Design and implementation of ETL processes using BPMN and relational algebra

Judith Awiti [a,*], Alejandro A. Vaisman [b], Esteban Zimányi [a]

[a] *Department of Computer and Decision Engineering, Université Libre de Bruxelles. Av. Roosevelt 50, B-1050, Bruxelles, Belgium* [b] *Instituto Tecnológico de Buenos Aires, Buenos Aires, Argentina*

A R T I C L E   I N F O

Keywords:
Data Warehousing
OLAP
ETL
BPMN

A B S T R A C T

Extraction, transformation, and loading (ETL) processes are used to extract data from internal and external sources of an organization, transform these data, and load them into a data warehouse. The Business Process Modeling and Notation (BPMN) has been proposed for expressing ETL processes at a conceptual level. A different approach is studied in this paper, where relational algebra (RA), extended with update operations, is used for specifying ETL processes. In this approach, data tasks in an ETL workflow can be automatically translated into SQL queries to be executed over a DBMS. To illustrate this study, the paper addresses the problem of updating Slowly Changing Dimensions (SCDs) with dependencies, that is, the case when updating a SCD table impacts on associated SCD tables. Tackling this problem requires extending the classic RA with update operations. The paper also shows the implementation of a portion of the TPC-DI benchmark that results from both approaches. Thus, the paper presents three implementations: (a) An SQL implementation based on the extended RA-based specification of an ETL process expressed in BPMN4ETL; and (b) Two implementations of workflows that follow from BPMN4ETL, one that uses the Pentaho DI tool, and another one that uses Talend Open Studio for DI. Experiments over these implementations of the TPC-DI benchmark for different scale factors were carried out, and are described and discussed in the paper, showing that the extended RA approach results in more efficient processes than the ones produced by implementing the BPMN4ETL specification over the mentioned ETL tools. The reasons for this result are also discussed.

1. Introduction

Extraction, transformation, and loading (ETL) processes extract data from internal and external sources of an organization, transform these data, and load them into a data warehouse (DW). Since ETL processes are complex and costly, it is important to reduce their development and maintenance costs. Modeling these processes at a conceptual level would contribute to achieve this goal. Since there is no agreed-upon conceptual model to specify such processes, existing ETL tools use their own specific language to define ETL workflows. Considering this, the paper discusses two methods for designing ETL processes. The first one, called BPMN4ETL, is based on the Business Process Modeling Notation (BPMN), a de-facto standard for specifying business processes, which provides a conceptual and implementation-independent specification of such processes, that can be then translated into executable specifications for ETL tools. The second one is a logical model based on relational algebra (RA), a formal language that provides a solid basis to specify ETL processes for relational databases. The rationale for studying these two alternatives is two-folded: on the one hand, since BPMN is widely used for specifying business processes, adopting this methodology for ETL would likely be smooth for users already familiar with that language. On the other hand, RA is not only a well-studied formal language, but its expressiveness allows providing a detailed view of the data flow of any ETL process as well.

* Corresponding author.
E-mail addresses: judith.awiti@ulb.ac.be (J. Awiti), avaisman@itba.edu.ar (A.A. Vaisman), ezimanyi@ulb.ac.be (E. Zimányi).

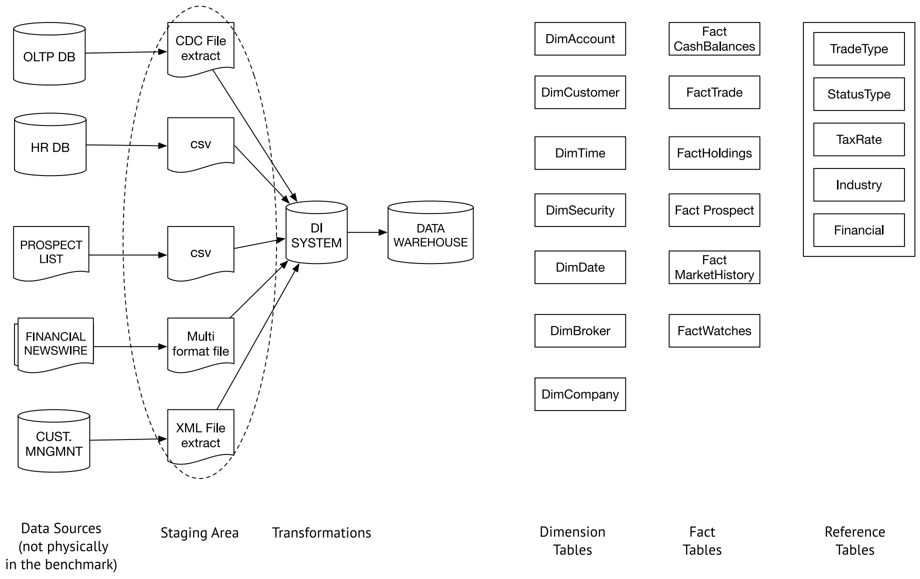


Fig. 1. Schema of the TPCI-DI system (left); Tables in the TPCI-DI DW (right).

1.1. *Running example*

The TPC-DI benchmark [1] is used as running example throughout the paper, with focus on the processes that update customers and their accounts. The benchmark has two phases: *Historical Load*, and *Incremental Updates*. In the former, destination tables are initially empty and then populated with new data. The TPC-DI scheme is shown on the left-hand side of Fig. 1, which depicts the overall process of loading data from the sources into the DW, whose tables are given on the right-hand side. The OLTP database represents transactional information about securities market trading and the entities involved, e.g., customers, accounts, and so on. There is a Change Data Capture (CDC) file, where data are sorted according with the time when the updates occurred. Thus, input files from the OLTP database are modeled as CDC extracts, which show the changes in the source data starting from the last data extraction. The CDC records also contain a flag, denoted CDC_FLAG, that may take the values I or U, telling which kind of operation each record represents (insert, or update, respectively). In addition, a CustomerMgmt.xml file represents actions resulting in new or updated customer and account information, extracted from a Customer Management System. This file is only used in the historical load, to populate the dimension tables for customers and their accounts, denoted DimCustomer and DimAccount, respectively.

It must be noted that, for each action in the ETL, only the properties involved in the update are given. For example, a ‘NEW’ action (an insertion of a new customer) will contain customer identifying information, many properties (e.g. name, address), and information about the customer's account. In contrast, an ’UPDCUST’ action updates a customer and all of his/her current accounts. Also note that all actions have at least one related customer, and each account is associated with a single customer. There are also other related tables (e.g., Prospect list, Financial Newsware, and so on) that are used by the different processes.

1.2. *Contributions*

The paper studies the two approaches introduced above for modeling and implementing ETL processes. For this study, the paper addresses the well-known case of Slowly Changing Dimensions (SCDs) with dependencies, that is, the case when updating a SCD table impacts on associated SCD tables. The problem of updating this kind of SCD is introduced in Section 2.

As a key contribution, an ETL development approach is proposed. It starts with a BPMN4ETL conceptual model, that is then translated into a RA extended with update operations at the logical level (this extended RA is presented in Section 4). Finally, RA operations are translated into an SQL-based specification of the ETL process. Although BPMN4ETL been already proposed in [2], and explained in detail in [3], the problem of modeling SCDs using this technique is discussed here for the first time (see Section 3).

Modeling ETL processes using RA was introduced by Santos and Belo [4–6], also addressing SCDs. The proposal presented in this paper goes far beyond those works (as it is discussed in Section 7, along with other related efforts), addressing the problem of updating SCDs with dependencies, and implementing the resulting ETL process using SQL. The relevance of tackling SCDs with dependencies is reflected by their inclusion in the TPC-DI benchmark introduced in Section 1.1. Modeling SCD with dependencies using the extended RA approach is described in Section 5.

As another contribution, the paper compares the implementation of a portion of the TPC-DI benchmark resulting from both approaches, namely: (a) An SQL implementation based on the extended RA-based specification of an ETL process expressed in BPMN4ETL; and (b) Two implementations of workflows that follow straightforwardly from BPMN4ETL, one that uses the Pentaho DI tool, and another one that uses Talend Open Studio for DI. Experiments over these implementations of the TPC-DI benchmark for different scale factors were carried out, and are described and discussed in Section 6. The results suggest that the extended RA approach results in more efficient processes than the ones produced by implementing the BPMN4ETL specification over the mentioned ETL tools. The reasons for this are also discussed in Section 6. Conclusions are given in Section 8.

2. Slowly changing dimensions with dependencies

Slowly Changing Dimensions (SCD) [3,7] are used in a DW to keep the history of changes that occur in data sources. Kimball [7] defined seven types of SCD. With SCDs of type 2, the history of changes is kept by augmenting the schema of the dimension table with two temporal attributes, called StartDate and EndDate. The former stores the time when the tuple was inserted into the dimension table. The latter stores the date when an update of the attribute was made in the dimension table. In general, a currently valid record has a NULL value or a date far off into the future as its EndDate. When a tuple is deleted from the source table, the EndDate attribute of its corresponding tuple in the dimension table is set to the current date. An additional current indicator attribute, IsCurrent, contains a Boolean value which is set to True to indicate that a tuple is the current record corresponding to the natural key.

