking at the PDF of the report.

In most cases prospective clients are keen and interested in the reports but they have a commitment to other reporting tools. When IBI made the commitment to build Report Services Reports we did so knowing that most companies would already have something else. Report Services was used because it gave very good examples of how to implement reports, it was free to all SQL Server Clients thereby reducing the barrier to entry for Business Intelligence for customers, and it was about to be pushed very hard by Microsoft.

We had expected Microsoft to get greater penetration with Report Services over the last 6 years.

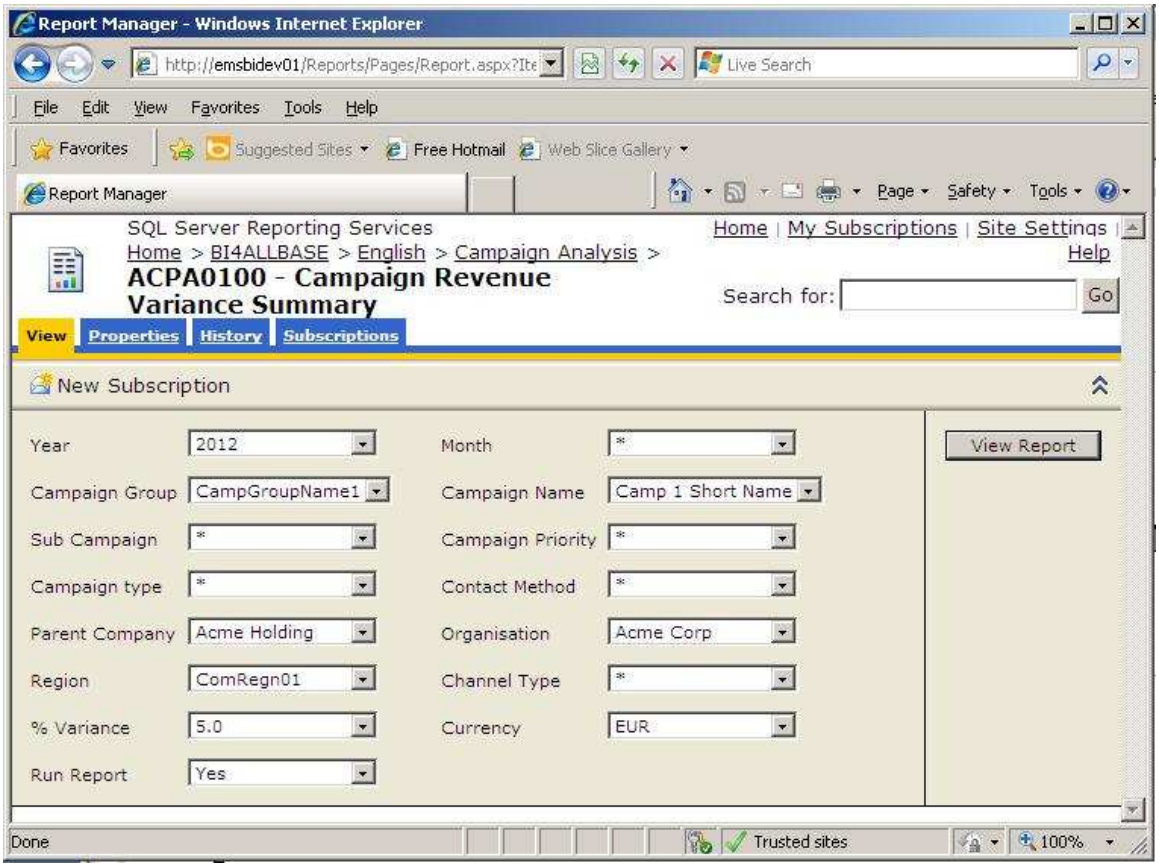
Report Services continues to be a very viable product for the deployment of reports of a relatively static nature where many parameters are set.

8.1. Connected Parameters

When designing Report Service Reports we wanted to emulate the ability to drill down and filter on parameters that other products provide as much as is possible. We feel we came as close as the product allows us. Each parameter is linked to a stored procedure behind it. Where there is a hierarchy in the parameters it is possible to pass the selection of one parameter down to the next parameter in the hierarchy. Then the drop down box for that next parameter will be automatically updated with the filtered list.

We think this is a very good idea as it means that there is no “other set of data” that needs to be refreshed and maintained. The drop down lists are populated directly from the data warehouse. Those who have used Business Objects List of Values over the years can speak to the frustrations and problems of consistently keeping them up to date.

Here is an example of Parameters being selected.



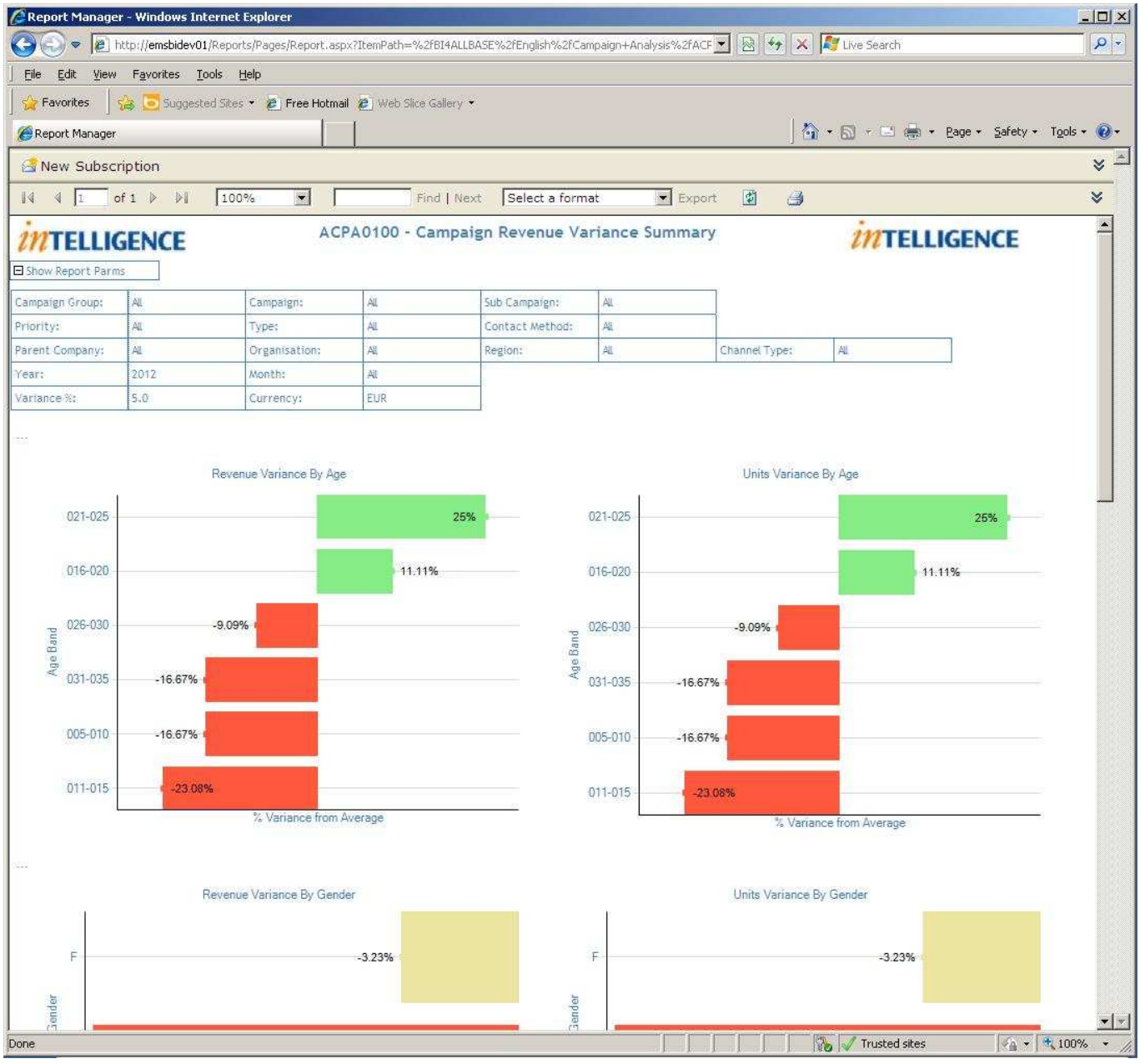
You can not see the hierarchy of the values in the drop down list in the diagram. We will put a video demonstration onto the web site to show that. However, you can clearly see how the filters are populated and set.

Currently there reports only support the selection of one value from the drop down list which can also be the “*” character as the wild card.

8.2. Dashboard Level Reports

BI4ALL Reports have been designed as best as is possible to present a higher level dashboard about some area of interest and to provide drilldown through the dashboard. The Campaign Management and Sales Reporting reports do this.

The drill through is achieved by placing three dots "…" near the item that can be drilled through and when the mouse floats over the item to have a popup to say "click here to drill down".



In the above report it can be clearly seen that there are three small dots to the top left of each of the two charts on the left.

During development IBI did contact Stephen Few at Perceptual Edge and IBI did review Stephens work. As much as IBI agree with Stephen the capabilities of Report Services were not up to the level that would be desirable. The decision made was to develop the best reports that could be developed given the Report Services 2005 product.

