

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Nino Ostrc

**Performančni modul za platformo
BI4Dynamics**

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Luka Šajn

Ljubljana 2015

UNIVERSITY OF LJUBLJANA, SLOVENIA
FACULTY OF COMPUTER AND INFORMATION SCIENCE

Nino Ostrc

**Performance module for
BI4Dynamics platform**

BACHELOR'S THESIS

UNDERGRADUATE UNIVERSITY STUDY PROGRAMME
COMPUTER AND INFORMATION SCIENCE

SUPERVISOR: Assist. Prof. Luka Šajn, PhD

Ljubljana 2015

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V okviru diplomske naloge izdelajte načrt za performančno orodje za analitično platformo BI4Dynamics. Za potrebe načrta najprej analizirajte in opišite obstoječo arhitekturo in razmislite, kako bi jo izkoristili in orodje realizirali kot modul. Posebno pozornost namenite funkcionalnosti za zajem statistik in zagotavljanju neodvisnosti od ostalih poslovnih podatkov.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Nino Ostrc, z vpisno številko **63060164**, sem avtor diplomskega dela z naslovom:

Performančni modul za platformo BI4Dynamics

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Luke Šajna,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 13. januarja 2015

Podpis avtorja:

I would like to thank my supervisor Assist. Prof. Luka Sajn, PhD, for all the help and directions in writing this thesis. Also special thanks go out to my co-workers at BI4Dynamics d.o.o. who shared with me all the trials and tribulations in developing this project.

Most important of all, of course, my mother - the best mom in the world. Without you there would be no me.

Your dreams came true. Finally.

Contents

Povzetek

Abstract

1	Introduction	1
2	Businesses Intelligence	3
2.1	Lessons Of History	5
2.2	Definition	8
3	BI4Dynamics	11
3.1	BI4Dynamics NAV	12
3.2	Data Model	13
3.3	Interface	31
4	BI4Profiler	35
4.1	Module	35
4.2	Development	38
4.3	Putting it all together	44
4.4	Filling the void	47
5	Conclusion	51
	List of Figures	52
	List of Tables	53

List of abbreviations

1. BI: Business Intelligence
2. ERP: Enterprise Resource Planning
3. DBMS: Database management systems
4. .NET: Software framework developed by Microsoft, which runs primarily on Microsoft Windows
5. AMO: Analysis Management Object
6. ETL: Extract, Transform, Load
7. OLAP: Online Analytical Processing
8. SQL: Structured Query Language
9. XML: Extensible Markup Language
10. UI: User Interface
11. NAV: Navision
12. AX: Axapta
13. DMV: Dynamic Management Views
14. DMO: Database Management Objects
15. XAML: Extensible Application Markup Language

Povzetek

BI4Dynamics je analitična BI-rešitev, ki nadgradi transakcijske sisteme Microsoft Dynamics ERP s sposobnostjo kompleksnih analiz podatkov. V podjetju BI4Dynamics smo razvili dve produktni liniji platforme BI4Dynamics. *BI4Dynamics NAV* za transakcijske sisteme Microsoft Navision ter *BI4Dynamics AX* za transakcijske sisteme Microsoft Axapata. Ker je podatkovni model obeh produktov v samem jedru enak, razen manjših strukturnih specifičnosti, in ker je bil performančni modul BI4Profiler razvit za sisteme *BI4Dynamics NAV*, se v nadaljevanju osredotočamo le na te sisteme.

Platforma BI4Dynamics je zapakirana kot .NET-aplikacija. Poleg veliko dodatnih funkcionalnosti, ki jih tukaj ne omenjamo, je njen najpomembnejši del namestitveni čarovnik. Ta v šestih preprostih korakih, preko uporabniške interakcije, zgradi podatkovno skladišče in analitično bazo.

Podatkovno skladišče BI4Dynamics, ki se namesti na Microsoft SQL-strežnik, je zgrajeno po Kimballovmem principu zvezdne sheme [8] in predstavlja podatkovni vir za nabor OLAP-kock v analitični bazi. Razdeljeno je na področje priprave podatkov in prezentacijski nivo. Na področju priprave podatkov se kreirajo ekvivalenti tabel Navision, v katere se z ETL-paketi prečrpajo prečiščeni podatki iz podatkovnega vira. ETL-procesi nadalje skrbijo za transformacijo teh podatkov v zvezdno shemo na prezentacijskem nivoju [8]. Ta je sestavljen iz tabel dejstev in dimenzijskih tabel. Tabela dejstev predstavlja meritve nekega poslovnega področja (recimo prodaje), dimenzijske tabele pa predstavljajo tekstovne deskriptorje tega poslovnega področja (recimo artiklov). Transformacijo ETL-procesi izvedejo s transfor-

macijskim pogledom in polnitveno proceduro. Transformacijski pogled je najpomembnejši del te platforme, saj se tu nahaja vsa poslovna logika.

Med podatki prezentacijskega nivoja in poslovnimi uporabniki pa se nahaja še en nivo – analitična baza. Ta poslovnim uporabnikom še dodatno poenostavi in pohitri analizo podatkov, ker uporablja odlično kompresijo podatkov in predpripravljene agregate podatkov po različnih nivojih dimenzij. Osnovo analitične baze predstavljajo kocke, ki so množice sorodnih meritev in dimenzij. Kocka je ekvivalent paketu v aplikaciji, ki predstavlja eno poslovno področje (npr.: prodajo). Dimenzija pa je skupina atributov, ki predstavljajo interesno področje pri analizi sorodnih meritev znotraj kocke. Kocka torej črpa podatke iz prezentacijskega nivoja podatkovnega skladišča, s tem da združi sorodne tabele dejstev oz. meritev. Rešitev BI4Dynamics vsebuje 14 kock, ki pokrivajo tako standardna kot nestandardna poslovna področja.

Ko se namesti podatkovno skladišče in analitična baza in se podatki v njih tudi obdelajo, se lahko uporabnik poveže z analitičnim strežnikom s katerim koli BI-kompatibilnim uporabniškim orodjem. Najbolj pogosto uporabljeno orodje je Excel, za katerega nudimo že predpripravljena poročila za vsa standardna področja analize podatkov.

Vendar prava moč rešitve BI4Dynamics leži v njeni platformni zasnovi. Ker v celoti uporablja Microsoftov arhitekturni sklad, ki temelji na analitični in SQL-tehnologiji, v polni meri omogoča spreminjanje katere koli od komponent modula. Modul v našem primeru pomeni osnovni gradnik platforme BI4Dynamics. Modul je lahko dimenzija ali kocka in vse pripadajoče odvisne komponente v podatkovnem skladišču ter viru. Modul torej vsebuje analitični objekt (dimenzijo ali kocko), ETL-procedure (transformacijski pogled in polnitveno proceduro) ter seznam stolpcev in tabel na podatkovnem viru Navision, na osnovi katerih se zgradi področje priprave podatkov.

Prva verzija rešitve BI4Dynamics je vsebovala šest kock, kot del specifičnih projektov za stranke pa smo razvili tudi druge kocke in pripadajoče dimenzije, ki smo jih kasneje zaradi frekventnega povpraševanja po njih in

pogoste uporabe integrirali kot del standarda. Vsakič, ko smo implementirali nove skripte, še posebej v fazi optimizacije, so se porajala vprašanja, kako se ti skripti obnesejo, če bi jih lahko pohitrili, kje je ozko grlo in koliko prostora porabijo na strežniku. Če izvajamo meritve ročno brez avtomatike, je to lahko mučen in dolgotrajen proces. Sprva smo hoteli razviti preprosto performančno orodje za interno uporabo. Nato smo se odločili, da bi lahko izkoristili obstoječo platformo in to orodje zapakirali kot modul ter ga ponudili tudi našim partnerjem, ki so jih mučili podobni problemi, in nazadnje tudi končnim strankam, ki bodo lahko analizirale te podatke na isti način, kot bi katerokoli drugo poslovno področje.

BI4Profiler je torej performančni modul, ki pa poleg osnovne funkcionalnosti modula, se pravi svoje OLAP-kocke, nabora dimenzij ter pripadajoče komponente v podatkovnem skladišču, vsebuje še dodatno funkcionalnost za zajem statistik, ki smo jo morali implementirati v ogrodje BI4Dynamics. Moja naloga v tem procesu je bila načrtovanje arhitekture te funkcionalnosti in nadzor razvoja modula. Ker smo hoteli zagotoviti neodvisnost in obstojnost teh statističnih podatkov tudi v primeru, ko uporabnik na novo namesti podatkovno skladišče, smo za modul uvedli novo ločeno podatkovno skladišče in analitično bazo, ki pa se namestita samo ob prvi namestitvi originalnega podatkovnega skladišča in se nato ne spreminjata. Podatki v njih pa se obdelajo po tem, ko se obdelajo vsi podatki v originalnem podatkovnem skladišču in analitični bazi.

Podatkovno skladišče tega modula prav tako vsebuje področje priprave podatkov in prezentacijski nivo. Področje priprave podatkov predstavljata samo dve tabeli: glavna tabela s statistikami obdelav in tabela paketnih obdelav. Pred obdelavo originalnih podatkov se v tabelo paketnih obdelav doda zapis, ki predstavlja trenutno obdelavo s časovnim žigom. Med obdelavo se v tabelo s statistikami beležijo statistike za vsako od polnitvenih procedur, ki se izvaja v tej obdelavi. Ko se obdelava konča, se pripadajoči vrstici v tabeli paketnih obdelav doda časovni žig za konec obdelave in vsi zapisi v tabeli statistik se označijo, da pripadajo tej paketni obdelavi. Nato

se ti podatki transformirajo ter prenesejo v prezentacijski nivo in na koncu se tabela statistik sprazni. Nekaterne glavne statistike so: *ime objekta oz. polnitvene procedure, začetni časovni žig, končni časovni žig, neto čas izvajanja* ter *absolutni čas izvajanja oz. obračalni čas*.

Modul vsebuje samo eno OLAP-kocko s petimi meritvenimi skupami (kar pomeni pet tabel dejstev v prezentacijskem nivoju): *statistike obdelav, statistike tabel, statistike indeksov, statistike baze oz. podatkovnega skladišča* ter *statistike trdih diskov*. Prav tako vsebuje sledeče dimenzije (seveda z pripadajočimi dimenzijskimi tabelami v prezentacijskem nivoju): *paketna obdelava, datum, baza, datoteke na bazi, indeksi, procedure, tabele, tip obdelave, področje obdelave* ter *načrt obdelave*.

Vse tabele dejstev razen *statistika obdelav* se napolnijo iz trenutnega stanja baze v procesu transformacije. Ta pa se napolni iz ekvivalente tabele s področju priprave podatkov, ki pa podatke dobi med samo obdelavo, za kar skrbi funkcionalnost za zajem podatkov. Ta funkcionalnost je realizirana v obliki ločene niti, ki se zažene pred klicem originalne polnitvene procedure modula, ki se trenutno obdeluje, in se uniči, ko se konča izvajanje te procedure. Nit v tem času pošilja poizvedbe na SQL-strežnik, na katerem se nahaja originalno podatkovno skladišče, glede statistike izvajanja trenutne procedure, kot je koliko RAM-a ima trenutno na voljo. Preden se nit uniči, torej po tem, ko se je originalna polnitvena procedura izvedla, nit pošlje še zadnjo poizvedbo na SQL-strežnik, s katero dobi še vse ostale statistike, ki jih potem zapiše v tabelo statistik obdelav. To doseže z uporabo sistemskih pogledov ali DMV-jev (Dynamics Management Views), ki jih ponuja Microsoftov SQL-strežnik. Prvi, preko katerega izve za porabo spomina, je *sys.dm_exec_query_memory_grants*, drugi pa *sys.dm_exec_procedure_stats*. Tu nit dobi pet statistik: *začetek časovni žig, trajanje izvajanja, izvajanje na procesorju* ter *ime baze*. Na podlagi teh statistik in podatkov, ki jih prejme iz aplikacije, nit izračuna in dopolni še vse ostale statistike.