This paper (as well as the TPC-DI benchmark) tackles SCDs of type 2 with dependencies. Such a SCD table contains a surrogate key reference to another dimension table. This way, whenever an update to the referenced dimension table occurs, the referencing table must be updated as well. To illustrate this, consider the schemas two dimension tables, namely DimA and DimB, depicted in Eqs. (1) and (2), depicted below.

$$\mathsf{DimA} = (\mathsf{SkA}, \mathsf{PkA}, \dots, \mathsf{IsCurrent}, \mathsf{StartDate}, \mathsf{EndDate}) \tag{1}$$

$$\mathsf{DimB} = (\mathsf{SkB}, \mathsf{SkA}, \mathsf{PkB}, \dots, \mathsf{IsCurrent}, \mathsf{StartDate}, \mathsf{EndDate}) \tag{2}$$

Dimension DimB references the surrogate key (SkA) of DimA. Dimensions DimA and DimB have ‘natural’ keys PkA and PkB, respectively. When an update occurs over DimA, the (SkA) value of DimB must be replaced by the most current (SkA) value in DimA.

The general steps of an ETL that updates a tuple in DimA are listed below.

1. Retrieve the current tuple (the one with IsCurrent = True) from DimA.
2. Rename SkA to SkAOld.
3. Retrieve the maximum SkA value from DimA and increase it by 1.
4. Remove “logically” the current tuple from DimA. That means, set IsCurrent and EndDate attributes to False and the current date, respectively.
5. Insert a new tuple in DimA (set IsCurrent and EndDate attributes to True and NULL, respectively).
6. Retrieve the corresponding current tuples from DimB. These are the tuples with IsCurrent and SkA values of True and SkAOld, respectively.
7. Remove the current tuples in DimB, that means, set IsCurrent and EndDate attributes to False and the current date, respectively.
8. Insert new tuples in DimB (set IsCurrent and EndDate to True and NULL, respectively).

This process is represented using BPMN4ETL in Section 3 and using the relational algebra (RA) approach in Section 5.

3. Conceptual BPMN for slowly changing dimensions

The BPMN4ETL conceptual model [3,8], represents ETL processes as a combination of control and data tasks. Control tasks orchestrate groups of tasks, and data tasks detail how input data are transformed and output data are produced. For example, populating a DW is a control task composed of multiple subtasks, while populating fact or dimension tables is a data task. This section shows how BPMN4ETL can be applied to handle SCD with dependencies described above in the TPC-DI benchmark. First, a description of the possible changes is given. Then, a BPMN4ETL representation is shown for the cases of insertion of a new customer record, and updating an existing one.

3.1. Updates in the TPC-DI benchmark

In the TPC-DI benchmark introduced in Section 1, changes that occurred in the data sources before the historical load are stored in the `CustomerMmgt.xml` file. Thus, updates (in this case, over the dimensions DimCustomer and DimAccount) are not only present during the incremental load, but also in the historical load.

As mentioned in Section 2, the dimension tables for customers and accounts are related to each other in a way such the DimAccount table has a surrogate key that references the surrogate key (Sk_CustomerID) of the DimCustomer table, conforming type 2 SCDs with dependencies. Therefore, upon changes over DimCustomer, the DimAccount table may need to be updated (see below).

The type of changes over the dimension tables above can be: (a) NEW, where a new customer is inserted, always associated with a new account; (b) ADDACCT, where one or more new accounts are associated with an existing customer (this update does not affect customer records); (c) UPDACCT, which updates the information in one or more existing accounts (also not affecting customer records); (d) UPDCUST, which updates existing customer's information; (e) CLOSEACCT, which closes one or more existing accounts, not affecting customer records; (f) INACT, which sets the status of an existing customer, and his/her associated active accounts, to "inactive". Note that for UPDCUST and INACT actions, no account fields are included in the file. Also note that for UPDACCT, attributes that are not updated are not included. Thus, the CDC records in this case include the identifying data, and the updated attributes. Finally, since a change to an existing customer causes a new tuple to be inserted in DimCustomer with a new surrogate key, *all accounts belonging to that customer must be updated with the new customer surrogate key*. Thus, new tuples must be inserted in DimAccount.

Everything is ready now to explain the BPMN4ETL conceptual design. To avoid redundancy, the process will be explained for the case of *type 2 SCDs with dependencies.*

3.2. Conceptual BPMN design for slowly changing dimensions with dependencies

This section illustrates the BPNN4ETL representation for a type 2 SCD is described, by means of two cases: first, the update of a new customer is shown. Then, the update of an existing customer and its associated accounts is explained.

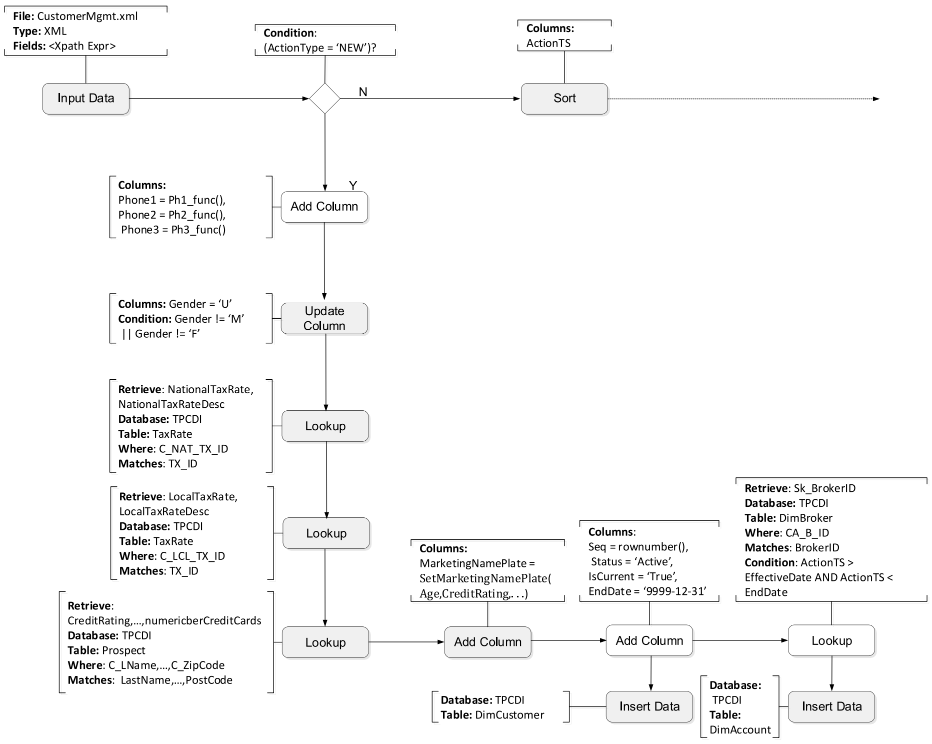
Fig. 2 depicts the BPMN4ETL specification for the case of the insertion of a new customer(ActionType = ''NEW''). The whole process starts with an input data task which reads all attributes of the `CustomerMmgt.xml` file. There is first an Input data task. A condition check on the action type is then performed. If ActionType is not ''NEW'', a Sort data task is performed based on the action's timestamp, to indicate the order in which each change occurred. Otherwise, an Add Column task adds the phone attributes to the flow. These attributes are the Phone1,Phone2, and Phone3 columns of table DimCustomer. Then, an Update Column task inserts the character "U" into the Gender column if the value in it is not 'M' or 'F'. Two Lookup tasks follow, retrieving columns from the TaxRate table, such that the TX_ID column of this table matches the C_NAT_TX_ID attribute of the flow (the national tax id) and the C_LCL_TX_ID (the local tax id). The next lookup task retrieves a number of columns from the table Prospect, which are used to calculate the MarketingNamePlate column, added to the flow in the next Add Column task. After that, an Add Column task adds the column Seq to the flow, to give a row number to each row (that will eventually be stored in the surrogate key columns for both DimCustomer and DimAccount). The status is also set to 'Active', and the new tuple made the most current one, for which the effective and end dates must be also set. The tuple then splits in two paths, namely one to continue with the insertion into DimCustomer, and the other to insert into DimAccount. The attributes for inserting into the latter are all found in the flow, except for Sk_BrokerID, which is retrieved from the DimBroker table by matching CA_B_ID with BrokerID where the action timestamp (ActionTS) falls between the EffectiveDate and EndDate of the table DimBroker. These date check prevents a situation where a tuple in the account dimension references a broker that did not exist when the account was created. Next, an Add Column task adds the Sk_AccountID column to the flow. Finally an Insert Data task inserts the tuple into DimAccount.

