Modeling the Data Warehouse Layer with BI

Version 1.0
July 19, 2006

Table of Contents:

Data Warehouse Layer

Data warehousing has developed into an advanced and complex technology. For some time it was assumed that it was sufficient to store data in a star schema optimized for reporting. However this does not adequately meet the needs for consistency and flexibility in the long run. Therefore data warehouses are now structured using a layer architecture. The different layers contain data at differing levels of granularity. We differentiate between the following layers:

- Persistent staging area
- Data warehouse
- Architected data marts
- Operational data store

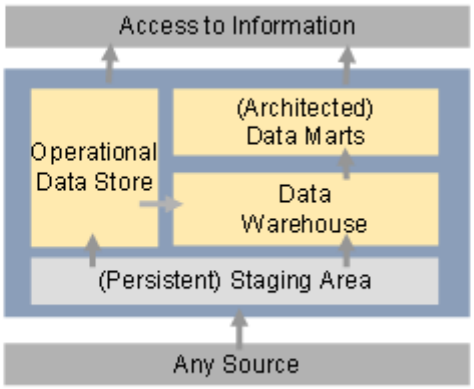


Figure 1 Conceptual Layers of Data Warehousing

The **data warehouse layer** offers integrated, granular, historic, stable data that has not yet been modified for a concrete usage and can therefore be seen as neutral. It acts as the basis for building consistent reporting structures and allows you to react to new requirements with flexibility.

1 Dos and Don'ts for Modeling a Data Warehouse Layer

- It is generally not recommended that you create a stovepipe data model with:
    - A direct dataflow from an extractor with document-type data to a highly aggregated data mart InfoCube
    - Proprietary creation of commonly-used central InfoObjects. It is recommended that you reuse central InfoObjects like business partner, product or company code.
- If you have a heterogenous source system landscape that comprises data from different components and systems: It is generally recommended that you create an intermediate consolidation layer within your data model (for example, within the dataflow from data source to a data mart InfoCube). The appropriate storage object for this layer is the DataStore object.

  See example: **full-blown content model** (see section 2.3 of this document)
- If you are extracting document-type data that is not preaggregated to BI: It is generally recommended that you build a data warehouse layer with DataStore objects where the data is stored in a slightly denormalized form at the most appropriate level of granularity.

  See example: **light-weighted content model** (see section 2.3 of this document)
- DataStore objects for a data warehouse layer should be modeled with the same granularity as the data that is delivered by the extractor:

- No aggregation of business-relevant data to retain information on operational level
- Slight denormalization is recommended: for example, header and item information from document-type data can be flattened into one extract structure (see section 4 of this document)
- Extractors for master and transactional mass data should be delta enabled

- Historical completeness of data to an appropriate extent where required: for example, adding a time element to the data.

  One example from SAP standard BI Content is the DataStore Object 0FIAR_O03 *FI-AR: Line Item*:

  - Financials documents are updated in BI when document field entries of non-key fields are changed: *status* and *clearing data*. The document *status* changes from *open* to *cleared* and the *clearing date* is set simultaneously when the document status is changed.
  - Thus without sending a separate change document, the changed information can be retained in the data warehouse.Thereby, for example, aging lists can be created with calculated business processing KPIs as a way of retaining the history of data changes.

2 Data Warehouse Layer (Enterprise Data Warehouse) in BI

2.1 Motivation and Benefits

Data warehousing provides data that is:

- Integrated as far as possible: master data is consolidated and master data is uniformly coded
- Consistent: central metadata models are shared to enable cross-application scenarios
- Historical: the history of the data is retained in dedicated data containers
- Complete: the data is not aggregated in dedicated data containers and is stored according to the granularity of the OLTP data

Organizations and businesses with multiple BI implementations and a heterogenous source system landscape face the challenge of avoiding islolated, inconsistent, stovepipe data warehouse solutions with:

- Redundant data flows
- Redundant extractions
- Redundant data stores
- Redundant data models

If this redundancy is not controlled, it is difficult to achieve integrated consistent reporting on the data and metadata. Moreover, the whole administration of the data and metadata becomes more complex and expensive.

A company-wide **Enterprise Data Warehouse** (EDW) concept helps to address these challenges. It comprises aspects of:

- Data storage: a multi-layer concept for persistent data storage
- Data model: BI objects for each layer and their relationships
- System landscape: this is not discussed in this paper

The following sections concentrate on data modeling. They explain how you can implement a multi-layer concept while focusing on the data warehouse layer as an element of this concept.

2.2 Conceptual Layers of Data Warehousing with BI

The main motivation for a layer concept is that each layer has its own optimized structure and services for the administration of data within an enterprise data warehouse. Therefore each layer also requires its own metadata modeling limitations constraints (see Figure 1 **Conceptual Layers of Data Warehousing**).

a) Architected Data Mart Layer

- Analysis and reporting layer
- Common master data definitions (consolidated InfoObjects)
- Aggregated data
- Data manipulation with business logic, for example, calculation of process time KPIs (for example, delivery time variance)
- Modeled using InfoCubes or DataStore objects

b) Data Warehouse Layer (DWH Layer)

- Corporate information repository of EDW
- Historical completeness - different levels of completeness are possible: from availability of latest version with change date to change history of all versions
- No aggregation of reporting-relevant data; for example, document line-item granularity for document-type data
- Normally no reporting targets – exception: operational reporting on line items
- Modeled using DataStore objects
- Common master data definitions (consolidated InfoObjects) to retain cross-system integration of system-dependent master data
- Optional: separation into
    - Propagation tier: data source-dependent, primary foundation for applications
    - Integration tier: integrates data from different processes

