

# A Materialized View-based Approach to Integrating ETL Process and Data Warehouse Applications

Tsae-Feng Yu

Oracle Corporation  
One Oracle Drive  
Nashua, NH 03062,  
U.S.A

Tsae-Feng.Yu@oracle.com

## Abstract

*The ETL process bridging the online transaction processing (OLTP) system and the online analytic processing system (OLAP) is often modeled as a separate and independent process. When transformed data are loaded into the data warehouse, the analysis-centric applications take place. To expedite the analysis process, materialized views are often created and used through the query rewrite mechanism in the data warehouse. In this paper, we propose an approach to integrate the ETL process and the data warehouse applications by using views and materialized views to model and perform the ETL process. This process integration has several advantages. First, it achieves fast data transformation and materialized view maintenance through one single materialized view refresh call. Changes in the OLTP system can be quickly and transparently applied to the materialized views so that a near real-time data analysis can be carried out. Second, the use of views and materialized views to model the ETL process provides the benefits of encapsulating data transformations in a multi-step SQL process. Compared to the commonly used scripting approach, this approach provides better readability and maintainability. Third, because the transformation SQL is processed inside the database, better query optimization improves the ETL performance. Lastly, the use of materialized view in the ETL process could facilitate the data cleansing so that clean data are passed through and processed while dirty data are intercepted and loaded into tables for correction. The transformation for the corrected data can be resumed at the point of errors spotted.*

Keywords: data transformation, data warehouse, materialized view.

## 1. Introduction

Data analysis and decision making are ultimate goals in the data warehouse applications. Consider a large data warehouse containing terabytes of data distributed in denormalized tables each of which has potentially millions or even billions of data rows. Retrieving data from such system is a very resource-intensive and time-consuming process. Queries that retrieve and calculate a large amount of data could require hours to be processed.

To speed up the query processing in the OLAP system, a commonly used approach is the use of materialized views. The materialized view is a database object that contains pre-calculated data typically for highly aggregate queries associated with complex joins of fact and dimension tables. The use of the materialized view in query processing is often transparent through a mechanism called *query rewrite*. In such mechanism, the original query is rewritten to make use of the materialized view instead. The query response time is thus improved. Consequently, the materialized view has become a common but required resident in the OLAP system.

Then, consider the subject of data loading to the OLAP system, a well-known process called ETL. It extracts the source data from the operation-centric OLTP system, transforms the data to resolve various kinds of heterogeneity problems and loads the transformed data into the target OLAP system (e.g., data warehouse). The ETL process is usually an independent process that bridges the two systems. It is often achieved by the third party tool which executes the ETL process in a specific window of time. Due to this asynchronous nature, the changes in the OLTP system are not transformed and propagated to the OLAP system in the real time. As a result, real-time data analysis and decision making is difficult to achieve with such process model.

On the other hand, since the materialized view has already become a common data warehouse object for improving query performance, it will be beneficial to use the materialized view to model the ETL process so that the ETL process and the data warehouse applications can be seamlessly integrated. This idea motivates us to

investigate and design an integration mechanism that is based on the materialized view technology.

In our approach, we combine the ETL process into the OLAP application by conditionally using views and materialized views. The buffering of intermediate results in the ETL process and the fact and dimension tables in the OLAP system are all represented in materialized view objects. The changes propagation and the data transformation are done in one single refresh call that refreshes all the materialized views in the dependency hierarchy. Such design generates an efficient integration and enables near real-time data analysis. Also, the approach can support data cleansing to filter out dirty data and allows corrected data to be re-submitted into the ETL process at the spot where the error is found.

The rest of the paper is organized as follows. In Section 2, we first review related work in the areas of the ETL modeling, data cleansing and processing performance. Section 3 presents our proposed approach and compares it with the current architecture. We then illustrate the use of the materialized view to model the ETL process and data cleansing with examples. Later, we discuss the advantages and concerns of our approach. Section 4 describes the conclusion.