Fig. 3 shows the conceptual design of the UPDCUST action for the historical load. Initially, an Add Column task updates the phone attributes of the sources. The next five tasks (Lookup, Update Column, Drop Column, Lookup and Add Column), implement the updates defined in the TPC-DI benchmark specifications. Note that Status is set to "active" in the Add Column task to indicate that an updated customer is still active. Since DimCustomer is a SCD-type 2 table, upon updating a customer record, a new Sk_CustomerID value must be inserted for the new current tuple. Thus, the current Sk_CustomerID value will not refer to the correct value in the DimAccount table anymore. Further, the only link to DimAccount is the Sk_CustomerID value of the current DimCustomer record. Thus, a lookup is performed using the customer identifier, followed by a Rename Column task that renames the current Sk_CustomerID attribute to Sk_CustomerID_OLD. In the next Add Column task, the maximum Sk_CustomerID value of DimCustomer is retrieved and incremented by one, becoming the Sk_CustomerID for the new tuple. This is followed by an Update Data task that sets the IsCurrent and EndDate values of the current tuple in DimCustomer to 'False' and 'ActionTS' (the action timestamp) respectively. Then the new tuple is inserted into DimCustomer, with IsCurrent and EndDate values to 'True' and '9999-12-31' respectively. Now, all the accounts of the updated customer must be updated, setting the new value of Sk_CustomerID. These accounts are found through a Lookup task with the Sk_CustomerID_OLD value that has been saved in the flow. This value is matched with the Sk_CustomerID value of DimAccount. After this, the current tuple of this account in DimAccount, for this customer, is logically deleted, by setting the IsCurrent and EndDate values to 'False' and 'ActionTS', respectively, using an Update Data task. Since a rule in the TPC-DI specification document requires that there must be at most one update to a 'natural' key record on any given day (even when the source data contains more than one update), a tuple is only deleted if the EffectiveDate value is not equal to the ActionTS value (which will become the EndDate of the deleted tuple). Otherwise, another Update Data task sets their Sk_CustomerID value to the new Sk_CustomerID value in the flow. Then, an Add Column task adds the Sk_AccountID column to the flow. The row number value of each tuple is added to the maximum Sk_AccountID value since more than one account could belong to the same customer being updated. Finally, an Insert Data task inserts the tuples in DimAccount.

4. An extended relational algebra for ETL processes

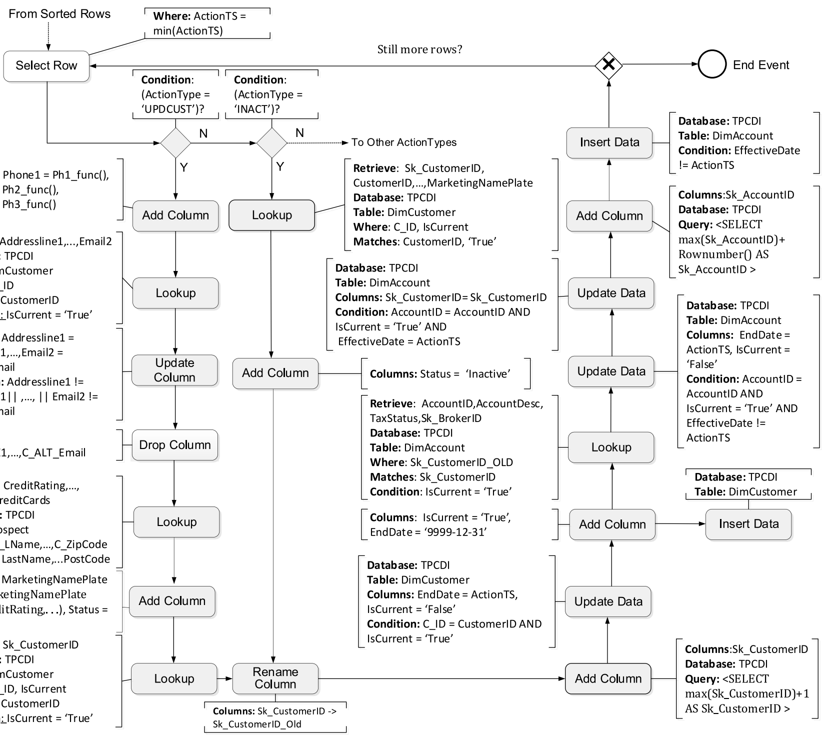
This section presents an extended RA that can be used for implementing ETL processes. RA is a well-studied formal language, that can be used to automatically generate SQL queries to be executed over any Relational Database Management System (RDBMS). It is assumed that relations may contain duplicate tuples, a typical situation in ETL processes that take input data for arbitrary data sources. Along the lines of [9], the paper uses an extension of RA that allows representing SQL data modification operations. In Section 5 these operations are applied to implement the ETL tasks specified using BPMN4ETL in Section 3.

The extended RA operations are shown in Fig. 4. On the left-hand side, the typical RA operations are depicted. A description of these operations can be found in classic database literature (see, e.g., [10]). The additional operators are indicated on the right-hand side of the figure, and detailed next.



File: CustomerMgmt.xml
Type: XML
Fields: <Xpath Expr>
Input Data
Condition: (ActionType = 'NEW')?
N
Y
Columns: ActionTS
Sort
Columns: Phone1 = Ph1_func(), Phone2 = Ph2_func(), Phone3 = Ph3_func()
Add Column
Columns: Gender = 'U'
Condition: Gender != 'M' || Gender != 'F'
Update Column
Retrieve: NationalTaxRate, NationalTaxRateDesc
Database: TPCDI
Table: TaxRate
Where: C_NAT_TX_ID
Matches: TX_ID
Lookup
Retrieve: LocalTaxRate, LocalTaxRateDesc
Database: TPCDI
Table: TaxRate
Where: C_LCL_TX_ID
Matches: TX_ID
Lookup
Retrieve: CreditRating,...,numericberCreditCards
Database: TPCDI
Table: Prospect
Where: C_LName,...,C_ZipCode
Matches: LastName,...,PostCode
Lookup
Columns: MarketingNamePlate = SetMarketingNamePlate(Age,CreditRating,...)
Add Column
Columns: Seq = rownumber(), Status = 'Active', IsCurrent = 'True', EndDate = '9999-12-31'
Add Column
Retrieve: Sk_BrokerID
Database: TPCDI
Table: DimBroker
Where: CA_B_ID
Matches: BrokerID
Condition: ActionTS > EffectiveDate AND ActionTS < EndDate
Lookup
Database: TPCDI
Table: DimCustomer
Insert Data
Database: TPCDI
Table: DimAccount
Insert Data

Fig. 2. Conceptual schema of the ETL process for updating of a customer using a type 2 SCD approach.



From Sorted Rows
Where: ActionTS = min(ActionTS)
Select Row
Still more rows?
End Event
Condition: (ActionType = 'UPDCUST')?
Condition: (ActionType = 'INACT')?
N
N
Y
Y
To Other ActionTypes
Columns: Phone1 = Ph1_func(), Phone2 = Ph2_func(), Phone3 = Ph3_func()
Add Column
Lookup
Retrieve: Sk_CustomerID, CustomerID,...,MarketingNamePlate
Database: TPCDI
Table: DimCustomer
Where: C_ID, IsCurrent
Matches: CustomerID, 'True'
Retrieve: Addressline1,...,Email2
Database: TPCDI
Table: DimCustomer
Where: C_ID
Matches: CustomerID
Condition: IsCurrent = 'True'
Lookup
Database: TPCDI
Table: DimAccount
Columns: Sk_CustomerID= Sk_CustomerID
Condition: AccountID = AccountID AND IsCurrent = 'True' AND EffectiveDate = ActionTS
Columns: Addressline1 = C_ADLINE1,...,Email2 = C_ALT_Email
Condition: Addressline1 != C_ADLINE1|| ,..., || Email2 != C_ALT_Email
Update Column
Add Column
Columns: Status = 'Inactive'
Columns: C_ADLINE1,...,C_ALT_Email
Drop Column
Retrieve: AccountID,AccountDesc, TaxStatus,Sk_BrokerID
Database: TPCDI
Table: DimAccount
Where: Sk_CustomerID_OLD
Matches: Sk_CustomerID
Condition: IsCurrent = 'True'
Retrieve: CreditRating,..., NumberCreditCards
Database: TPCDI
Table: Prospect
Where: C_LName,...,C_ZipCode
Matches: LastName,...PostCode
Lookup
Columns: IsCurrent = 'True', EndDate = '9999-12-31'
Columns: MarketingNamePlate = SetMarketingNamePlate (Age,CreditRating,...), Status = 'Active'
Add Column
Database: TPCDI
Table: DimCustomer
Columns: EndDate = ActionTS, IsCurrent = 'False'
Condition: C_ID = CustomerID AND IsCurrent = 'True'
Retrieve: Sk_CustomerID
Database: TPCDI
Table: DimCustomer
Where: C_ID, IsCurrent
Matches: CustomerID
Condition: IsCurrent = 'True'
Lookup
Rename Column
Columns: Sk_CustomerID -> Sk_CustomerID_Old
Add Column
Columns:Sk_CustomerID
Database: TPCDI
Query: <SELECT max(Sk_CustomerID)+1 AS Sk_CustomerID >
Update Data
Add Column
Insert Data
Database: TPCDI
Table: DimCustomer
Lookup
Update Data
Database: TPCDI
Table: DimAccount
Columns: EndDate = ActionTS, IsCurrent = 'False'
Condition: AccountID = AccountID AND IsCurrent = 'True' AND EffectiveDate != ActionTS
Update Data
Add Column
Columns:Sk_AccountID
Database: TPCDI
Query: <SELECT max(Sk_AccountID)+ Rownumber() AS Sk_AccountID >
Insert Data
Database: TPCDI
Table: DimAccount
Condition: EffectiveDate != ActionTS

Fig. 3. BPMN4ETL design for updating a customer (SCD type 2 table).

