

D1 Solutions White Paper

# BO Universe Design Best Practices



**Abstract** According to the book of Genesis, the universe was created in seven days.

The following day-by-day analogy provides a best practices guide for the creation of an SAP Business Objects Universe.

Seven days actually might just be enough.

## The First Day - „Let There Be Light“

Understanding the business requirements is essential for any successful implementation. “Let there be light” stands for understanding the success factors and constraints in a project:

- What type of reporting will be done?
- What type of usage can be expected?
- What are the security requirements?
- What are the processes in this organization?

### What type of reporting will be done?

Each type of reporting, like financial reporting, ad hoc analysis, dashboards, data mining, etc., requires specific data modelling.

Before starting with the universe design, it is advisable to transform the data into a model which suits the type of reporting. Starting with a good data model will enable the developer to create better universes. For example, multidimensional cubes are ideal for financial reporting because there is a need for aggregation and navigation along unbalanced hierarchies (e.g. account hierarchies).

On the other hand, a denormalized datamart might be preferred for producing standard KPI reporting. Then again, a normalized data model close to the model of the source application is best suited for analytical querying and report prototyping.

### What type of usage can be expected?

Will the universe be used in a basic rather than an advanced way? Basic usage is about standard reports and key figures. The main concern for basic usage is that the reports as well as the underlying universe are correct, well understood, and that the data are available as soon as possible. As a consequence the universe development should focus on these goals. On the other hand, advanced usage is about new insights that can be derived from the data. A universe supporting advanced usage should enable prototyping and in depth analysis. The focus of development should lie on extending the data model in order to support as many requests as possible.

Frequently, an unjustified bias in favor of advanced universe design can be observed. This is an easy trap to fall into because the person in charge of creating the universe naturally uses it like a prototype at the start of the project.

### For basic usage the following design choices should be considered:

- Less is more: The number of objects available in the universe should be limited to those that the end user really needs. In practice however, the developer is often faced with the contradictory requirements of both basic and advanced usage. A good way to handle this situation is using security access levels: Advanced users will be able to use all objects of the universe while users with basic needs just get displayed a subset of objects. If security access levels are no option, then consider creating a linked universe on the full scope master universe. The object hiding technique on that secondary linked universe can then be used in order to reduce the scope and complexity of an advanced universe.
- One truth: A universe intended for basic usage should refrain from offering several “solutions” to the same business request. For example, dimension attributes should be provided either in current or historical truth. Providing both current and historical truth for the same dimension attribute will only confuse a user with basic needs. Furthermore, for the sake of simplicity, do not provide several definitions of a given KPI.
- Self-explanatory: Focus on providing good and clear business names and descriptions. The naming will be most easily understood by business users if it is adopted from the source application front end. On the other hand, IT users might prefer the original column names to avoid confusion when mapping columns from the source tables to the reporting database.
- Eliminate report logic: Variables like weighted averages are normally created on report level, because their value is weighted depending on the grouping context of a table/ graph within the report. However, creating variables in a report requires skill, time and a clear understanding of the exact definition. The universe measure property “database delegated” allows for the creation of such variables at universe level. The drawback of “database delegated” objects is that every change in the report design will require a query refresh. However, the better reliability will pay off in the long run. Consider also using analytical SQL functions in your queries in order to provide measures that will be independent of the query and the report grouping context (e.g. team average).

### What are the security requirements?

Requirements regarding data access security can have a decisive impact on the development cost of a universe, and consequently of the whole reporting solution. Requirements can range from no security restrictions up to very complex security models based on organizational charts. Consider the following options when deciding on an implementation:

- Row level security: The BO user credentials are linked to data on row level. For such an option to work use a bridge table managing the row level access on database level. Such a table can often be derived from the source data itself, e.g. if it contains a table with the correct organizational hierarchy. In other cases, it has to be maintained manually.
- Folder security: The built-in BO security relies on folders, which pass on their rights to the underlying objects like reports and universes. This means that access is either granted to all of the underlying data or to none of them. Access to certain universe objects like dimensions and measures can be further restricted.
- Encryption: Sometimes there is a need for encryption of delicate & sensitive objects (e.g. customer names).
- Advanced Security: When security requirements are based on hierarchy levels, consider building a universe on top of the security mechanism built into an OLAP cube. Most OLAP cubes can be accessed by Business Objects. The cube will independently manage security based on the BO user id.