Ključne besede: BI, podatkovno skladišče, zmogljivostna analiza, DMV, OLAP, BI4Dynamics.

Abstract

BI4Dynamics is an analytical Business Intelligence (BI) solution for Microsoft Dynamics ERPs. The solution deploys a data warehouse on a Microsoft SQL server and creates an analytical database on a Microsoft Analysis server; accordingly, due to SQL's flexibility, it offers an excellent platform for developing custom BI modules; OLAP cubes and dimensions.

This paper describes how the BI4Profiler, which is a performance module for BI4Dynamics platform, was developed. In this process my task was to design the core architecture of the module and oversee project development. The BI4Profiler module grew out of our partners' and our own need for a tool when benchmarking and optimizing custom developed SQL scripts.

We wanted to use the existing framework so this tool was packed as a module, thus it can be analyzed using any BI compatible front end tool. To ensure independency of statistic data, the BI4Profiler deploys its own separate data warehouse with staging and presentation area, as well as an OLAP cube with a set of dimensions. Although module, it has an additional core functionality that we had to integrate into the existing framework. During the execution of the scripts, this separate thread pools the SQL Server using DMVs (Dynamics Management Views) for statistics, which first saves said statistics to the data warehouse, thence builds and processes the analytical database.

Keywords: BI, BI4Dynamics, OLAP, data warehouse, benchmarking, DMV.

Chapter 1

Introduction

There have been a lot of BI providers on the market of late. Some offer complete solutions, while others only cover certain functionality of BI - such as visualization. Here at BI4Dynamics d.o.o. we developed a complete BI solution for Microsoft Dynamics ERPs. This solution provides a platform that enables users to develop their own modules or customize standard ones. For the sake of better integration it is built on top of the Microsoft Technology Stack, with Microsoft SQL for the data warehouse, Microsoft Analysis Services for the OLAP cubes and .NET framework for the interface.

The BI4Dynamics platform contains 14 standard cubes that cover most common businesses areas. Back in 2008 we started with only 6 standard cubes but over the years we and our partners - as a part of client specific projects - have developed custom cubes that later became part of our standard cube repository. Yet every time we started implementing these customizations we faced the same challenge, especially in the optimization phase. How do those new scripts perform? And how much space do they use? You can do the benchmarking manually, but it's a really strenuous task.

This is how the *BI4Profiler* was born. Initially we intended to build a simple benchmarking application, but then decided to package it as a module capable of enabling users to analyze statistical data in much the same way as they would their business data.

Chapter 2

Businesses Intelligence

In the modern consumer age, one cannot imagine running a large company in which all records are kept in paper format. Indeed, why would we. Since the advent of computers, we have been given an amazing opportunity to take evolution to a whole new level. Every year our technological capacities grow exponentially, and an enormous amount of data is being collected, processed and stored every day. That's why companies - big and small - need robust, rapid and reliable systems. These are so-called transaction systems, mostly known as ERP (Enterprise Resource Planning) systems. Even though these systems offer some form of static analysis, they are primarily optimized for data collection. But data does not equal information. Data without context is meaningless. Implemented as an ERP add-on, the first BI solutions emerged during the late 1990s as an answer to the desire to obtain useful information from analytical data which ultimately provides knowledge and a better understanding of the market, products, services and customers, which in turn ensures a march on competition.

You can find a vast variety of definitions of BI on the internet, but to really understand it we have to answer two fundamental questions:

1. What is its purpose?
2. How is it used?

Over the years, with evolution of technology, we have witnessed great improvements in BI systems but their sole purpose remains the same. Sun Tzu's magnum opus *Art of War* [1] talks about the importance of knowing both your own and your opponent's greatest strengths and weaknesses in order to win a war; such is also the underlying notion of modern economic warfare: only a complete overview and understanding guarantees optimal decisions. A company must know all its strengths and weaknesses and has to be able to exploit its opponent's weaknesses if it wishes to successfully compete on the market battlefield. So the main purpose of BI systems is the timely provision of quality information in order to achieve better and more reliable decision making processes and the subsequent provision of a head start in the market through higher quality business processes, services and products as a consequence of such decisions.

Important decisions are usually based on the conclusions of multiple experts from various departments within the organization. These experts rely on the fact that the latest information is up to date and reflects company's current state. BI systems therefore have to reflect a company's objective reality; if it does not, said system doesn't hold any value to the company, and indeed incorrect information in relation to business processes may even have deleterious consequences. Besides providing sound information, one of the main goals of BI systems is the timely delivery of relevant useful information. These factors do, of course, vary from field to field. For example in stock trading the on-time factor is of manifold greater importance than it is in the retail sector, because information has to be available right away else the lucrative investment opportunity will be missed. Likewise the operational health of the company (e.g. reduced production costs, increased sales, better HR management, greater productivity) is proportional to the quality of the intelligence provided.

For this reason, the realization of BI systems is nowadays one of the most important tasks within modern organizations, if not the most important. Indeed, Garnet's 2008 survey of 1,500 leading CEOs concurs with this asser-

tion, while BI systems rank among the top 3 priority tasks undertaken by the IT department. It can only be surmised that the importance of this task can only have increased in the interim.

2.1 Lessons Of History

BI systems have been around in their primitive form since the dawn of man. All organizations are forever in need of insightful information, on the basis of which decisions are made. Any organization which makes decisions based on a whim or hunch is ultimately destined to fail and fall apart. Before the advent of computers, various media - from stone inscriptions to paper accounts - were used to store data, while the system's building blocks were people. Analysts who possessed the knowledge and skill to extract useful information from stored data, and mathematical and statistical methods were employed in this. Early data were records of transactions and the trade of goods and commodities such as silk and sugar. For example, the Sumerians kept records of wheat shipments carved into stone plates, and this is the first recorded use of a written language for the purpose of storing and analyzing data. As can be seen, the questions bothering the Sumerians were not so different from the ones we have today. Which product is the most profitable? And which buyers are late with payments? Just the time needed to provide answers was significantly longer.

From the data inscribed in stone to the data stored in a RAM (Random Access Memory) or on a hard drive, there's always been a challenge in storing the largest possible amount of data in the smallest possible space. The need for increasing the capacity of a storage medium is even more relevant today. The possibility of storing large amounts of data became significant when the first magnetic tapes emerged in the early 1950s¹ and really took off with the

¹Magnetic tape was first used to record computer data in 1951 on the Eckert-Mauchly UNIVAC I. The recording medium was a thin strip of half-inch (12.65 mm) wide nickel-plated bronze (called Vicalloy). The recording density was 128 characters per inch (198 micrometer/character) on eight tracks.

advent of the hard drive, which represented a huge step forward in mass data storage. This set the stage for the arrival in late 1960s of the Hierarchical database model and mass data collection.

Edgar F. Codd published *A Relational Model of Data for Large Shared Data Banks* [2] in 1970, and this laid the foundations for the database systems we know today. This set off an avalanche of relational DBMS such as Ingres, SyBase, System R and later DB2, while the first tools for analyzing and reporting emerged not long after. Such tools were, however, far from handy; the analyst needed help from a programmer - who knew the DBMS inside out- to transfer the data from the server. Not only was there a low limit for the amount of data that could be transferred, but also, from this point on, the transferred data was no longer synchronized with the server. This was a huge blow due to the fact that the amount of transferred data was rarely sufficient to satisfy the need for quality analysis, hence reliability remained a limiting factor. Combining this with a command line user interface and inherent inability to handle missing values or partial records, systems were highly prone to error, which was the reason in itself why those tools were destined not to become the next big revolution.

With the arrival of personal computers in the early 1980s ² the world of data analysis started to evolve rapidly. Personal computers basically invaded the realm of business almost over night, and a prime mover in this invasion was Daniel Bricklin. In 1978, while still a student at Harvard Business School, Bricklin embarked on the development of VisiCalc. His idea was to make a program that would be intuitive and enable users to handle numbers with the same ease they handled a text when using a text editor. This was the first electronic spreadsheet; readily available for home and office use, its impact was enormous.

²With the introduction of the original Macintosh computer on 24 January 1984, Steve Jobs revolutionized the world of computers. This was the first mass-market personal computer featuring an integral graphic user interface and mouse.

It ran on non other than the Apple II and is therefore also widely credited for fueling the rapid growth of the personal computer industry.

Instead of producing financial projections using manually calculated spreadsheets, and having to recalculate every single cell in the sheet, VisiCalc allowed the user to edit any cell, and have the entire sheet automatically recalculated. This turned 20 hours of work into a 15 minute task and provided greater creativity. Upon graduation in 1979, Bricklin and Frankston founded Software Arts Inc. and began selling VisiCalc. It is interesting to note that they didn't patent VisiCalc, which enabled a company called Lotus to develop widely know application Lotus 1-2-3.

From there on afterwards there was no need for the analyst to sit behind big computers in a huge data center and go through a great deal of technical training just to learn how to use those mean machines. Now they could do it on a small screen in the comfort of their own office, a far more user-friendly and intuitive solution. But still one problem remained. There was still a dependence on programmers to transfer data, which was a slow and highly impractical process. It wasn't until the standardization of SQL language in the late 1980s, when the first two-tier client-server DMBSs arrived on the scene, that these systems fully developed their true potential. SQL had been used for queries before, but the standardization milestone enabled tools to independently switch between different DBMS providers. Upon transition to a new a system, companies were given an excellent opportunity to restructure their businesses process which subsequently caused additional counseling costs in addition to the costs of new hardware and software; nevertheless, many companies dived right into it, seeing greater returns in the long run, and these migrations fueled the IT industry. The first ERP systems - such as SAP, Baan and Peoplesoft - were born in the late 1990s in the midst of all this euphoria. Most known were. These new systems came fully packed, and covered a wide variety of other businesses areas besides manufacturing.

2.2 Definition

BI slowly starts gaining widespread recognition in the late 1990s, but the term - Business Intelligence - was originally coined by Richard Millar Devens in his 1865 *Cyclopædia of Commercial and Business Anecdotes* [4]. Devens used the term to describe how the banker, Sir Henry Furnese, gained profit by receiving and acting upon information about his environment, prior to his competitors. *“Throughout Holland, Flanders, France, and Germany, he maintained a complete and perfect train of business intelligence. The news of the many battles fought was thus received first by him, and the fall of Namur added to his profits, owing to his early receipt of the news.”* (Devens, p.210).

The ability to collect and react accordingly based on the information retrieved - a skill that Furnese excelled in - is today still at the very heart of BI. IBM researcher Hans Peter Luhn became the first to use the term business intelligence in a more modern sense in his 1958 article [3]; in it he employed a Webster’s dictionary definition of business: a collection of activities carried on for whatever purpose, be it science, technology, commerce, industry, law, government, defense, et cetera. The communication facility serving the conduct of a business (in the broader sense) may be referred to as an intelligence system. The notion of intelligence was also defined therein in a more general sense as *“the ability to apprehend the interrelationships of presented facts in such a way as to guide action towards a desired goal.”* It was, however, Howard Dresner who popularized it by proposing *business intelligence* as an umbrella term to describe concepts and methods to improve business decision making by using fact-based support systems.