Operator	Notation
Selection	$\sigma_C(R)$
Projection	$\pi_{A_1,..,A_n}(R)$
Cartesian Product	$R_1 \times R_2$
Union	$R_1 \cup R_2$
Intersection	$R_1 \cap R_2$
Difference	$R_1 - R_2$
Join	$R_1 \bowtie_C R_2$
Natural Join	$R_1 * R_2$
Left Outer Join	$R_1 ⟕_C R_2$
Right Outer Join	$R_1 ⟖_C R_2$
Full Outer Join	$R_1 ⟗_C R_2$
Semijoin	$R_1 \ltimes_C R_2$
Division	$R_1 \div R_2$

Operator	Notation
Aggregate	$\mathcal{A}_{A_1,..,A_m \mid C_1=\mathsf{F}_1(B_1),..,C_n=\mathsf{F}_n(B_n)}(R)$
Delete	$R \leftarrow R - \sigma_C(R)$
Extend	$\mathcal{E}_{A_1=\mathsf{Expr}_1,\ldots,A_n=\mathsf{Expr}_n}(R)$
Input	$R \leftarrow \mathcal{I}_{A_1,\ldots,A_n}(F)$
Insert	$R \leftarrow R \cup S$ or $R \leftarrow S$
Lookup	$R \leftarrow \pi_{A_1,\ldots A_n}(R_1 \bowtie_C R_2)$
Remove duplicates	$\delta(R)$
Rename	$\rho_{A_1 \leftarrow B_1,\ldots,A_n \leftarrow B_n}(R)$ or $\rho_S(R)$
Sort	$\tau_A(R)$
Update	$\mathcal{U}_{A_1=\mathsf{Expr}_1,\ldots,A_n=\mathsf{Expr}_n \mid C}(R)$
Update Set	$R \leftarrow \mathcal{U}(R)_{C \mid A_1=\mathsf{Expr}_1,\ldots,A_n=\mathsf{Expr}_n}(S)$

Fig. 4. Relational algebra operators (left). Extended relational operators (right).

- **Aggregate**: Let F be an aggregate function such as Count, Min, Max, Sum, or Avg. The aggregate operator $\mathcal{A}_{A_1,\ldots,A_m|C_1=\mathsf{F}_1(B_1),\ldots,C_n=\mathsf{F}_n(B_n)}(R)$ partitions the tuples of R in groups that have the same values in attributes A_i and computes for each group new attributes C_i by applying the aggregate function F_i to the values of attribute B_i in the group, or the cardinality of the group if $F_i(B_i)$ is Count(∗). If no grouping attribute is given (i.e., $m = 0$), the aggregate functions are applied to the whole relation R. The schema of the resulting relation has the attributes $(A_1, \ldots, A_m, C_1, \ldots, C_n)$.
- **Delete**: The delete operation, denoted by $R \leftarrow R - \sigma_C(R)$, removes from relation R the tuples that satisfy the Boolean condition C.
- **Extend**: Given a relation R, the extension operation, denoted $\mathcal{E}_{A_1=Exp_1,\ldots,\ A_n=Exp_n}(R)$, returns a relation where each tuple in R is extended with new attributes A_i obtained by computing the expression Exp_i. The schema of the resulting relation also contains the attributes in R.
- **Input**: This operation, denoted by $R \leftarrow \mathcal{I}_{A_1,\ldots,A_n}(F)$ returns a relation with schema $R(A_1, \ldots, A_n)$ that contains a set of tuples constructed from the content of the file F. This is a meta-operation with several variants allowing to input files of various formats. Only CSV and XML formats are considered here.
- **Insert**: Given two relations R and S, this operation, denoted $R \leftarrow R \cup S$, adds to R the tuples from S. When a new relation R is created with the contents of S, the operation is denoted $R \leftarrow S$. Also if the two relations have different arity, the attributes of the second relation must be explicitly stated as, e.g., $R \leftarrow R \cup \pi_{B_1,\ldots,B_n} S$.
- **Lookup**: The lookup operation is given by $R \leftarrow \pi_{A_1,\ldots,A_n}(R_1 \bowtie_C R_2)$, where the join operation can be any of the six types in Fig. 4.
- **Remove Duplicates**: This operation, denoted $\delta(R)$, returns a relation that contains the tuples of R without duplicates.
- **Rename**: This operation is applied over relation names or over attribute names. For the former, the operation is denoted by $\rho_S(R)$, where the input relation R is renamed to S. For attributes, $\rho_{A_1 \leftarrow B_1,\ldots,A_n \leftarrow B_n}(R)$, returns a relation where the attributes A_i in R are renamed to B_i, respectively.
- **Sort**: This operation, denoted $\tau_A(R)$, sorts a relation that contains the tuples of R sorted by the attribute A.
- **Update Column**: This operation, denoted $\mathcal{U}_{A_1=Expr_1,\ldots,A_n=Expr_n|C}(R)$, returns a relation where for each tuple in R that satisfies the Boolean condition C, the value of the attribute A_i is replaced, respectively, by the value of Expr_i.
- **Update Data**: Denoted $R \leftarrow \mathcal{U}(R)_{C|A_1=\mathsf{Expr}_1,\ \ldots,A_n=\mathsf{Expr}_n}(S)$, this operation updates tuples in R that correspond to tuples in S that satisfy the Boolean condition C. The value of attribute A_i is replaced by the value of the expression Expr_i. Unlike the Update Column operation, the Update data operation matches the tuples of two relations before the boolean condition is applied.

Fig. 5 shows how tasks in the BPMN4ETL methodology can be translated into RA. For example, the Add Column task, which adds columns to the flow, is translated as an Extend operation. Analogously, the Aggregate Data task, which adds columns to the flow, based on aggregating data from the input flow, is translated as an Aggregate operation. The Drop Column task shown in Fig. 5 removes the last two columns A_{n-1}, A_n from the flow. The Drop Column operation is specified in RA as a projection of all columns except the removed ones. In the Input Data task, the Fields (attributes) are generated by XPath expressions. The Update Data task is translated as an Update Set operation with matching attributes explicitly stated in the condition.

BPMN Data task	Relational Algebra Expression
Columns: A_1= Expr$_1$(),…, A_n= Expr$_n$() Add Column	$\mathcal{E}_{A_1=\mathsf{Expr}_1,\ldots,A_n=\mathsf{Expr}_n}(R)$
Group By: A_1,…, A_m **Columns:** C_1= $F_1(B_1)$,…,C_n= $F_n(B_n)$ Aggregate	$\mathcal{A}_{A_1,\ldots,A_m \mid C_1=\mathsf{F}_1(B_1),\ldots,C_n=\mathsf{F}_n(B_n)}(R)$
Table: R **Condition:** C Delete Data	$R \leftarrow R - \sigma_C(R)$
Columns: A_{n-1},A_n Drop Column	$R \leftarrow \pi_{A_1,\ldots,A_{n-2}} R$
File: F.xml **Type:** XML **Fields:** <XPath Expr> Input Data	$R \leftarrow \mathcal{I}_{A_1,\ldots,A_n}(F)$
Input: R_1.*, R_2.* **Output:** R.* Union	$R \leftarrow R_1 \cup R_2$
Database: **Table:** R **Coumns:** R.A_1=S.B_1,…,R.A_n=S.B_n Insert Data	$R \leftarrow R \cup \pi_{B_1,\ldots,B_n} S$
Condition: C **Join Type:** Inner Join Join	$R_1 \bowtie_C R_2$
Retrieve: A_1,…,A_n **Database:** **Table:** R_2 **Condition:** C Lookup	$R \leftarrow \pi_{A_1,\ldots,A_n}(R_1 \bowtie_C R_2)$
Columns: A Sort	$\tau_A(R)$
Columns: A_1->B_1,…, A_n->B_n Rename	$\rho_{A_1 \leftarrow B_1,\ldots,A_n \leftarrow B_n}(R)$
Columns: A_1= Expr$_1$(),…, A_n= Expr$_n$() **Condition:** C Update Column	$\mathcal{U}_{A_1=\mathsf{Expr}_1,\ldots,A_n=\mathsf{Expr}_n \mid C}(R)$
Table: R **Columns:** A_1= Expr$_1$(),…, A_n= Expr$_n$() **Where:** S.B_1 **Matches:** R.A_1 Update Data	$R \leftarrow \mathcal{U}(R)_{R.A_1=S.B_1 \mid A_1=\mathsf{Expr}_1,\ldots,A_n=\mathsf{Expr}_n}(S)$

Fig. 5. Translation of BPMN4ETL tasks to RA operations.

5. Relational algebra specification for updating a type 2 SCD with dependencies

This section shows how the representation of the ETL workflow that loads new and updated customers in DimCustomer and DimAccount dimension tables of TPC-DI, using the RA operations presented in Section 4. That means, this is the RA equivalent to the BPMN4ETL representation described in Section 3.