## 2. Related Work

### 2.1. ETL modeling and implementation

The ETL process has been widely modeled in the data flow approach [1][2][3][5][6][7][9][12][13]. In this approach, the Extraction-Transformation-Loading process is considered as a data flow from the source systems to the target systems. In general, there are two approaches to implement the process:

1. *Script based approach*: In this approach, the ETL process is implemented in the program code [5][6]. The user can specify the hard-coded transformation logic in low-level code such as C, Perl, PL/SQL, etc. The advantages of this approach are its flexibility and efficiency. The disadvantages are that it is hard to debug and maintain.
2. *Fine-step GUI-based transformation approach*: This approach relies on a GUI-based tool to specify individual transformation steps. Typically, a set of primitive transformation tool boxes [3][9] are provided to allow the user to design and construct the ETL plan. The advantages of this approach are that it is easy to maintain and modify the plan as the transformation logic is visualized in a process flow manner. With the GUI, it is more user-friendly to build an ETL plan. However, the disadvantages are that when the ETL process becomes complex, the ETL plan could have a

large number of transformation boxes which could easily lead the user to generate an inefficient plan. For example, some intermediate columns of data could stay around in the plan so that an additional optimization [14] for this plan is required.

Distinct from the script-based and fine-step GUI-based approaches, our materialized view based ETL approach is a neutral mechanism that encapsulates a number of transformation operations in a single materialized view SQL query while preserving sufficient logic representation for better maintenance.

### 2.2. Data cleansing

Data quality is a critical requirement that affects data analysis and decision making. To meet the requirement, many research efforts [4][10][11] have been invested. However, the problem domain is very broad and complicated so that it could include a variety of data and schematic heterogeneity problems and their combinations. Dirty data could appear in the source system in any arbitrary forms or any value representations. It is virtually impossible to achieve automatic data correction during the ETL process execution. As a result, human intervention is inevitable in order to correct the dirty data.

It is highly possible that dirty data can be identified and intercepted in the ETL process and recorded in some areas for human experts to examine and make corrections [11]. However, the human involvement in the data cleansing often compromises the ETL process performance since in some ETL system, the process needs to be stopped and incrementally modified to meet the need.

Without compromising the performance, our proposed approach utilizes the materialized view whose defining query contains data condition checks to filter out potential dirty data without affecting the transformation of correct data. Dirty data are redirected into a table for the human expert to examine and correct. The corrected data will be resubmitted to the ETL process as part of the materialized view incremental maintenance operation.

### 2.3. Processing performance

To improve the ETL processing performance, some research efforts were made in [4], [11], [14] and [15]. The work proposed in [11] is to improve the ETL execution by compiling the sequence of transforms into form of C or Perl program to achieve better execution performance. This language-based method only provides limited benefit since the Plan Execution Engine can be improved to achieve better performance. However, the key performance factor still depends on the amount of data being processed. While in [4], another optimization approach is tailored to improve data matching, targeting the reduction of data joining cost. Main techniques used

include mixed evaluation, operation re-ordering, neighborhood hash join, etc. However, in the ETL process, data matching is not the only performance challenge. Many of other transformations, such as data filtering, conversion, etc., need to be considered as well. Considering the ETL plan-level optimization, a rule-based approach is proposed in [14] to achieve automatic optimization for the ETL plan by altering the original plan into an equivalent but more efficient one. It is enabled by relieving the transformation load from the Plan Execution Engine and by moving part of transformation operations into more efficient processing systems such as source databases. The plan modification is automatically done by the ETL optimization component using heuristics. In [15], a transform merge mechanism was proposed to combine intermediate transformation steps into a single one with multi-threading parallel execution.

Our approach of using materialized view performs data transformation inside the target OLAP system. Similar to [14], it utilizes the processing power of a database system. But different from [14] where the use of database engine is through the ETL plan optimization, our approach uses the target database engine through the incremental refresh of the materialized view. The execution of the internal refresh statements is internally optimized by the query processor and optimizer.

