

Designing Distributed Data Warehouses and OLAP Systems

Jane Zhao

Massey University, Information Science Research Centre
& Department of Information Systems
Private Bag 11222, Palmerston North, New Zealand
j.zhao@massey.ac.nz

Abstract: On-line analytical processing (OLAP) systems deal with analytical tasks in businesses. As these tasks do not depend on the latest updates by transactions, it is assumed that the data used in OLAP systems are kept in a data warehouse, which separates the input coming from operational databases from the output going to dialogue interfaces for OLAP. In this article we present a 3-tier architecture for data warehouses and OLAP systems capturing the fundamental requirement of separating input from operational databases from output to OLAP systems. On this basis we start developing refinement rules to enable step-wise refinement for such systems, which includes pragmatic guidelines for the application of such rules.

1 Introduction

Data Warehouses are data-intensive systems that are used for analytical tasks in businesses such as analysing sales/profits statistics, cost/benefit relation statistics, customer preferences statistics. The term used for these tasks is “on-line analytical processing” (OLAP) in order to distinguish them from operational data-intensive systems, for which the term “on-line transaction processing” (OLTP) has become common. Thus, whenever we are confronted with data warehouse design, this includes the design of an OLAP system.

The idea of a data warehouse [In96, Ki96] is to extract data from operational databases and to store them separately. The justification for this approach is that OLAP largely deals with condensed data, thus does not depend on the latest updates by transactions. Furthermore, OLAP requires only read-access to the data, so the separation of the data for OLAP from OLTP allows time-consuming transaction management to be dispensed with.

Thus, the first problem in data warehouse design is to integrate views from various source databases. This point of view of data warehouse design as a view integration problem has been strongly promoted in [Wi95] and [KM99]. On the other hand it has been observed that the structure of the data needed for OLAP, i.e. the structure of data warehouse schemata, is somehow simpler than the structure of operational databases. This has led to the notion of multi-dimensional databases, in which “facts” needed for OLAP such as number of sales, prices, are separated from “dimensions” such as time, location, product, i.e. parameters that characterise the facts. Formally, we still obtain relations, in which the dimensions form a key, but the multi-dimensional (relational) database schemata usually

have the form of star or snowflake schemata [In96]. The work in [GL96] presents a formal model for multi-dimensional databases.

The main idea of data warehouses implies a separation of input from operational databases and output to views that contain the data for particular OLAP tasks. In the data warehouse literature these views are often called “data marts”. The work in [LST99] presented a different view on data warehouse design emphasising not just the input, but also the output, i.e. the data marts and OLAP. In doing this, each data mart together with the OLAP functions working on it defines a so-called “dialogue object”. Following this idea it is astonishing that a lot of work is put into the design of data warehouses, whereas the major emphasis should be on the OLAP functions that are based on views over the warehouse. This motivates us to take a closer look into the systems dynamics, not just the static data structures.

Furthermore, as dialogue objects over a data warehouse lead to views over a view, it may be questioned, whether it makes sense to take a holistic approach to data warehouse design or whether it might be better to replace the data warehouse by a collection of materialised views on the operational databases. This view is also underlying the work in [TS98, Th99].

In [MSZ05] an approach to data warehouse design was presented that explicitly refers to the fundamental idea of separating input from operational databases from output to OLAP systems. The emphasis was on the distribution of data warehouses. In this paper here we discuss step-wise refinement from the 3-tier architecture, aiming at achieving a complete set of rules for the refinement. We roughly classify the rules into two groups: those that reflect additional application needs and those that reorganise and improve a specification without adding new requirements. For the first group we have to pick up techniques for view integration, reorganisation of data structures according to a formal data warehouse model [GL96], distribution design techniques [MSZ05], and constraint manipulation techniques. For the second group techniques for introducing materialised views [TS98] and more standard operational refinement rules have to be considered.

This approach is related to a formal approach to the design of data warehouses and OLAP systems using the method of Abstract State Machines (ASMs, [BS03, Bö03]). This formal approach was presented in [ZS04, ZM04b, SZ05]. A formalisation of refinement rules incorporating ideas from [Bö03] was introduced in [ZM04a].

We present the 3-tier architecture in Section 2. We discuss the step-wise refinement in Section 3, with examples that show how the rules should be applied, i.e. we give pragmatic guidelines for the refinement method. We conclude with a short summary.

2 Architecture of Distributed Data Warehouses

We follow the general idea of data warehouses, i.e. the separation of input from operational databases and output to data marts. This idea is illustrated using the three-tier architecture in Figure 1.