First, the insertion of a new customer is addressed. For this, Fig. 6 describes the ETL process for tuples with ActionType = 'NEW'. First, all tuples in the XML file input into a relation called All (Eq. 3). The variable Temp0 holds all tuples except those with ActionType = 'NEW' (Eq. 4). Equations 5 through 17 process the tuple if the value of the ActionType attribute is 'NEW'. In Eq. 5, a tuple from Temp0 is stored into variable NTemp1 if ActionType = 'NEW'. After that, the phone columns are added, by applying functions that take as parameters the phone attributes in the XML file (Eq. 6). For simplicity, the parameters of the phone functions are not listed. In Eq. 7, the attribute Gender is updated, using a function that inserts 'U' if the value is neither 'M' or 'F'. NationalTaxRate, NationalTaxRateDesc, LocalTaxRate, LocalTaxRateDesc are set by obtaining the corresponding TX_NAME, TX_RATE values in from the DW table TaxRATE. After this, TX_NAME and TX_RATE are renamed (Eqs. 8 through 11). The attributes from the DW table Prospect are then retrieved (Eq. 12). These attributes are used by the function SetMarketingNamePlate() to compute the value of MarketingNamePlate (Eq. 13). The Seq column is added in Eq. 14, to provide a row number to each row that will be stored as the surrogate key columns for both DimCustomer (in Eq. 15) and DimAccount (in Eq. 17). Values to other columns are set as well (Status, IsCurrent, EffectiveDate, EndDate). Then, the new customer's data are inserted into the dimension table DimCustomer (Eq. 15). Finally, data must be inserted into the DimAccount table. For this, the Sk_BrokerID is obtained by matching the attributes Customer/Account/CA_B_ID and DimBroker.BrokerID, such that the action's timestamp falls within the EffectiveDate and EndDate of DimBroker (Eq. 16). Finally, the new account data are inserted into DimAccount (Eq. 17).

Updating a new customer record is addressed next. The implementation in RA of the process described in the running example for tuples with ActionType = 'UPDCUST', is shown in Fig. 7. Variable Temp0 holds all tuples except for the ones with ActionType='NEW', and Temp1 holds the tuple pointed to by a cursor from Temp0 at any particular time. For the sake of space,

$$\text{All} \leftarrow \mathcal{I}_{\text{ActionType,ActionTS,C\_ID,}\ldots}(\text{Customer.XML}) \tag{3}$$

$$\text{Temp0} \leftarrow \tau_{\text{ActionTS}}(\sigma_{\text{ActionType}\neq\text{'NEW'}}(\text{All})) \tag{4}$$

$$\text{NTemp1} \leftarrow \sigma_{\text{ActionType = 'NEW'}}(\text{All}) \tag{5}$$

$$\text{NTemp2} \leftarrow \mathcal{E}_{\text{Phone1 = GetPhone1(),..., Phone3 = GetPhone3()}}(\text{NTemp1}) \tag{6}$$

$$\text{NTemp2} \leftarrow \mathcal{U}_{\text{Gender = UpdateGender()}}(\text{NTemp2}) \tag{7}$$

$$\text{NTemp3} \leftarrow \pi_{\ldots\text{,TX\_NAME,TX\_RATE}}(\text{NTemp2} \ltimes_{\text{C\_NAT\_TX\_ID=TX\_ID}} \text{TaxRate}) \tag{8}$$

$$\text{NTemp3} \leftarrow \rho_{\text{TX\_Name}\leftarrow\text{NationalTaxRateDesc, TX\_Rate}\leftarrow\text{NationalTaxRate}}(\text{NTemp3}) \tag{9}$$

$$\text{NTemp3} \leftarrow \pi_{\ldots\text{,TX\_NAME,TX\_RATE}}(\text{NTemp2} \ltimes_{\text{C\_LCL\_TX\_ID=TX\_ID}} \text{TaxRate}) \tag{10}$$

$$\text{NTemp3} \leftarrow \rho_{\text{TX\_Name}\leftarrow\text{LocalTaxRateDesc, TX\_Rate}\leftarrow\text{LocalTaxRate}}(\text{NTemp3}) \tag{11}$$

$$\text{NTemp4} \leftarrow \pi_{\ldots\text{,AgencyID,Age,}\ldots}(\text{NTemp3} \ltimes_{\text{C\_LName,}\ldots\text{,C\_ZipCode= LastName,...PostCode}} \text{Prospect}) \tag{12}$$

$$\text{NTemp5} \leftarrow \mathcal{E}_{\text{MarketingNamePlate = SetMarketingNamePlate(Age,CreditRating,}\ldots)}(\text{NTemp4}) \tag{13}$$

$$\text{NTemp6} \leftarrow \mathcal{E}_{\text{Seq = rownumber(), Status = 'Active', IsCurrent = 'True', EffectiveDate = ActionTS, EndDate = '9999-12-30'}}(\text{NTemp5}) \tag{14}$$

$$\text{DimCustomer} \leftarrow \text{DimCustomer} \cup (\pi_{\text{Seq, C\_ID, C\_Tax\_ID, Status,...,EndDate}}(\text{NTemp6})) \tag{15}$$

$$\text{NTemp7} \leftarrow \pi_{\ldots\text{,Sk\_BrokerID}}(\text{NTemp6} \ltimes_{\text{CA\_B\_ID = BrokerID}\wedge \text{ActionTS}\geq\text{EffectiveDate}\wedge\text{ActionTS}<\text{EndDate}} \text{DimBroker}) \tag{16}$$

$$\text{DimAccount} \leftarrow \text{DimAccount} \cup (\pi_{\text{Seq, CA\_ID, SK\_BrokerID, Status,...,EndDate}}(\text{NTemp7})) \tag{17}$$

Fig. 6. RA expressions to model the historical load for a new customer.

only the part concerning SCDs will be explained (Eqs. 28 through 30). Equations 28 and 29 obtain the Sk_CustomerID of the current customer tuple in DimCustomer, and rename it to Sk_CustomerID_OLD, to keep the current surrogate key of DimCustomer in the flow, for the reasons already explained. Eqs. 30–31 add Sk_CustomerID to the flow (the new surrogate key value is computed by adding 1 to the maximum Sk_CustomerID value in DimCustomer). The corresponding current tuple in DimCustomer is then "deleted" (Eq. 32). Then, the remaining columns needed are added (Eq. 33), and the tuple is inserted into DimCustomer (Eq. 34). After this, all current accounts of the customer are obtained (Eq. 35) and "deleted" (by setting, e.g., IsCurrent as 'False', in Eq. 36). Again, only one tuple is inserted, for accounts such that EffectiveDate $\neq$ ActionTS. For accounts with EffectiveDate = ActionTS, only their Sk_CustomerID values are updated (Eq. 37). Finally, Eq. 38 adds Sk_AccountID to the flow. This is the maximum Sk_AccountID in DimAccount plus the rownumber() value of each current account tuple. Finally, the tuples are inserted into DimAccount (Eq. 39).

6. Performance evaluation

This section describes the experimental evaluation aimed at showing that modeling ETL workflows with Extended RA and translating it to SQL queries results in more efficient processes than the ones that result from translating BPMN4ETL to typical ETL tools. In this case, the chosen tools were Pentaho DI and Talend, probably the most popular open source data integration tools. First, the experimental setting is described. Then, results of the different tests are reported. Finally, these results are discussed.

6.1. Experimental setting

The experiments were run over the TPC-DI benchmark introduced in Section 1. The benchmark contains one historical load and two identical incremental loads. Data sources for these loads are of different formats (xml, csv, txt, and so on). Also, the benchmark performs several kinds of operations and transformations, including error checking, surrogate key lookups, data type conversions, aggregate operations, data updates, among other ones. Rules for both, the historical load and incremental updates, are described in the specification document and the reader is referred to the documentation for details. The data of the benchmark are extracted into files in a staging area (see Fig. 1) and the ETL implementation inserts them in the DW.