      This separation into two tiers produces the **full-blown content model** (See example 'Global Spend Analysis' in section 2.3.2 of this document).

c) Operational Data Store Layer

- For operational list reporting
- Common master data definitions (consolidated infoObjects)
- Transaction-near data
- Optional: Near real-time access
- Modeled using DataStore objects

What are the benefits of a specific DWH layer? Customers expect a DWH layer embedded in their overall EDW strategy because it is used predominantly as:

- **Information hub** to distribute OLTP data from multiple source systems to BI targets and subsequently to SAP or non-SAP applications
- **Historical basis** for archiving OLTP data from multiple source systems in BI (timeframe 5-7 years)
    - storage of document version (actual version)
    - exceptional and case-dependent: storage of change history (for example, order change history)

- **Integration basis** to integrate OLTP data from multiple source systems or components; in many cases more than one layer of DataStore objects is necessary.

2.3 Modeling Examples

This section contains three modeling examples from BI Content of a DWH layer for transaction data.

2.3.1 Bank Analyzer

The data model comprises the data flow from the operational banking systems (for example, CML, AM) to an analytical solution like the Bank Analyzer. The inbound and outbound data part of the DWH layer is modeled using DataStore objects. This scenario emphasizes the integration and consolidation aspects of the DWH concept.

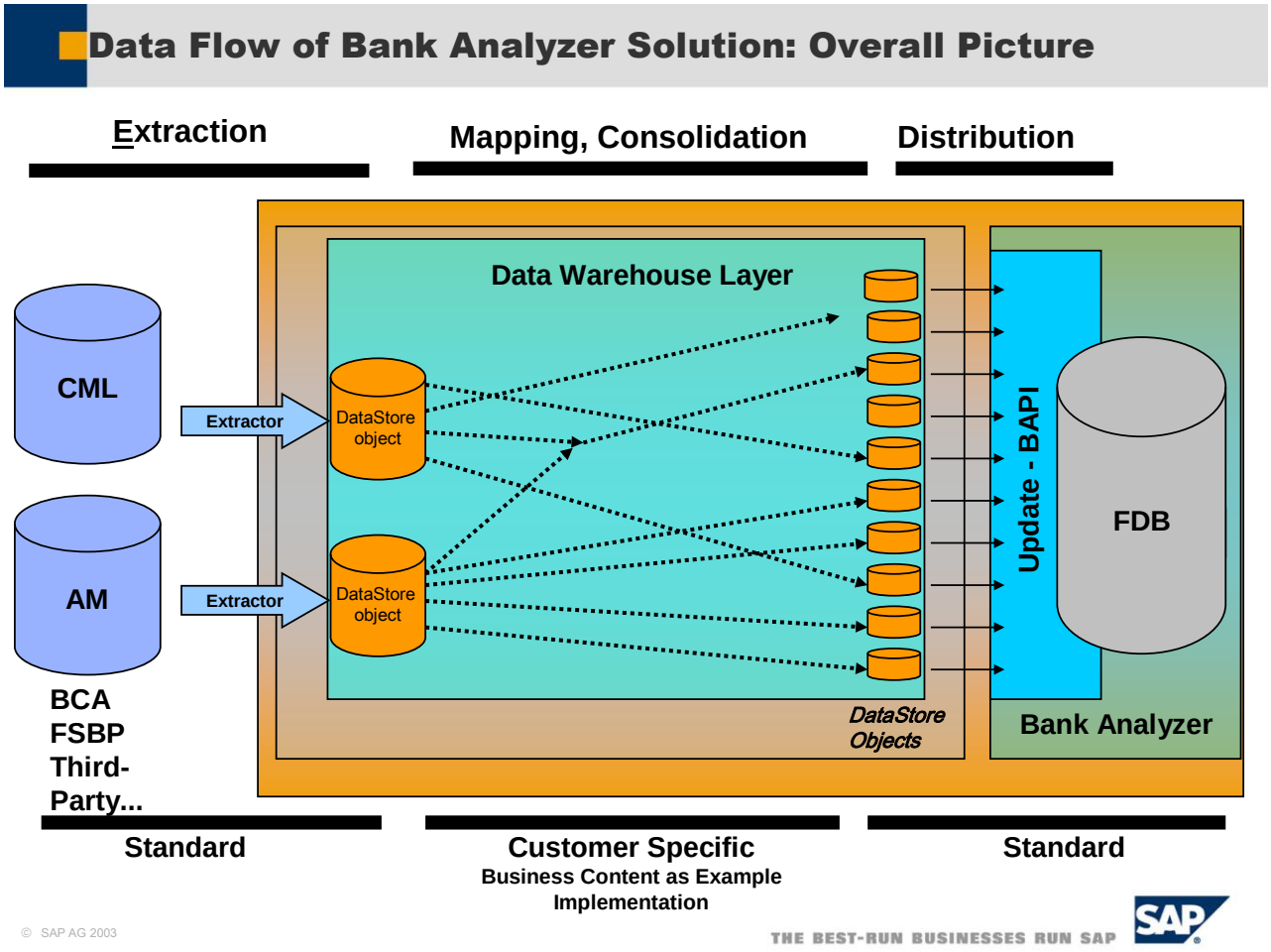


Figure 2 Data Flow of Bank Analyzer Solution in BI

In this example, the most important aspects of the DWH layer are:

- Integration of data from different operational finance systems for distribution to other applications (*integration basis* function)
- Storage of consolidated data and consistent metadata in BI: retain cross-system integration of system-dependent data (*integration basis* function)
- Data is distributed to subsequent non-BI data targets/analytical application, for example, Financial Database (FDB) / Bank Analyzer (*information hub* function)

2.3.2 SRM Global Spend Analysis

This scenario is used to analyze the expenditure of an affiliated group over all its companies and systems. A typical system landscape can include more than one SAP back-end system and enterprise buyer systems connected to a BI.