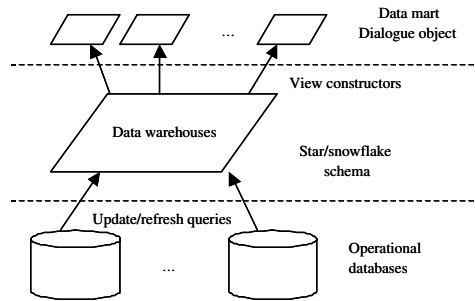


Figure 1: General Data Warehouse Architecture

2.1 The Operational Database Tier

On the bottom tier we have operational databases set up for purposes that are of no particular interest for the data warehouse. However, we assume that the data stored in the data warehouse are extracted from these operational databases. In other words, the input of data into the data warehouse is defined by update- or refresh operations, which take data from these operational databases and insert them into the data warehouse. In particular, these refresh-operations can be formulated by queries on the operational databases. Assuming that the relational data model has been used for the operational databases, we can express the refresh-operations in relational algebra or SQL.

For example, suppose we only have a single operational database with the schema that is modelled by the Entity-Relationship diagram in the left hand diagram in Figure 2. That is, in a relational representation we would have five relation schemata Store, Part, Customer, Buys and Offer of arities 2, 2, 3, 4 and 5, respectively. The following SQL query could well be used to extract data for a data warehouse defined on top of this operational database schema:

```

select C.cid, P.pid, S.sid, B.time,
        Sum(O.quantity) as quantity,
        Sum(O.quantity) * O.price as money_sales,
        Sum(O.quantity) * )(O.price - O.cost)
        as profit
from Customer C, Part P, Store S,
        Buys B, Offer O
where B.time.(day,month,year) = O.date
group by C.cid, P.pid, S.sid, B.time

```

In general, if there is some conflict between these operational data, i.e. data from different operational databases have to be integrated before they can be inserted into the data warehouse, we need an integrator component (see [Wi95]). However, we may assume that this integrator is part of the extraction functions.

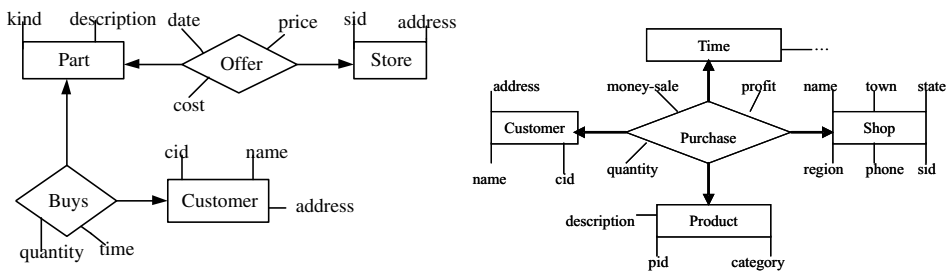


Figure 2: The operational database and data warehouse star schemata

2.2 The Data Warehouse Tier

The second tier of the architecture in Figure 1 is made up by the data warehouse itself. In principle, we just have a database system here with the only differences that we may assume a simpler schema, i.e. star or snowflake schema, and we do not have to consider complex transactions. The only write-operations are the refresh operations that connect the data warehouse to the operational databases. All other operations only read data from the data warehouse. In fact, we just build views for the “data marts” or dialogue objects that are used as the OLAP interface. So, the view construction operations are the only ones that link the middle tier to the top tier, which deals with OLAP.

For example, the data warehouse schema may be given by the Entity-Relationship diagram in the right hand diagram in Figure 2. That is, it can be represented by a relational database schema with five relation schemata Shop, Product, Customer, Purchase and Time with arities 6, 3, 3, 7 and 4, respectively. Then the SQL query in the previous subsection could indeed be used to refresh Purchase.

Similarly, the following SQL query can be used to define the view underlying a data mart for sales analysis:

```

select S.sid, S.region, S.state,
        T.month, T.quarter, T.year,
        Sum(P.quantity) as quantity,
        Sum(P.money_sale) as money_sale
from Shop S, Time T, Purchase P
group by S.sid, S.region, S.state,
        T.month, T.quarter, T.year

```

The schema of this data mart is represented by the (higher-order) Entity-Relationship schema in Figure 3.