The historical and incremental loads were run in three ways: (a) Using Pentaho DI,[1] translating the BPMN4ETL specification directly into Pentaho; (b) Using Talend Open Studio for Data Integration[2]; translating the BPMN4ETL specification directly into

[1] https://github.com/pentaho/pentaho-kettle
[2] https://www.talend.com/products/talend-open-studio

$$\text{Temp1} \leftarrow \sigma_{\text{FetchCursorRow()}}(\text{Temp0}) \tag{18}$$

$$\text{UCTemp1} \leftarrow \sigma_{\text{ActionType = 'UPDCUST'}}(\text{Temp1}) \tag{19}$$

$$\text{UCTemp2} \leftarrow \mathcal{E}_{\text{Phone1 = GetPhone1(),.. . . , Phone3 = GetPhone3()}}\ (\text{UCTemp1}) \tag{20}$$

$$\text{UCTemp3} \leftarrow \pi_{\ldots,\text{Addressline1},\ldots,\text{Email2}}(\text{UCTemp2} \bowtie_{\text{C\_ID = CusomerID} \wedge \text{IsCurrnet = 'True'}} \text{DimCustomer}) \tag{21}$$

$$\text{UCTemp4} \leftarrow \sigma_{\text{Addressline1} \neq \text{C\_ADLINE1} \vee,\ldots,\vee \text{Email2} \neq \text{C\_ALT\_EMAIL}}(\text{UCTemp3}) \tag{22}$$

$$\text{UCTemp5} \leftarrow \sigma_{\text{Addressline1} = \text{C\_ADLINE1} \vee,\ldots,\vee \text{Email2} = \text{C\_ALT\_EMAIL}}(\text{UCTemp4}) \tag{23}$$

$$\text{UCTemp6} \leftarrow \mathcal{U}_{\text{Addressline1} = \text{C\_ADLINE1},\ldots,\text{Email2} = \text{C\_ALT\_EMAIL}}(\text{UCTemp4}) \tag{24}$$

$$\text{UCTemp7} \leftarrow \text{UCTemp5} \cup \text{UCTemp6} \tag{25}$$

$$\text{UCTemp8} \leftarrow \pi_{\ldots,\text{AgencyID},\text{Age},\ldots}(\text{UCTemp7} \bowtie_{\text{C\_LName},\ldots,\text{C\_ZipCode} = \text{LastName},\ldots\text{PostCode}} \text{Prospect}) \tag{26}$$

$$\text{UCTemp9} \leftarrow \mathcal{E}_{\text{MarketingNamePlate = SetMarketingNamePlate(Age,CreditRating,...)}}(\text{UCTemp8}) \tag{27}$$

$$\text{UCTemp10} \leftarrow \pi_{\ldots,\text{Sk\_CustomerID}}(\text{UCTemp9} \bowtie_{\text{C\_ID=CustomerID} \wedge \text{IsCurrent = 'True'}}\ \text{DimCustomer}) \tag{28}$$

$$\text{UCTemp11} \leftarrow \rho_{\ldots,\text{Sk\_CustomerID} \leftarrow \text{Sk\_CustomerID\_OLD}}(\text{UCTemp10}) \tag{29}$$

$$\text{UCTemp12} \leftarrow \sigma_{\text{Sk\_CustomerID = max(Sk\_CustomerID)} + 1}(\text{DimCustomer}) \tag{30}$$

$$\text{UCTemp13} \leftarrow \pi_{\ldots,\text{Sk\_CustomerID}}(\text{UCTemp11} \times \text{UCTemp12}) \tag{31}$$

$$\text{DimCustomer} \leftarrow \mathcal{U}(\text{DimCustomer})_{\text{EndDate = ActionTS, IsCurrent = False}|\text{CustomerID=C\_ID} \wedge \text{IsCurrent = 'True'}}(\text{UCTemp13}) \tag{32}$$

$$\text{UCTemp14} \leftarrow \mathcal{E}_{\text{Status = 'Active', IsCurrent = 'True', EffectiveDate = ActionTS, EndDate = '9999-12-30'}}(\text{UCTemp14}) \tag{33}$$

$$\text{DimCustomer} \leftarrow \text{DimCustomer} \cup (\pi_{\text{C\_ID},\ldots,\text{EndDate}}(\text{UCTemp14})) \tag{34}$$

$$\text{UCTemp15} \leftarrow \pi_{\text{AccountID, AccountDesc, TaxStatus}}(\text{UCTemp14} \bowtie_{\text{Sk\_CustomerID\_OLD = Sk\_CustomerID} \wedge \text{IsCurrent = 'True'}} \text{DimAccount}) \tag{35}$$

$$\text{DimAccount} \leftarrow \mathcal{U}(\text{DimAccount})_{\text{EndDate = ActionTS, IsCurrent = False}|\text{AccountID=AccountID} \wedge \text{IsCurrent = 'True'} \wedge \text{EffectiveDate} \neq \text{ActionTS}}(\text{UCTemp15}) \tag{36}$$

$$\text{DimAccount} \leftarrow \mathcal{U}(\text{DimAccount})_{\text{Sk\_CustomerID = Sk\_CustomerID }|\text{AccountID=AccountID} \wedge \text{IsCurrent = 'True'} \wedge \text{EffectiveDate=ActionTS}}(\text{UCTemp15}) \tag{37}$$

$$\text{UCTemp16} \leftarrow \sigma_{\text{Sk\_AccountID = max(Sk\_AccountID) + rownumber()}}(\text{DimAccount}) \tag{38}$$

$$\text{DimAccount} \leftarrow \text{DimAccount} \cup (\pi_{\text{AccountID, SK\_BrokerID, Status,...,EndDate}}(\sigma_{\text{EffectiveDate} \neq \text{ActionTS}} \text{UCTemp16}))) \tag{39}$$

Fig. 7. RA expressions to model the historical load for an updated customer.

Talend[3] (c) Translating the BPMN4ETL specification into RA, and then implementing the RA operations using Postgres PLSQL.[4] Both Pentaho Data Integration (PDI) and Talend Open Studio (TOS) are ETL tools developed in Java. They offer a Graphical User Interface (GUI) that allow defining data integration jobs and transformations. The PDI Client (called Spoon), is based on the Standard Widget Toolkit. Spoon enables building transformations, and creating jobs, also scheduling these jobs. PDI is not a code generator but a metadata-driven ETL tool. That means, no code has to be written to perform complex data transformations or ETL tasks. TOS provides an Eclipse-based GUI. Unlike PDI, TOS is a code generator, therefore ETL tasks and jobs are translated into Java or Perl code. The target language is chosen when creating a new project.

The TPC-DI benchmark defines different sizes for the tests, through a scale factor (SF). For instance, for scale factor 3, the benchmark processes 4.5 million records. For scale factors 5 and 10, the benchmark processes 7.8 and 16.1 million records, respectively. The data warehouse sizes were 4 GB, 7 GB, and 15 GB, for scales factors 3, 5, and 10, respectively.

To optimize the performance of the Pentaho DI implementation, the tool's performance recommendations were applied.[5] The memory limit for Pentaho DI was increased from 2GB to 4GB in order to prevent Java out of memory exceptions and to improve

[3] Pentaho and Talend were chosen since most tools do not allow publishing experimental results due to licensing restrictions.

[4] https://www.postgresql.org/docs/9.2/static/plpgsql.html

[5] https://help.pentaho.com/Documentation/7.1/0P0/100/040/010

Table 1
Results of implementing TPCDI with relational Algebra (PLSQL) and BPMN (Pentaho PDI) in Hours:Minutes:Seconds.

		Historical	Incremental 1	Incremental 2
SF-3	PLSQL	00:12:50	00:00:09	00:00:07
	Pentaho PDI	11:23:52	00:01:32	00:01:40
	Talend	02:29:00	00:00:43	00:00:43
SF-5	PLSQL	00:22:31	00:00:15	00:00:14
	Pentaho PDI	20:25:32	00:03:03	00:03:11
	Talend	04:05:00	00:01:53	00:01:09
SF-10	PLSQL	02:11:15	00:00:39	00:00:36
	Pentaho PDI	25:08:13	00:11:35	00:12:38
	Talend	09:03:00	00:4:00	00:03:00

performance. Also, subjobs that are not dependent on each other were executed in parallel. To optimize the performance of the TOS implementation, the memory limit of most jobs was increased to 4GB. As in the case of PDI, this reduces the execution time and prevents out of memory exceptions. Also, whenever possible, many columns were derived in a single testsftMap component, instead of including different components for different column derivations. Like in the case of the PDI implementation, jobs were run in parallel whenever possible. All tests were run over an Intel i7 computer, with a RAM of 16 GB, running the Windows 10 Enterprise operating system, and using the PostgreSQL database for the DW storage. Using PostgreSQL database for all implementation makes the comparison fair.

6.2. Results

Table 1 reports the total execution times of the processes, for different SFs. The historical and incremental loads were run, for all three SFs. It can be noticed that for the historical load, PLSQL implementation is orders of magnitude faster, for all scale factors. Differences are also relevant for incremental loads.

To get further insight on the results, Fig. 8 shows the execution times for each table in the TPC-DI DW, for SF = 3 using PLSQL (resulting from the RA approach), Pentaho PDI and Talend (the results for the other SFs are similar). It can be noticed that the PLSQL implementation is faster for loading all tables. Also, results for TOS are better than the ones obtained for PDI. Moreover, it is orders of magnitude faster when it comes to implementing SCDs with dependencies, DimCustomer/DimAccount and DimCompany/DimSecurity. The reasons for these results are discussed in the next section.