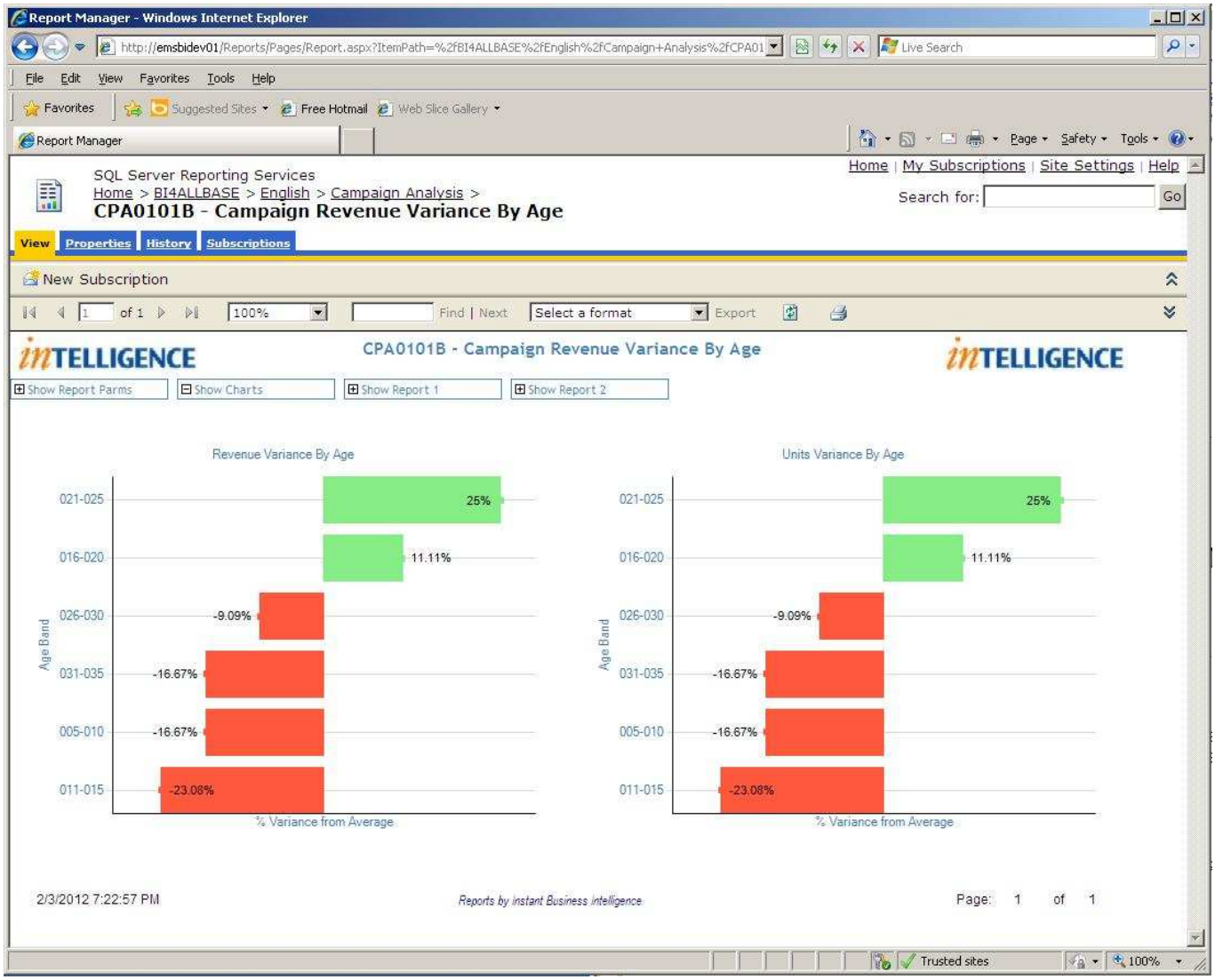
In this screen shot you can also notice the parameters of the report can be shown by clicking on the "Show Report Parameters" toggle switch. This is important because when printed it is not possible to see the parameters from the report unless they are some how printed.

8.3. Detailed Level Reports

The detailed level reports contain some interesting features. We attempted to design them to be as similar as possible to other much more expensive reporting tools like Business Objects.

The first thing to notice is that there are a series of toggle buttons along the top of each detail report. The reports actually render all the tables or charts on the report and then they are hidden according to the toggle buttons that are clicked. Usually the summary charts are defaulted to visible, as in the example below, and the detailed reports that are beneath that are defaulted to hidden.

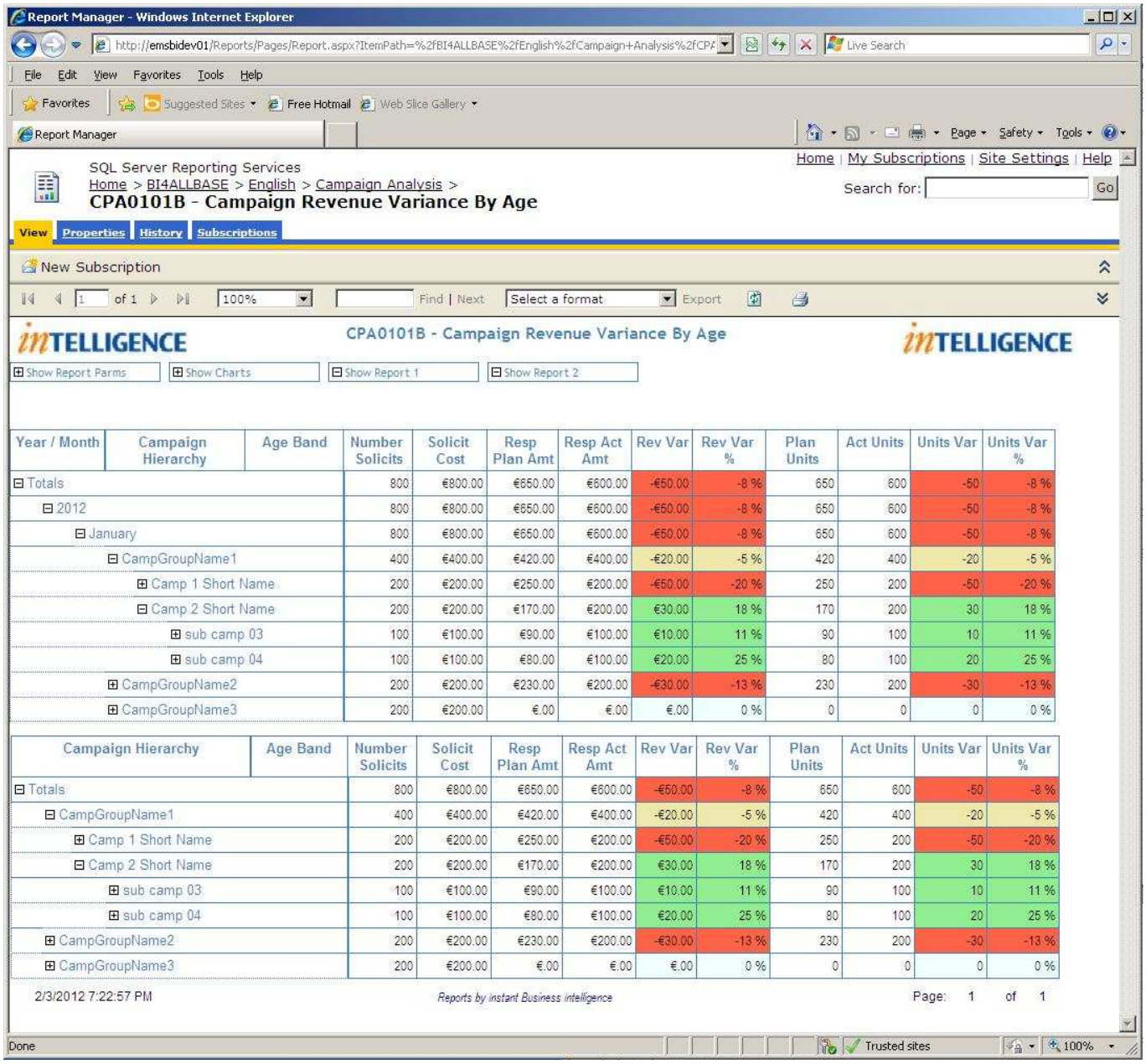
Naturally all this is customisation at implementation time by the client.



Notice how “Show Report 1” and “Show Report 2” have toggle buttons that indicate these objects are hidden.

Similarly “Show Report Parameters” is also set to indicate the report parameters are hidden.

In this next screen shot we can see how the “Show Charts” toggle button has now been turned off and the charts are now gone. In the place of the charts are Report 1 which includes the year and the month in the drill hierarchy and Report 2 which does not contain the year and month in the drill hierarchy.



Year / Month	Campaign Hierarchy	Age Band	Number Solicits	Solicit Cost	Resp Plan Amt	Resp Act Amt	Rev Var	Rev Var %	Plan Units	Act Units	Units Var	Units Var %
⊟ Totals			800	€800.00	€650.00	€600.00	-€50.00	-8 %	650	600	-50	-8 %
⊟ 2012			800	€800.00	€650.00	€600.00	-€50.00	-8 %	650	600	-50	-8 %
⊟ January			800	€800.00	€650.00	€600.00	-€50.00	-8 %	650	600	-50	-8 %
⊟ CampGroupName1			400	€400.00	€420.00	€400.00	-€20.00	-5 %	420	400	-20	-5 %
⊞ Camp 1 Short Name			200	€200.00	€250.00	€200.00	-€50.00	-20 %	250	200	-50	-20 %
⊟ Camp 2 Short Name			200	€200.00	€170.00	€200.00	€30.00	18 %	170	200	30	18 %
⊞ sub camp 03			100	€100.00	€90.00	€100.00	€10.00	11 %	90	100	10	11 %
⊞ sub camp 04			100	€100.00	€80.00	€100.00	€20.00	25 %	80	100	20	25 %
⊞ CampGroupName2			200	€200.00	€230.00	€200.00	-€30.00	-13 %	230	200	-30	-13 %
⊞ CampGroupName3			200	€200.00	€.00	€.00	€.00	0 %	0	0	0	0 %

Campaign Hierarchy	Age Band	Number Solicits	Solicit Cost	Resp Plan Amt	Resp Act Amt	Rev Var	Rev Var %	Plan Units	Act Units	Units Var	Units Var %
⊟ Totals		800	€800.00	€650.00	€600.00	-€50.00	-8 %	650	600	-50	-8 %
⊟ CampGroupName1		400	€400.00	€420.00	€400.00	-€20.00	-5 %	420	400	-20	-5 %
⊞ Camp 1 Short Name		200	€200.00	€250.00	€200.00	-€50.00	-20 %	250	200	-50	-20 %
⊟ Camp 2 Short Name		200	€200.00	€170.00	€200.00	€30.00	18 %	170	200	30	18 %
⊞ sub camp 03		100	€100.00	€90.00	€100.00	€10.00	11 %	90	100	10	11 %
⊞ sub camp 04		100	€100.00	€80.00	€100.00	€20.00	25 %	80	100	20	25 %
⊞ CampGroupName2		200	€200.00	€230.00	€200.00	-€30.00	-13 %	230	200	-30	-13 %
⊞ CampGroupName3		200	€200.00	€.00	€.00	€.00	0 %	0	0	0	0 %

The small boxes with the + or – in them next to the values are drill paths on the reports. Where it is – this indicates that the drill path has been clicked on to show the lower level of information. Where it is + it means there is more information left underneath to review.