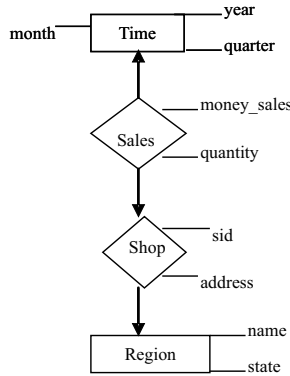


Figure 3: Data Mart Schema

2.3 The OLAP Tier

The top tier itself is constructed out of the dialogue objects and the OLAP operations working on them. This tier realises the idea of using dialogue objects for this purpose. The general idea from [SS00] is that each user has a collection of open dialogue objects, i.e. data marts. At any time we may get new users, and each user may create new dialogue objects without closing the existing ones. Thus, we maintain the lists of users and their data marts in the system. Part of the functionality of the OLAP tier deals with adding and removing users and data marts. In particular, if a user leaves the system, all data marts owned by him/her must be removed as well.

The major functionality, however, deals with running operations on existing data marts or creating new data marts. In the latter case we have to use a view creation query such as the one described in the previous subsection. In this case we choose a new identifier for this data mart / dialogue object, and initialise its data content. If the user selects some of the data of the data mart and an operation other than quit, i.e. the user does not want to leave the system, or close, i.e. the user does not want to finish work on the current data mart, or open, i.e. no new data mart is to be created, then we request to receive additional input from the user, before the selected operation will be executed.

2.4 Distributed Data Warehouses

We have seen that the central data warehouse tier in Figure 1 can be represented by a relational database schema. Therefore, in dealing with distributed data warehouses we may apply the relational theory of distribution design [OV99], i.e. we first fragment the schema, then allocate the fragments to the nodes of a network. This allocation may replicate fragments, i.e. store the same fragment at more than one node. The goal of the distribution is to optimise global performance. Having this goal in mind, fragmentation and allocation

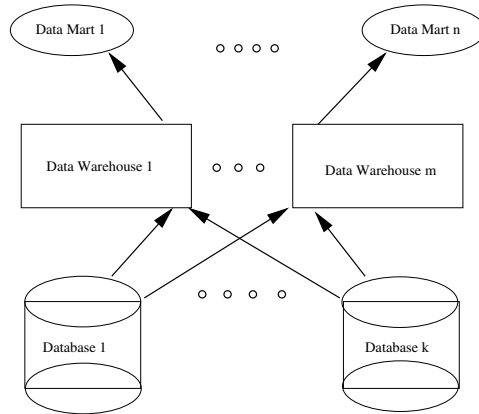


Figure 4: Distributed Data Warehouse Architecture

mutually depend on each other, and thus both steps are usually combined.

However, as refreshing of the data warehouse content can be assumed to be executed in an off-line mode and the OLAP tier requires only read-access to the warehouse, we favour an architecture with as much replication as necessary, such that all view creation operations for data marts can be performed locally. This assumption is illustrated by the distributed data warehouse architecture in Figure 4. It is even possible that the whole data warehouse is replicated at all nodes.

For instance, in the sales example we used so far regional sales statistics may only be provided to regional branch offices, whereas a general sales statistics for a headquarter may disregard individual shops. This implies that different data marts would be needed for the locations participating in the distributed warehouse and OLAP system.

3 Step-wise Refinement

We take a pragmatic look at the data warehouse and OLAP system design process starting from the 3-tier architecture of the previous section. We then discuss refinements dealing with changes that are due to identified requirements and other refinements that mainly deal with system optimisation.

3.1 Requirements Capture

Using the 3-tier architecture as presented in the previous section we can assume separate schemata for the operational databases, the data warehouse and the OLAP tier. For each of these tiers we use separate *types* (i.e. entity or relationship types) to model states of the

system by logical structures and *operations* expressing transitions between these states. The tiers are linked together via queries that extract data from the lower tier and reorganise it for the higher one.

Refinements dealing with changes that are due to identified requirements will therefore start on the top level, i.e. the OLAP tier. We just iterate the following steps each of which corresponds to the application of some refinement rules:

1. *Add a new operation to the OLAP tier:* This is used to model an additional OLAP function. It basically means that we first do not take much care about the existing specification, when we encounter new requirements. Just write down a new operation in a rather rough way.