6.3. Discussion

Consider the fact table FactHoldings in the benchmark. Figs. 9 and 10 show the ETL processes for the historical load of the table, using Pentaho PDI and Talend, respectively. The PLSQL queries resulting from the RA approach that implements the process are shown below. It can be seen that with the Pentaho PDI implementation, the Merge Sort step must be preceded by two sort steps. The Merge Sort step requires the join condition columns to be already sorted. These sort steps add up to the total execution time since the sort components must finish execution before the ETL process can go on. The absence of these sort queries and components for all joins in PLSQL and Talend lowers the cost of the ETL process. The PLSQL join query uses a hash join which does not require both inputs to be sorted.

```
DROP FUNCTION IF EXISTS FactholdingsLoad();
CREATE FUNCTION FactholdingsLoad()
RETURNS VOID AS $$
BEGIN

 DROP TABLE IF EXISTS FHTemp1,FHTemp2;

 CREATE TEMPORARY TABLE FHTemp1 AS
 SELECT HH_H_T_ID, HH_T_ID, HH_BEFORE_QTY, HH_AFTER_QTY,
 SK_CustomerID,  SK_AccountID,  SK_SecurityID,
 SK_CompanyID, SK_CloseDateID, SK_CloseTimeID,TradePrice
 FROM holdinghistory H, DimTrade D
 WHERE H.HH_T_ID = D.TradeID;

 INSERT INTO Factholdings(TradeID, CurrentTradeID,
 SK_CustomerID, SK_AccountID,  SK_SecurityID,
```

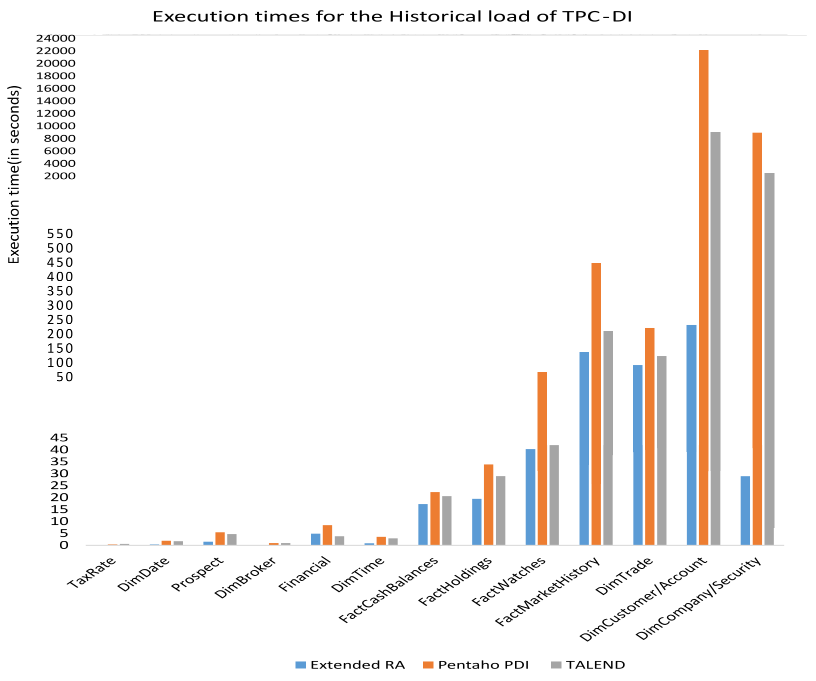


Fig. 8. Execution times for each table for the historical load in the TPC-DI benchmark, using PLSQL, Pentaho Data Integration and Talend.

```
  SK_CompanyID, SK_DateID, SK_TimeID,
  CurrentPrice, CurrentHolding, BatchID)
  SELECT HH_H_T_ID, HH_T_ID, SK_CustomerID, SK_AccountID,
  SK_SecurityID, SK_CompanyID,
  SK_CloseDateID, SK_CloseTimeID, TradePrice, HH_AFTER_QTY,1
  FROM
  FHTemp1 ;

END;
$$ LANGUAGE 'plpgsql';
```

Further, the results obtained are also partially explained by analyzing the way in which each approach implements the loops needed to perform the updates over the SCDs in the benchmark. Recall that the tables DimCustomer and DimAccount are populated from the CustomerMmgt.xml file, which contains all updates to customers and their accounts prior to the historical load. Apart from tuples with ActionType="NEW", which are loaded in batch form, all other action types are loaded using loops, one row at a time. Pentaho PDI handles loops with the Copy Rows to Result step, that stores the rows in main memory and retrieves them one row at a time. For SF = 3, it takes Pentaho PDI 22,156 s out of the total running time for the historical load, to finish running the DimCustomer and DimAccount dimension tables due to this loop. Talend handles loops with the tFlowToIterate component, which reads data line by line from the input flow; data are then stored in iterative global variables flowed by the tFixedFlowInput component. For example, for SF = 3, it takes Talend 9,000 s out of the total running time for the historical load, to finish running the DimCustomer and DimAccount dimension tables due to this loop. The same reasoning applies to the DimCompany and DimSecurity tables.

Finally, both Pentaho PDI and Talend are run in a Java-enabled environment. They provide a Java engine that controls data processing. That means, data from a staging area are extracted into the Java environment for processing. Any ETL task in Pentaho PDI or Talend that requires a lookup for data from the DW increases the ETL execution time, since the data are first loaded into the tool's environment before processing it. For example, several lookup tasks are needed to find values from the most current tuples of the DimCustomer/DimAccount and (DimCompany/DimSecurity) SCDs. Implementing ETL processes based on a translation from the extended RA specification proposed in this paper, as in the case of PLSQL, the above drawbacks are avoided. This is because the data for lookup already exist in the database and hence there are no additional costs due to extraction of the data into the ETL environment. Thus, the ETL execution time increases exponentially when it comes to implementing loops in Pentaho and Talend data integration tools. For each tuple, the tasks that look up values in the DW have to reload the data from the DW into the tool's environment to obtain the correct value(s). An example of the above occurs when finding the surrogate key of the most current

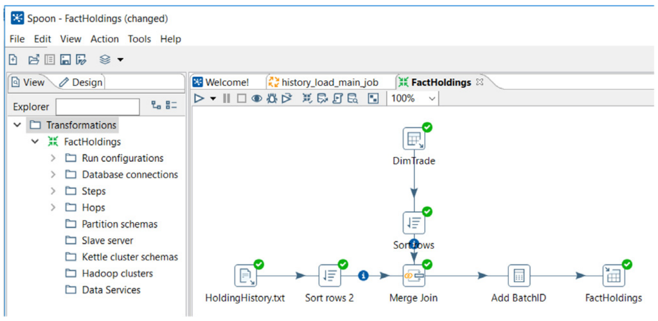


Fig. 9. Load of DimBroker table for Pentaho PDI.

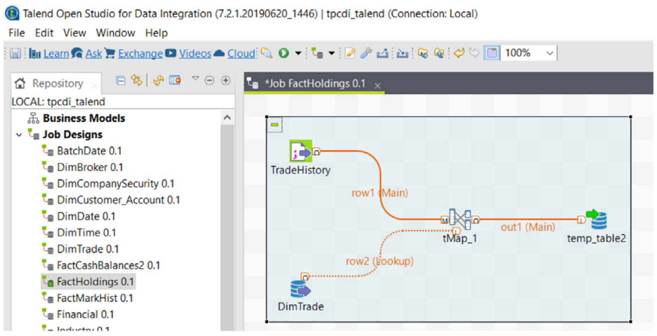


Fig. 10. Load of DimBroker table for Talend.

tuple in DimCustomer or DimAccount, and increasing it by one as the surrogate key of the new tuple to be inserted, as explained in Section 3.

Automatically translating a BPMN4ETL specification to RA would have a crucial impact on the use of the approach discussed in this paper to real world scenarios. There are many BPMN modeling tools in the market. Some of these tools can be extended for BPMN4ETL modeling, and translated automatically to RA (and therefore to SQL, given that this translation is straightforward), with the use of the XML language. An example of an extendable tool is Camunda Modeler.[6] The idea of this automated translation is as follows. First, a model is created using the BPMN4ETL tool (that is, the extension of a BPMN modeling tool). Then, an XML file is automatically generated for such model. The information in this XML file could be used as the input of an algorithm, in order to finally produce the SQL code that will execute the ETL process. This automated translation is ongoing work by the authors, and a full explanation of the process falls outside the scope of this paper.

In summary, the results reported here suggest that implementing ETL SQL-based processes based on a translation from a RA specification is plausible and competitive. The tests showed that the extended RA-to-SQL approach is more efficient in terms of overall execution time for running the TPC-DI benchmark, than the implementations over data integration tools. Of course, a great part of the observed difference is due to the overhead introduced by the higher level of abstraction provided by the process-specification tools. However, an automated translation from BPMN4ETL to RA (or SQL) commented above, would bring the best of both worlds: a high-level of abstraction together with an efficient execution time.

7. Related work

Several different strategies have been proposed to model ETL processes. The conceptual model proposed by Vassiliadis et al. [11] analyzes the structure and data of the data sources and their mapping to a target DW. A set of frequently used ETL activities are identified, like the assignment of surrogate keys, the check for null values, etc. The conceptual model proposed by Trujillo and Luján-Mora [12], uses UML to design ETL processes. Here, each ETL process is represented by a stereotyped class which is in turn represented by an icon. This icon can be used in a UML model instead of the standard representation of a class. This model, however, does not capture some important ETL transformations and mappings between attributes of data sources and data warehouses, and is refined by Muñoz et al. [13], using UML activity diagrams instead of UML class diagrams to model the ETL processes.