The data flow consists of invoice or purchase order data from the procurement systems (SRM or MM). Different DataStore objects contain data for each document type at line-item level. The detailed data is consolidated in a subsequent DataStore object. An InfoProvider designed for analytical reporting contains data from all enterprise buyer and purchasing systems that feed the prelimary DataStore objects. This scenario is an EDW example of a **full-blown content model** with DataStore objects**.**

In this example, the most important aspects of the DWH layer are:

- Integration of data from different procurement systems and the appliance of business rules accross two layers of DataStore objects (*integration basis* functionality)
- 1st layer: data is stored with document line-item granularity with no business rule manipulations. The delta method for the extractors is AIMD: after-image delta records with delete records; in the change log table of the DataStore objects, you can trace the history of after-image records; during data extraction, header and item data is combined into one OLTP structure.
- 2nd layer: flat-list operational reporting accross the whole purchasing process data at document level. Calculations using business logic are applied to generate KPIs such as *Delivery Time from Purchase Order to Confirmation*.

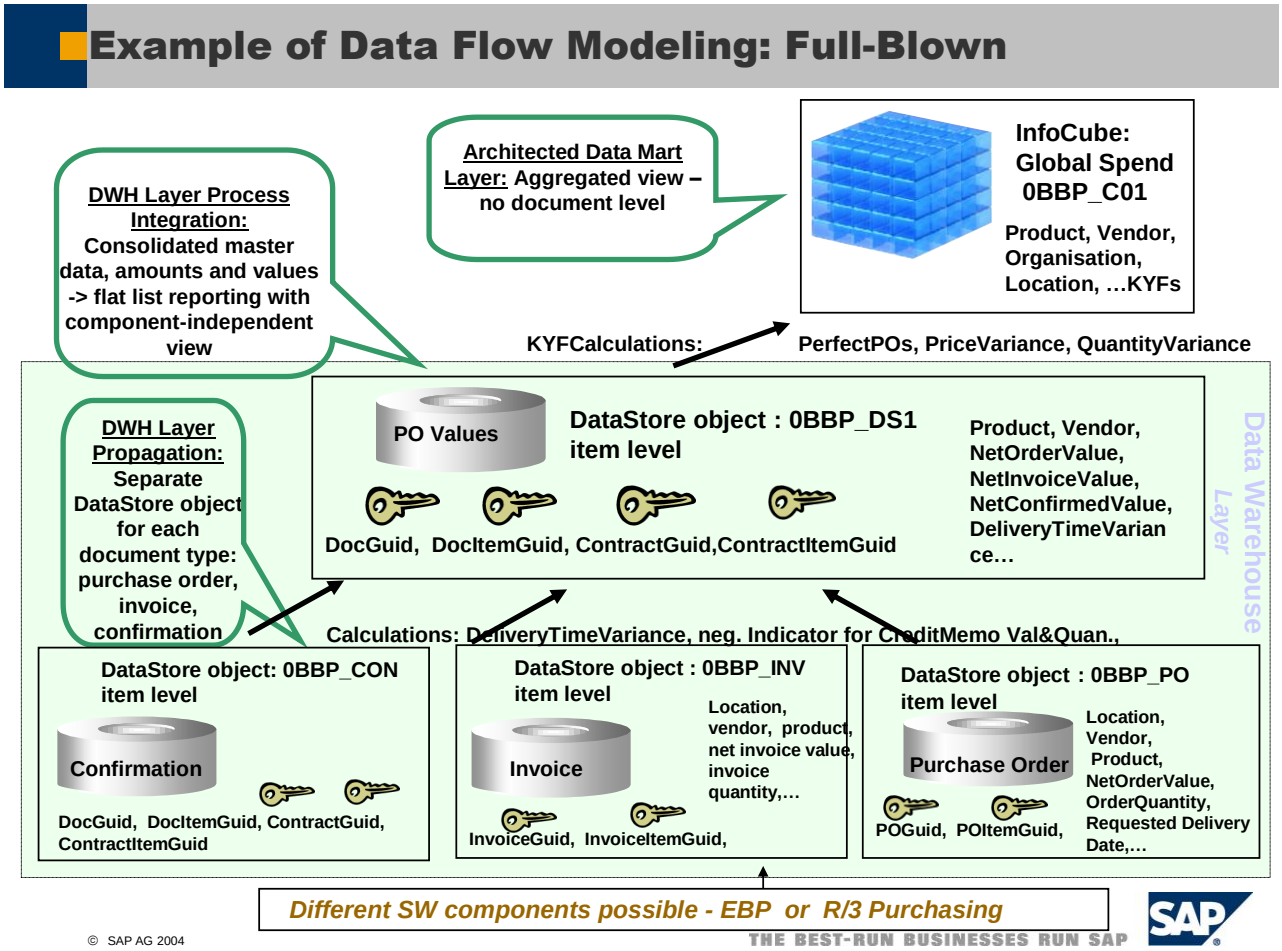


Figure 3 Data Flow of Full-Blown Content Model

2.3.3 CRM Sales Analysis

This scenario is an EDW example of a **light-weighted content model** with DataStore objects. That means that it is not necessary to integrate data from heterogenous source systems and complex processes are not required.