Example 1 Let us look at the grocery store example as illustrated in the left hand HERM diagram in Figure 2. At the very beginning, we may not consider OLAP operations such as roll-up, drill-down, slice, and dice, but we add them in during the refinement. Similarly, the data marts are also built up gradually. When the new operation roll-up is to be added, we will sketch its requirements without detailing the implementation in this rule. We will name the operation as *roll_up*, and parameters as such, in star schema case, the data mart selected, *dm_sel*, the dimension selected, *dim_sel*, the level in the dimension the roll-up starts from, *from_lev*, and the level in the dimension the roll-up ends, *to_level*, where $from_lev \in dim_sel.DIM_LEV$ and $to_lev \in dim_sel.DIM_LEV$ and $from_lev < to_lev$. The operation roll-up is defined as $roll_up(dm_sel, dim_sel, from_lev, to_lev)$. When this operation is called by, say, $roll_up(total_sales, location, sid, region)$, it is equivalent to the following SQL query:

```

select S.region, S.state,
        T.month, T.quarter, T.year,
        Sum(P.quantity) as quantity,
        Sum(P.money_sale) as money_sale
from Shop S, Time T, Purchase P
group by S.region, S.state,
        T.month, T.quarter, T.year

```

All the other OLAP operations are added in the same way.

2. *Add a new type to the OLAP tier:* This will be used to model a view that is needed for the support of any new OLAP function, provided the existing view definitions are not yet sufficient.

Example 2 Since we have no data mart built at the beginning, we will apply the rule above to add a new operation such as, *open_total_sales*, which will result in a new type, *total_sales*, added to the OLAP tier. Then we sketch its requirements such as, the arity, and the attributes. Its equivalent SQL query is shown in Section 2.2. We build up the data marts in such way.

3. *Integrate types on the data warehouse tier:* This will be used whenever the types for the OLAP tier that are expressed by these types are extended. As a consequence, the type creation rules on the data warehouse tier must be changed accordingly.

Example 3 When new data mart is added to OLAP tier, its source, the data warehouse will need to be adapted to provide the data. In the case that summarised views are materialised in the data warehouse for better performance, adding a new data mart may result in integration of views in data warehouse tier. If this happens, the operation to create these views will be changed accordingly.

4. *Add additional types to the data warehouse tier:* This will be a consequence of the view integration.
5. *Reorganise the types on the data warehouse tier:* This will result from the additional types that are added in the previous step. Accordingly the operations dealing with the refreshing of the data warehouse have to be adapted.
6. *Change the queries on the operational database tier:* These queries actually model the data extraction from the operational databases and thus will be used by the refresh operations. Thus, each change to a refresh rule has to be reflected in a change to the queries on the operational database tier.

3.2 System Optimisation and Implementation

In addition to the refinement rules that are used to capture requirements in the 3-tier specification, we also apply rules that will reorganise the specification independently from the requirements. These refinement rules roughly fall into three groups dealing with operational refinements, distribution design, and with view materialisation, i.e. the definition of additional intermediate views that will help to facilitate the efficient execution of OLAP functions.

According to this classification we iterate the following steps each of which corresponds again to the application of some refinement rules:

7. *Apply operational refinements:* These refinement rules apply to the operations on all three levels and consist of rearranging the types and the operations preserving the semantics.
8. *Distribute the data warehouse and the OLAP tier:*
 - (a) *Replicate the data warehouse and the OLAP tier:* For each node in the network assume the same copy of the data warehouse and the OLAP schema and operations.
 - (b) *Remove types and operations in local OLAP tier copies:* If the needed OLAP functionality is different at different network nodes, then this rule will simply reduce the corresponding OLAP tier.

- (c) *Fragment types in local data warehouse tier copies*: This rule will reorganise and reduce a local data warehouse schema and operations, if the corresponding OLAP tier copy does not need all of the replicated data warehouse. The refresh queries are then adapted accordingly.
 - (d) *Recombine fragments in local data warehouse tier copies*: This rule will reorganise a local data warehouse copy according to query cost considerations. The refresh queries are then adapted accordingly. A cost model and pragmatics for choosing which fragments to recombine was presented in [MSZ05] and will not be repeated here.
9. *Integrate views in the Data Warehouse tier*: In this case the refinement will materialise additional views for efficiently building up the dialogue objects. The view creation queries will be adapted accordingly.

4 Conclusion

In this paper we presented a continuation of the work in [MSZ05] focusing on the step-wise refinement process in the design of data warehouses and OLAP systems. Based on the general idea of data warehouses separating input from operational databases from output to dialogue-based on-line analytical processing (OLAP), we start from a 3-tier architecture. Further development of the data warehouse and the OLAP interfaces can be based on step-wise refinement of such a design.