The works by El Akkaoui et al. [8,14,15] propose a vendor-independent conceptual metamodel for designing ETL processes based on BPMN. This ETL workflow design combines two perspectives, a control process view, and a data process view. A control process

[6] https://camunda.com/products/modeler/

view consists of all the data processes in the ETL workflow, while the data process view provides a more detailed information of the input and output data of each data process in the control process view. Using BPMN to specify ETL processes makes this model simple and easy to understand at the cost of expressiveness: it is not possible to visualize the transformation of attributes and attribute constraints at any point in the workflow. This approach has been later used in a proposal for assessing of the quality of an ETL design [16].

Santos and Belo have proposed using relational algebra for modeling ETL processes [4–6,17]. In particular, ETL processes for slowly changing dimensions specifications [4], data quality enforcement tasks [5], and ETL conciliation tasks [6], were modeled with relational algebra and applied to a real-world ETL scenario [17]. Since he authors use the classic RA, these works cannot address more complex scenarios like, for instance, the problem of updating SCDs that are referenced by other SCDs (SCDs with dependencies). The extended RA in the present paper makes it easy to translate RA specifications to any ETL scenario. Further, the work above does not include an implementation of the proposal, but only descriptions of possible ones. On the contrary, the present paper shows an implementation over the TPC-DI benchmark (described in Section 6), allowing to compare the approach against alternatives popularly used in real-world scenarios, like Pentaho DI and Talend Open Studio for DI.

Finally, relevant to the work presented here, a research reported in [18] presents a framework for ETL development based on writing software code, instead of specifying the process using commercial tools like Pentaho PDI, Integration Services, etc., and discusses the advantages of this approach.

8. Conclusion

This paper discussed the use of an extension of RA for modeling and implementing ETL processes at the conceptual level, and compared this approach against BPMN4ETL, a conceptual model for ETL processes based on the Business Process Modeling Notation (BPMN), commonly used for specifying business processes. For this comparison, the paper addresses the problem of Slowly Changing Dimensions (SCDs) with dependencies, that is, the case when updating a SCD table impacts on associated SCD tables. To evaluate the efficiency of the ETL processes resulting from the two approaches above, the loading of a portion of the TPC-DI benchmark was used. That is, the TPC-DI benchmark was used in two ways: one, using RA to translate the specification into SQL. The other one, implementing the benchmark using BPMN4ETL, and translating this directly into the Pentaho Data Integration (PDI), and the Talend Open Studio for Data Integration (TOS) tools. The experiments showed that the SQL implementation runs orders of magnitude faster than the other two ones. Therefore, ongoing work by the authors, on the automated translation of BPMN to RA and SQL would allow to keep the high level of abstraction provided by BPMN, while obtaining the important efficiency gain reported here. Finally, since this work did not consider structural changes of the data sources, this issue remains open to be addressed in future research.

CRediT authorship contribution statement

Judith Awiti: Conceptualization, Investigation, Methodology, Writing - original draft, Writing - review & editing. **Alejandro A. Vaisman:** Conceptualization, Methodology, Writing - review & editing, Project administration. **Esteban Zimányi:** Conceptualization, Investigation, Methodology, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work of Judith Awiti is supported by the European Commission through the Erasmus Mundus Joint Doctorate project *Information Technologies for Business Intelligence-Doctoral College* (IT4BI-DC). Alejandro Vaisman has been partially supported by the Erasmus BDMA program, and PICT 2017 Project 1054, from the Argentina Scientific Agency.

References

[1] M. Poess, T. Rabl, H.-A. Jacobsen, B. Caufield, TPC-DI: the first industry benchmark for data integration, Proc. VLDB Endow. 7 (13) (2014) 1367–1378.
[2] Z.E. Akkaoui, E. Zimányi, Defining ETL worfklows using BPMN and BPEL, in: Proceedings of the 12th ACM International Workshop on Data Warehousing and OLAP, ACM, Hong Kong, China, 2009, pp. 41–48.
[3] A.A. Vaisman, E. Zimányi, Data Warehouse Systems: Design and Implementation, Springer, 2014.
[4] V. Santos, O. Belo, Slowly changing dimensions specification a relational algebra approach, Int. J. Inf. Technol. 1 (3) (2011) 63–68.
[5] V. Santos, O. Belo, Modeling ETL data quality enforcement tasks using relational algebra operators, Proc. Technol. 9 (2013) 442–450.
[6] V. Santos, O. Belo, Modelling ETL conciliation tasks using relational algebra operators, in: Proceedings of the 2014 European Modelling Symposium, IEEE, Pisa, Italy, 2014, pp. 275–280.
[7] R. Kimball, J. Caserta, The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data, John Wiley & Sons, 2011.
[8] Z. El Akkaoui, E. Zimányi, Defining ETL worfklows using BPMN and BPEL, in: Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP, ACM, 2009, pp. 41–48.
[9] E. Baralis, J. Widom, An algebraic approach to rule analysis in expert database systems, in: Proceedings of the 20th International Conference on Very Large Data Bases, Morgan Kaufmann, Santiago, Chile, 1994, pp. 475–486.

[10] R. Elmasri, S.B. Navathe, Fundamentals of Database Systems, seventh ed., Addison Wesley, 2015.
[11] P. Vassiliadis, A. Simitsis, S. Skiadopoulos, Conceptual modeling for ETL processes, in: Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP, ACM, 2002, pp. 14–21.
[12] J. Trujillo, S. Luján-Mora, A uml based approach for modeling ETL processes in data warehouses, in: International Conference on Conceptual Modeling, Springer, 2003, pp. 307–320.
[13] L. Muñoz, J.-N. Mazón, J. Pardillo, J. Trujillo, Modelling ETL processes of data warehouses with UML activity diagrams, in: OTM Confederated International Conferences "on the Move to Meaningful Internet Systems", Springer, 2008, pp. 44–53.
[14] Z.E. Akkaoui, J. Mazón, A.A. Vaisman, E. Zimányi, BPMN-based conceptual modeling of ETL processes, in: Proceedings of the 14th Data Warehousing and Knowledge Discovery International Conference, DaWaK 2012, Springer, Vienna, Austria, 2012, pp. 1–14.
[15] Z. El Akkaoui, E. Zimányi, J.-N. Mazón, J. Trujillo, A BPMN-based design and maintenance framework for ETL processes, Int. J. Data Warehous. Min. 9 (3) (2013) 46–72.
[16] Z.E. Akkaoui, A.A. Vaisman, E. Zimányi, A quality-based ETL design evaluation framework, in: Proceedings of the 21st International Conference on Enterprise Information Systems, Vol. 1, ICEIS 2019, Heraklion, Crete, Greece, May 3–5, 2019, 2019, pp. 249–257.
[17] V. Santos, O. Belo, Using relational algebra on the specification of real world ETL processes, in: Proceedings of 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, IEEE, 2015, pp. 861–866.
[18] T.B. Pedersen, Programmatic ETL, in: Business Intelligence and Big Data: 7th European Summer School, eBISS 2017, Bruxelles, Belgium, July 2–7, 2017, in: Tutorial Lectures, vol. 324, 2018, p. 21.

Judith Awiti received a B.Sc Computer Engineering degree from the Kwame Nkrumah University of Science and Technology (KNUST), Ghana, and an M.Sc Information Technology degree from Sikkim Manipal University, India. She was a Senior ICT Assistant at University of Energy and Natural Resources (UENR) from August 2013 to January 2018 and is now a researcher in the IT4BI-DC Erasmus Mundus programme. She has authored and co-authored some scientific papers presented at major database conferences and her current research interests include data warehouses and ETL processes.

Alejandro Vaisman received a BA degree in Civil Engineering, a BA in Computer Science, and a PhD in Computer Science from UBA, under the supervision of Prof. Alberto Mendelzon, from the University of Toronto, Canada. He has been a post-doctoral researcher at the University of Toronto. He has been an Associate Professor at UBA between 1994 and 2013, and is currently full professor at the Instituto Tecnológico of Buenos Aires (ITBA). He has been Vice-Head of the Computer Science Department at UBA, and chair of the Masters Program in Data Mining (between 2005 and 2010). He was a visiting researcher at the University of Toronto, Universidad Politecnica de Madrid, University of Hasselt, and Universidad de Chile. His research interests are in the field of databases, particularly in Business Intelligence, OLAP and Data Warehousing, the Semantic Web and Geographic Information Systems. He has authored and co-authored several scientific papers presented at major database conferences and journals.

Esteban Zimányi is a full professor and the director of the Department of Computer & Decision Engineering (CoDE) at ULB. He started his studies at the Universidad Autónoma de Centro América, Costa Rica. He received a B.Sc. (1988) and a doctorate (1992) in computer science from the Faculty of Sciences at the ULB. During 1997, he was a visiting researcher at the Database Laboratory of the Ecole Polytechnique Fédérale de Lausanne, Switzerland. He co-authored and co-edited 16 books, 18 book chapters, 19 articles in international journals, and 82 papers in international conferences. He is coordinator of the BDMA and IT4BI-DC Erasmus Mundus programmes. He is Editor-in-Chief of the Journal on Data Semantics published by Springer. His current research interests include data warehouses, spatio-temporal databases, geographic information systems, and semantic web.