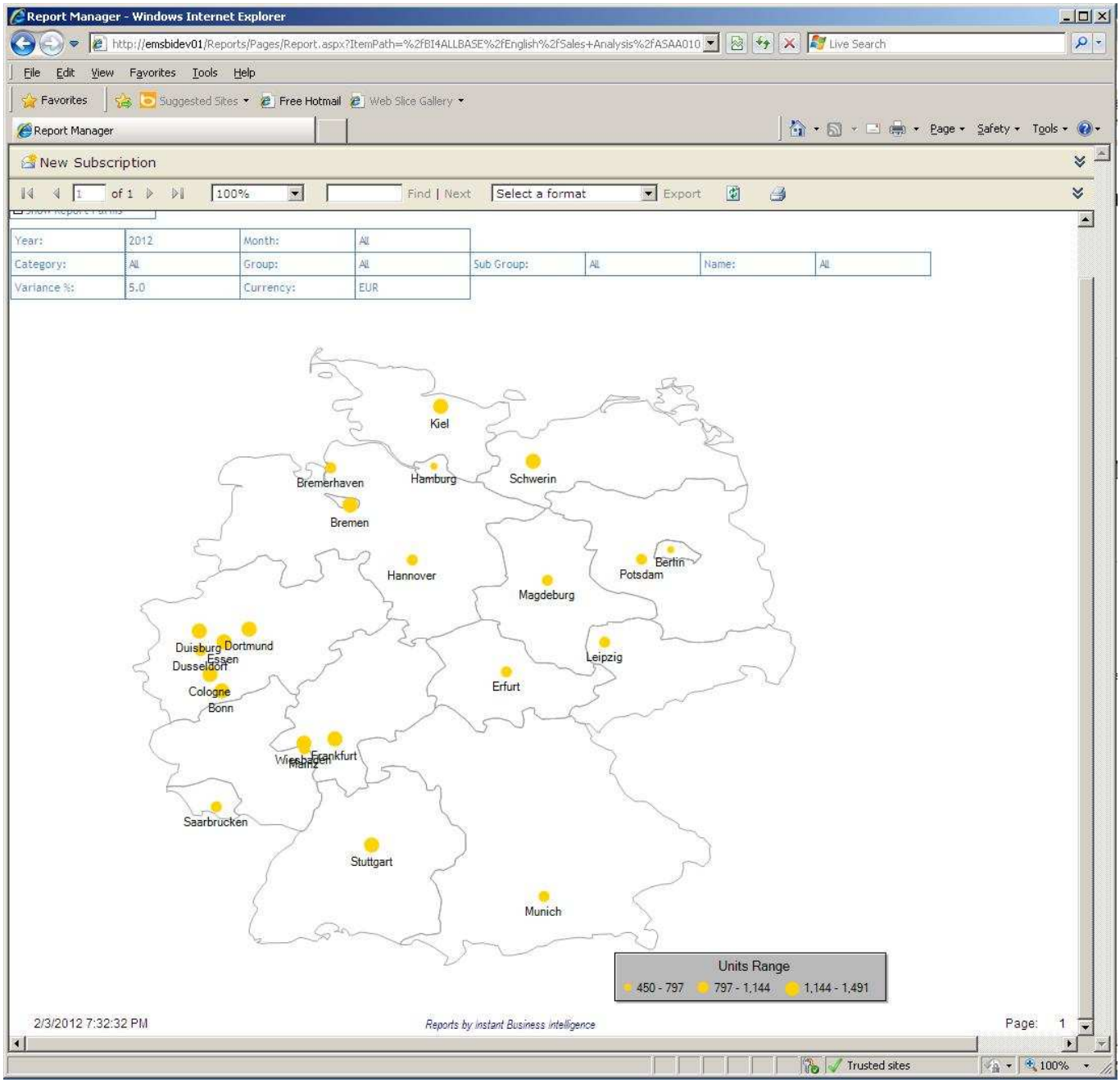
This is a very standard technique that can be easily implemented with Report Services with just a little coding effort on the report developers behalf.

Notice too how cells can be coloured depending on their variance to plan being above or below the variance tolerance.

8.4. Report Services and Maps

It is also possible to create report dashboards as maps and to drill down through the map by clicking on the name of the city you with to drill down into. This is quite useful for people who want to be able to see geographic regions compared quickly and easily along with access to the underlying data for any drill item quickly and easily.

There is, of course, the argument that bar charts are preferable to maps and there is some truth in that. This screen shot is really just to demonstrate the BI4ALL Reports can do maps if that is what the client wants.



9. APPENDIX 1 – SAMPLE STORED PROCEDURES

9.1. Drilling Down A Hierarchy

A good set of sample stored procedures to look at for how a hierarchy is established is the set of SPs around the product hierarchy. These SPs are as follows.

```
SP0005_Select_Product_Category
SP0005_Select_Product_Group
SP0005_Select_Product_Sub_Group
SP0005_Select_Product_Name

/*****************************************************************************************************************/
/* SP0005 - Stored procedures related to product dimension.                                                      */
/*                                                                                                               */
/*                                                                                                               */
/*                                                                                                               */
/*****************************************************************************************************************/

create procedure dbo.SP0005_Select_Product_Category
 @UserID               varchar(255)
AS
select distinct vm_product.prod_category from dbo.vm_product vm_product
where vm_product.pk_vm_product > 0                              Remove the zero record from the list.

 union
select '*'                                                      Add the wild card.
order by 1
go

create procedure dbo.SP0005_Select_Product_Group
 @prod_category          varchar(255)                           Pass the previously selected product category
,@UserID                 varchar(255)
AS
select distinct vm_product.prod_group from dbo.vm_product vm_product
where   1=1
and     (@prod_category = vm_product.prod_category  or   @prod_category = '*')
and     (vm_product.pk_vm_product > 0)
union
select '*'
order by 1
go
```

```
create procedure dbo.SP0005_Select_Product_Sub_Group
 @prod_category          varchar(255)                          Pass the previously selected product category
,@prod_group             varchar(255)                          Pass the previously selected product group
,@UserID                 varchar(255)
AS
select distinct vm_product.prod_sub_group from dbo.vm_product vm_product
where   1=1
and     (@prod_category = vm_product.prod_category      or @prod_category = '*')
and     (@prod_group = vm_product.prod_group            or @prod_group = '*')
and     (vm_product.pk_vm_product > 0)
union
select '*'
order by 1
go



create procedure dbo.SP0005_Select_Product_Name
 @prod_category          varchar(255)                          Pass the previously selected product category
,@prod_group             varchar(255)                          Pass the previously selected product group
,@prod_sub_group         varchar(255)                          Pass the previously selected product sub group
,@UserID                 varchar(255)
AS
select distinct prod_name from dbo.vm_product
where   1=1
and     (@prod_category = vm_product.prod_category      or @prod_category = '*')
and     (@prod_group = vm_product.prod_group            or @prod_group = '*')
and     (@prod_sub_group = vm_product.prod_sub_group    or @prod_sub_group = '*')
and     (vm_product.pk_vm_product > 0)
union
select '*'
order by 1
go
```

9.2. Plan Versus Actual Templates

There are many template stored procedures in the Data Access Layer as well as many SPs that are used actively in reports. The SP documented below is one that is a template for Sales Plan vs Actual by Year, Month, Company Reporting Structure, Geography and Product. It is not meant to be used as is. It is meant to provide a template as to how to approach sales plan vs actual across these very standard dimensions.

The most significant parts of the SP will be pointed out in comments.

`SP1015_Select_Sale_PlnvAct_By_Year_Month_Company_Geog_Product_tpl`

```
create procedure dbo.SP1015_Select_Sale_PlnvAct_By_Year_Month_Company_Geog_Product_tpl
 @year_name             varchar(255)
,@month_name            varchar(255)
,@parent_company        varchar(255)
,@org_name              varchar(255)
,@region_name           varchar(255)
,@channel_type          varchar(255)
,@economic_block        varchar(255)
,@country               varchar(255)
,@country_region        varchar(255)
,@province              varchar(255)
,@state                 varchar(255)
,@state_region          varchar(255)
,@county                varchar(255)
,@city                  varchar(255)
,@suburb                varchar(255)
,@postal_code           varchar(255)
,@prod_category         varchar(255)
,@prod_group            varchar(255)
,@prod_sub_group        varchar(255)
,@prod_name             varchar(255)
,@iso_currency_code     varchar(255)
,@UserID                varchar(255)

AS
```

Large set of input parameters.

This is actually the final outer select. You need to read the inner selects to understand what this is presenting.

```
select
      vf_sale_meas_all_tmp.iso_currency_code
     ,vf_sale_meas_all_tmp.year_name
     ,vf_sale_meas_all_tmp.month_in_year
     ,vf_sale_meas_all_tmp.month_name
     ,vf_sale_meas_all_tmp.parent_company
     ,vf_sale_meas_all_tmp.org_name
     ,vf_sale_meas_all_tmp.region_name
     ,vf_sale_meas_all_tmp.channel_type
     ,vf_sale_meas_all_tmp.economic_block
     ,vf_sale_meas_all_tmp.country
     ,vf_sale_meas_all_tmp.country_region
     ,vf_sale_meas_all_tmp.state
     ,vf_sale_meas_all_tmp.province
     ,vf_sale_meas_all_tmp.state_region
     ,vf_sale_meas_all_tmp.county
     ,vf_sale_meas_all_tmp.city
     ,vf_sale_meas_all_tmp.suburb
     ,vf_sale_meas_all_tmp.postal_code
     ,vf_sale_meas_all_tmp.prod_category
     ,vf_sale_meas_all_tmp.prod_group
     ,vf_sale_meas_all_tmp.prod_sub_group
     ,vf_sale_meas_all_tmp.prod_name
     ,vf_sale_meas_all_tmp.pln_sale_unit_amount
     ,vf_sale_meas_all_tmp.pln_sale_extended_amount
     ,vf_sale_meas_all_tmp.pln_cost_unit_amount
     ,vf_sale_meas_all_tmp.pln_cost_extended_amount
     ,vf_sale_meas_all_tmp.pln_tax1_unit_amount
     ,vf_sale_meas_all_tmp.pln_tax1_extended_amount
     ,vf_sale_meas_all_tmp.pln_tax2_unit_amount
     ,vf_sale_meas_all_tmp.pln_tax2_extended_amount
     ,vf_sale_meas_all_tmp.pln_sale_units
     ,vf_sale_meas_all_tmp.act_sale_unit_amount
     ,vf_sale_meas_all_tmp.act_sale_extended_amount
     ,vf_sale_meas_all_tmp.act_cost_unit_amount
     ,vf_sale_meas_all_tmp.act_cost_extended_amount
     ,vf_sale_meas_all_tmp.act_tax1_unit_amount
     ,vf_sale_meas_all_tmp.act_tax1_extended_amount
     ,vf_sale_meas_all_tmp.act_tax2_unit_amount
     ,vf_sale_meas_all_tmp.act_tax2_extended_amount
     ,vf_sale_meas_all_tmp.act_sale_units
```

Dimension fields listed.

Plan fields listed.

Actual fields listed. So it is presented to the query tool as one row of data with plans and actuals on the same row.