In our future work we will extend the standard refinement rules. At the end we expect to arrive at a rather complete list of refinement rules similar to those in [Sc97] that address the formal development of relational database applications.

References

- [Bö03] Börger, E.: The ASM refinement method. *Formal Aspects of Computing*. 15:237–257. 2003.
- [BS03] Börger, E. und Stärk, R.: *Abstract State Machines*. Springer-Verlag. Berlin Heidelberg New York. 2003.
- [GL96] Gyssens, M. und Lakshmanan, L.: A foundation for multidimensional databases. In: *Proc. 22nd VLDB Conference, Mumbai (Bombay), India*. 1996.
- [In96] Inmon, W.: *Building the Data Warehouse*. Wiley & Sons. New York. 1996.
- [Ki96] Kimball, R.: *The Data Warehouse Toolkit*. John Wiley & Sons. 1996.
- [KM99] Kedad, Z. und Métais, E.: Dealing with semantic heterogeneity during data integration. In: Akoka, J., Bouzeghoub, M., Comyn-Wattiau, I., und Métais, E. (Eds.), *Conceptual Modeling – ER'99*. volume 1728 of *LNCIS*. pp. 325–339. Springer-Verlag. 1999.

- [LST99] Lewerenz, J., Schewe, K.-D., und Thalheim, B.: Modelling data warehouses and OLAP applications using dialogue objects. In: Akoka, J., Bouzeghoub, M., Comyn-Wattiau, I., und Métais, E. (Eds.), *Conceptual Modeling – ER’99*. volume 1728 of *LNCS*. pp. 354–368. Springer-Verlag. 1999.
- [MSZ05] Ma, H., Schewe, K.-D., und Zhao, J.: Cost optimisation for distributed data warehouses. In: *Proc. HICSS 2005*. 2005.
- [OV99] Özsu, T. und Valduriez, P.: *Principles of Distributed Database Systems*. Prentice-Hall. 1999.
- [Sc97] Schewe, K.-D.: Specification and development of correct relational database programs. Technical report. Clausthal Technical University, Germany. 1997.
- [SS00] Schewe, K.-D. und Schewe, B.: Integrating database and dialogue design. *Knowledge and Information Systems*. 2(1):1–32. 2000.
- [SZ05] Schewe, K.-D. und Zhao, J.: Balancing redundancy and query costs in distributed data warehouses – an approach based on abstract state machines. In: Hartmann, S. und Stumptner, M. (Eds.), *Conceptual Modelling 2005 – Second Asia-Pacific Conference on Conceptual Modelling*. volume 43 of *CRPIT*. pp. 97–105. Newcastle, Australia. 2005. Australian Computer Society.
- [Th99] Theodoratos, D.: Detecting redundancy in data warehouse evolution. In: Akoka, J., Bouzeghoub, M., Comyn-Wattiau, I., und Métais, E. (Eds.), *Conceptual Modeling – ER’99*. volume 1728 of *LNCS*. pp. 340–353. Springer-Verlag. 1999.
- [TS98] Theodoratos, D. und Sellis, T.: Data warehouse schema and instance design. In: *Conceptual Modeling – ER’98*. volume 1507 of *LNCS*. pp. 363–376. Springer-Verlag. 1998.
- [Wi95] Widom, J.: Research problems in data warehousing. In: *Proceedings of the 4th International Conference on Information and Knowledge Management*. ACM. 1995.
- [ZM04a] Zhao, J. und Ma, H.: Asm-based design of data warehouses and on-line analytical processing systems. Technical Report 10/2004. Massey University, Department of Information Systems. 2004. available from http://infosys.massey.ac.nz/research/rs_techreports.html.
- [ZM04b] Zhao, J. und Ma, H.: Quality-assured design of on-line analytical processing systems using abstract state machines. In: Ehrich, H.-D. und Schewe, K.-D. (Eds.), *Proceedings of the Fourth International Conference on Quality Software (QSIC 2004)*. pp. 224–231. Braunschweig, Germany. 2004. IEEE Computer Society Press.
- [ZS04] Zhao, J. und Schewe, K.-D.: Using abstract state machines for distributed data warehouse design. In: Hartmann, S. und Roddick, J. (Eds.), *Conceptual Modelling 2004 – First Asia-Pacific Conference on Conceptual Modelling*. volume 31 of *CRPIT*. pp. 49–58. Dunedin, New Zealand. 2004. Australian Computer Society.