### 3. Proposed Approach

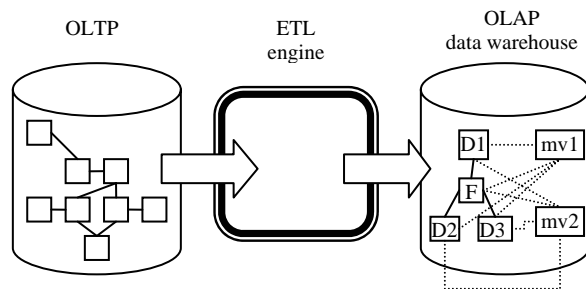
In this section, we first show the current ETL process model and compare it with the proposed architecture that integrates the ETL and data warehouse applications. Second, we show the use of views and materialized views to achieve the integration and illustrate it using examples. Finally, we evaluate the integration model and discuss its benefits and issues.

#### 3.1. Architectures of current ETL and data warehouse applications

The ETL process is to bridge and transform the data from an operational system (OLTP) to an analysis-centric data warehouse system (OLAP). In an OLTP system, data are often modeled in normalized tables that work well for frequent DML-based transactions. On the other hand, in an OLAP system like a data warehouse, data are commonly modeled in de-normalized tables (e.g., star schema) to support better data retrieval and business analysis. The data model discrepancy causes an extra complexity for the ETL process in addition to the classic data discrepancy problem.

Figure 1 shows the current architecture of the ETL process. In this figure, the source data are extracted from the OLTP system in an ETL engine. This ETL engine is often a third-party tool which could be different from the source and target systems. In such execution

model, the ETL performance highly depends on the capability of the ETL engine. The data extracted from the source system are transformed in the ETL engine and finally loaded in the target OLAP system such as a data warehouse or a data mart. It is noted that the data in the OLAP system are used for data analysis or data mining and eventually for decision making purposes. Such data analysis process is resource intensive and requires the access to highly aggregated data from multiple tables each of which could contain a huge number of rows of data. It is very timing-consuming to process such query requests as it may require many hours of execution time to generate a result report.



**Figure 1: Current architecture of ETL and data warehouse applications**

To support efficient query processing in the OLAP system, in addition to the use of indexes, a commonly used approach is to create a number of materialized views. The materialized view contains pre-calculated query result from joining multiple tables such as fact and dimension tables. The query is highly aggregated and includes various analytic functions such as COUNT, SUM, AVERAGE, MAX, MIN, RANK, LAG, etc. The materialized view speeds up the query processing by means of a mechanism called query rewrite. When an analytic query is submitted, the query is analyzed to find out any applicable materialized views to rewrite with. Once found, the query optimizer evaluates and selects the best materialized view to use and rewrite the original query with a new query that access data from the materialized view. Since the materialized view contains the pre-calculated data, the response time for the query is greatly reduced.

The maintenance of the materialized views is often through an incremental approach [8] in which only changes made to the base tables (e.g., fact and dimension tables) are applied to the materialized views. The materialized view maintenance can be done when the change transaction to the base tables commits or can be launched on demand. The incremental maintenance ensures the materialized view's availability for query rewrite.

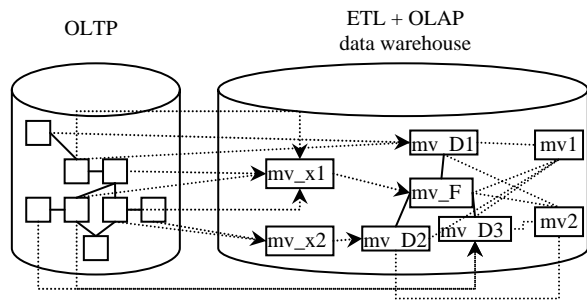
In the current architecture, the ETL process and the data warehouse applications are independent

processes. It has many disadvantages. First, the data of the fact and dimension tables in the OLAP system is not synchronized with the data in the source OLAP system until the ETL process is finished. That is, the ETL is done in an asynchronous fashion. Second, after the fact and dimension tables are loaded with new data, the materialized views still need to be refreshed to pick up the changes. The mismatch between the ETL and the warehouse processes affects the availability of the materialized view and its use. Due to the delay in the change propagation, real time data analysis is not possible. Finally, since the ETL engine only resides in a third site, the data transmission costs for extraction from the source systems and loading to the target systems could be significant. Also, the ETL performance may not be satisfying if the ETL engine is not efficient.