- Document header data and item data are extracted and stored with line-item granularity in DataStore object 0CRM_SALO
- The data structure is slightly denormalized: document header data and item data are extracted using one DataSource and are stored in one DataStore object
- Data is updated to the subsequent data mart InfoCube 0CSAL_C03 to enable OLAP analysis on the aggregated data

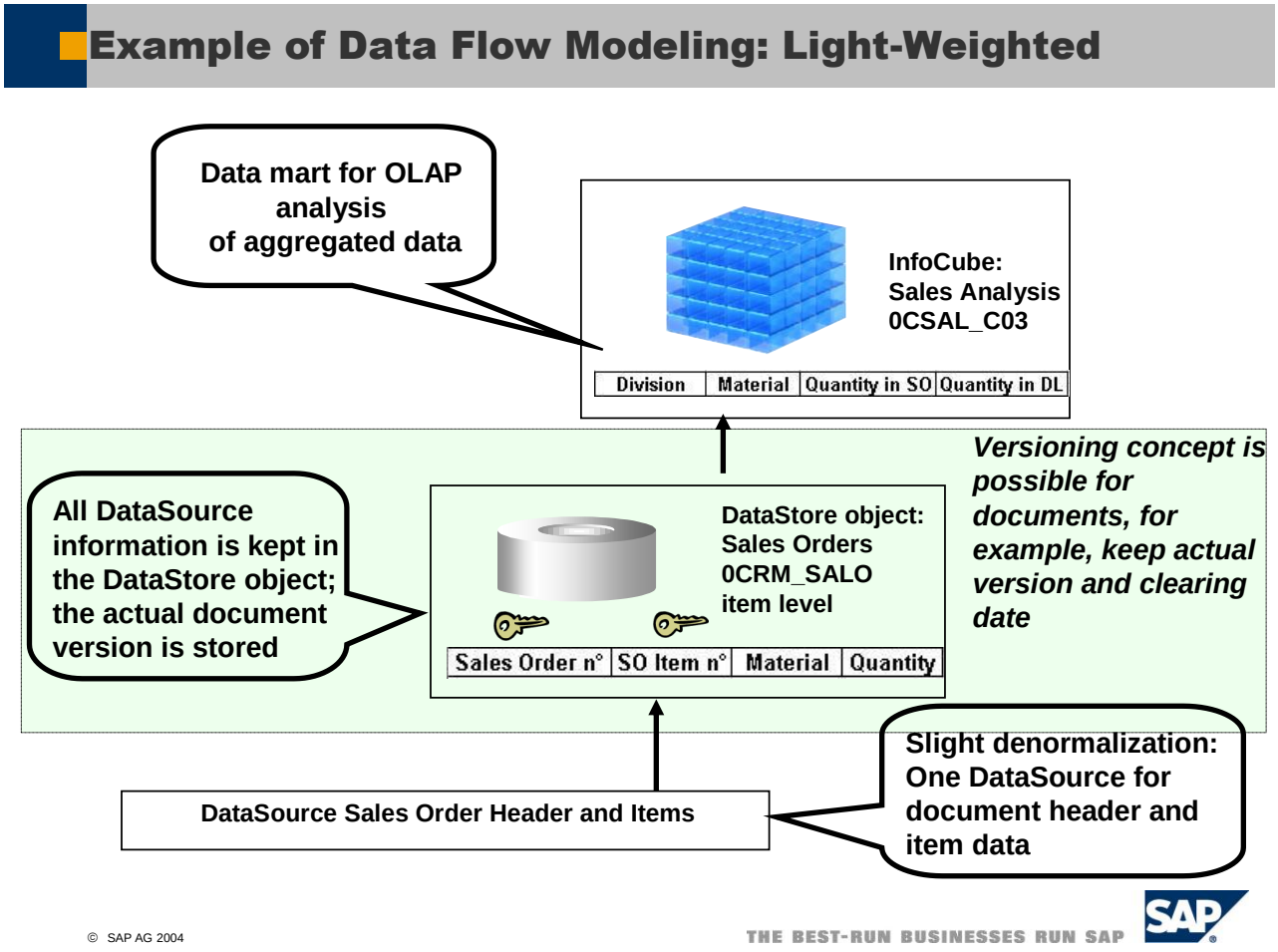


Figure 4 Data Flow of a Light-Weighted Content Model

3 Document-Type Data (Line Items) in DataStore Objects and InfoCubes

The following table compares the usage of DataStore objects and InfoCubes as data storage objects in BI systems. When should you load granular document-type data (line items) into a DataStore object and when should you use an InfoCube?

Line Items in DataStore Objects and InfoCubes

	DataStore object	InfoCube
Usage	Unification and consolidation of data in the data warehouse layer Determination of (additive) delta records that can be loaded into InfoCubes or master data tables in a subsequent step Operative reporting (when used in operational data store layer)	Aggregation and performance optimization for multidimensional reporting Analytic and strategic reporting
Type of data	Non-volatile data (when used in the data warehouse layer) Volatile data (when used in the operational data store layer) Transaction data, document-type data (line items)	Non-volatile data Aggregated data, totals records
Type of data upload	Overwrite/modify (in rare cases: add)	Add only
Data structure in BI	Flat and relational database tables, semantic key fields	Extended star schema (fact tables and dimension tables)
Reporting methods	Reporting at a high level of granularity, flat reporting Number of query records is greatly restricted by selecting qualified key fields Display a single document	Multidimensional reporting at a low level of granularity (OLAP analysis) Usage of InfoCube aggregates Drill-through to line items (stored in the DataStore object) using the report-report interface
Prerequisites	To update data to a DataStore object, the InfoSource should have the technical field '0recordmode'; the DataSource should not have the delta method 'D' or 'E'	No restrictions

Figure 5 DataStore Object Versus InfoCube for Line-Item Data

To summarize, it is recommend that you use **DataStore objects** to store document-type data (line items). You can also use DataStore objects as buffer storage for flexibly staging master data InfoObjects. In BI reporting, the key fields of the DataStore object should be filled before the DataStore object is read so that a single document or a limited number of documents are displayed. You usually require (secondary) indexes to access this subset of records so that you can avoid full table scans. You can define secondary indexes in DataStore object maintainance. If large volumes of data are stored in the DataStore object, it is not recommended that you run a complex analysis on the DataStore object for performance reasons. In a complex (multidimensional) OLAP query based on an InfoCube, the report-report interface is used to access the selected data for a single line item (stored in a DataStore object).