The true definition we know today, however, refined itself in the last ten to fifteen years. According to Forrester Research when referring to BI in the broader sense we refer to *“a set of methodologies, standards, processes, architectures, and technologies that transform raw data into meaningful and useful information used to enable more effective strategic, tactical, and operational insights and decision-making.”*[5] Therefore BI systems are also sometimes referred to data driven Decision Support Systems - DSS. Under this

definition, business intelligence also encompasses technologies such as data integration, data quality, data warehousing, master-data management, text- and content-analytics, and many other processes that the market sometimes lumps into the “Information Management” segment.

On the other hand, when talking about BI in a more narrow sense, i.e. only in terms of data usage, we refer to just the top layers of the BI architectural stack, such as reporting, analytics and dashboards [6]. Data usage and data preparation are indeed two separate things, but they are closely linked segments of the business-intelligence architectural stack.

Looking closely at the definition it’s obvious that-even the first ERP systems possessed some elements of BI system. The statically compiled reports were able to provide the answers to a few specific questions. But in order to achieve this, the data had to be taken from the data source and rearranged in a certain way in order to yield answers to a substantial range of so-called ad-hoc questions. The term data warehouse first surfaced back in 1988, however it was in the late 1990s when Bill Inmon and Ralph Kimball ensured its widespread recognition. In 1992 Inmon, recognized by many as the father of the data warehouse, published his most notable work: *Building the Data Warehouse* [11]. Kimball, widely regarded as one of the original architects of data warehousing, followed with *The Data Warehouse ToolKit* [8] in 1996; in it he discussed his own dimensional modeling methodology - also known as Star Schema [8] - which later became the de facto standard in the sphere of decision support.

According to Thomas Davenport [7] BI systems are divided into Business Analytics, querying, reporting (standard, ad-hoc), online analytical processing (OLAP) and alert tools. Because they offer greater flexibility and rapid answers to arbitrary ad-hoc questions per single businesses events, technologies for advanced reporting that utilize a data warehouse and OLAP offer a huge advantage and improvement over simple ERP queries.

Business analytics take everything a step further by making extensive use of statistical and quantitative analysis, explanatory and predictive modeling,

forecasting and stochastic optimization, not to find out what has happened in the past but all that which is yet to come. In other words: Querying, reporting, OLAP and alert tools can answer questions in relation to what has happened, how many, how often, where the problem is, and what actions are necessary. Business analytics can answer such questions as why is this happening, what will happen next if these trends continue (prediction), and what is the best outcome (optimization).

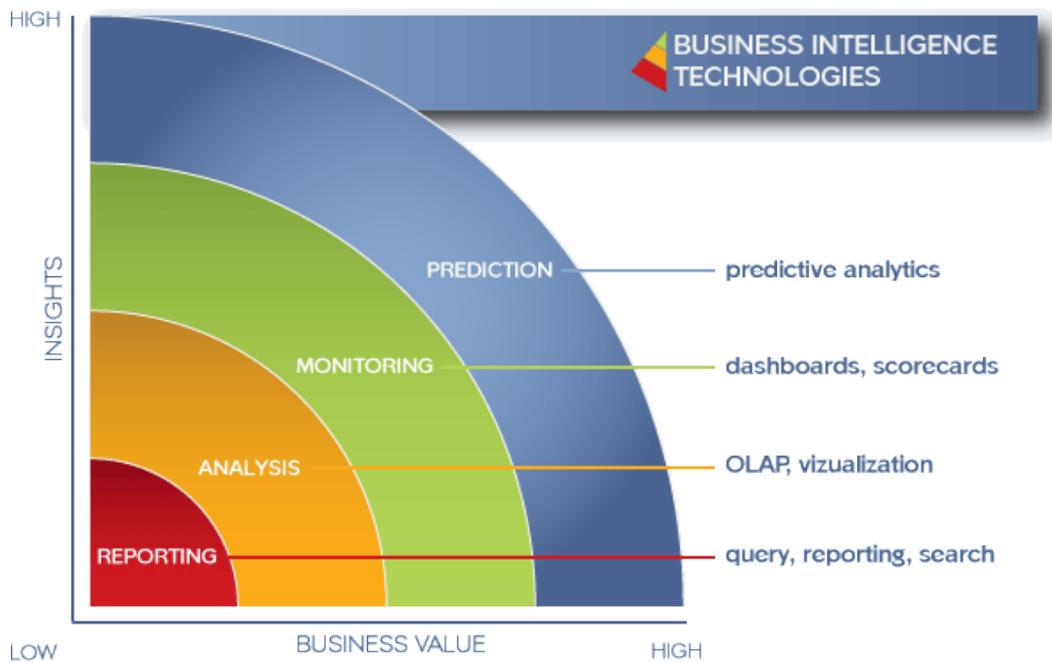


Figure 2.1: The outer layer represents Business Analytics.

Chapter 3

BI4Dynamics

BI4Dynamics is an analytical BI solution for Microsoft Dynamics ERP system. It comes in two available product lines:

- BI4Dynamics NAV - for Microsoft Navision ERPs
- BI4Dynamics AX - for Microsoft Axapta ERPs

Since both product lines have the same core framework and data model, with the exception of some minor structural differences, and since the performance tool was initially developed for BI4Dynamics NAV product line, the following chapters focus solely on the BI4Dynamics NAV.



Figure 3.1: BI4Dynamics Transformation Process.

3.1 BI4Dynamics NAV

BI4Dynamics NAV is a BI system designed to equip the Microsoft Navision ERP system with some heavy analytical firepower. Like all ERP systems, Microsoft Navision has been optimized for fast and efficient data collection and this can only be achieved with a normalized data model which minimizes data redundancy by employing a large number of tables. This can be a big problem since queries that are executed during data analysis as opposed to those that are executed from data storage, have to combine data scattered over large amount of tables and use computationally demanding operations, which is a heavy draw on resources. The solution lies in restructuring data from the Navision data source into a data warehouse that uses a dimensional model as its basis. This highly simplifies the structure of the relational database and enables more rapid query execution (although it produces some redundancy).

Built on Kimball's star schema principle [8], the BI4Dynamics data warehouse is a data source for a set of OLAP Cubes. Each cube contains analytical data for one of Navision's business areas and provides hierarchically ordered description data and aggregated transactional data for each of the hierarchies. Using appropriate tools, this enables business users connected to those cubes to undertake the desired analysis in a rapid and easy fashion. Most BI4Dynamics customers use Excel as a BI front-end client, but any other BI client capable of connecting to Analysis services can be used to access that information. The BI4Dynamics platform also enables the combination of data from different NAV database versions in the same BI4Dynamics data warehouse, thus scripts are executed by the source version in all Navision products - from NAV 3.1 to NAV 2015 - enabling the user to join different operational database versions.

The BI4Dynamics system fully utilizes a Microsoft technology stack, which means it can be installed on any version of the SQL Server. But the greatest power lies in its architecture, as it provides a platform for building custom analytical modules thus any part of the solution can be customized.

3.2 Data Model

The BI4Dynamics system is packed as a .NET application which through an especially designed installation wizard deploys a fully functional data warehouse (with ETL procedures) and a set of desired OLAP cubes and corresponding dimensions. ETL procedures transfer the data from the Navision data source into the staging area of the BI4Dynamics data warehouse. In the next step, data is restructured, cleaned and then saved into dimensional and fact tables within the presentation area, using a set of views and stored procedures. The final step transfers data from the data warehouse presentation area into cubes and dimensions within the analytical database.

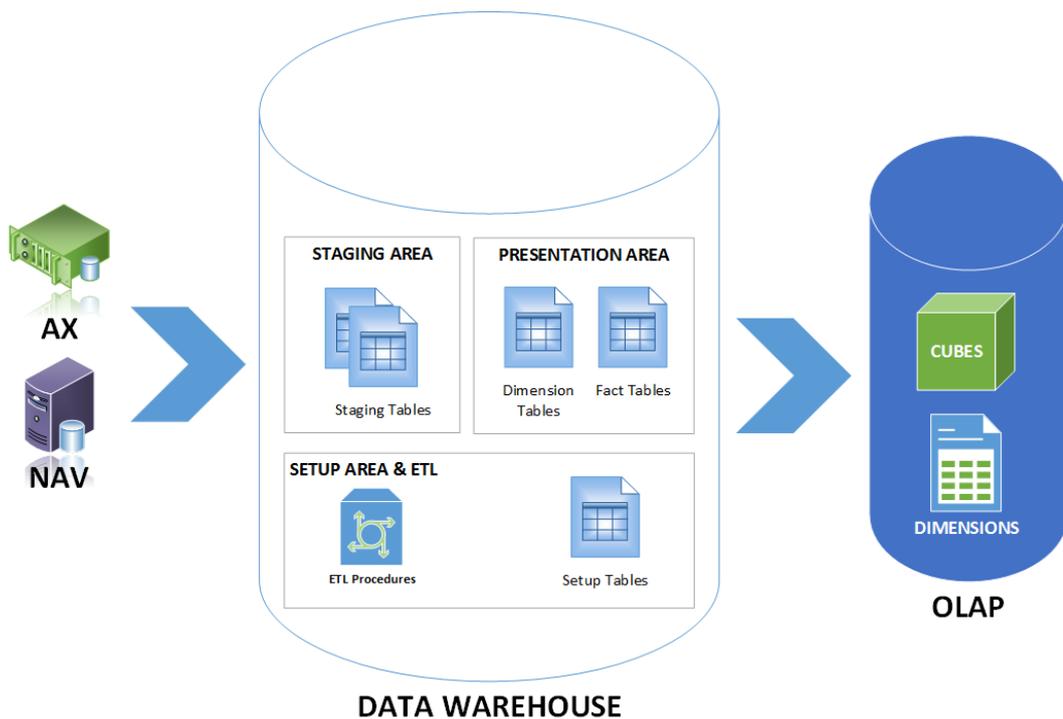


Figure 3.2: BI4Dynamics Data Model.

3.2.1 Data Warehouse

Setup Area

The Setup Area is the home of setup tables and setup procedures which contain all the necessary setup data for building the staging and presentation area. Most of the setup data is entered through the installation wizard by the user, but some is also automatically generated. Tables and procedures are part of the *setup.** schema which is automatically created when deploying the data warehouse.

Table	Type	Description
setup.DataSource	Table	Contains info about selected Navision data sources such as version, server name, etc.
setup.Company	Table	Contains selected companies
setup.DataSourceTable	Table	Contains tables from Navision data source that are needed for the staging area
setup.DataSourceColumn	Table	Contains columns for the tables
setup.DataSourceIndex	Table	Contains indexes for the tables
setup.Translation	Table	Contains option field translations
setup.InitializeDimension	Procedure	Stored procedure that initializes dimensional table with 0 ID and default values for every field in case of missing or incomplete values in the transactional data
setup.CreateStage	Procedure	See under ETL section
setup.UpdateStage	Procedure	See under ETL section
setup.IndexStage	Procedure	See under ETL section

Table 3.1: Setup Tables and Procedures.

Staging Area

The staging area is where transactional data is readied for reorganization and restructuring. It is a set of NAV equivalent tables that are part of *stage.** schema. They are filled with ETL procedures which are not visible to the end user.

ETL

The first step Extracts the transactional data from the data source. In this step only the relevant data is transferred, therefore a complete understanding of Navision's data model is of key importance. This part of the ETL process is represented by three procedures in the Setup area:

- **Create Stage** procedure for every Navision table in *setup.DataSourceTable*, builds an equivalent stage table with equivalent columns in *setup.DataSourceColumn*. Stage tables are unified for all data sources inside *setup.DataSource* table, meaning a union of tables and columns is made across all data sources. Foreign keys *CompanyID* and *DataSourceID* are added to distinguish between companies and data sources.
- **Update Stage** procedure copies the data from tables on data sources into the stage tables within the presentation area, also adding primary keys *CompanyID* from *setup.Company* and *DataSourceID* from *setup.DataSource*. *NULL* values are inserted if the column doesn't exist in the current data source.
- **Index Stage** procedure creates the same indexes on the stage table as they are in the Navision equivalent. These indexes are stored in *setup.DataSourceIndex* table. Besides the original indexes, a user can also create their own and these too may be stored in this table.