Now we have the first inner select. Notice how this inner select has to deal with either portion of the subsequent subselect being null. There may be months where this is no plan but actuals exist and there may be months where no actuals exists but plans do. Both cases must be taken care of.

```
from
(
select

 CASE WHEN vf_sale_meas_pln_tmp.iso_currency_code      is NULL THEN vf_sale_meas_act_tmp.iso_currency_code      ELSE vf_sale_meas_pln_tmp.iso_currency_code      END as iso_currency_code
,CASE WHEN vf_sale_meas_pln_tmp.year_name              is NULL THEN vf_sale_meas_act_tmp.year_name              ELSE vf_sale_meas_pln_tmp.year_name              END as year_name
,CASE WHEN vf_sale_meas_pln_tmp.month_in_year          is NULL THEN vf_sale_meas_act_tmp.month_in_year          ELSE vf_sale_meas_pln_tmp.month_in_year          END as month_in_year
,CASE WHEN vf_sale_meas_pln_tmp.month_name             is NULL THEN vf_sale_meas_act_tmp.month_name             ELSE vf_sale_meas_pln_tmp.month_name             END as month_name
,CASE WHEN vf_sale_meas_pln_tmp.parent_company         is NULL THEN vf_sale_meas_act_tmp.parent_company         ELSE vf_sale_meas_pln_tmp.parent_company         END as parent_company
,CASE WHEN vf_sale_meas_pln_tmp.org_name               is NULL THEN vf_sale_meas_act_tmp.org_name               ELSE vf_sale_meas_pln_tmp.org_name               END as org_name
,CASE WHEN vf_sale_meas_pln_tmp.region_name            is NULL THEN vf_sale_meas_act_tmp.region_name            ELSE vf_sale_meas_pln_tmp.region_name            END as region_name
,CASE WHEN vf_sale_meas_pln_tmp.channel_type           is NULL THEN vf_sale_meas_act_tmp.channel_type           ELSE vf_sale_meas_pln_tmp.channel_type           END as channel_type
,CASE WHEN vf_sale_meas_pln_tmp.economic_block         is NULL THEN vf_sale_meas_act_tmp.economic_block         ELSE vf_sale_meas_pln_tmp.economic_block         END as economic_block
,CASE WHEN vf_sale_meas_pln_tmp.country                is NULL THEN vf_sale_meas_act_tmp.country                ELSE vf_sale_meas_pln_tmp.country                END as country
,CASE WHEN vf_sale_meas_pln_tmp.country_region         is NULL THEN vf_sale_meas_act_tmp.country_region         ELSE vf_sale_meas_pln_tmp.country_region         END as country_region
,CASE WHEN vf_sale_meas_pln_tmp.state                  is NULL THEN vf_sale_meas_act_tmp.state                  ELSE vf_sale_meas_pln_tmp.state                  END as state
,CASE WHEN vf_sale_meas_pln_tmp.province               is NULL THEN vf_sale_meas_act_tmp.province               ELSE vf_sale_meas_pln_tmp.province               END as province
,CASE WHEN vf_sale_meas_pln_tmp.state_region           is NULL THEN vf_sale_meas_act_tmp.state_region           ELSE vf_sale_meas_pln_tmp.state_region           END as state_region
,CASE WHEN vf_sale_meas_pln_tmp.county                 is NULL THEN vf_sale_meas_act_tmp.county                 ELSE vf_sale_meas_pln_tmp.county                 END as county
,CASE WHEN vf_sale_meas_pln_tmp.city                   is NULL THEN vf_sale_meas_act_tmp.city                   ELSE vf_sale_meas_pln_tmp.city                   END as city
,CASE WHEN vf_sale_meas_pln_tmp.suburb                 is NULL THEN vf_sale_meas_act_tmp.suburb                 ELSE vf_sale_meas_pln_tmp.suburb                 END as suburb
,CASE WHEN vf_sale_meas_pln_tmp.postal_code            is NULL THEN vf_sale_meas_act_tmp.postal_code            ELSE vf_sale_meas_pln_tmp.postal_code            END as postal_code
,CASE WHEN vf_sale_meas_pln_tmp.prod_category          is NULL THEN vf_sale_meas_act_tmp.prod_category          ELSE vf_sale_meas_pln_tmp.prod_category          END as prod_category
,CASE WHEN vf_sale_meas_pln_tmp.prod_group             is NULL THEN vf_sale_meas_act_tmp.prod_group             ELSE vf_sale_meas_pln_tmp.prod_group             END as prod_group
,CASE WHEN vf_sale_meas_pln_tmp.prod_sub_group         is NULL THEN vf_sale_meas_act_tmp.prod_sub_group         ELSE vf_sale_meas_pln_tmp.prod_sub_group         END as prod_sub_group
,CASE WHEN vf_sale_meas_pln_tmp.prod_name              is NULL THEN vf_sale_meas_act_tmp.prod_name              ELSE vf_sale_meas_pln_tmp.prod_name              END as prod_name

,CASE WHEN vf_sale_meas_pln_tmp.pln_sale_unit_amount       is NULL THEN 0 ELSE vf_sale_meas_pln_tmp.pln_sale_unit_amount       END as pln_sale_unit_amount
,CASE WHEN vf_sale_meas_pln_tmp.pln_sale_extended_amount   is NULL THEN 0 ELSE vf_sale_meas_pln_tmp.pln_sale_extended_amount   END as pln_sale_extended_amount
,CASE WHEN vf_sale_meas_pln_tmp.pln_cost_unit_amount       is NULL THEN 0 ELSE vf_sale_meas_pln_tmp.pln_cost_unit_amount       END as pln_cost_unit_amount
,CASE WHEN vf_sale_meas_pln_tmp.pln_cost_extended_amount   is NULL THEN 0 ELSE vf_sale_meas_pln_tmp.pln_cost_extended_amount   END as pln_cost_extended_amount
,CASE WHEN vf_sale_meas_pln_tmp.pln_tax1_unit_amount       is NULL THEN 0 ELSE vf_sale_meas_pln_tmp.pln_tax1_unit_amount       END as pln_tax1_unit_amount
,CASE WHEN vf_sale_meas_pln_tmp.pln_tax1_extended_amount   is NULL THEN 0 ELSE vf_sale_meas_pln_tmp.pln_tax1_extended_amount   END as pln_tax1_extended_amount
,CASE WHEN vf_sale_meas_pln_tmp.pln_tax2_unit_amount       is NULL THEN 0 ELSE vf_sale_meas_pln_tmp.pln_tax2_unit_amount       END as pln_tax2_unit_amount
,CASE WHEN vf_sale_meas_pln_tmp.pln_tax2_extended_amount   is NULL THEN 0 ELSE vf_sale_meas_pln_tmp.pln_tax2_extended_amount   END as pln_tax2_extended_amount
,CASE WHEN vf_sale_meas_pln_tmp.pln_sale_units             is NULL THEN 0 ELSE vf_sale_meas_pln_tmp.pln_sale_units             END as pln_sale_units


,CASE WHEN vf_sale_meas_act_tmp.act_sale_unit_amount       is NULL THEN 0 ELSE vf_sale_meas_act_tmp.act_sale_unit_amount       END as act_sale_unit_amount
,CASE WHEN vf_sale_meas_act_tmp.act_sale_extended_amount   is NULL THEN 0 ELSE vf_sale_meas_act_tmp.act_sale_extended_amount   END as act_sale_extended_amount
,CASE WHEN vf_sale_meas_act_tmp.act_cost_unit_amount       is NULL THEN 0 ELSE vf_sale_meas_act_tmp.act_cost_unit_amount       END as act_cost_unit_amount
,CASE WHEN vf_sale_meas_act_tmp.act_cost_extended_amount   is NULL THEN 0 ELSE vf_sale_meas_act_tmp.act_cost_extended_amount   END as act_cost_extended_amount
```

```
,CASE WHEN vf_sale_meas_act_tmp.act_tax1_unit_amount          is NULL THEN 0 ELSE vf_sale_meas_act_tmp.act_tax1_unit_amount          END as act_tax1_unit_amount
,CASE WHEN vf_sale_meas_act_tmp.act_tax1_extended_amount      is NULL THEN 0 ELSE vf_sale_meas_act_tmp.act_tax1_extended_amount      END as act_tax1_extended_amount
,CASE WHEN vf_sale_meas_act_tmp.act_tax2_unit_amount          is NULL THEN 0 ELSE vf_sale_meas_act_tmp.act_tax2_unit_amount          END as act_tax2_unit_amount
,CASE WHEN vf_sale_meas_act_tmp.act_tax2_extended_amount      is NULL THEN 0 ELSE vf_sale_meas_act_tmp.act_tax2_extended_amount      END as act_tax2_extended_amount
,CASE WHEN vf_sale_meas_act_tmp.act_sale_units                is NULL THEN 0 ELSE vf_sale_meas_act_tmp.act_sale_units                END as act_sale_units
```

Now we have the second inner select. This select fetches the plan numbers from the vf_sale_plan table.

```
from
(
select
```

Select the dimension fields.

```
 vm_currency.iso_currency_code                iso_currency_code
,vm_sale_period.year_name                     year_name
,vm_sale_period.month_in_year                 month_in_year
,vm_sale_period.month_name                    month_name
,vm_comp_reporting_struct.parent_company      parent_company
,vm_comp_reporting_struct.org_name            org_name
,vm_comp_reporting_struct.region_name          region_name
,vm_comp_reporting_struct.channel_type        channel_type
,vm_geography.economic_block                  economic_block
,vm_geography.country                         country
,vm_geography.country_region                  country_region
,vm_geography.state                           state
,vm_geography.province                        province
,vm_geography.state_region                    state_region
,vm_geography.county                          county
,vm_geography.city                            city
,vm_geography.suburb                          suburb
,vm_geography.postal_code                     postal_code
,vm_product.prod_category                     prod_category
,vm_product.prod_group                        prod_group
,vm_product.prod_sub_group                    prod_sub_group
,vm_product.prod_name                         prod_name
```

Select the Fact fields.

```
      ,sum(a.sale_unit_amount)                pln_sale_unit_amount
      ,sum(a.sale_extended_amount)            pln_sale_extended_amount
      ,sum(a.cost_unit_amount)                pln_cost_unit_amount
      ,sum(a.cost_extended_amount)            pln_cost_extended_amount
      ,sum(a.tax1_unit_amount)                pln_tax1_unit_amount
      ,sum(a.tax1_extended_amount)            pln_tax1_extended_amount
      ,sum(a.tax2_unit_amount)                pln_tax2_unit_amount
      ,sum(a.tax2_extended_amount)            pln_tax2_extended_amount
      ,sum(a.sale_units)                      pln_sale_units
```