The **InfoCube** is suitable for storing data that is used in multidimensional OLAP queries. Since these queries select a larger number of data records, aggregates should be defined (based on the InfoCube) for all reporting-relevant questions. It is only recommended that you store line items in InfoCubes in exceptional cases where multidimensional reporting is required at document level (for example, OLAP navigation with respect to document number). In this case, the *document number* characteristic has to be defined within a line-item dimension of the InfoCube. Storing mass data in InfoCubes at document level is generally not recommended because when data is loaded, a huge SID table is created for the document number line-item dimension.

4 BI Data Models for Line Item and Header Information

Many document-type data models have a similar structure. In OLTP systems, for example, sales order, purchase order, delivery note and so on all have structure:

Header data and **item** data tables

This section provides rough ideas and suggestions for the best way to extract document-type data into BI and the most suitable BI data model. Four different BI data model scenarios are discussed. This section ends with a comparison and an evaluation of the different scenarios (pros and cons).

4.1 BI Data Model Scenarios

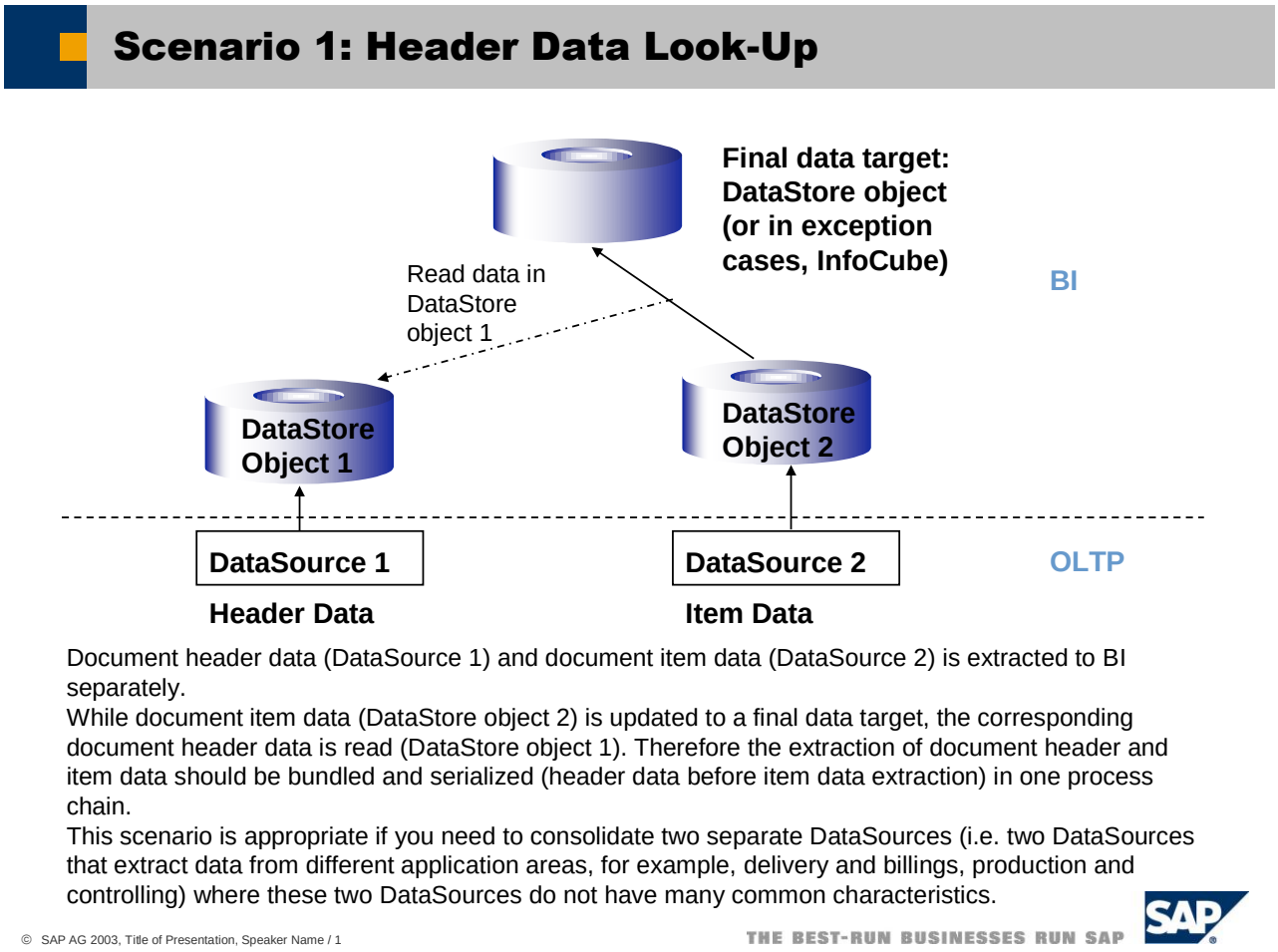
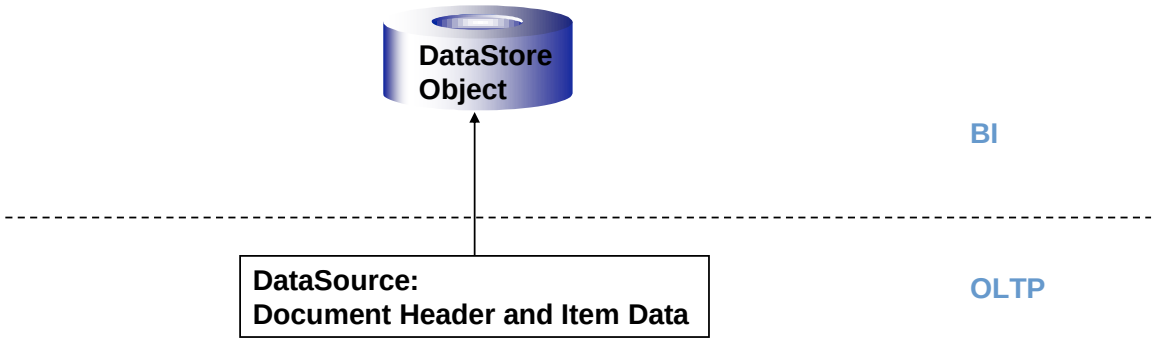