Before the data is transformed and saved, the *CreateTable Procedure* first builds the presentation area by creating presentation tables.

The next step is Transformation of the data. This process includes cleaning the data (handling missing values, correction of grammatical errors, resolving domain conflicts and reorganization into standard formats), joining data from multiple data sources and removing duplicates. This step is carried out by *Transformation View 1* and *Transformation View 2*.

- **Transformation View 1** is where all the Navision businesses knowledge and logic is concentrated, and it is also where all the transformations are made. Transformation View 1 first combines the data from stage tables, help tables and setup tables; it then cleans, filters, calculates and represents the data with the granularity of the targeted dimensional table. Any SQL function compatible with the target server may be used.
- **Transformation View 2** is specific only for fact tables. In Kimball star schema [8], dimension tables are directly linked to fact tables. In the BI4Dynamcis data warehouse, Transformation View 2 provides that link between fact and dimension tables. Based on the conditions, a set of dimensions is provided for each record in the fact table. Missing values are transformed into 0 key so that they are linked with the default record inserted by the *setup.InitializeDimension* procedure.

Finally the data is Loaded into the dimensional and fact tables in the presentation area. After this step the data is ready to be used by the analytical server.

- **Load Procedure** takes the data provided by *Transformation View 2* and stores it into the tables in the presentation area. Its structure is slightly different for dimension and fact tables. The fact table procedure first truncates the data in the fact table, or in the event of an incremental update, only the data that is newer than the incremental time stamp is truncated. Dimensional tables are not truncated since they are connected to fact tables with primary keys, thus the load procedure only inserts new data and updates old records.

Presentation Area

The Presentation Area is where the data is organized, saved and directly accessible in relation to user queries, reports and other analytical applications. This data is organized in a manner that follows the dimensional modeling principle.

Dimensional modeling differs a lot from modeling approaches such as third normal form (3NF) which is a design technique that minimizes redundancy. The normalization approach enables rapid processing of the data because saving and updating transactions update data in one place only, hence this type of approach is really good for ERP systems, as was mentioned earlier. At the same time the normalized form is not appropriate for higher structurally complex queries typical to data warehouses. In relation to the DBMS, the complexity of the normalized form exceeds the ability of query optimization. Specifically due to these shortcomings, the normalized form is not appropriate for modeling data in the presentation area of data warehouse. It is most important for the dimensional model structure to foresee all possible types of user queries. Despite the fact that data marts contain summarized data which speed up the queries, they must also contain atomic data that guarantees the smallest possible granularity. This way a user can, by drilling down through the dimensions, access the most detailed data. It is important that granularity of the data in the presentation area is detailed enough to provide the user with answers to the most detailed and exacting questions. All data marts must use shared dimension and fact tables. These tables, which form part of multiple business or subject areas, are known as conformed tables. Usage of such tables is crucial in assuring a robust and integrated data warehouse. Architecture that has a lot of conformed dimensional and fact tables is known as bus architecture, and is crucial in the provision of a distributed data warehouse.

The BI4Dynamics platform follows best practice and has a centralized data warehouse with bus architecture, by way of which all the dimensions are shared across different data marts. This results in low data redundancy.

Fact Tables

The Fact Table is a primary table of the dimension model and consists of the measurements, metrics or facts of a business process. The term “fact” describes one business measure, and data for a single business process (business or subject area) is usually stored in one data mart. Fact tables contain the content of the data warehouse and store different types of measures - such as additive, non additive, and semi additive measures.

Often defined by their grain, fact tables provide the additive values which act as independent variables, by way of which dimensional attributes are analyzed. The grain of a fact table represents the fundamental atomic level by which the facts may be defined. If, for example, we decide to follow sales for a store chain, then as a result, for each product sold across all stores, we get measures for quantity and amount. A measurement is therefore uniquely defined by a day, product and store. Other dimensions might be members of this fact table (such as location/region) but these add nothing to the uniqueness of the fact records. Such “affiliate dimensions” allow for additional slices of the independent facts, but generally provide insight at a higher level of aggregation (i.e. the region contains a number of stores).

A fact table typically has two types of columns: those that contain facts and those that are a foreign key to the dimension tables. The fact table assures referential integrity when all foreign keys in said table match with the primary keys in the dimension table. The primary key of a fact table is usually a composite key which is made up of all of its foreign keys. In the dimension model each table which contains a composite key is a fact table, and as such it has many-to-many connections. All other tables are thus dimension tables.

In the BI4Dynamics platform, fact tables don't have a primary key, because all data is deleted and newly inserted during update.

Dimension Tables

Unlike fact tables, dimension tables contain descriptive attributes (or fields) which are typically textual (or discrete numbers that behave like text). These attributes are designed to serve two critical purposes: query constraining and/or filtering, and query result set labeling. In well designed dimensional models, dimension tables have many descriptive attributes. It's therefore not unusual for a dimension table to contain 50 to 100 attributes. Compared to fact tables, dimension tables usually contain a relatively small number of records, about half a million.

Dimension table rows are uniquely identified by a single key field. It is recommended that the key field be a simple integer because a key value is meaningless, and used only for joining fields between the fact and dimension tables. Dimension tables often use primary keys which are also surrogate keys. Surrogate keys are often auto-generated (e.g. a Sybase or SQL Server "identity column", a PostgreSQL or Informix serial, an Oracle *SEQUENCE* or a column defined with *AUTO_INCREMENT* in MySQL).

Attributes play a key role within the data warehouse because they assure usefulness and understanding. That's why when designing a data warehouse it is crucial to label using a terminology that best describes businesses process. Best practice is to replace codes with description fields.

Dimension tables often include hierarchical relations that appear in the businesses system; for example: a dimensional table for items. Items usually belong to a brand (grouping), and brands are combined into categories. For each record we then also have to save its brand and category. It should be noted that such data increases redundancy, which in this case is welcomed because it simplifies things and lowers the execution time of queries.

Dimensional tables in the BI4Platform also include a large amount of descriptive attributes. Most of the attributes are restructured and combined to form new ones.

Dimensions can define a wide variety of characteristics, but some of the most common attributes defined by dimension tables include:

- Time dimension tables describe time at the lowest level of time granularity for which events are recorded in the star schema
- Geography dimension tables describe location data, such as country, state, or city
- Product dimension tables describe products
- Employee dimension tables describe personnel, such as sales staff
- Range dimension tables describe ranges of time, currency values, or other measurable quantities which simplify reporting

Star Schema

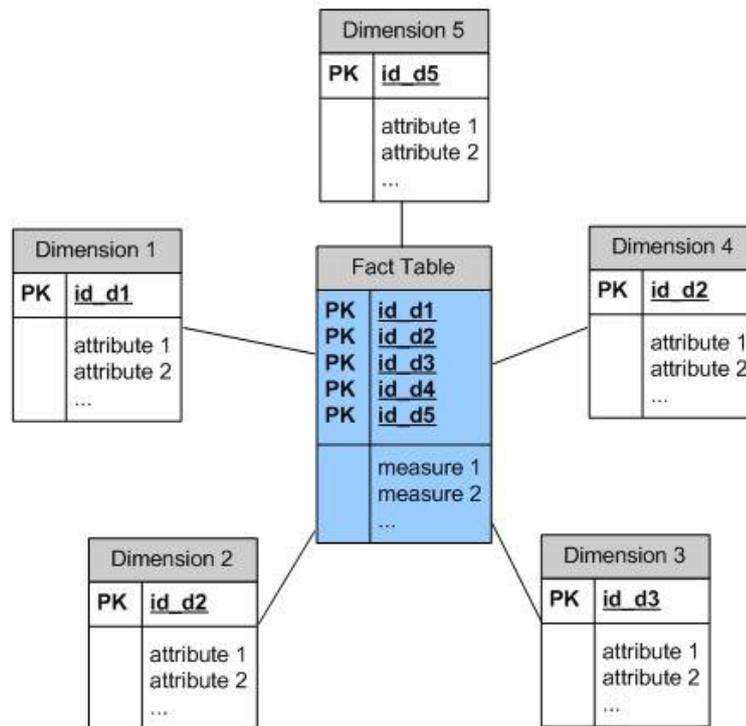


Figure 3.3: Star schema used by example query.

The Fact Table and its measures is located at the center of a Star Schema surrounded by dimension tables containing descriptive attributes for those measures. The Star Schema gets its name from the physical model's resemblance to a star, with a fact table at its center and the dimension tables surrounding it representing the star's points. The Star Schema (also called a star-join schema) is the simplest style of data mart schema.

The Star Schema is denormalized, meaning the normal rules of normalization applied to transaction relational databases are relaxed during star schema design and implementation. The benefits of Star Schema denormalization are:

- Simpler queries - star schema join logic is generally simpler than the join logic required to retrieve data from a normalized transactional schema.
- Simplified business reporting logic - in comparison with highly normalized schemas, the star schema expedites common business reporting logic, such as period-over-period and as-of reporting.
- Query performance gains - in comparison with highly normalized schemas, the star schema can provide performance enhancements for read-only reporting applications.
- Fast aggregations - the simpler queries in a star schema can result in improved performance in aggregated operations.
- Feeding cubes – the star schema is used by all OLAP systems to build proprietary OLAP cubes efficiently; in fact, most major OLAP systems provide a ROLAP mode of operation which can use a star schema directly as a source without building a proprietary cube structure.

3.2.2 OLAP

Online Analytical Processing is an approach to answering multi-dimensional analytical queries swiftly. The term OLAP was created as a slight modification of the traditional database term Online Transaction Processing (OLTP). OLAP tools enable users to analyze multidimensional data interactively from multiple perspectives. OLAP consists of three basic analytical operations: consolidation (roll-up), drill-down, and slicing and dicing. Consolidation involves the aggregation of data that can be accumulated and computed in one or more dimensions. For example, all sales offices are rolled up to the sales department or sales division to anticipate sales trends. By contrast, the drill-down is a technique that allows users to navigate through the details. For instance, users can view sales by individual product, and likewise can do this on a regional basis. Slicing and dicing is a feature whereby users can take out (slice) a specific set of data from the OLAP cube and view (dice) the slices from different perspectives. Databases configured for OLAP use a multidimensional data model or SSAS, facilitating the rapid execution of complex analytical and ad-hoc queries by borrowing aspects of navigational databases, hierarchical databases and relational databases.

Despite the fact that data in the presentation area of the BI4Dynamics data warehouse is structured in a way that is ready for end users to access and analyze, BI4Dynamics introduces another layer – the Analytical Database. The analytical database further simplifies and speeds up the way end users may handle the data by compressing the analytical database and pre-calculating aggregates for every level of dimension. BI4Dynamics uses MOLAP, which is the default go-to form of online analytical processing and is sometimes referred to as just OLAP. MOLAP stores data in optimized multi-dimensional array storage, rather than in a relational database. Therefore it requires the pre-computation and storage of information in the cube - the operation known as processing. MOLAP tools have a very rapid response time as well as the ability to quickly write back data into the data set.

Data Source

Data source is a link between the analytical database and data warehouse, and identifies to the analytical server where the data is coming from. It is defined in the form of a connection string that describes the manner of connection between the analysis server and the data warehouse where the data is stored. The connection string contains the *Server name*, *Database name*, *Security protocol* and *Timeout interval*.