Since this is the PLAN portion of the query set the actuals to zeros.

```
      ,0                                      act_sale_unit_amount
      ,0                                      act_sale_extended_amount
      ,0                                      act_cost_unit_amount
      ,0                                      act_cost_extended_amount
      ,0                                      act_tax1_unit_amount
      ,0                                      act_tax1_extended_amount
      ,0                                      act_tax2_unit_amount
      ,0                                      act_tax2_extended_amount
      ,0                                      act_sale_units
from
       dbo.vf_sale_plan                 a
      ,dbo.vm_sale_period               vm_sale_period
      ,dbo.vm_product                   vm_product
      ,dbo.vm_geography                 vm_geography
      ,dbo.vm_currency                  vm_currency
      ,dbo.vm_comp_reporting_struct     vm_comp_reporting_struct
```

Redundant joins

```
/* #0001 redundant joins start tag */

      ,dbo.vm_unit_of_measure           f
      ,dbo.vm_plan                      g
      ,dbo.vm_sub_campaign               h

/* #0002 redundant joins end tag */

where   1=1
```

The sql server 2005 partitioning strategy.

```
/* #003 SQL Server 2005 Partitioning Strategy Joins start tag */

and    vm_sale_period.dummy_join_col   = vm_comp_reporting_struct.dummy_join_col
and    a.partitioning_column           = vm_comp_reporting_struct.partitioning_column_part_01 + vm_sale_period.pk_vm_sale_period

/* #004 SQL Server 2005 Partitioning Strategy Joins end tag */
```

Normal joins .

```
and    a.dk_vm_sale_period             = vm_sale_period.pk_vm_sale_period
and    a.dk_vm_product                 = vm_product.pk_vm_product
and    a.dk_vm_geography               = vm_geography.pk_vm_geography
and    a.dk_vm_currency                = vm_currency.pk_vm_currency
and    a.dk_vm_comp_reporting_struct   = vm_comp_reporting_struct.pk_vm_comp_reporting_struct
```

Redundant joins

```
/* #0001 redundant joins start tag */

and    a.dk_vm_unit_of_measure         = f.pk_vm_unit_of_measure
and    a.dk_vm_plan                    = g.pk_vm_plan
and    a.dk_vm_sub_campaign            = h.pk_vm_sub_campaign

/* #0002 redundant joins end tag */
```

Where clauses to filter out data depending on the parameters.

```
and   (vm_comp_reporting_struct.parent_company      = @parent_company      or  @parent_company = '*')
and   (vm_comp_reporting_struct.org_name            = @org_name            or  @org_name = '*')
and   (vm_comp_reporting_struct.region_name         = @region_name         or  @region_name = '*')
and   (vm_comp_reporting_struct.channel_type       = @channel_type        or  @channel_type = '*')

and   (vm_sale_period.year_name                    = @year_name           or @year_name = '*' )
and   (vm_sale_period.month_name                   = @month_name          or @month_name = '*')

and   (vm_product.prod_category                    = @prod_category       or @prod_category = '*')
and   (vm_product.prod_group                       = @prod_group          or @prod_group = '*')
and   (vm_product.prod_sub_group                   = @prod_sub_group      or @prod_sub_group = '*')
and   (vm_product.prod_name                        = @prod_name           or @prod_name = '*')

and   (vm_geography.economic_block                 = @economic_block      or  @economic_block = '*')
and   (vm_geography.country                        = @country             or  @country = '*')
and   (vm_geography.country_region                 = @country_region      or  @country_region = '*')
and   (vm_geography.state                          = @state               or  @state = '*')
and   (vm_geography.province                       = @province            or  @province = '*')
and   (vm_geography.state_region                   = @state_region        or  @state_region = '*')
and   (vm_geography.county                         = @county              or  @county = '*')
and   (vm_geography.city                           = @city                or  @city = '*')
and   (vm_geography.suburb                         = @suburb              or  @suburb = '*')
and   (vm_geography.postal_code                    = @postal_code         or  @postal_code = '*')

and   (vm_currency.iso_currency_code               = @iso_currency_code   or  @iso_currency_code = '*')
```

Group by to summarise

```
group by

      vm_currency.iso_currency_code
     ,vm_sale_period.year_name
     ,vm_sale_period.month_in_year
     ,vm_sale_period.month_name
     ,vm_comp_reporting_struct.parent_company
     ,vm_comp_reporting_struct.org_name
     ,vm_comp_reporting_struct.region_name
     ,vm_comp_reporting_struct.channel_type
     ,vm_geography.economic_block
     ,vm_geography.country
     ,vm_geography.country_region
     ,vm_geography.state
     ,vm_geography.province
     ,vm_geography.state_region
     ,vm_geography.county
     ,vm_geography.city
     ,vm_geography.suburb
     ,vm_geography.postal_code
     ,vm_product.prod_category
     ,vm_product.prod_group
     ,vm_product.prod_sub_group
     ,vm_product.prod_name
```

Give the result set a name for the subsequent join.

```
) vf_sale_meas_pln_tmp
```

Notice the full outer join of the two most deeply nested selects.

```
full outer join
```

Now we have the third inner select. This select fetches the actual numbers from the vf_sale_meas table.

```
(
select
```

Select the dimension fields.

```
 vm_currency.iso_currency_code                iso_currency_code
,vm_sale_meas_period.year_name                year_name
,vm_sale_meas_period.month_in_year            month_in_year
,vm_sale_meas_period.month_name               month_name
,vm_comp_reporting_struct.parent_company      parent_company
,vm_comp_reporting_struct.org_name            org_name
,vm_comp_reporting_struct.region_name         region_name
,vm_comp_reporting_struct.channel_type        channel_type
,vm_geography.economic_block                  economic_block
,vm_geography.country                         country
,vm_geography.country_region                  country_region
,vm_geography.state                           state
,vm_geography.province                        province
,vm_geography.state_region                    state_region
,vm_geography.county                          county
,vm_geography.city                            city
,vm_geography.suburb                          suburb
,vm_geography.postal_code                     postal_code
,vm_product.prod_category                     prod_category
,vm_product.prod_group                        prod_group
,vm_product.prod_sub_group                    prod_sub_group
,vm_product.prod_name                         prod_name
```

Since this is the ACTUAL portion of the query set the plans to zeros.

```
,0                                            pln_sale_unit_amount
,0                                            pln_sale_extended_amount
,0                                            pln_cost_unit_amount
,0                                            pln_cost_extended_amount
,0                                            pln_tax1_unit_amount
,0                                            pln_tax1_extended_amount
,0                                            pln_tax2_unit_amount
,0                                            pln_tax2_extended_amount
,0                                            pln_sale_units
```

Select the Fact fields.

```
      ,sum(a.sale_unit_amount)              act_sale_unit_amount
      ,sum(a.sale_extended_amount)          act_sale_extended_amount
      ,sum(a.cost_unit_amount)              act_cost_unit_amount
      ,sum(a.cost_extended_amount)          act_cost_extended_amount
      ,sum(a.tax1_unit_amount)              act_tax1_unit_amount
      ,sum(a.tax1_extended_amount)          act_tax1_extended_amount
      ,sum(a.tax2_unit_amount)              act_tax2_unit_amount
      ,sum(a.tax2_extended_amount)          act_tax2_extended_amount
      ,sum(a.sale_units)                    act_sale_units
from
       dbo.vf_sale_meas                     a
      ,dbo.vm_sale_meas_period              vm_sale_meas_period
      ,dbo.vm_product                       vm_product
      ,dbo.vm_geography                     vm_geography
      ,dbo.vm_currency                      vm_currency
      ,dbo.vm_comp_reporting_struct         vm_comp_reporting_struct
```

Redundant joins

```
/* #0001 redundant joins start tag */

      ,dbo.vm_sub_campaign                  c
      ,dbo.vm_all_demographic               e
      ,dbo.vm_unit_of_measure               h

/* #0002 redundant joins end tag */

where   1=1
```

The sql server 2005 partitioning strategy.

```
/* #003 SQL Server 2005 Partitioning Strategy Joins start tag */

and    vm_sale_meas_period.dummy_join_col     = vm_comp_reporting_struct.dummy_join_col
and    a.partitioning_column                  = vm_comp_reporting_struct.partitioning_column_part_01 + vm_sale_meas_period.pk_vm_sale_meas_period

/* #004 SQL Server 2005 Partitioning Strategy Joins end tag */
```

Normal joins .

```
and    a.dk_vm_sale_meas_period              = vm_sale_meas_period.pk_vm_sale_meas_period
and    a.dk_vm_product                       = vm_product.pk_vm_product
and    a.dk_vm_customer_geography            = vm_geography.pk_vm_geography
and    a.dk_vm_currency                      = vm_currency.pk_vm_currency
and    a.dk_vm_comp_reporting_struct         = vm_comp_reporting_struct.pk_vm_comp_reporting_struct
```

Redundant joins

```
/* #0001 redundant joins start tag */

and    a.dk_vm_sub_campaign                  = c.pk_vm_sub_campaign
and    a.dk_vm_customer_demographic          = e.pk_vm_all_demographic
and    a.dk_vm_unit_of_measure               = h.pk_vm_unit_of_measure

/* #0002 redundant joins end tag */
```

Where clauses to filter out data depending on the parameters.

```
and    (vm_comp_reporting_struct.parent_company   = @parent_company      or  @parent_company = '*')
and    (vm_comp_reporting_struct.org_name         = @org_name           or  @org_name = '*')
and    (vm_comp_reporting_struct.region_name      = @region_name        or  @region_name = '*')
and    (vm_comp_reporting_struct.channel_type     = @channel_type       or  @channel_type = '*')

and    (vm_sale_meas_period.year_name             = @year_name          or @year_name = '*' )
and    (vm_sale_meas_period.month_name            = @month_name         or @month_name = '*')

and    (vm_product.prod_category                  = @prod_category      or @prod_category = '*')
and    (vm_product.prod_group                     = @prod_group         or @prod_group = '*')
and    (vm_product.prod_sub_group                 = @prod_sub_group     or @prod_sub_group = '*')
and    (vm_product.prod_name                      = @prod_name          or @prod_name = '*')

and    (vm_geography.economic_block               = @economic_block     or  @economic_block = '*')
and    (vm_geography.country                      = @country            or  @country = '*')
and    (vm_geography.country_region               = @country_region     or  @country_region = '*')
and    (vm_geography.state                        = @state              or  @state = '*')
and    (vm_geography.province                     = @province           or  @province = '*')
and    (vm_geography.state_region                 = @state_region       or  @state_region = '*')
and    (vm_geography.county                       = @county             or  @county = '*')
and    (vm_geography.city                         = @city               or  @city = '*')
and    (vm_geography.suburb                       = @suburb             or  @suburb = '*')
and    (vm_geography.postal_code                  = @postal_code        or  @postal_code = '*')

and    (vm_currency.iso_currency_code             = @iso_currency_code  or  @iso_currency_code = '*')
```