Figure 6 Header Data Look-Up

Scenario 2: Slight Denormalization



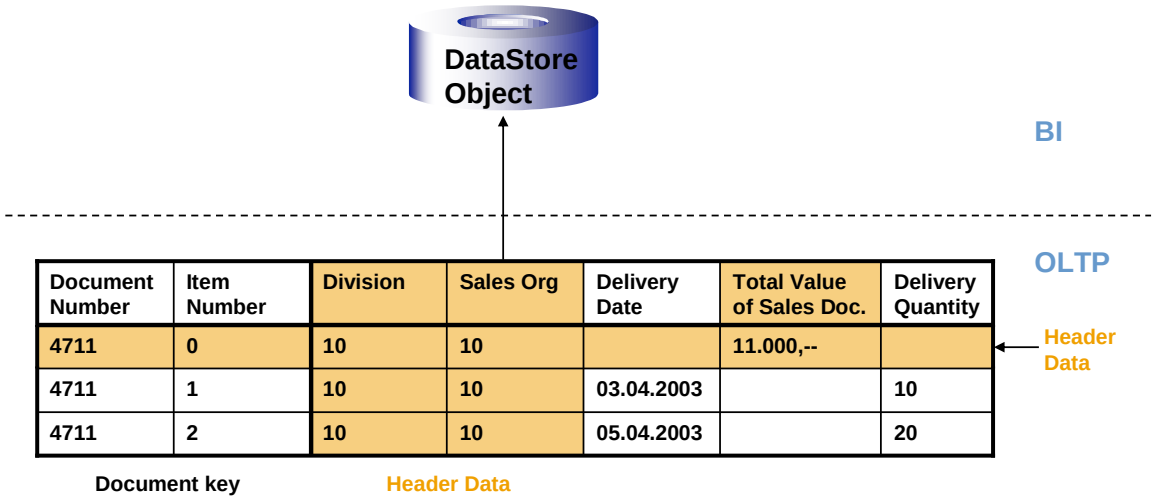
A single DataSource combines document header and item data. Data is transferred to BI data targets using one flat extract structure for both document header and item data.

This scenario is appropriate if you have two DataSources that are very closely connected, for example, if the data extracted using DataSource 1 changes, the data extracted using DataSource 2 is also likely to change.

A typical example is document header and document item DataSources: If the sales document is changed, the data extracted by both DataSources is likely to change at the same time. In this case, you can combine these two DataSource in one single DataSource in the OLTP system to ensure that the data is consistent.

Figure 7 Slight Denormalization

Scenario 2.1: Slight Denormalization (Incl. Header Data)



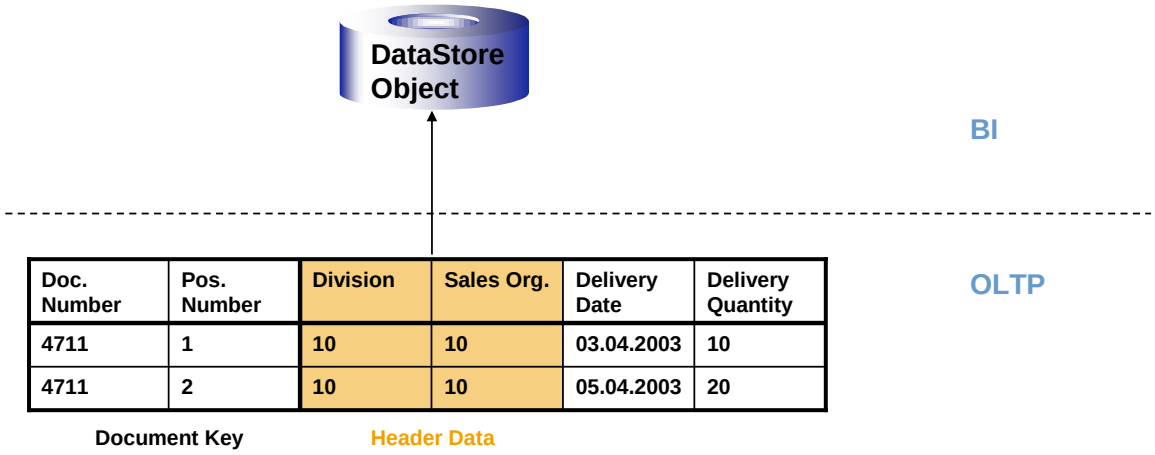
Document Number	Item Number	Division	Sales Org	Delivery Date	Total Value of Sales Doc.	Delivery Quantity
4711	0	10	10		11.000,--	
4711	1	10	10	03.04.2003		10
4711	2	10	10	05.04.2003		20

Scenario 2.1 is the same approach as scenario 2: flatten document header and item data into one extract structure.