### 3.2. Proposed integration of the ETL and data warehouse applications

Due to the common use of the materialized view in the OLAP system for quick query processing, the materialized view has become a required object. This fact motivates the idea of using materialized view to integrate the ETL and the data warehouse applications.

It is noted that the ETL process is essentially a number of individual data transformation steps that perform data conversion, filtering, assembly, etc. Regardless of how complicated the ETL process is, it can be represented in one or many steps of transformation SQL queries. On the other hand, the materialized view is query-based so that it is possible to be used to model the ETL process and further enables the integration of the ETL and the data warehouse applications.



**Figure 2: Integrated ETL and data warehouse applications using materialized views**

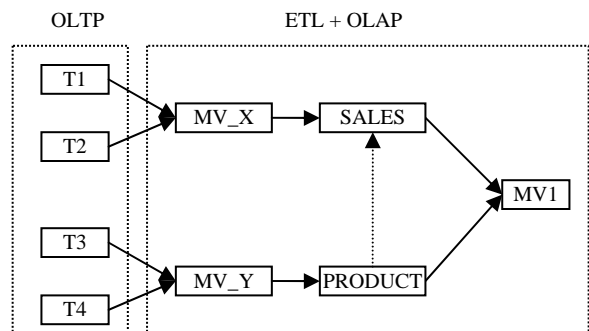
As seen in Figure 2, both the ETL process and the data warehouse applications are integrated using the materialized views. The complete ETL process is included in the target OLAP system with additional materialized views. For example, the original fact table F in Figure 1 is modeled as a materialized view mv\_F in Figure 2. mv\_F is based on an intermediate materialized view mv\_x1 that performs the transformation for two tables in the OLTP source system. In this architecture, the

original fact and dimension tables (F, D1-D3) in the target OLAP system are all modeled as materialized views (mv\_F, mv\_D1-mv\_D3) whose defining queries contain the data transformation logic.

The collection of the materialized views in the OLAP system forms a hierarchical structure in which some materialized views depending on one or more base materialized views. Such dependent materialized views are called *nested* materialized views. To enable the real time maintenance of the materialized views in the OLAP system, those materialized views (e.g., mv\_D1, mv\_x1) directly based on the tables in the OLTP system, called *base* materialized views, are refreshed on demand while other nested materialized views (e.g., mv\_F, mv1) are specified as refresh on commit. Such settings will enable the automatic maintenance of all materialized views in one refresh call. When changes in the tables of the OLTP system commit, change data are recorded in the corresponding materialized view logs of those OLTP tables. Then, on the OLAP side, the user can launch a single refresh call to refresh the group of the base materialized views. When the group of base materialized views finishes their refresh executions and commits, the refresh of the nested materialized views automatically gets started (as they are specified as on-commit refresh). The changes from the OLTP system are therefore propagated through the materialized view incremental refresh maintenance and applied to all the materialized views in the OLAP system. Thus, the materialized views are ready for access in support of real-time decision making.

### 3.3. Example of ETL process modeled in views and materialized views

Figure 3 shows an example of using materialized views to integrate the ETL process and the OLAP applications. In this example, four tables T1-T4 reside in the OLTP source system, while in the target OLAP system, two tables SALES and PRODUCT are in the warehouse schema. In addition, a materialized view, MV1 is built on top of tables SALES and PRODUCT to support quick query reporting.



**Figure 3: Example of using materialized views to integrate ETL and OLAP applications**

It is noted that there are two additional materialized views, MV\_X and MV\_Y, are created to bridge the OLTP tables (T1-T4) and OLAP tables (SALES and PRODUCT). The materialized view MV\_X is to perform part of the ETL process to transform data from T1 and T2 to SALES, while MV\_Y is for transforming data from T3 and T4 to PRODUCT. In this architecture, table SALES is also a materialized view that is based on MV\_X. The definition of the materialized view MV\_X can be specified as follows:

```
CREATE MATERIALIZED VIEW MV_X
FAST REFRESH ON DEMAND
AS
SELECT T1.ROWID T1RID, T2.ROWID T2RID,
       CONV1(T1.PRICE_SALES) PRICE_SALES,
       CONV2(T2.UNIT_SALES) UNIT_SALES,
       CONV3(T1.AMOUNT) AMOUNT_SOLD,
       T2.PRODUCT_ID PROD_ID,
       T1.C1
FROM T1@oltp, T2@oltp
WHERE T1.C1 = T2.C1;
```