### What are the processes in this organization ?

In many organizations there is an inherent conflict between IT operations and the business triggering new report requirements. The goal of IT operations is to stabilize the environment in order to guarantee service availability. Therefore, IT operations has no interest in quick and dirty solutions, and IT departments tend to enforce processes that slow down or even prevent changes.

On the other hand, business people have little patience with cumbersome and slow processes. In many organizations, the BO universe is used to work around this conflict. BO universes are often owned by the business. IT operations does not take any responsibility, and there is no risk that the operational processes will be affected by changes in the universe.

The drawback is that things that should have been built into the database itself end up as somewhat volatile constructs in the universe. A favorite tool in this context is the “derived table” functionality which enables the business users to

create something similar to database views, but – of course – without creating real objects in the database. This approach cannot be recommended as best practice due to tool dependency and due to lack of reusability. It is always preferable to implement business logic on a database level, where the same values are visible to all tools and users.

### The Second Day - “The Separation of Waters”

The time has come to structure & organize the data chaos. The data have to be analyzed and then separated into different business areas. For each of these business areas a separate universe should be created. The question remains: how should these business areas be grouped? A “one universe fits all” strategy might become too complex and difficult to use. On the other hand, creating a universe for each single business query leads to low reusability and entails the risk of inconsistency. The following rules of thumb apply when deciding on what data should be grouped:

- Business questions, which use the same or similar reporting dimensions, should be grouped into the same business area.
- Data that is strongly related to each other should be grouped into the same business area. Try not to use the same fact table in more than one universe. However it is no problem to use several different fact tables within one universe. Use contexts in order to control correct querying.
- Aggregated fact tables and the underlying raw fact table should be made available in the same universe. Implement such a universe with aggregate aware functionality in order to get the best performance and consistency.
- The number of different contexts should be limited to about seven. Combining too many fact tables within the same universe leads to unmanageable universes with too many contexts to maintain.

Certain dimensions are used repeatedly in several universes (e.g. time, employee). Sometimes these dimensions are a common denominator when merging data from different universes in the same report. For reports combining data from different universes consider creating a super-universe by linking the underlying universes to it.

Unnecessary dimensions, which are specific to the underlying universes, can be hidden in the super-universe in order to reduce complexity and avoid inconsistent queries. The universe should only be used for KPI's and dimensions that actually match. Therefore, the universe contexts can be kept simple because only the links to the common denominators have to be maintained.

**The Third Day - “Solid Ground**

You are not on solid ground unless your results are reliable. It is important that any query on a universe should always return correct results, at least in a technical sense. This is probably the most important guideline for universe design. Before going live, extensive testing is necessary to ensure correct configuration of all contexts and to weed out any incorrect joins. If too much logic is built into the universe, testing becomes intractable. It is always a good idea to consider ways of reducing universe complexity. Business logic can be implemented anywhere along the data chain from the source application up to the report and query. The best place to put it largely depends on budget restrictions, as well as on the level of credibility and reusability that is required.

implementation level/needs & restrictions	IMPLEMENTATION COSTS		CREDIBILITY		MAINTENANCE COSTS		REUSABILITY	
SOURCE APPLICATION	--	++	--	++	--	++	--	++
DWH TABLE	-	+	-	+	-	+	-	+
DWH VIEW	+	~	+	~	+	~	+	~
UNIVERSE	+	~	+	~	+	~	+	~
REPORT/QUERY	++	-	-	+	-	+	-	+

Very often, logic is implemented on a query and report level. This is quite tempting because of the low initial cost, and because of the fact that the project starts out as a prototype. Yet keeping logic on a report level for a longer period can lead to high maintenance costs and error-prone reports. In most reporting environments there is the need to move business logic further up the data chain rather than down to the report. For example, if we know that employee classifications are going to be reused in several universes, such classifications should be implemented as a database view. The employee information will then be rolled out to all universes automatically. Especially people who prefer running SQL queries directly on the DWH, without using a BO universe, will be grateful for any business logic (e.g. grouping) that is provided for them on a DWH level.