If the document header contains header-specific key figures, for example, ‘total number of sales document items’ or ‘total value of sales document’, this header information should be extracted as an additional record with *item number* = 0.

Figure 8 Slight Denormalization with Header Data

Scenario 2.2: Slight Denormalization Without Header Key Figure

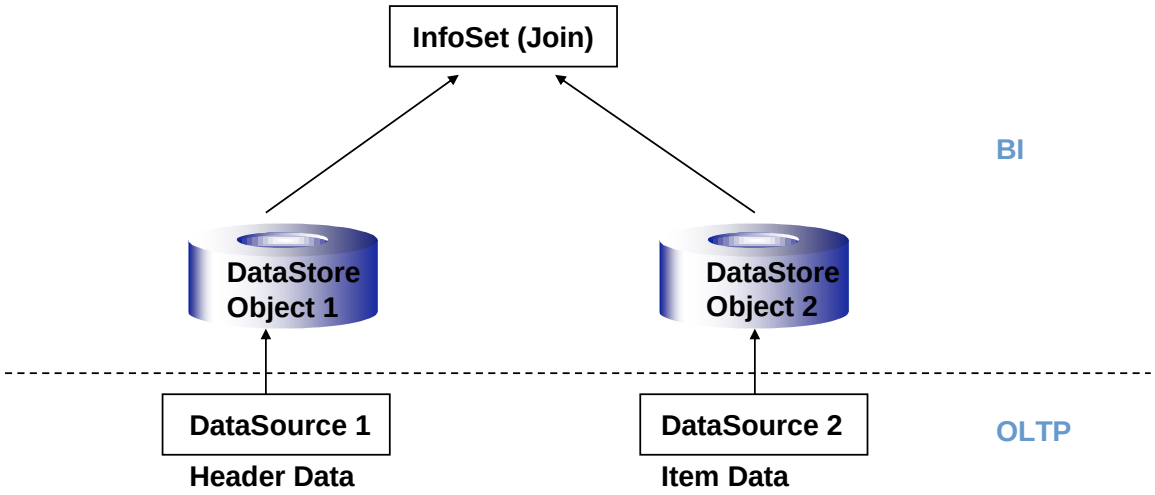


Scenario 2.2 is the same approach as scenario 2: flatten header and item data into one extract structure.

If the document header contains characteristics only (for example, division, sales organization) and does not contain header-specific key figures, these header characteristics should be extracted with the respective document-item records.

Figure 9 Slight Denormalization Without Header Key Figure

Scenario 3: Header and Item Data Join Using InfoSet

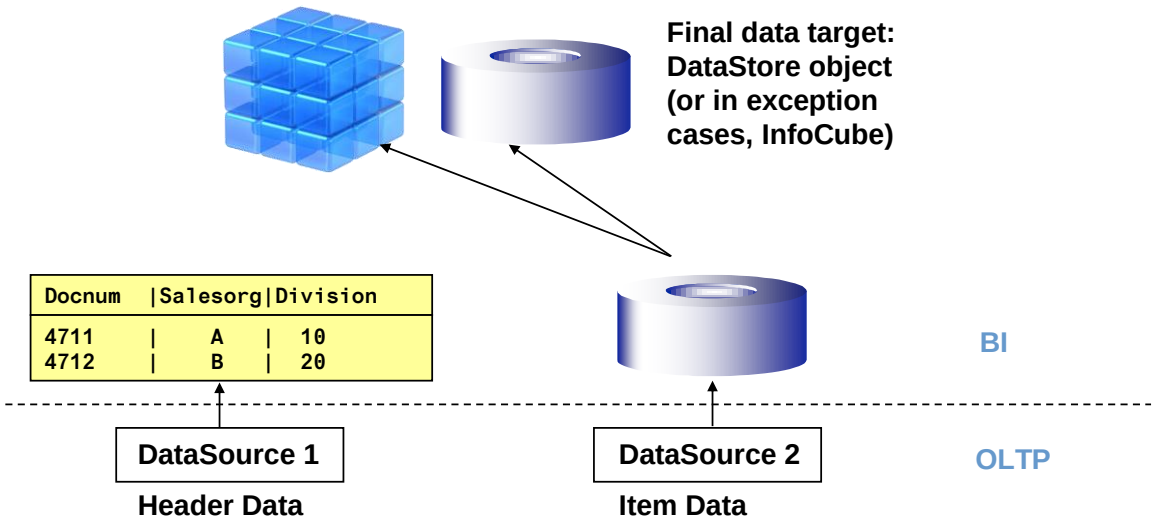


Document header and item data is extracted to BI separately and is updated to two Datastore objects. An InfoSet is defined based on these two Datastore objects and represents a join of the data in both Datastore objects.

At query runtime for a query that uses this InfoSet, there is no mechanism that can guarantee that document header data and item data are extracted into BI consistently. This scenario is appropriate for test purposes where the volume of data is relatively small.