BI4Dynamics uses an integrated SQLNCLI provider to connect to the data warehouse and ensures safety with Windows domain authentication.

Data Source Views

A data source view contains the logical model of the schema used by Analysis Services multidimensional database objects - namely cubes and dimensions. A data source view is the metadata definition, stored in an XML format. A Data Source View:

- Contains the metadata that represents selected objects from one or more underlying data sources, or the metadata that will be used to generate an underlying relational data store if one is following the top-down approach to schema generation.
- Can be built over one or more data sources, letting the user define multidimensional and data mining objects which integrate data from multiple sources.
- Can contain relationships, primary keys, object names, calculated columns, and queries that are not present in an underlying data source and which exist separately from the underlying data sources.
- Is not visible to, or available to be queried by, client applications.

Data in the BI4Dynamics presentation area is mapped to the analytical database using 1:1 relation, which means that it contains all dimensional and fact tables.

Cube

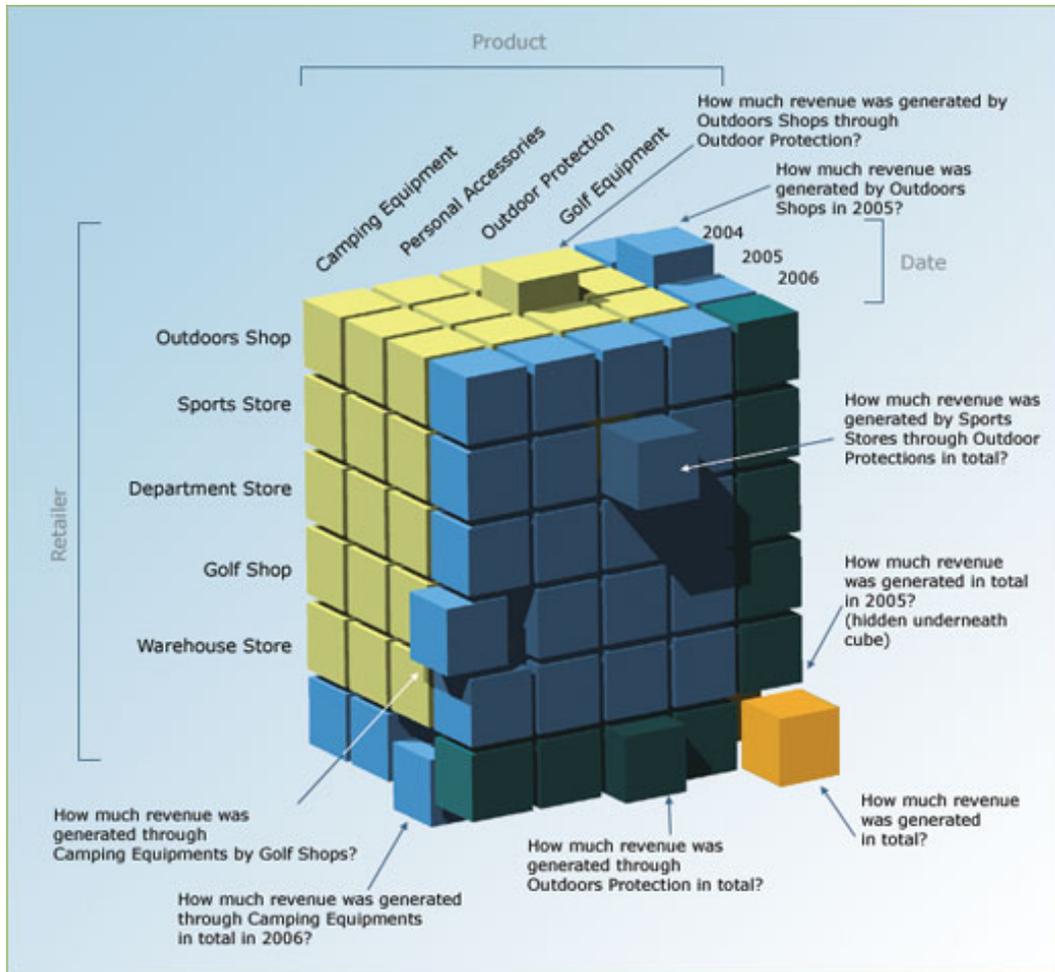


Figure 3.4: OLAP Cube.

An OLAP cube is a method of storing data in a multidimensional form. The cube is defined by the dimensions and measures which are categorized by its dimensions and represent transactional values used in the aggregations. A measure can be based on one or more columns drawn from one or more fact tables combined into a measure group. A dimension is a set of attributes that present a field of interest inside a measure group. Attributes can be based on one or more columns drawn from one or more dimensional tables and are often organized into hierarchies to enable users more detailed analysis.

For complex queries, OLAP cubes can produce an answer in around 0.1% of the time required for the same query using OLTP relational data. The most important mechanism in OLAP, which allows it to achieve this high performance level is the use of aggregations. Aggregations are constructed from the fact table by changing the granularity of specific dimensions, and aggregating the data along these dimensions.

The OLAP Cube also contain:

- **Calculations**, which are MDX expressions that enable the user to add additional objects, in most cases complex new measures, that can use other object (measures) both within and outside the cube, or even outside the analytical database. The MDX language was originally developed by Microsoft in the late 1990s, and has been adopted by many other vendors of multidimensional databases. Almost all cubes in the BI4Dynamics analytic database contain complex calculations.
- **Partitions**, which divide a cube into logical parts. Each partition can then be reprocessed (refreshed) independent of the other partitions. For example, a cube containing a huge amount of sales data, can be split into partitions based on each quarter of a year. When new data is being written, you can process a partition that represents the current quarter, a feature which speeds up processing enormously. In BI4Dynamics, each cube partition is processed individually each time.
- **Translations**, which ensure multilingual support. Data is often analyzed by users of different nationality, hence it is therefore convenient if object names are displayed in the local language. A translation, defined by the language code and caption, can be provided for every object inside the cube, these can encompass dimensions, hierarchies, the cube name, measure groups, measures and calculations etc... BI4Dynamics provides out of the box translations for all cubes in relation to ten different languages.
- **Actions** are not used in the BI4Dynamics platform.

Common cube operations:

- **Slicing** is the act of picking a rectangular subset of a cube by choosing a single value for one of its dimensions, creating a new cube with one fewer dimension. [9]

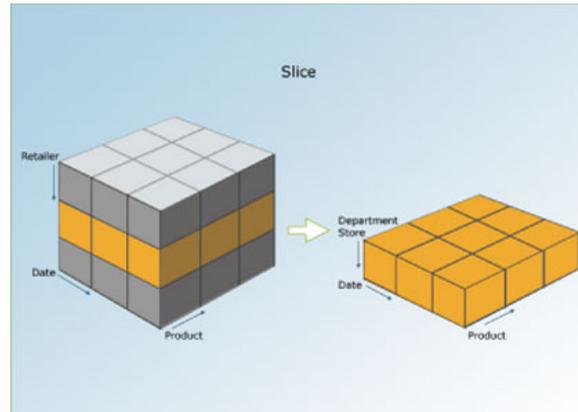


Figure 3.5: The slice contains information on date and product in relation to a department store.

- **Dicing** produces a sub-cube by allowing the analyst to pick specific values of multiple dimensions. [10]

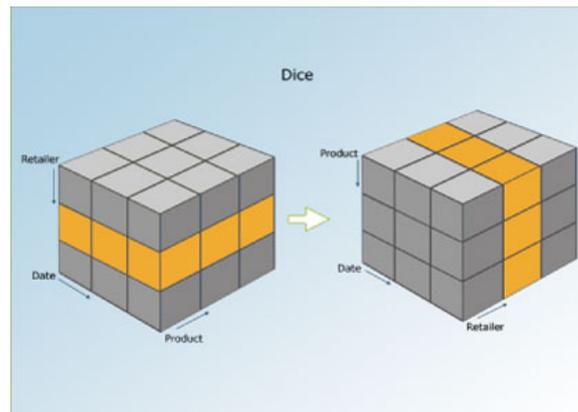


Figure 3.6: Dicing a data cube – retrieving more detailed information in relation to date and retailer of a specific product.

- **Pivoting** allows an analyst to rotate the cube in space to see its various facets. For example: while viewing data for a particular quarter, locations could be arranged vertically and products horizontally. Pivoting could be used to replace products with time periods to reveal data across time for a single product.
- **Drill Down/Up** allows the user to navigate among levels of data ranging from the most summarized (up) to the most detailed (down).

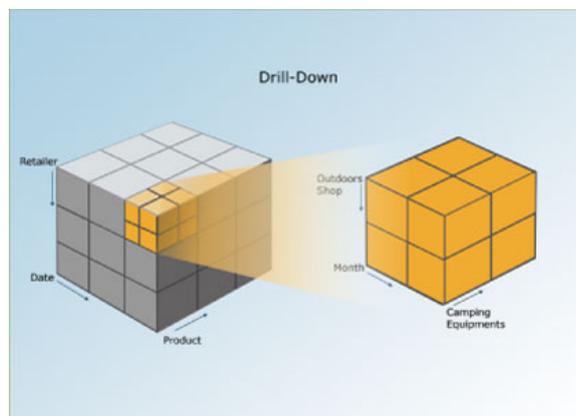


Figure 3.7: The Retailer dimension can be drilled-down to reveal specific retailers, while the Date dimension can be drilled-down to reveal a given month; Product can be explored in more detail per individual products.

- **Roll-up** involves summarizing the data along a dimension. The sum rule might be computing totals along a hierarchy, or the application of a set of formulae, such as $profit = sales - expenses$.

All standard cubes and dimensions of the BI4Dynamics platform are saved as an embedded resource in the form of an XML object. When deploying XML it becomes de-serialized into an AMO object inside a .NET application that is sent directly to the analytical database.

BI4Dynamics standard Cubes:

- **Sales**

Comes with 72 measures that in combination with 47 dimensions provides insight into all parts of the sales area. Analyzing sales trends, margin, parallel period and year-to-date sales reports extend standard reporting as well as makes analysis simple, powerful and quick.

- **Sales Orders History**

BI4Dynamics makes a daily snapshot of all sales quotes and orders and stores them in the BI4Dynamics' Data Warehouse. This feature enables the analysis of sales quotes and sales orders, even after they are amended, processed or deleted.

- **Receivables**

The receivables module is most commonly used as standard in all organizations and comes with more specific measures such as Average Balance, Sales on Credit, Average Due and Open Days, Receivables Turnover in Days, Customer Net Change, % of Total Balance, etc...

- **Purchase**

The purchase module enables complete purchase analysis with multiple measures and dimensions over multiple companies and currencies. Purchase by Type, Purchase by Top Vendors by Type, Purchase by Top 10 Vendors by Purchased Items

- **Purchase Orders History**

Powerful analysis of orders and blanket orders is mandatory for efficient supply chain management. It is crucial to track the status of purchased and supplied items. Daily data warehouse snapshots ensure that purchase order analysis is both fast and accurate.

- **Payables**

Analysis of specific invoices and groups of invoices with smart Payables measures like Payables Balance, Payables coefficient, Turnover in days, Purchase of credit, Vendor credit, Vendor discount per different currencies, date, companies, etc...

- **Inventory**

BI4Dynamics enables powerful inventory value and quantity analysis across multiple dimensions. To avoid typical difficulties with inventory valuation in ERP systems, daily snapshots of data are created in the data warehouse to ensure rapid and agile analysis.

- **Account Schedules**

Account schedules are used to prepare financial reports based on the general ledger. Using account schedules, users can choose specific accounts and perform calculations. In Microsoft Dynamics NAV, users define their own account schedule.