There are times when it is best to not just rely on a general purpose reporting system. Logic implemented and/or displayed within the source application is often perceived as more credible than data on some report. If you want to make sure your numbers are used, tested, and finally trusted, it might be a good choice to display them in the source system. For example, there could be information displayed on the CRM application screen showing whether a customer is eligible for discounts. Even though the data quality of such information might be bad, it is normally more acceptable and certainly more accessible than the same information within a report. Also, any report information will be perceived as more

credible if the same information is visible in the source application. Consistency is sometimes valued more highly than correct data interpretation. This is a business choice often observed and also might make sense for the sake of comparability. Implementations in the source application will avoid discussions about data interpretation by giving the business the responsibility and data ownership for important KPI's and classifications. For example, it might make sense to have the product hierarchy and catalogue maintained within the CRM system instead of maintaining a product hierarchy within reporting.

Some information is so generally useful that it should never be fragmented across various source systems or multiple reports. The natural place for such data is the DWH and then provided as shared dimension universe. Examples include:

- Generic date table/time universe that contains all possible date objects, hierarchies, time filters and groupings (e.g. Year To Date)
- Organization (Employee) hierarchy view / universe
- Mapping table(s) for grouping of values, e.g. products, etc. If it is difficult to create tables for the business then one mapping table that fits all dimensions might be a good choice, otherwise a mapping table per object/dimension type is better since security can be applied directly and the purpose is clear.

**The Fourth Day - “Sun, Moon and Stars”**

If the reporting system were to be the Earth - what would the Sun be? The source application (e.g. CRM System) actually fits quite well into such an analogy. The sun enables life on Earth just like the source application provides the data to be analyzed to a reporting system. Making use of the source application during development is very useful for various reasons:

- It helps to better understand the processes and the inconsistencies of a system.
- Business terms are often linked to the wording used in source applications. Adopting the business terms from the source application is often a good way to go because it is best understood by the users.
- Reviewing data on a detailed level. Often it is faster to review detailed data in the source application, since such systems are designed for the detailed case.

It is important to get access to source applications at an early stage of the project. Getting some training or actually working with these systems for a few days would also be a good idea.

**Moons & Stars:**

Additional tools and helpers from the world of SAP BO partners may brighten up the nightly reporting sky. Needs for automation can be implemented using the BO SDK library. Many SAP BO partners have implemented specific solutions to automation problems. The following tools could be of interest:

- A tool that provides an overview of dependencies between reports and universes.  
Reference: D1 Solutions DBOX
- A tool that automates report documentation by looking at the universe object descriptions.  
Reference: D1 Solutions DBOX
- A tool for maintenance of access levels, users and rights.  
Reference: D1 Solutions DBOX
- A tool for maintenance of hierarchies & navigation trees to be used in the universe.
- BO Performance Monitoring & other helpers:  
<http://www.apos.com>

Another important resource for all BO developers is the BOB forum at <http://www.forumtopics.com/busobj> which will provide you with helpers and additional tips and tricks as you go about building your universe.

**The Fifth Day - "Birds & Sea Creatures"**

A very useful technique when designing universes for a larger audience is using the hide object functionality. We can divide the objects to be created into birds and sea creatures. Sea creatures are not visible; they are below the surface and will only be available to the universe developer. You can create as many and as strange objects you want. If they turn out to be dinosaurs, let them go extinct. Follow the following guidelines:

- Never build class hierarchies with more than two levels.
- Always pre-format your date and number objects.
- Always name your objects with the class name and then the object name. There should not be two objects with the same name in one universe.
- Order the objects within the class in a consistent way. This could be as follows:
  1. Physical primary key
  2. Logical primary key
  3. Referential Dimensions  
(if less than 5 objects use same class, else add to class)
  4. Date Objects (Top Down)
  5. Other Dimensions (Top Down) – Alphabetical order
  6. Measure objects – Alphabetical order

7. Unused objects hidden at the bottom
- Replace null values caused by left outer joins with the string 'N/A'. Depending on your DB system, the command to do this is COALESCE(), ISNULL(), or NVL().

