From Conceptual to Logical ETL Design Using BPMN and Relational Algebra

Judith Awiti[1(✉)], Alejandro Vaisman[2], and Esteban Zimányi[1]

[1] Université Libre de Bruxelles, Brussels, Belgium
{judith.awiti,ezimanyi}@ulb.ac.be
[2] Instituto Tecnológico de Buenos Aires, Buenos Aires, Argentina
avaisman@itba.edu.ar

Abstract. 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 Notation (BPMN) has been proposed for expressing ETL processes at a conceptual level. This paper extends relational algebra (RA) with update operations for specifying ETL processes at a logical level. In this approach, data tasks can be automatically translated into SQL queries to be executed over a DBMS. An extension of RA is presented, as well as a translation mechanism from BPMN to the RA specification. Throughout the paper, the TPC-DI benchmark is used for comparing both approaches. Experiments show the efficiency of the resulting ETL flow with respect to the Pentaho Data Integration tool.

Keywords: OLAP · ETL · BPMN · Data warehousing

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 concep-tual and implementation-independent specification of such processes, that can be then translated into executable specifications for ETL tools. The second is a logical model based on Relational Algebra (RA), a formal language that provides a solid basis to specify ETL processes for relational databases.

Running Example. The TPC-DI benchmark [6] is used as running example throughout the paper, with focuses 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 OLTP database represents transactional information about securities market trading and the entities involved, e.g., customers, accounts, and so on. A CustomerMgmt.xml file represents actions resulting in new or updated customer and account information. For each action, 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. An 'UPDCUST' action updates a customer and all her current accounts. For that action, only the properties used to identify the updated properties of the customer are given. 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.

Contributions. The paper discusses the modeling of Slowly Changing Dimensions with Dependencies, that is, the case when updating a SCD table impacts on associated SCD tables (Sect. 2). As a key contribution, an ETL development approach is proposed, which begins with a BPMN4ETL conceptual model (Sect. 3) translated into RA extended with update operations (Sect. 4) at the logical level. Common ETL tasks and their extended RA specifications are also shown. Although BPMN4ETL has been already proposed (see [1,11]), the problem of modeling SCDs with dependencies using this technique is discussed here for the first time. Related work is covered in Sect. 7. Experiments over the TPC-DI benchmark are carried out and results are reported, suggesting that the above-mentioned approach results in more efficient processes than the ones produced by BPMN4ETL conceptual model translated into the Pentaho Data Integration (PDI) tool (Sect. 6). Conclusions are given in Sect. 8.

2 Slowly Changing Dimensions with Dependencies

Slowly Changing Dimensions (SCD) [3,11] are used in a DW to keep the history of changes that occurs in data sources. Kimball [3] 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.

$$\mathsf{DimA} = \langle \mathsf{SkA}, \mathsf{PkA}, \ldots, \mathsf{IsCurrent}, \mathsf{StartDate}, \mathsf{EndDate} \rangle \quad (1)$$

$$\mathsf{DimB} = \langle \mathsf{SkB}, \mathsf{SkA}, \mathsf{PkB}, \ldots, \mathsf{IsCurrent}, \mathsf{StartDate}, \mathsf{EndDate} \rangle \quad (2)$$

Fig. 1. Schema of DimA and DimB

This paper (as well as the TPC-DI benchmark) tackles SCDs of type 2 with dependencies. Such a Slowly Changing Dimension table contains a surrogate key reference to another dimension table. If an update occurs to the referenced dimension table, the referencing table must be updated as well. To illustrate this, the schemas of two dimension tables are defined, namely DimA and DimB, in Equations 1 and 2. DimB references the surrogate key (SkA) of DimA. DimA and DimB have 'natural' keys PkA and PkB, respectively. When an update occurs in DimA, the (SkA) value of DimB must be replaced by the most current (SkA) value in DimA. Below, the general steps of an ETL to update a tuple in DimA are listed, implementing the dependency stated above.

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. Retire the current tuple from DimA (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. Retire the current tuples in DimB (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).

3 Conceptual BPMN for Slowly Changing Dimensions

The BPMN4ETL conceptual model [1,11], 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.

In the TPC-DI benchmark introduced in Sect. 1, changes that occurred in the data sources before the historical load, are stored in the `CustomerMmgt.xml` file. The type of changes 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; (c) UPDACCT, which updates the information in one or more existing accounts; (d) UPDCUST, which updates existing customer's information; (e) CLOSEACCT, which closes one or more existing accounts; (f) INACT, which sets the status of an existing customer, and her

associated active accounts, to "inactive". Note that for UPDCUST and INACT actions, no account fields are included in the file. Updates over the dimensions DimCustomer and DimAccount are not only present during the incremental load, but also in the historical load. The DimAccount table has a surrogate key that references the surrogate key (Sk_CustomerID) of the DimCustomer table. These two tables are modelled as type 2 SCDs with dependencies.