The materialized view MV\_X is a materialized join view based on tables T1 and T2. All required columns of data from both tables are extracted and transformed through proper conversion functions. This materialized view is created with the specification of REFRESH FAST ON DEMAND. That is, it supports log-based incremental refresh maintenance and the refresh invocation is on demand. To achieve that, both T1 and T2 have a materialized view log built on each of them before the materialized view MV\_X is created. The materialized view log captures the information for all rows that has been changed in the table (e.g., T1). The refresh of the materialized view MV\_X is done through the following statements:

```
EXECUTE DBMS_MVIEW.REFRESH('MV_X');
```

It is also noted that there are two ROWID columns included in the defining query of MV\_X. They are included to support log-based incremental refresh as the ROWID column contains the physical address of the data rows and can be used to speed up the refresh process.

Besides, the fact table SALES is also modeled as a materialized view as follows:

```
CREATE MATERIALIZED VIEW SALES
FAST REFRESH ON COMMIT
AS
SELECT X.ROWID XRID, P.ROWID PRID,
       FUNC1(X.PRICE_SALES) DOLLAR_SALES,
       X.UNIT_SALES,
       X.AMOUNT_SOLD, X.PROD_ID,
       X.C1
FROM MV_X X, PRODUCT P
WHERE X.PROD_ID= P.ID;
```

Similar to MV\_X, SALES is a materialized join view. It looks up the product key from the dimension table PRODUCT (which implies that PRODUCT is built up and maintained first). The ROWID columns are for the same purpose of supporting incremental refresh. Both MV\_X and PRODUCT have materialized view logs created on them to support incremental refresh of SALES. It is noted that SALES is a nested materialized view based on MV\_X and is specified as REFRESH FAST ON COMMIT. That is, when the materialized view MV\_X gets refreshed, the incremental refresh of SALES is automatically kicked off to propagate and apply the changes.

The original materialized view used for speeding up query performance is MV1. Its definition statement is unchanged except the REFRESH mode is changed to ON COMMIT. It is to achieve automatic change propagations as done on SALES. The materialized view creation statement for MV1 could be defined as follows:

```
CREATE MATERIALIZED VIEW MV1
FAST REFRESH ON COMMIT
ENABLE QUERY REWRITE
AS
SELECT S.PROD_ID,
       COUNT(*) CT_STAR,
       COUNT(DOLLAR_SALES) CT_DSALES,
       SUM(DOLLAR_SALES) SUM_DSALES,
       COUNT(AMOUNT_SOLD) CT_ASOLD,
       SUM(AMOUNT_SOLD) SUM_ASOLD
FROM SALES S, PRODUCT P
WHERE S.prod_id = P.id
GROUP BY S.prod_id;
```

In this example, to trigger the automatic change propagations from the OLTP tables to the OLAP tables and materialized views are through the following refresh statement execution:

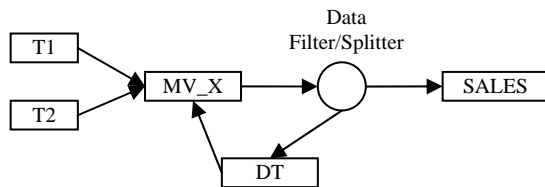
```
EXECUTE DBMS_MVIEW.REFRESH('MV_X, MV_Y');
```

All materialized views based on the above two materialized view in the dependency hierarchy will be automatically refreshed. The materialized view MV1 will therefore be available for query rewrite and support near real-time data analysis.

### 3.4. Example of using materialized views for data cleansing

The use of the materialized view in the ETL process can support data cleansing as well. As previously mentioned, automatic conflict resolutions and fixes are nearly impossible due to the complexity of data representations. Part of the conflict resolutions and fixes will have to rely on human experts to examine the dirty data and make corrections. Typically, when transformed data are

identified as dirty, those rows are recorded in a separate file or table for the human expert to examine. After corrections, the data can be resubmitted to the transformation process.