**The Sixth Day - "Land Animals"**

We are now about to turn the birds and sea creatures into land animals. A bit like the turn from fish to frog, we turn the combination of a dimension with a basic measure into a pivoted measure (land animal).

Let us assume you have the following employee table which gives you an overview about how many FTE's were employed during a certain time period per department and employee type:

**EMPLOYEE TABLE :**

TIME	DEPARTMENT	EMPTYTYPE	FTE
2011/01	MARKETING	INTERNAL	30
2011/01	MARKETING	EXTERNAL	2
2011/01	SALES FORCE	INTERNAL	20
2011/01	SALES FORCE	EXTERNAL	90
2011/02	MARKETING	INTERNAL	35
2011/02	SALES FORCE	INTERNAL	20
2011/02	SALES FORCE	EXTERNAL	90
2011/03	MARKET&SALES	INTERNAL	45
2011/03	MARKET&SALES	EXTERNAL	90
2011/03	COMMUNICATIONS	INTERNAL	10

With the above data input it can be a tedious task to calculate the share of internal FTE's versus the total number of FTE's within the report itself. In this case it is better to provide pivoted objects by constraining the FTE aggregation to the respective employee type value.

The following syntax would be used to create the new measure object "Internal FTE".

```
-- "INTERNAL FTE"
SUM(CASE
    WHEN EMPTYTYPE='INTERNAL'
    THEN FTE
    ELSE 0
END) -- AS "INTERNAL FTE"
```

1. Note that the "AS" part has to be commented out or left away. Else the query will raise an error when the new measure object is used in a where clause.
2. Note that formatting and commenting the SQL will be greatly appreciated by anyone using/analyzing the SQL
3. Note that "ELSE 0" can be optionally used in order to enforce a zero return value. In most cases this behavior is preferred to NULL value returns. An advantage of the zero value return is that no special functions have to be applied when summing up the pivoted measures (remember that NULL + 234 returns NULL and not 234)

Pivoted measures are often the better choice with regard to usability. The user will not need to build such objects within the report. Also, such pivoted measures might actually be required as an input when working with certain BI tools (e.g. Xcelsius).

A positive side effect of creating pivoted values is that they can be named in a way that is more understandable to business users because it is based on something real they already know. In our example, EMPTYTYPE is more or less meaningless while "Internal FTE" is very clear.

However, each additional pivoted object in the universe entails some maintenance overhead, because its definition will have to change if the possible values of the underlying object change.

Pivoted measures should be created for objects that will be used constantly and which are expected to have a long life. In the EMPLOYEE example we would avoid creating a pivoted measure for each of the departments since they are numerous as well as subject to change every once in a while.

Another type of pivoted object is the use of MIN and MAX. Often the business user does not trust aggregated data. This is especially true when it looks erroneous or illogical. The best way of dispelling distrust in data quality (stemming from wrong queries) is to provide all the detail records used for the aggregation and KPI calculation.

The user will then be able to drill down to these records and understand why the data is the way it is. If there is too much effort involved in providing such functionality, it is a very convenient shortcut to provide MAX and MIN objects of the underlying detail record. The query will still be performing well and return a reduced set of data.

Nevertheless, it will also provide the end user with two examples for each aggregation. Even better examples can be provided by using analytical functions like first\_value() with a well chosen argument. In the employee example such a min or max object would be the maximum and minimum employee number. Or you could choose some average guy like David as an example.

### The Seventh Day - "Rest & Enjoy"

- Rest & Enjoy ■



#### Author

Candid Rutz graduated with a degree in economics from the University of Lausanne (HEC) in 2000. His areas are customer care, finance, controlling and process performance management. He is managing projects in the manufacturing, financial and telecom industries. Candid Rutz has been with D1 Solutions since 2007.

#### Your contact for further information

Simon Hefti, Chairman

simon.hefti@d1solutions.ch, P +41 (0)44 435 10 10