- **General Ledger**

Consolidation of information across multiple companies, dimensions and currencies has never been easier. Charts of Accounts and Income Statement reports can be created, while budgets over the years can be compared with YTD (year-to-date) KPI (key performance indicators).

- **Jobs and Resources**

Powerful analysis of Jobs and Resources, which can reveal answers in relation to budgets, costs and profits in relation to disparate open jobs in a single report. The Job and Resources module provides the ability to concurrently compare budgets, costs and profits in one report for a specific project.

- **Manufacturing**

Powerful, simple and agile analysis across different manufacturing processes. The Manufacturing module enables the monitoring of Production Orders, Item Composition and Consumption, Actual and Expected Quantities with Variance. Average Costs or Work in Progress can be analyzed across many dimensions.

- **Fixed Assets**

The Fixed Assets module provides the possibility to analyze fixed assets value, inventory and depreciation.

- **Bank Account**

The Bank Account module enables balance, credit, and debit analyses of multiple bank accounts across multiple currencies, companies and data sources.

- **Service Management**

The Service Management module extends to the Microsoft Dynamics NAV Service Management module and enhances it with powerful KPIs.

Dimension

Grouping the data into similar fields of interest, Dimension is the cubes' key building block. Based on the columns of tables in the data source view, Dimension is a set of attributes that are linked to the dimensional table columns in the presentation area of the data warehouse. They can exist independently of cubes. The same dimension can be used in multiple cubes and can be used multiple times in same cube. A dimension that exists independently of a cube is referred to as an analytical dimension. Instance of a dimension inside the cube is called cube dimension. A dimension can contain a hierarchy that is used to organize cube measures.

3.3 Interface

The BI4Dynamics platform was developed on top of the .NET framework (v4.5) using C# programming language. Employing XAML, an XML-based language, to define and link various interface elements, the WPF (Windows Presentation Foundation) graphical subsystem provides the user interface. Considering this is a pretty big application, herein we shall only touch upon the most essential part of this application: the Installation Wizard.

3.3.1 Installation Wizard

The Installation wizard is an automated package within the application that serves as a tool to simplify the whole installation and deployment process. It breaks the process down into 6 individual consecutive steps:

1. **License step**

Here the user enters the license key provided upon purchasing the solution. The application connects to our licensing server and verifies the license key.

2. Instance step

Here the user enters Name of the instance (the application supports multiple instances), selects the language, selects SQL and Analysis server and authentication type to connect to those servers as well as enters the name of the data warehouse and analytical database.

3. Data Source step

Here the user adds one or more NAV data sources. These are defined by SQL server, database name and Navision version.

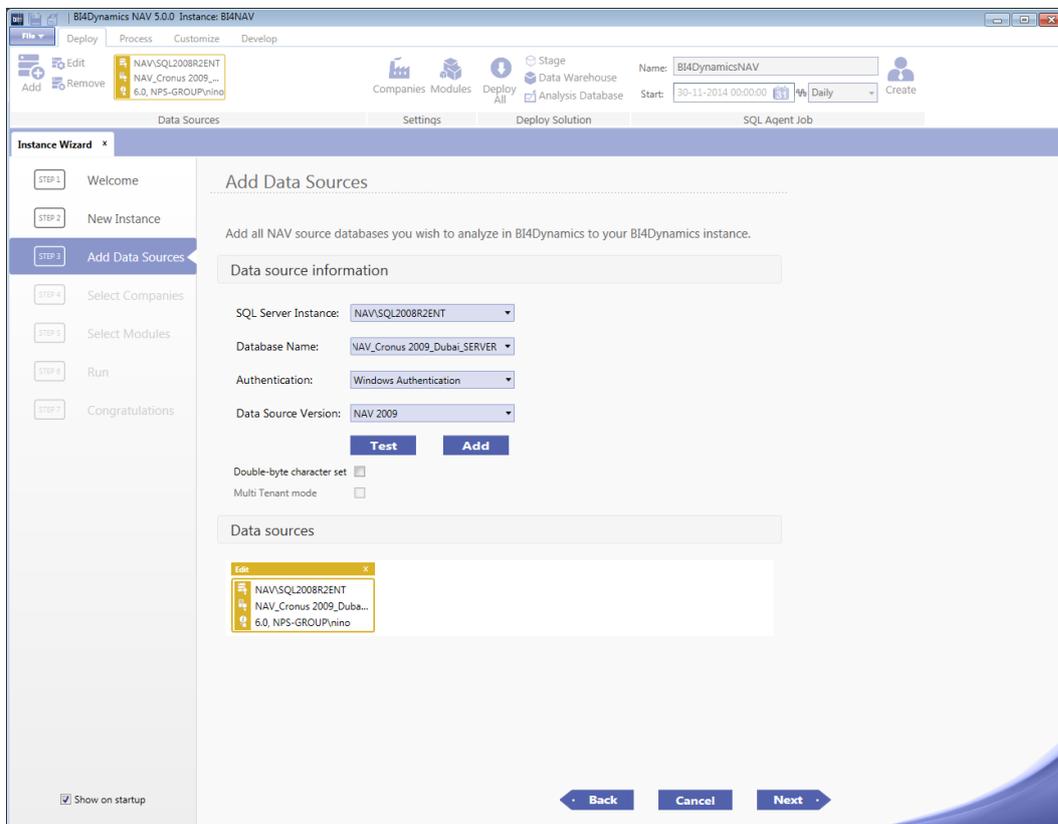


Figure 3.8: Data Source Step.

4. Company step

Here the user selects one or more companies inside the added NAV data sources, local and additional currency and adds 1-8 global dimensions (A global dimension is a dimension that can be used as a filter anywhere in Microsoft Dynamics NAV).

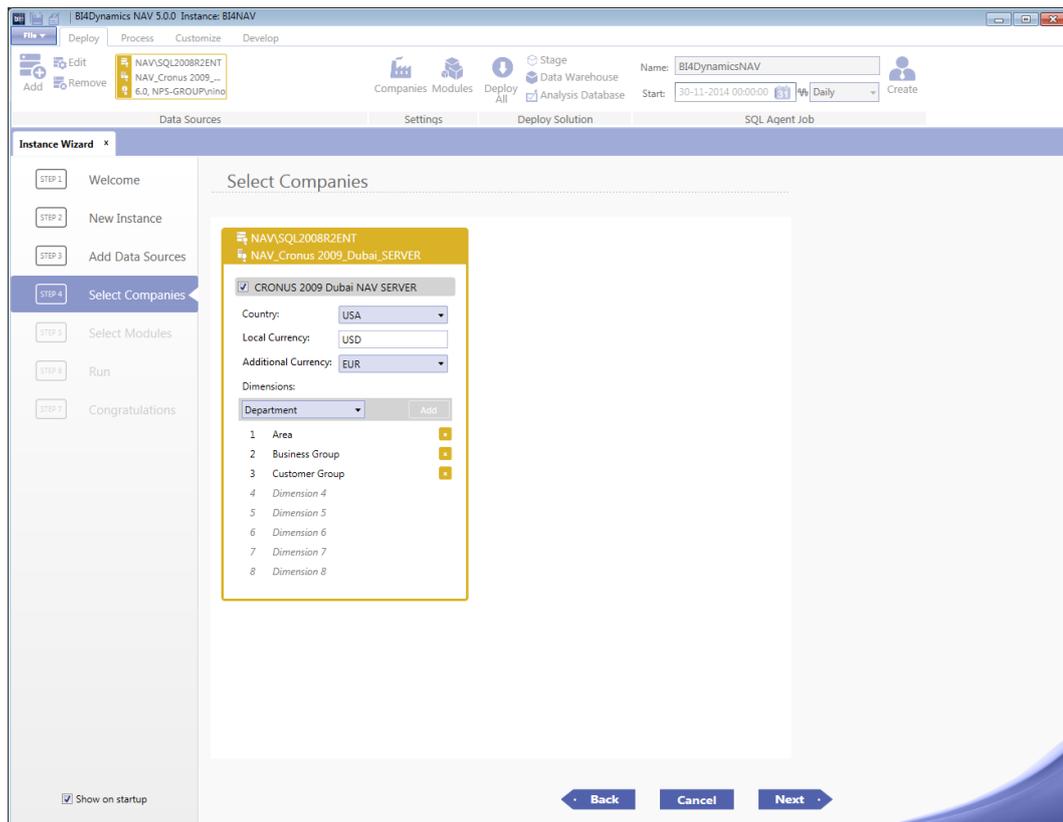


Figure 3.9: Company Step.

5. Modules step

Here the user selects their desired modules (described in the previous section) each corresponding to one business area which the user wants to analyze, choosing between standard and vertical solutions.

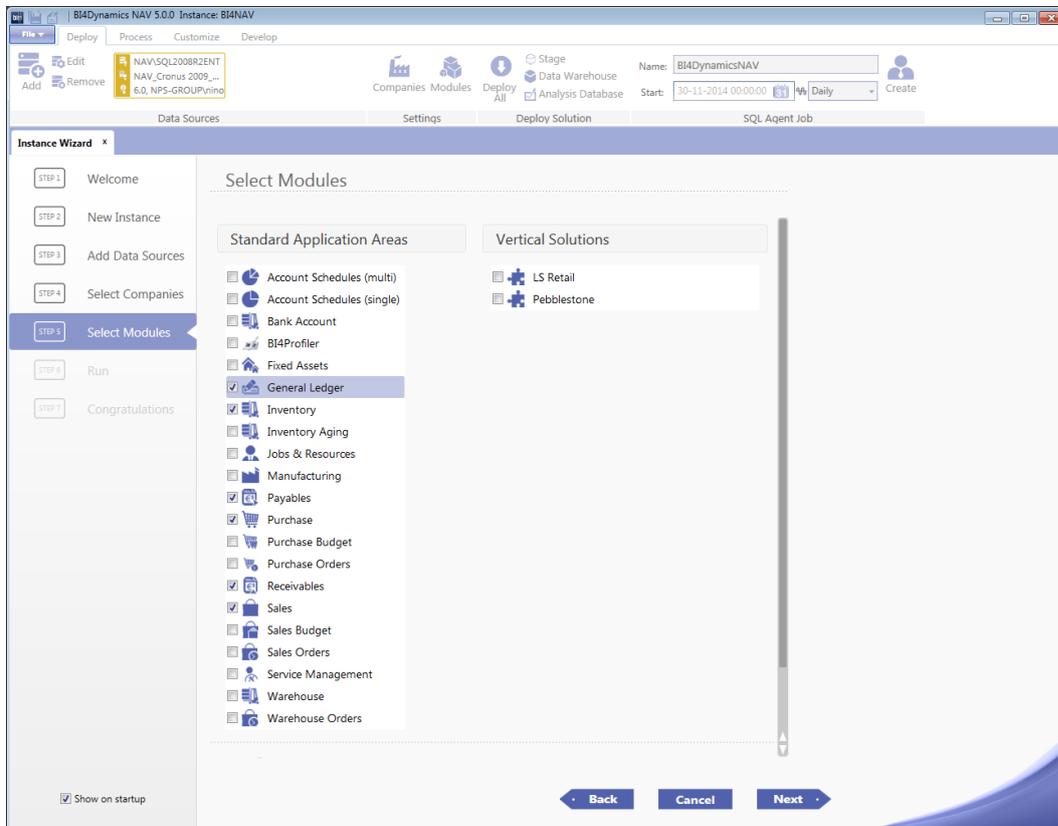


Figure 3.10: Modules Step.

6. Run step

This is the final step. Here the user can select/deselect whether they only want to deploy data or process it as well. An SQL Agent job can also be created which, once is set up, automatically executes this process daily, weekly or monthly, depending on users' requirements.