Group by to summarise

```
group by
      vm_currency.iso_currency_code
     ,vm_sale_meas_period.year_name
     ,vm_sale_meas_period.month_in_year
     ,vm_sale_meas_period.month_name
     ,vm_comp_reporting_struct.parent_company
     ,vm_comp_reporting_struct.org_name
     ,vm_comp_reporting_struct.region_name
     ,vm_comp_reporting_struct.channel_type

     ,vm_geography.economic_block
     ,vm_geography.country
     ,vm_geography.country_region
     ,vm_geography.state
     ,vm_geography.province
     ,vm_geography.state_region
     ,vm_geography.county
     ,vm_geography.city
     ,vm_geography.suburb
     ,vm_geography.postal_code

     ,vm_product.prod_category
     ,vm_product.prod_group
     ,vm_product.prod_sub_group
     ,vm_product.prod_name
```

Give the result set a name for the subsequent join.

```
) vf_sale_meas_act_tmp
```

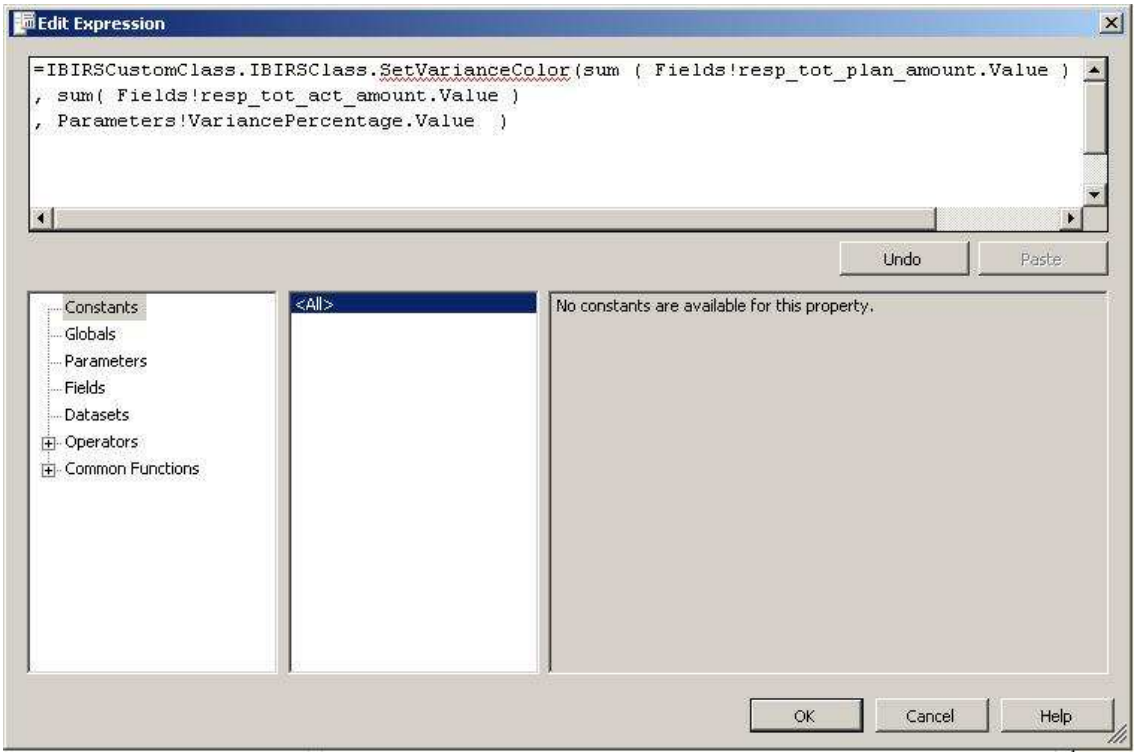
The on for the full outer join.

```
on
      vf_sale_meas_pln_tmp.iso_currency_code          = vf_sale_meas_act_tmp.iso_currency_code
and   vf_sale_meas_pln_tmp.year_name                  = vf_sale_meas_act_tmp.year_name
and   vf_sale_meas_pln_tmp.month_in_year              = vf_sale_meas_act_tmp.month_in_year
and   vf_sale_meas_pln_tmp.month_name                 = vf_sale_meas_act_tmp.month_name
and   vf_sale_meas_pln_tmp.parent_company             = vf_sale_meas_act_tmp.parent_company
and   vf_sale_meas_pln_tmp.org_name                   = vf_sale_meas_act_tmp.org_name
and   vf_sale_meas_pln_tmp.region_name                = vf_sale_meas_act_tmp.region_name
and   vf_sale_meas_pln_tmp.channel_type               = vf_sale_meas_act_tmp.channel_type
and   vf_sale_meas_pln_tmp.economic_block             = vf_sale_meas_act_tmp.economic_block
and   vf_sale_meas_pln_tmp.country                    = vf_sale_meas_act_tmp.country
and   vf_sale_meas_pln_tmp.country_region             = vf_sale_meas_act_tmp.country_region
and   vf_sale_meas_pln_tmp.state                      = vf_sale_meas_act_tmp.state
and   vf_sale_meas_pln_tmp.province                   = vf_sale_meas_act_tmp.province
and   vf_sale_meas_pln_tmp.state_region               = vf_sale_meas_act_tmp.state_region
and   vf_sale_meas_pln_tmp.county                     = vf_sale_meas_act_tmp.county
and   vf_sale_meas_pln_tmp.city                       = vf_sale_meas_act_tmp.city
and   vf_sale_meas_pln_tmp.suburb                     = vf_sale_meas_act_tmp.suburb
and   vf_sale_meas_pln_tmp.postal_code                = vf_sale_meas_act_tmp.postal_code
and   vf_sale_meas_pln_tmp.prod_category              = vf_sale_meas_act_tmp.prod_category
and   vf_sale_meas_pln_tmp.prod_group                 = vf_sale_meas_act_tmp.prod_group
and   vf_sale_meas_pln_tmp.prod_sub_group             = vf_sale_meas_act_tmp.prod_sub_group
and   vf_sale_meas_pln_tmp.prod_name                  = vf_sale_meas_act_tmp.prod_name

)
vf_sale_meas_all_tmp

order by 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21
```

10. APPENDIX 2 – THE VISUAL BASIC CUSTOM ASSEMBLY

Report Services allows the designer of reports to customise the appearance of reports at run time. This can be done via a custom assembly. When designing and building BI4ALL IBI staff determined that the custom assembly could be used for far more than simple colouring of characters and lines. It could be used for performing quite substantial calculations thereby removing these calculations from being coded into the report itself.

The custom assembly is invoked in a report as follows.



In almost all cases the custom assembly can be called to return an expression that would otherwise be selected from some list or entered as is. The one that is invoked in this case is the setting of the colour of a bar depending on the variance of the response planned amount and the response actual amount. The variance percentage is the amount of variance to set the bar to red, amber, or green.

The actual code that is executed from this call is as follows.

```
Public Shared Function SetVarianceColor(ByVal Plan As Double, ByVal Actual As Double, ByVal VarPercentage As Double) As String

    Dim Variance As Double

    Variance = 0
    VarPercentage = VarPercentage / 100

    If Plan = 0 Or Actual = 0 Then
       Return "Azure"
    End If

    Variance = (Actual - Plan) / Plan

    If (Variance < (VarPercentage * -1)) Then
       Return "Tomato"
    End If

    If (Variance > VarPercentage) Then
       Return "LightGreen"
```

```
    End If

    Return "PaleGoldenrod"

End Function
```

Clearly, what the code is doing is as follows:

1. If the plan or actual is 0 then it sets the colour of the bar to Azure. You can not have a variance if one of the amounts is zero.

2. If the actual is worse than the plan by more than VariancePercentage then the bar is coloured Tomato, meaning that it is a shade of red.

3. If the actual is better than the plan by more than VariancePercentage then the bar is coloured LightGreen.

4. If the actual is neither of the above, meaning it is within VariancePercentage of the plan then PaleGoldenrod is used. This is the closest colour to “amber” as was available.

This is one simple example of setting colours for bars on charts. In the example reports you can see how this works. To download sample reports click here.

http://www.instantbi.com/LinkClick.aspx?link=BI4ALL+Example+Reports.zip&tabid=59&mid=499

Obviously, because the custom assembly is merely code, it is possible to extend the custom assembly and to create more and more complex functions as the client might wish.

Let us review more of the functions in the custom assembly so that the reader can understand what the custom assembly is doing.

10.1. Calculate a Percentage

This is a simple calculation of percentage while protecting for zero divide errors.

```
Public Shared Function CalcPercantage(ByVal Numerator As Double, ByVal Denominator As Double) As Double

    Dim Percentage As Double

    If Numerator = 0 Or Denominator = 0 Then
       Return 0
    End If

    Percentage = Numerator / Denominator
    Return Percentage

End Function
```

10.2. Calculate a Variance Percentage

This is a simple variance percentage of the variance of the actual to plan while protecting for the case where either the plan or the actual is zero.

```
Public Shared Function CalcVariancePercentage(ByVal Plan As Double, ByVal Actual As Double) As Double

    Dim Variance As Double

    Variance = 0

    If Plan = 0 Or Actual = 0 Then
       Return 0
    End If

    Variance = (Actual - Plan) / Plan
    Return Variance

End Function
```

10.3. Calculate the Estimated Response for a Campaign

When reporting on campaigns there is a need to estimate the response from a sub group if the sub group responded according to average. These types of calculations are performed to determine the variance of a sub group to the overall group. In these cases it is then possible to see the sub groups that over perform or under perform the average.

```
Public Shared Function CalcEstResponses(ByVal GroupNbrSols As Double, ByVal GroupNbrResp As Double, ByVal TotNbrSols As Double,
ByVal TotNbrResp As Double) As Double

    Dim ws_tot_response_rate As Double
    Dim ws_exp_response As Double

    If GroupNbrSols = 0 Or GroupNbrResp = 0 Or TotNbrSols = 0 Or TotNbrResp = 0 Then
       Return 0
    End If

    ws_tot_response_rate = TotNbrResp / TotNbrSols
    ws_exp_response = ws_tot_response_rate * GroupNbrSols
    Return ws_exp_response
End Function
```