**Figure 4: Example of using materialized views for data cleansing**

Figure 4 shows an example of using materialized views to support data cleansing. Similar to the example given in Section 3.3, the materialized view MV\_X contains intermediate transformation results which are to be passed to SALES. However, the data quality needs to be ensured before the data passing. Therefore, we use a data filter/splitter (as shown in an oval shape in Figure 4) that represents one or more condition checks. The data that pass the condition checks are forwarded to populate SALES. Otherwise, the data are treated as “dirty” and redirected into a table DT for the human expert to examine and correct. The following statement is an example to populate the table DT:

```

MERGE INTO DT
(SELECT T1RID, T2RID, PRICE_SALES,
      UNIT_SALES, AMOUNT_SOLD, PROD_ID,C1,
      DECODE (PRICE_SALES < 0, TRUE,
              'invalid PRICE_SALES',
              DECODE (PROD_ID > 250, TRUE,
                      'invalid PROD_ID','OK')) MSG
FROM MV_X
WHERE PRICE_SALES < 0 or
      T2.PRODUCT_ID > 250)
WHEN NOT MATCHED
  INSERT;
  
```

It is noted that the table DT has one extra column, MSG. It is to provide the reason why the data is considered as dirty. The human expert will use the information to correct the data. After the correction, the corrected data in the table DT is resubmitted to the ETL process through MV\_X. To support this, the creation statement for MV\_X is modified as the following:

```

CREATE MATERIALIZED VIEW MV_X
FAST REFRESH ON DEMAND
AS
SELECT '1' UMARKER, '0' DTRID
      T1.ROWID T1RID, T2.ROWID T2RID,
      CONV1(T1.PRICE_SALES) PRICE_SALES,
      CONV2(T2.UNIT_SALES) UNIT_SALES,
      CONV3(T1.AMOUNT) AMOUNT_SOLD,
  
```

```

      T2.PRODUCT_ID PROD_ID,
      T1.C1
FROM T1, T2
WHERE T1.C1 = T2.C1
UNION ALL
SELECT '2' UMARKER, DT.ROWID DTRID
      T1RID, T2RID, PRICE_SALES,
      UNIT_SALES, AMOUNT_SOLD, PROD_ID,C1
FROM DT;
  
```

The modified MV\_X to support data cleansing has one extra query block connected to the original query through a UNION ALL operator. To support log-based incremental refresh for MV\_X, two additional columns (UMARKER and DTRID) are added. This way, the materialized view MV\_X is able to include corrected data in the ETL process.

Another minor modification is needed for SALES whose defining query is added with the filter conditions so that only correct data are passed to SALES.

### 3.5. Benefits and Issues

The above proposed approach that integrates the ETL process and the data warehouse applications has the following advantages:

1. It integrates the ETL process and the data warehouse applications in a common framework. Instead of using the third party ETL tool that potentially causes high data transmission overhead, the ETL process is combined into the data warehouse system in a seamless fashion which saves the data transmission overhead from the ETL engine to the target system.
2. It achieves fast data transformation and materialized view maintenance through one single materialized view refresh call. The use of on-commit nested materialized views enables the changes in the OLTP system to be quickly and transparently applied to the terminal materialized views so that a near real-time data analysis can be realized.
3. The use of views and materialized views to model the ETL process provides the benefits of encapsulating data transformations in a multi-step SQL process. Compared to the scripting and fine-step GUI-based transformation approaches, this approach offers a neutral representation of the ETL plan which provides better readability and maintainability. The modification of the ETL plan is as simple as the defining query change of the materialized view.
4. Because the materialized views that model the ETL plan are processed inside the database, efficient query processing strategies by the optimizer can improve the ETL process performance. Common operations such as data

matching, sorting and filtering can be done much more efficiently.

5. The use of materialized view in the ETL process could facilitate the data cleansing as well. In this approach, clean data are passed and processed through the normal path while dirty data are intercepted and loaded into tables for correction. The transformation for the corrected data can be resumed at the point of errors spotted and fed into the transformation process.