Chapter 4

BI4Profiler

The BI4Profiler is an especially designed module that captures and stores data in relation to performed daily activities (such as processing time, disk size, RAM etc.) within the SQL server while processing BI solution. This information is used for sizing and performance optimization which is required in the corporate environment.

4.1 Module

The true power of BI4Dynamics lies in its platform-like architecture that provides an environment for developing custom modules. The platform is based on SQL therefore it can be fully customized. Besides the previously mentioned standard cubes, so-called verticals, custom modules developed by our partners and later implemented into an internal module repository are also offered. Most renowned among these are **LSRetail**¹ and **Pebblestone**².

So what's a module? A module is a basic building block of BI4Dynamics platform.-It's basically an abstraction of a OLAP object and all of its dependencies on a data warehouse and data source.

¹LSRetail® is a NAV add-on that covers Point of Sale (POS) to Back-Office and Head-Office Functionality

²Pebblestone Fashion® is the leading NAV solution for companies in the fashion industry

There are three types of module in the BI4Dynamics's architecture: *dimension module*, *cube module* and *framework module*.

The dimension module and cube module are almost identical and they consist of:

- **TAC file**

TAC is the Tables And Columns file which is an XML document containing all the source tables and columns needed for the module. This file is the basis for building the staging area and is used for filling *setup.DatasourceTables*, *setup.DataSourceColumns* and *setup.DataSourceIndex* tables.

- **DataWarehouse objects**

All previously mentioned objects, such as *CreateTable procedure*, *Transformation View 1* (*Transformation View 2* for fact tables) and *Load Procedure*, used for building the presentation area of the data warehouse are stored as SQL scripts in this XML container file.

- **OLAP objects**

OLAP objects are also stored in an XML container file as AMO (Analysis Management Objects) providing a complete library of programmatically accessed objects. This enables an application to manage a running instance of Microsoft SQL Server Analysis Services.

- **Deployment file**

Stored in an XML container file, the Deployment file is an XML document which defines priority in deploying data warehouse and OLAP objects to servers.

- **ProcessFlow file**

The ProcessFlow file is an XML document in an XML container file that defines priority in processing objects deployed to servers.

The Framework module is slightly different in that it does not contain any TAC file or OLAP objects. Instead of usual Data Warehouse objects (CreateTable etc..) the Framework module contains all the objects necessary for deploying the Setup Area (addressed in the Setup section). Besides these objects it also contains a stored *dbo.DropObject* procedure used for deleting custom objects in the Data Warehouse.

All four module components are stored in an XML container file which enables the versioning of the objects. Each XML container file holds multiple instances of an object that can be selected on the basis of the following parameters:

1. NAV Datasource version
2. SQL Server version
3. SQL Server type
4. Analysis version

Developing a custom module is undertaken in 3 steps:

- Importing necessary tables and columns into the staging area of the data warehouse, which is accomplished through an interface inside the application
- Developing ETL procedures - Transformation views, any necessary additional help tables and load procedure
- Developing dimensions and cubes on an analytical database (can be carried out with an OLAP compatible tool)

4.2 Development

Upon commencing the design of the BI4Profiler we faced the challenge of how to make the module totally independent. When you customize any part of standard or custom modules, either on Data Warehouse or OLAP, you have to redeploy new objects; this means that the database gets dropped and recreated, and you lose all the history. However, we wanted to make a module capable of holding all the history, unless there was an explicit demand that history was to be deleted. Another major issue that arose is the processing of OLAP objects. The analysis server supports the processing of single objects from the AMO API used in our .NET framework, but in order to obtain coherent data when customizing for example a single dimension you first have to process the dimension itself and then all the cubes that use this shared dimension in order to rebuild the indexes that are used in a star schema connecting dimensions to the fact table. And if you customize two dimensions, you would have to accomplish this in the right order. This means that all the dependencies and priorities must be implemented internally, in itself a time consuming approach since the Analysis server implements all such logic with one single AMO API call: *Process Analysis*.

The solution? We implemented a separate data warehouse and analytical database for this module. This "SYSTEM" database is called *NameOfTheDataWarehouse_SYSTEM*. For now its only purpose is to isolate the BI4Profiler module from the standard data warehouse and analytical database, but in future versions the plan is to entirely move the BI4Dynamics Setup area into this database. Both the SYSTEM data warehouse and analytical database are deployed upon first deployment of the BI4Dynamics standard data warehouse and analytical database. From initial deployment onwards, the SYSTEM data warehouse and analytical database are not dropped or redeployed unless the user connects to the SQL server and drops them manually. As with the BI4Dynamics data warehouse, the SYSTEM data warehouse consists of three areas: *Setup area*, *Staging Area* and *Presentation Area*.

4.2.1 Setup Area

The Setup Area of the SYSTEM data warehouse contains 2 objects: the *dbo.DropObject* and *setup.InitalizeDimension* stored procedures (both of which are described in previous sections). The Setup Area introduces 3 new schema:

- *p_stage.** used for the *Staging Area*
- *p_dim.** used by dimensions in the *Presentation Area*
- *p_fact.** used by facts in the *Presentation Area*

4.2.2 Staging Area

p_stage.Batch

Every record in this table corresponds to one process run. Thus, each time a user or SQL agent processes the data, a new record is provided in this table. Below is a tabular 4.1 presentation of *p_stage.Batch* structure.

Column	Description
BatchID	Automatically generated ID
ProcessType	0 - Full 1 - Incremental
RunTag	0 - ProcessAll 1 - ProceStage 2 - ProcessDatawarehouse 3 - ProcessAnalysisDatabase
AutoProcess	0 - Manualy from UI 1 - SQL Agent Job
StartDateTime	Starting time stamp
EndDateTime	Ending time stamp

Table 4.1: *p_stage.Batch* structure.

p_stage.ProcessFlowStatistics

This table forms the heart of the BI4Profiler module. It is responsible for filling the most important part of the BI4Profiler data warehouse presentation area which is later used for the majority of calculations and reports. It also holds all the statistical data for each stored procedure executed in a particular batch. Where this data comes from shall be discussed in the later chapters, but for now let's just focus on its content and structure:

- **ProcessFlowName**

BI4Dynamics supports multiple process flows. This column contains the process flow name used in the current run.

- **DatabaseObjectName**

Name of the executed stored procedure.

- **RefDatabaseObjectName**

Stored procedures for creating and filling the staging area - such as *setup.LoadStage* - are executed with parameters. This is the name of the stage table passed as a parameter for specific procedures; for every other stored procedure it remains empty.

- **DatabaseName**

Name of the database currently being processed.

- **ExecutionStartDate**

Starting time stamp of the execution of a stored procedure.

- **ExecutionEndDate**

Ending time stamp of the execution of a stored procedure.

- **ExecutionTime**

Duration of the execution in milliseconds (ms).

- **CpuTime**

Effective time stored procedure spent on the CPU, as opposed to waiting for resources.

- **GrantedMemoryKb**

Highest granted memory, in kilobytes.

- **DeletedRecordsCount**

Number of records which the stored procedure deleted.

- **InsertedRecordsCount**

Number of records which the stored procedure inserted.

- **UpdatedRecordsCount**

Number of records which the stored procedure updated.

- **DbSpaceUsed**

Size of the database at the time of current batch run.

- **LogPhysicalSpaceUsed**

Physical size of the log at the time of current batch run.

- **LogLogicalSpaceUsed**

Logical size of the log at the time of current batch run.

- **BatchID**

ID of the corresponding batch run.

- **CallStartDate**

Time stamp of the call of stored procedure.

- **CallEndDate**

Time stamp of the return from call of stored procedure.

4.2.3 Presentation Area

Dimension Tables

Most of the dimensional tables are filled with the data from the Object Catalog Views (system views which provide database metadata such as list of tables, stored procedures etc.). Let's look at the list of dimensional tables and what they represent:

- **p_dim.Batch**

This dimensional table is a copy of the *p_stage.Batch* table with some minor descriptive transformations.

- **p_dim.Date**

System generated date dimensional table.

- **p_dim.DB**

List of server databases, from *sys.databases*

- **p_dim.DBFile**

List of log and database files (for each database) from *sys.master_files*

- **p_dim.DBIndex**

List of all indexes on the server (for each table) from *sys.indexes*

- **p_dim.DBProc**

List of all stored procedures on the server from *sys.procedures*

- **p_dim.DBTable**

List of all tables on server from *sys.tables*

- **p_dim.ProcessArea**

List of all process areas in *p_stage.ProcessFlowStatistics*

- **p_dim.ProcessFlow**
List of all of process flow names in *p_stage.ProcessFlowStatistics*
- **p_dim.ProcessType**
List of all process types in *p_stage.ProcessFlowStatistics*
- **p_dim.Volume**
List of all volumes on the server used by data and log files.

Fact Tables

As with dimensional tables, most of the fact tables are filled with data from Object Catalog Views. Their rows basically present a snapshot of the statistics taken at the time.

- **p_fact.DBFileStatistics**
Snapshot of Database statistics, such as Size in MB, a combination of data from *sys.master_files*, *sys.databases*.
- **p_fact.IndexStatistics**
Snapshot of Index statistics, such as Average Index Fragmentation as %, Total Space in KB, Used Space in KB, etc., a combination of data from *sys.dm_db_index_physical_stats*, *sys.dm_db_index_usage_stats*, *sys.tables*, *sys.schemas*, *sys.indexes*, *sys.partitions*, *sys.allocation_units*.
- **p_fact.ProcessStatistics**
Main table containing all the statistics from *p_stage.ProcessFlowStatistics* described above.
- **p_fact.TableStatistics**
Snapshot of Table statistics such as Row Count, Total Data in KB, Index Size in KB (on the table), Data Used in KB, Index Used in KB. A combination of data from *sys.tables*, *sys.schemas*, *sys.indexes*, *sys.partitions*, *sys.allocation_units*.

- **p_fact.VolumeStatistics**

Snapshot of Volume statistics, such as Drive Letter, Total Space in MB and Available Space in MB A combination of data from *sys.master_files*, *sys.dm_os_volume_stats*.

4.3 Putting it all together

Each dimension is a separate module, which means that it contains its own deployment and process file, data warehouse objects (*CreateTable procedure*, *Transformation View 1* and *Load Procedure*) and analytical object.

Fact tables, however, form part of the BI4Profiler module, which means that they reside in the BI4Profiler's deployment and process flow. Each fact table contains a *CreateTable procedure*, *Transformation View 1* and *Load Procedure*.

Although they also form part of the BI4Profiler module, the Setup and Staging area deployment files have a priority section, meaning that they are deployed first. The process order is similar to the deployment order, though only the load procedures for dimensions and facts are executed; create table procedures are executed when the BI4Profiler is deployed:

1. **BI4Profiler Priority Section**

- (a) `dbo.DropObject`
- (b) `dbo.InitializeDimension`
- (c) `p_stage.TruncateProcessFlowStatistics`
- (d) `p_stage.ProcessFlowStatistics`
- (e) `p_stage.Batch`

2. **Dimensions**

First is the *CreateTable procedure* then *Transformation View 1* and lastly *Load Procedure* for each dimension.

(a) Batch

For example:

- i. p_dim.CreateTableBatch
- ii. p_dim.BatchView
- iii. p_dim.LoadBatch

(b) Date

(c) DB

(d) DB File

(e) DB Index

(f) DB Proc

(g) DB Table

(h) Process Area

(i) Process Flow

(j) Process Type

3. Facts

The *Transformation View 1* is first deployed for all Facts, then the *CreateTable procedure* and lastly *Load Procedure*.