10.4. Calculate the Over/Under Variance Percentage

This function calculates the over or under variance percentage of a sub group to the overall group.

```
Public Shared Function CalcOverUnderVariancePct(ByVal GroupNbrSols As Double, ByVal GroupNbrResp As Double, ByVal TotNbrSols As Double, ByVal TotNbrResp As Double) As Double

    Dim VariancePct As Double
    Dim ws_tot_response_rate As Double
    Dim ws_exp_response As Double
    Dim ws_diff_response As Double

    VariancePct = 0

    If GroupNbrSols = 0 Or GroupNbrResp = 0 Or TotNbrSols = 0 Or TotNbrResp = 0 Then
       Return 0
    End If

    ws_tot_response_rate = TotNbrResp / TotNbrSols
    ws_exp_response = ws_tot_response_rate * GroupNbrSols
    ws_diff_response = GroupNbrResp - ws_exp_response
    VariancePct = ws_diff_response / ws_exp_response
    Return VariancePct
End Function
```

10.5. Calculate the Over/Under Variance Number

This function calculates the over or under variance number of a sub group to the overall group.

```
Public Shared Function CalcOverUnderVarianceNbr(ByVal GroupNbrSols As Double, ByVal GroupNbrResp As Double, ByVal TotNbrSols As Double, ByVal TotNbrResp As Double) As Double

    Dim VarianceNbr As Double
    Dim ws_tot_response_rate As Double
    Dim ws_exp_response As Double

    VarianceNbr = 0

    If GroupNbrSols = 0 Or GroupNbrResp = 0 Or TotNbrSols = 0 Or TotNbrResp = 0 Then
       Return 0
    End If

    ws_tot_response_rate = TotNbrResp / TotNbrSols
    ws_exp_response = ws_tot_response_rate * GroupNbrSols
    VarianceNbr = GroupNbrResp - ws_exp_response
    Return VarianceNbr
End Function
```

10.6. Colour the Over Under Bars and Lines By Variance

This function sets the colour of bars, lines and cells on reports based on the variance percentage of the sub group to the overall group for the particular amounts being represented on the report.

```
Public Shared Function ColorOverUnderVariance(ByVal GroupNbrSols As Double, ByVal GroupNbrResp As Double, ByVal TotNbrSols As Double, ByVal TotNbrResp As Double, ByVal VarPercentage As Double) As String

    Dim VariancePct As Double
    Dim ws_tot_response_rate As Double
    Dim ws_exp_response As Double
    Dim ws_diff_response As Double

    VariancePct = 0

    If GroupNbrSols = 0 Or GroupNbrResp = 0 Or TotNbrSols = 0 Or TotNbrResp = 0 Then
       Return "Azure"
    End If

    ws_tot_response_rate = TotNbrResp / TotNbrSols
    ws_exp_response = ws_tot_response_rate * GroupNbrSols
    ws_diff_response = GroupNbrResp - ws_exp_response
    VariancePct = ws_diff_response / ws_exp_response

    VarPercentage = VarPercentage / 100


    If (VariancePct < (VarPercentage * -1)) Then
       Return "Tomato"
    End If

    If (VariancePct > VarPercentage) Then
       Return "LightGreen"
    End If


    Return "PaleGoldenrod"

End Function
```

10.7. Formatting Currencies

This function allows the proper formatting of currencies on the reports. It means that reports can be in mixed currencies and the formatting will still be appropriate. Further the report developer does not need to know how to format the currency. The report developer simply sends the currency to the custom assembly and the formatting will be done for him/her.

```
Public Shared Function FormatCurrencies(ByVal InCurrency As String) As String
    Dim CurrencyFormat As String

    CurrencyFormat = "X#,#.00"

    Select Case InCurrency
      Case "USD"
        CurrencyFormat = "U$#,#.00"
      Case "EUR"
        CurrencyFormat = "€#,#.00"
      Case "GBP"
        CurrencyFormat = "£#,#.00"
    End Select

    Return CurrencyFormat

  End Function
```