On the other hand, the use of materialized views to integrate with the ETL process could introduce the following requirements and questions:

1. There are additional storage requirements for the materialized views. We observed that and suggest conditional use of view in place of materialized view when the intermediate transformation result is not necessarily stored and the incremental refresh of the next materialized view is preserved. This alleviates the concern of storage needs. Also, a range-based incremental refresh can be used to support the change propagations only for certain areas of data (e.g., a period of time). This allows old and no longer needed intermediate transformation results in the materialized views can be aged out to conserve storage space.
2. A common but fundamental question is: can the materialized view sufficiently handle all possible ETL scenarios? The answer appears to be positive as all ETL processes that can be written in scripts or programs are able to be represented in a sequence of SQL statements. To handle some schematic transformations, using an intermediate materialized view is sufficient to achieve the goal.

## 4. Conclusions

In this paper, we presented an approach of integrating the ETL process into the OLAP data warehouse system using the materialized views and views. This approach provides a seamless integration that can quickly and efficiently propagate the changes from the OLTP source system all the way to the materialized views in the OLAP data warehouse system. Thus, the materialized views used by query rewrite contain up-to-date information and are able to support near real-time data analysis/mining and consequently benefit decision making.

The approach is carried out by a number of intermediate materialized views or views to model the ETL process in the OLAP system. This enables the ETL execution inside the database in which more efficient optimization mechanisms for data processing can be utilized. The ETL process can therefore be more efficient. In addition, the use of on-commit nested materialized

views automates the data transformation and the refresh maintenance of terminal materialized views in one single refresh call which greatly eases the process that is currently handled in a complicated way.

Finally, some concerns about this approach such as storage requirements are alleviated by conditionally using views to replace materialized views in the ETL process. Also, the unnecessary transformation results can be aged out from the intermediate materialized views by range refresh maintenance of the materialized view.

## REFERENCES

- [1] Eckerson, W. W. (1996), "Understanding Data Marts: A Look at Architectures, Products, and Real-World Implementations", Patricia Seybold Group.
- [2] Eckerson, W. W. (1997), "Next-Generation Data Marts: Scaling and Integrating Data Marts for Enterprise Decision Support", Patricia Seybold Group.
- [3] Dunbar V., et al, (1997), "The Oracle Data Mart Suite Cookbook", Oracle product documentation, Oracle Corporation.
- [4] Galhardas, H., Florescu, D., Shasha, D. (2000), "An Extensible Framework for Data Cleaning", in Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, Texas, U.S.A.
- [5] Informatica Corporation (2006), Informatica PowerCenter, [www.informatica.com](http://www.informatica.com).
- [6] Microsoft Corporation (2002), Microsoft SQL Server Data Transformation Services, [www.microsoft.com](http://www.microsoft.com).
- [7] Miller, M. (1999), "Features of Data Transformation Services for SQL Server 7.0", presentation slides, Microsoft Corp.
- [8] Oracle Corporation (2005), "Oracle 10gR2 Data Warehouse Guide", [www.oracle.com](http://www.oracle.com).
- [9] Prism Solutions, Inc. (1998), "Prism Executive Suite: Building the Best Information Foundation", Prism Web Document, Prism Solutions, Inc.
- [10] Rahm, E., Do, H. H. (2000), "Data Cleaning: Problems and Current Approaches", in IEEE bulletin, Vol. 23, No. 4, Dec 2000.
- [11] Raman, V., Hellerstein, J. M. (2001), "Potter's Wheel: An Interactive Data Cleaning System", in Proceedings of the 27<sup>th</sup> VLDB Conference, Roma, Italy.
- [12] White, C. J. (1997), "Building a Corporate Information System: The Role of the Data Mart", Database Associate Internal, Inc.
- [13] White, C. J. (2003), "Intelligent Business Strategies: ETL in the Database", Published in DM Review in April 2003
- [14] Yu, T., (2001) "Optimizing Data Transformation Plan Execution", published in International Symposium on Information Systems and Engineering (ISE'2001), Las Vegas, Nevada, USA, June 2001.
- [15] Yu, T., (2003) "Transform Merging of ETL Data Flow Plan", published in International Conference on Information and Knowledge Engineering (IKE'2003), Las Vegas, Nevada, USA, June 2003.