In the following order:

- (a) ProcessStatistics
- (b) TableStatistics
- (c) IndexStatistics
- (d) DBFileStatistics
- (e) VolumeStatistics

For example:

- (a) p_fact.ProcessStatisticsView
- (b) p_fact.TableStatisticsView
- (c) p_fact.IndexStatisticsView
- (d) ...

Most of the BI4Profiler users are our partners or advanced users who are developing complicated custom cubes, dimensions or customizing standard modules. They use the BI4Profiler when optimizing their scripts and usually focus on reducing execution times, index fragmentation as well as index and table size.

Below is the most commonly used report in Excel:

The screenshot shows an Excel PivotTable report with two tables. The upper table displays process statistics, and the lower table displays table statistics. The PivotTable Field List on the right shows the configuration for the report.

Row Labels	Exec Time - hh:mm:ss	Exec Time - sec	Net Exec Time - hh:mm:ss	Net Exec Time - sec	Granted Memory KB
Incremental Update	00:04:40	280,32	00:04:02	242,14	4.154.968
BI4Dynamics	00:04:40	280,32	00:04:02	242,14	4.154.968
Batch: 1, Start: 29.11.14 19:57:34,					
End: 20:02:14	00:04:40	280,32	00:04:02	242,14	4.154.968
1 - Stage	00:01:30	90,40	00:01:24	84,12	248.056
setup.DeleteStage	00:00:00	0,55	00:00:00	0,14	
setup.IndexStage	00:00:33	33,32	00:00:28	28,21	70.656
setup.LoadStage	00:00:56	56,53	00:00:55	55,78	177.400
2 - Data Warehouse	00:02:13	133,24	00:01:41	101,34	3.906.912
3 - Cubes	00:00:56	56,68	00:00:56	56,68	
Grand Total	00:04:40	280,32	00:04:02	242,14	4.154.968

Row Labels	Data Total KB	Data Used KB	Index Total KB	Index Used KB	Row Cnt
Batch: 1, Start: 29.11.14 19:57:34,					
End: 20:02:14	1.222.648	1.217.520	665.712	653.320	13.151.424
dim.BusinessUnit	16	16			11
dim.ClosedPeriod	16	16	16	16	3
dim.Company	16	16			3
dim.Currency	40	40	16	16	154
dim.Customer	200	160	16	16	214
dim.CustomerPostingGroup	16	16	16	16	7
dim.DataSource	16	16			1
dim.Date	2.568	2.552	64	64	2.558
dim.Dimension1	64	64	16	16	63

Figure 4.1: The upper table shows process statistics, while the lower table shows table statistics.

4.4 Filling the void

This paper has still to address a most important issue: where does the data come from?

When designing the BI4Profiler we had one single goal in mind: Cover all the angles of benchmarking as thoroughly and accurately as possible, especially in consideration of the fact that this was a network distributed solution. When dealing with network solutions it is necessary to consider and provide contingency for all the latency which might occur due to increased unrelated network traffic and stored procedural call initialization within the application. With all this in mind, it was decided to use a hybrid approach using our own internal benchmarking in combination with DMVs (Dynamic Management Views).

4.4.1 Dynamic Management Views

With the advent of the Dynamic Management Views (DMVs) in SQL Server 2005, Microsoft vastly expanded the range and depth of metadata that could be exposed in relation to connections, sessions, transactions, statements and processes that are, or have been, executed against a database instance. These DMOs provide insight into the resultant workload generated on the server, where the pressure points are, and so on, and are a significant and valuable addition to the DBA's troubleshooting armory.

We used the following two DMVs:

- **sys.dm_exec_procedure_stats**

This DMV returns aggregate performance statistics for cached stored procedures. The view returns one row for each cached stored procedure plan, and the lifetime of the row is as long as the stored procedure remains cached. When a stored procedure is removed from the cache, the corresponding row is eliminated from this view. The view contains approximately 30 columns but for the first version we only decided to use just 4.

Column	Description
last_execution_time	Last time at which the stored procedure was executed.
last_elapsed_time	Elapsed-time, in microseconds, for the most recently completed execution of this stored procedure.
last_worker_time	CPU time, in microseconds, that were consumed the last time the stored procedure was executed.
dbname	This column is actually not part of the view, but it is returned as a part of the query. As the name suggests it returns the name of the database: DB_NAME(<i>databaseid</i>)

Table 4.2: Columns we used from sys.dm_exec_procedure_stats.

- **sys.dm_exec_query_memory_grants**

This DMV returns the memory stored procedure currently requested and granted to it. However, the issue with this DMV is that it's not cached as the previous one. Thus, in order to get the highest memory grant, the SQL server has to be polled the entire time this stored procedure is executing. In order to achieve this we implemented a separate thread which polls the server for this DMV. This thread is created prior to the initial call of the stored procedure, and while running it maintains the highest memory grant. When the original stored procedure execution ends, it also terminates the thread. Upon termination the thread combines the memory grant with the stats returned by the previous DMV and saves them into the *p_stage.ProcessFlowStatistics*.

One minor issue was that *sys.dm_exec_procedure_stats* DMV was introduced with SQL Server 2008, which meant that there was no backward compatibility. This is where our internal benchmarking approach comes in. When the polling thread is initialized, just before the stored procedure call, the time stamp (CallStartDate) is saved; when the thread is terminated the ending time stamp (CallEndDate) is again saved.

The time between these two time stamps is a turnaround time from the application's perspective, and indeed during this time another stored procedure cannot be called and executed. Turnaround time contains lead time (latency due to network traffic, waiting on resources etc.) and the actual processing time. Since

sys.dm_exec_procedure_stats DMV is not supported on older SQL servers we at least have the turnaround time.

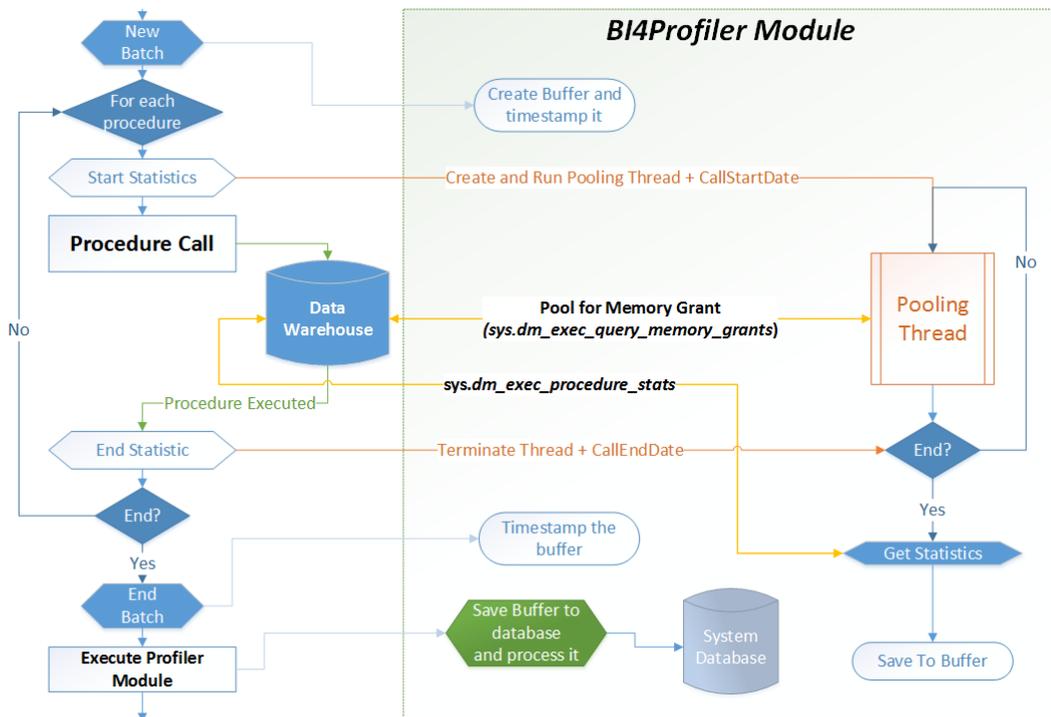


Figure 4.2: BI4Profiler Core Architecture.

Chapter 5

Conclusion

With the development of the BI4Module, testing, benchmarking and optimization processes were improved tremendously, at the same time we also created a strong selling point for our marketing and sales department. Even though this is the first version of the module, we have already benefited enormously from it. For example, after refactoring scripts for standard cubes when testing the incremental update, we noticed that there was a bug in one of the load scripts which dragged down overall performance of the scripts by 20%. This slowdown may not be much re the relatively small databases used in our testing environment, but when using such scripts in a real-life production database up to 300 GB in size, we are looking at few hours difference.

We also plan to port this module to our AX product line, and in the next version we are also looking to add more measures and additional dimensions. Now that we have a strong core, our only limitation is the power and flexibility of the DMVs. Microsoft also upgrades DMVs with every new SQL version, and this can be a little tricky at times since we have to ensure backward compatibility.

One of the next features shall definitely be a set of precompiled most commonly used reports, provided in much the same way as the reports for standard cubes we offer. This shall ensure that the general user is provided an out-of-the-box module solution ready for analysis.

For advanced users - including us at BI4Dynamics - we plan to further explore *sys.dm_exec_query_stats* which contain performance statistics for cached query plans. Breaking down the stored procedure into several separate queries will ensure even greater control and insight re the optimization process.

List of Figures

2.1	Business Intelligence Technologies.	10
3.1	BI4Dynamics Transformation Process	11
3.2	BI4Dynamics Data Model	13
3.3	The Star Schema	20
3.4	OLAP Cube	24
3.5	Slicing	26
3.6	Dicing	26
3.7	Drill Down	27
3.8	Data Source Step	32
3.9	Company Step	33
3.10	Modules Step	34
4.1	BI4Profiler Report	46
4.2	BI4Profiler Core Architecture	49

List of Tables

3.1	Setup Tables and Procedures	14
4.1	p_stage.Batch	39
4.2	sys.dm_exec_procedure_stats	48

Bibliography

- [1] Tzu, Sun. “The Art of War, Samuel B. Griffith, trans.” Oxford University Press 1 (1971): 963.
- [2] Codd, Edgar F. “A relational model of data for large shared data banks.” Communications of the ACM 13.6 (1970): 377-387.
- [3] Luhn, Hans Peter. “A business intelligence system.” IBM Journal of Research and Development 2.4 (1958): 314-319.
- [4] Devens, Richard Miller. “Cyclopaedia of Commercial and Business Anecdotes: Comprising Interesting Reminiscences and Facts, Remarkable Traits and Humors... of Merchants, Traders, Bankers... Etc. in All Ages and Countries...” D. Appleton, 1868.
- [5] Evelson, Boris, and N. Norman. “Topic overview: business intelligence.” Forrester Research (2008). More at:
<https://www.forrester.com/home/>
- [6] Evelson, Boris. “Want to know what Forrester’s lead data analysts are thinking about BI and the data domain?.” (2010). More at:
<https://www.forrester.com/home/>
- [7] Henschen, Doug. “Analytics at Work; Q and A with Tom Davenport (Interview).” (2010). More at:
<http://www.informationweek.com/news/software/bi/222200096>

- [8] Kimball, Ralph. "The data warehouse toolkit: practical techniques for building dimensional data warehouse." New York, itd: Wiley (1996).
- [9] The OLAP Council. "OLAP and OLAP Server Definitions." (1995).
More at:
<http://www.olapcouncil.org/research/glossaryly.htm>
- [10] University of Alberta. "Glossary of Data Mining Terms." (1999). More at:
<http://webdocs.cs.ualberta.ca/zaiane/courses/cmput690/glossary.html>
- [11] Inmon, William H. "Building the data warehouse." John wiley & sons, 1992.