Figure 10 Header Data and Item Data Join Using an InfoSet

Scenario 4: Document Header Modeled as Master Data Table



Document header data and item data are extracted to BI separately. The extraction of header data and item data should be combined in one process chain.

Document header data is modeled as a master data table in BI, whereas document item data is modeled as a DataStore object.

This scenario is appropriate for document header data that contains characteristics only where you do not expect these header characteristics to change frequently.

Figure 11 Document Header Modeled as Master Data Table

4.2 Comparison of Different Scenarios

Document-type Data: Comparison of the 4 Scenarios

	Scenario 1: Header Data Look-Up (Not recommended)	Scenario 2: Slight Denormalisation (Recommended)	Scenario 3: Join Using InfoSet (Not recommended)	Scenario 4: Header as Master Data (Recommended in limited cases)
Performance	Contra: Updating the final data target can be performance intensive as another DataStore object has to be read	Pro: Good performance for data staging and query runtime	Contra: Performance can be poor at query runtime	Pro/contra: Performance depends on the size and complexity of the master data table of the document header
Quality of data (Consistency, up-to-dateness)	Contra: Once the header data is staged into the final data target it cannot be changed unless all the data (header and item data) is reloaded; if the header data changes, the changes cannot be applied automatically.	Pro: The combination of item data and header data (using one DataSource) always delivers the most up-to-date status of the data. It is more accurate and up-to-date than in scenario 4.	Contra: At query runtime for a query that uses the InfoSet join, it is possible that either the header data or item data is not yet extracted to BI. The quality of the data is worse than in scenario 4.	Pro/contra: Header data and item data are combined at query runtime. Since the document header data is modeled as a master data table, it can be modified.
Transparency of data	Pro: If errors occur, you can determine on the BI side whether the errors occur in the document header data or the item data, without having to look in the OLTP system.	Pro/contra: If errors occur, it may be necessary to look into the OLTP system to find out whether the errors occur in the document header data or the item data.	Contra: If errors occur, it is necessary to analyze the OLAP process in addition to the steps mentioned for scenario 4 and 1.	Pro: If errors occur, you can determine on the BI side whether the errors occur in the document header data or the item data, without having to look in the OLTP system.
Flexibility (development efforts)	Contra: Implementation efforts on BI side exceed those of scenario 2. An additional look-up of a DataStore objects has to be coded.	Pro: Most easy to implement on BI side. The major implementation efforts are on OLTP side.	Contra: Implementation efforts on BI side exceed those of scenario 2. An additional InfoSet and join conditions have to be modeled.	Contra: Implementation efforts on BI side exceed those of scenario 2.
Complexity of data flow	Complex	Simple	Simple	Simple

Figure 12 Document-Type Data: Comparison of the Four Scenarios

4.3 General Recommendations

1. If two DataSources are very closely related they should be combined in a single DataSource in the OLTP system with one flat extract structure. A typical example is the combination of document header and item DataSources. When the document is changed, the changes are applied to both DataSources at the same time. In this case **scenario 2** is the appropriate data model.
2. If two separate DataSources do not extract data from one common application area (for example, delivery and billings, production and controlling) and these two DataSources do not have common characteristics, **scenario 1** is the appropriate data model.
3. **Scenario 4** is the appropriate data model if the volume of document data is relatively small and the document header only contains characteristics.
4. **Scenario 3** is the appropriate data model only if the volume of data is very limited. It should not be used in a productive scenario. It can generally only be used if the mechanism that guarantees data consistency (for example, that the appropriate header data and item data are available in BI when the InfoSet is running) is provided using an additional process chain.

5 Performance Aspects of the Data Warehouse Layer

5.1 Performance When Activating Data and the BEx Reporting Indicator

If the *BEx Reporting* indicator is set in DataStore object maintainance, SIDs are stored instead of chararcteristic key values. This improves flexibility in reporting but slows down the activation of the DataStore object.

The creation of SIDs is time-consuming and can be avoided in the following cases:

- Do not set the *BEx Reporting* indicator if you are not planning to report on DataStore objects in BEx or on the Web. You should not set the *BEx Reporting* indicator if you intend to use the DataStore object as a data store only.
- If your reporting requirements with regard to DataStore objects are limited (for example, you only want to be able to display a few, selected records), use InfoSets on top of DataStore objects and deselect the *BEx Reporting* indicator.
- If you are using line items (for example, document number, time stamp and so on) as characteristics in the DataStore object, mark these as *Attribute only* in characteristics maintenance.

5.2 Unique Records in DataStore Objects

If you are only loading unique data records to the DataStore object (unique data records are data records that have a unique key combination), you can improve load performance by setting the *Unique data record* indicator in DataStore object maintenance.

If this indicator is set, the system does not look up existing key values and only performs (mass) inserts into the active table of the DataStore object. Furthermore, the before-image can be omitted from the change log and the data does not have to be sorted before the DataStore object is activated.

Note:

When you select the *Unique data record* indicator, BI cannot guarantee that all the data records are unique; this has to be guaranteed externally (outside of the the BI system) by the extractor, for example. Otherwise the BI system creates a short dump.

5.3 Indexes

If you filter special characteristics values within DataStore objects (for reporting or uploading into other data targets), make sure that these characteristics are indexed so that you avoid full table scans on the DataStore object tables. Secondary indexes accelarate the selective reading of data from an DataStore object. This improves performance when you update data from the DataStore object to other data targets and when you report on the DataStore object.

You define secondary indexes on the DataStore object maintainance screen.

Note:

If you define too many secondary indexes for one DataStore object, this has a negative impact on performance during data update as all the secondary indexes have to be maintained during the load.