10.8. Setting the Fonts and Colours of the Report

This set of functions are self explanatory. These functions are called to set the colours, fonts and sizes of the items they are named for. The reader can see how many objects are able to have their characteristics changed merely by changing these functions.

```
Public Shared Function ColorTitleBackground() As String
    Return "White"
End Function

Public Shared Function ColorTitleText() As String
    Return "SteelBlue"
End Function

Public Shared Function ColorTitleBorder() As String
    Return "White"
End Function

Public Shared Function TitleFont() As String
    Return "Trebuchet MS"
End Function

Public Shared Function TitleFontSize() As String
    Return "12pt"
End Function

Public Shared Function ColorShowBoxesBackground() As String
    Return "White"
End Function

Public Shared Function ColorShowBoxesText() As String
    Return "SteelBlue"
End Function

Public Shared Function ColorShowBoxesBorder() As String
    Return "SteelBlue"
End Function

Public Shared Function ShowBoxesFont() As String
    Return "Trebuchet MS"
End Function

Public Shared Function ShowBoxesFontSize() As String
    Return "8pt"
End Function

Public Shared Function ColorParametersBackground() As String
    Return "White"
End Function

Public Shared Function ColorParametersText() As String
    Return "SteelBlue"
End Function

Public Shared Function ColorParametersBorder() As String
    Return "SteelBlue"
End Function

Public Shared Function ParametersFont() As String
    Return "Trebuchet MS"
End Function
```

```
Public Shared Function ParametersFontSize() As String
    Return "8pt"
End Function

Public Shared Function ColorHeading01Background() As String
    Return "White"
End Function

Public Shared Function ColorHeading01Text() As String
    Return "SteelBlue"
End Function

Public Shared Function ColorHeading01Border() As String
    Return "SteelBlue"
End Function

Public Shared Function Heading01Font() As String
    Return "Arial"
End Function

Public Shared Function Heading01FontSize() As String
    Return "10pt"
End Function

Public Shared Function ColorHeading02Background() As String
    Return "White"
End Function

Public Shared Function ColorHeading02Text() As String
    Return "SteelBlue"
End Function

Public Shared Function ColorHeading02Border() As String
    Return "SteelBlue"
End Function

Public Shared Function Heading02Font() As String
    Return "Arial"
End Function

Public Shared Function Heading02FontSize() As String
    Return "10pt"
End Function

Public Shared Function ColorHeading03Background() As String
    Return "White"
End Function

Public Shared Function ColorHeading03Text() As String
    Return "SteelBlue"
End Function

Public Shared Function ColorHeading03Border() As String
    Return "SteelBlue"
End Function


Public Shared Function Heading03Font() As String
    Return "Arial"
End Function

Public Shared Function Heading03FontSize() As String
```

```
    Return "10pt"
End Function

Public Shared Function ColorCellBackground() As String
    Return "White"
End Function

Public Shared Function ColorCellBorder() As String
    Return "SteelBlue"
End Function
Public Shared Function ColorCellText() As String
    Return "Black"
End Function

Public Shared Function CellFont() As String
    Return "Arial Narrow"
End Function

Public Shared Function CellFontSize() As String
    Return "10pt"
End Function

Public Shared Function ColorRowHeader01Background() As String
    Return "White"
End Function

Public Shared Function ColorRowHeader01Border() As String
    Return "SteelBlue"
End Function

Public Shared Function ColorRowHeader01Text() As String
    Return "SteelBlue"
End Function

Public Shared Function RowHeader01Font() As String
    Return "Arial"
End Function

Public Shared Function RowHeader01FontSize() As String
    Return "10pt"
End Function

Public Shared Function ColorRowHeader02Background() As String
    Return "White"
End Function

Public Shared Function ColorRowHeader02Border() As String
    Return "SteelBlue"
End Function

Public Shared Function ColorRowHeader02Text() As String
    Return "Black"
End Function

Public Shared Function RowHeader02Font() As String
    Return "Arial"
End Function

Public Shared Function RowHeader02FontSize() As String
    Return "8pt"
End Function

Public Shared Function ColorRowHeader03Background() As String
```

```
    Return "White"

End Function

Public Shared Function ColorRowHeader03Border() As String
    Return "SteelBlue"
End Function

Public Shared Function ColorRowHeader03Text() As String
    Return "SteelBlue"
End Function

Public Shared Function RowHeader03Font() As String
    Return "Arial"
End Function

Public Shared Function RowHeader03FontSize() As String
    Return "8pt"
End Function

Public Shared Function ColorRowHeader04Background() As String
    Return "White"
End Function

Public Shared Function ColorRowHeader04Border() As String
    Return "SteelBlue"
End Function

Public Shared Function ColorRowHeader04Text() As String
    Return "SteelBlue"
End Function

Public Shared Function RowHeader04Font() As String
    Return "Arial"
End Function

Public Shared Function RowHeader04FontSize() As String
    Return "8pt"
End Function

Public Shared Function ColorRowHeader05Background() As String
    Return "White"
End Function

Public Shared Function ColorRowHeader05Border() As String
    Return "SteelBlue"
End Function

Public Shared Function ColorRowHeader05Text() As String
    Return "SteelBlue"
End Function

Public Shared Function RowHeader05Font() As String
    Return "Arial"
End Function

Public Shared Function RowHeader05FontSize() As String
    Return "8pt"
End Function

Public Shared Function HighDensityPercentageFont() As String
    Return "Arial Narrow"
End Function
```

```
Public Shared Function HighDensityPercentageFontSize() As String
    Return "8pt"
End Function

Public Shared Function ChartAreaBackColor1() As String
    Return "White"
End Function

Public Shared Function ChartAreaBackColor2() As String
    Return "White"
End Function

Public Shared Function ChartAreaBackGradient() As String
    Return "Center"
End Function

Public Shared Function ChartAreaBorderStyle() As String
    Return "Dotted"
End Function

Public Shared Function ChartAreaBorderHatching() As String
    Return "None"
End Function

Public Shared Function ChartAreaBorderColor() As String
    Return "SteelBlue"
End Function

Public Shared Function ChartAreaBorderLineStyle() As String
    Return "Dotted"
End Function

Public Shared Function ChartAreaBorderLineColor() As String
    Return "SteelBlue"
End Function

Public Shared Function ChartAreaBorderLineThickness() As String
    Return "1pt"
End Function

Public Shared Function ChartTitleFont() As String
    Return "Arial"
End Function

Public Shared Function ChartTitleSize() As String
    Return "16pt"
End Function

Public Shared Function ChartTitleColor() As String
    Return "SteelBlue"
End Function

Public Shared Function ChartTitleStyle() As String
    Return "Normal"
End Function

Public Shared Function ChartTitleWeight() As String
    Return "Normal"
End Function

Public Shared Function ChartTitleDecoration() As String
    Return "Underline"
```

```
End Function

Public Shared Function ChartTitleColor1() As String
    Return "White"
End Function

Public Shared Function ChartTitleColor2() As String
    Return "White"
End Function

Public Shared Function ChartTitleGradient() As String
    Return "Center"
End Function

Public Shared Function ChartTitleHatching() As String
    Return "None"
End Function

Public Shared Function ChartTitleLineColor() As String
    Return "White"
End Function

Public Shared Function ChartTitleLineStyle() As String
    Return "Solid"
End Function

Public Shared Function ChartTitleLineThickness() As String
    Return "1pt"
End Function

Public Shared Function ChartPictureApprnceColor1() As String
    Return "White"
End Function

Public Shared Function ChartPictureApprnceColor2() As String
    Return "White"
End Function

Public Shared Function ChartPictureApprnceGradient() As String
    Return "Center"
End Function
Public Shared Function ChartPictureApprnceHatching() As String
    Return "None"
End Function

Public Shared Function ChartPictureApprnceLineColor() As String
    Return "Transparent"
End Function

Public Shared Function ChartPictureApprnceLineStyle() As String
    Return "Solid"
End Function

Public Shared Function ChartPictureApprnceLineThickness() As String
    Return "1pt"
End Function

Public Shared Function ChartBorderApprnceColor1() As String
    Return "White"
End Function

Public Shared Function ChartBorderApprnceColor2() As String
    Return "White"
```

```
End Function

Public Shared Function ChartBorderApprnceGradient() As String
    Return "Center"
End Function

Public Shared Function ChartBorderApprnceHatching() As String
    Return "None"
End Function

Public Shared Function ChartBorderApprnceLineColor() As String
    Return "SteelBlue"
End Function

Public Shared Function ChartBorderApprnceLineStyle() As String
    Return "Solid"
End Function

Public Shared Function ChartBorderApprnceLineThickness() As String
    Return "1pt"
End Function

Public Shared Function ChartDataPointApprnceColor1() As String
    Return "White"
End Function

Public Shared Function ChartDataPointApprnceColor2() As String
    Return "White"
End Function

Public Shared Function ChartDataPointApprnceGradient() As String
    Return "Center"
End Function

Public Shared Function ChartDataPointApprnceHatching() As String
    Return "None"
End Function

Public Shared Function ChartDataPointApprnceLineColor() As String
    Return "SteelBlue"
End Function

Public Shared Function ChartPlanLineColor() As String
    Return "SkyBlue"
End Function

Public Shared Function ChartActualLineColor() As String
    Return "Gold"
End Function

Public Shared Function ChartDataLineApprnceLineThickness() As String
    Return "1pt"
End Function

Public Shared Function ChartDataPointApprnceLineStyle() As String
    Return "Solid"
End Function

Public Shared Function ChartDataPointApprnceLineThickness() As String
    Return "1pt"
End Function

Public Shared Function ChartDataPointLabelFontAngle() As String
```

```
    Return "0"
End Function

Public Shared Function ChartDataPointLabelBackColor() As String
    Return "White"
End Function

Public Shared Function ChartDataPointLabelBorderColor() As String
    Return "Black"
End Function

Public Shared Function ChartDataPointLabelBorderStyle() As String
    Return "Solid"
End Function

Public Shared Function ChartDataPointLabelBorderWidth() As String
    Return "1pt"
End Function

Public Shared Function ChartDataPointMarkerStyle() As String
    Return "Square"
End Function

Public Shared Function ChartDataPointMarkerSize() As String
    Return "5"
End Function

Public Shared Function ChartDataPointMarkerColor() As String
    Return "White"
End Function

Public Shared Function ChartDataPointMarkerBorderColor() As String
    Return "Black"
End Function

Public Shared Function ChartDataPointLabelMarkerFont() As String
    Return "Ariel"
End Function

Public Shared Function ChartDataPointLabelMarkerSize() As String
    Return "8pt"
End Function

Public Shared Function ChartDataPointLabelMarkerColor() As String
    Return "White"
End Function

Public Shared Function ChartDataPointLabelMarkerStyle() As String
    Return "Regular"
End Function

Public Shared Function SparkLinePlanLineColor() As String
    Return "SkyBlue"
End Function

Public Shared Function SparkLineActualLineColor() As String
    Return "Black"
End Function

Public Shared Function SparkLineDataPointApprnceGradient() As String
    Return "Center"
End Function
```

```
Public Shared Function SparkLineDataPointApprnceHatching() As String
    Return "None"
End Function

Public Shared Function SparkLineDataPointApprnceLineStyle() As String
    Return "Solid"
End Function

Public Shared Function SparkLineDataPointApprnceLineThickness() As String
    Return "2pt"
End Function

Public Shared Function BulletChartPointer1Color() As String
    Return "Gold"
End Function

Public Shared Function BulletChartPointer1Gradient() As String
    Return "Center"
End Function

Public Shared Function BulletChartPointer1Hatching() As String
    Return "None"
End Function

Public Shared Function XAxisTitleFont() As String
    Return "Arial"
End Function

Public Shared Function XAxisTitleSize() As String
    Return "12pt"
End Function

Public Shared Function XAxisTitleColor() As String
    Return "SteelBlue"
End Function

Public Shared Function XAxisTitleStyle() As String
    Return "Normal"
End Function

Public Shared Function XAxisTitleWeight() As String
    Return "Normal"
End Function

Public Shared Function XAxisTitleDecoration() As String
    Return "Underline"
End Function

Public Shared Function XAxisLableFont() As String
    Return "Arial"
End Function

Public Shared Function XAxisLableSize() As String
    Return "12pt"
End Function

Public Shared Function XAxisLableColor() As String
    Return "SteelBlue"
End Function

Public Shared Function XAxisLableStyle() As String
    Return "Normal"
End Function
```

```
Public Shared Function XAxisLableWeight() As String
    Return "Normal"
End Function

Public Shared Function XAxisLableDecoration() As String
    Return "Underline"
End Function

Public Shared Function XAxisMajorGridLineColor() As String
    Return "Black"
End Function

Public Shared Function XAxisMajorGridLineStyle() As String
    Return "Solid"
End Function

Public Shared Function XAxisMajorGridLineThickness() As String
    Return "1pt"
End Function

Public Shared Function XAxisMinorGridLineColor() As String
    Return "Silver"
End Function

Public Shared Function XAxisMinorGridLineStyle() As String
    Return "Solid"
End Function

Public Shared Function XAxisMinorGridLineThickness() As String
    Return "1pt"
End Function

Public Shared Function YAxisTitleFont() As String
    Return "Arial"
End Function

Public Shared Function YAxisTitleSize() As String
    Return "12pt"
End Function

Public Shared Function YAxisTitleColor() As String
    Return "SteelBlue"
End Function

Public Shared Function YAxisTitleStyle() As String
    Return "Normal"
End Function

Public Shared Function YAxisTitleWeight() As String
    Return "Normal"
End Function

Public Shared Function YAxisTitleDecoration() As String
    Return "Underline"
End Function

Public Shared Function YAxisLableFont() As String
    Return "Arial"
End Function

Public Shared Function YAxisLableSize() As String
    Return "12pt"
```

```
End Function

Public Shared Function YAxisLableColor() As String
    Return "SteelBlue"
End Function

Public Shared Function YAxisLableStyle() As String
    Return "Normal"
End Function

Public Shared Function YAxisLableWeight() As String
    Return "Normal"
End Function

Public Shared Function YAxisLableDecoration() As String
    Return "Underline"
End Function

Public Shared Function YAxisMajorGridLineColor() As String
    Return "Navy"
End Function

Public Shared Function YAxisMajorGridLineStyle() As String
    Return "Solid"
End Function

Public Shared Function YAxisMajorGridLineThickness() As String
    Return "1pt"
End Function

Public Shared Function YAxisMinorGridLineColor() As String
    Return "SkyBlue"
End Function

Public Shared Function YAxisMinorGridLineStyle() As String
    Return "Solid"
End Function

Public Shared Function YAxisMinorGridLineThickness() As String
    Return "1pt"
End Function

Public Shared Function ChartLegendColor1() As String
    Return "White"
End Function

Public Shared Function ChartLegendColor2() As String
    Return "White"
End Function

Public Shared Function ChartLegendGradient() As String
    Return "Center"
End Function

Public Shared Function ChartLegendHatching() As String
    Return "None"
End Function

Public Shared Function ChartLegendBorderLineColor() As String
    Return "Black"
End Function

Public Shared Function ChartLegendBorderLineStyle() As String
```

```
    Return "Solid"
End Function

Public Shared Function ChartLegendBorderLineThickness() As String
    Return "1pt"
End Function

Public Shared Function ChartLegendFont() As String
    Return "Arial"
End Function

Public Shared Function ChartLegendFontSize() As String
    Return "8pt"
End Function

Public Shared Function ChartLegendFontColor() As String
    Return "Black"
End Function

Public Shared Function ChartLegendFontStyle() As String
    Return "Normal"
End Function

Public Shared Function ChartDataFont() As String
    Return "Arial"
End Function

Public Shared Function ChartDataSize() As String
    Return "8pt"
End Function

Public Shared Function ChartDataStyle() As String
    Return "Normal"
End Function

Public Shared Function ChartDataWeight() As String
    Return "Normal"
End Function

Public Shared Function ChartDataColor() As String
    Return "Black"
End Function

Public Shared Function ChartDataDecoration() As String
    Return "Underline"
End Function

Public Shared Function PlotAreaBackColor1() As String
    Return "White"
End Function

Public Shared Function PlotAreaBackColor2() As String
    Return "White"
End Function

Public Shared Function PlotAreaBackGradient() As String
    Return "Center"
End Function

Public Shared Function PlotAreaBorderStyle() As String
    Return "Dotted"
End Function
```

```
Public Shared Function PlotAreaBorderWidth() As String
    Return "1pt"

End Function
Public Shared Function PlotAreaBorderColor() As String
    Return "Black"
End Function

Public Shared Function RadarPlotAreaSurfaceColor() As String
    Return "115, 255, 215, 0"
End Function

Public Shared Function RadarPlotAreaSurfaceColorPlan() As String
    Return "112, 0, 255, 255"
End Function

Public Shared Function RadarPlotAreaLegendColorPlan() As String
    Return "SkyBlue"
End Function

Public Shared Function RadarPlotAreaLineColorPlan() As String
    Return "Navy"
End Function

Public Shared Function RadarPlotAreaSurfaceColorActual() As String
    Return "115, 255, 215, 0"
End Function

Public Shared Function RadarPlotAreaLegendColorActual() As String
    Return "Gold"
End Function

Public Shared Function RadarPlotAreaLineColorActual() As String
    Return "Goldenrod"
End Function
```

10.9. Reports By Text

This function allows the reports to be credited to the client or to anyone else for that matter. This text appears on the footer of all BI4ALL Reports.

```
Public Shared Function ReportsByText() As String
    Return "Reports by Instant Business Intelligence"
End Function
```