Figure 2 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 specified 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_CustomerIDOLD. 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_CustomerIDOLD 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 can be used to automatically generate SQL queries to be executed in any Relational Database Management System (RDBMS). The typical

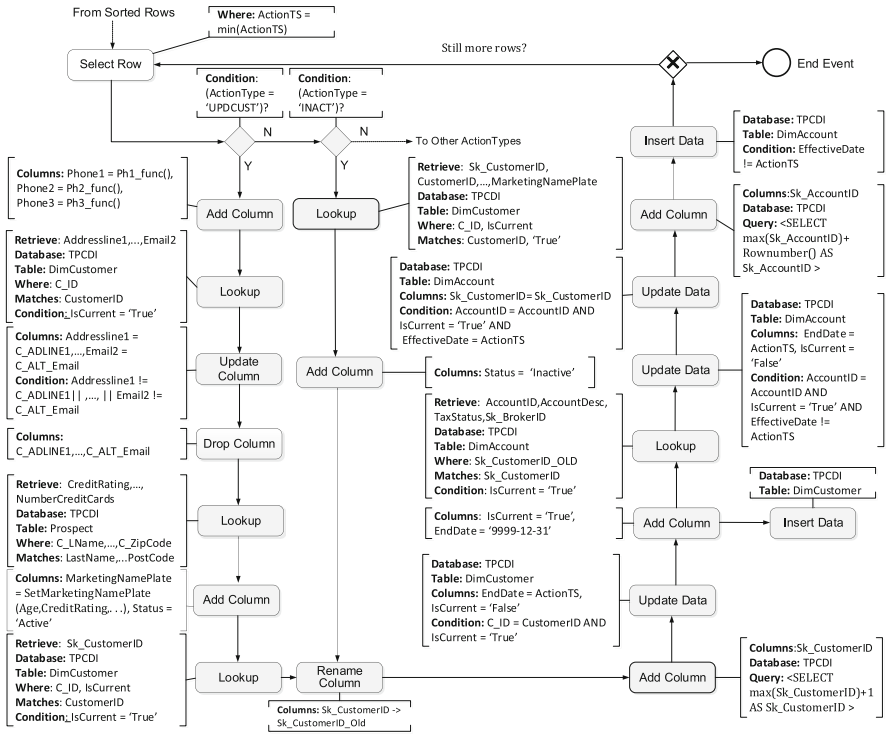

Fig. 2. BPMN4ETL design for updating a customer (SCD type 2 table).

RA operations are shown on the left-hand side of Fig. 3. A description of these operations can be found in classic database literature. In order to model different scenarios of ETL processes, these operations are extended with additional operators as indicated on the right-hand side of the figure, and detailed next.

- **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 of 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 $\mathsf{Count}(*)$. If no grouping attributes are 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.

Operator	Notation
Selection	$\sigma_C(R)$
Projection	$\pi_{A_1,\ldots,A_n}(R)$
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	$\mathcal{A}_{A_1,\ldots,A_m \mid C_1=\mathsf{F}_1(B_1),\ldots,C_n=\mathsf{F}_n(B_n)}(R)$
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	$R \leftarrow \mathcal{U}(R)_{A_1=\mathsf{Expr}_1,\ldots,A_n=\mathsf{Expr}_n \mid C}(S)$

Fig. 3. Relational Algebra operators (left). Extended relational operators (right).

- **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.
- **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. 3.
- **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 $\mathcal{U}_{A_1=Expr_1,\ldots,A_n=Expr_n \mid C}(R)$, 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 Set**: Denoted $R \leftarrow \mathcal{U}(R)_{A_1=\mathsf{Expr}_1,\ldots,A_n=\mathsf{Expr}_n \mid C}(S)$, 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 condition of the Update Set operation includes matching the tuples of two relations.

Figure 4 shows how tasks in the BPMN4ETL methodology can be translated into RA. For example, the Aggregate Data task is translated as an Aggregate operation. The Drop Column operation is specified in RA as a projection of all columns except the removed ones (in this case, A_n and A_{n-1}). 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
Add Column	$\mathcal{E}_{A_1=\mathsf{Expr}_1,\ldots,A_n=\mathsf{Expr}_n}(R)$
Aggregate	$\mathcal{A}_{A_1,\ldots,A_m\mid C_1=\mathsf{F}_1(B_1),\ldots,C_n=\mathsf{F}_n(B_n)}(R)$
Delete Data	$R \leftarrow R - \sigma_C(R)$
Drop Column	$R \leftarrow \pi_{A_1,\ldots,A_{n-2}} R$
Input Data	$R \leftarrow \mathcal{I}_{A_1,\ldots,A_n}(F)$
Union	$R \leftarrow R_1 \cup R_2$
Insert Data	$R \leftarrow R \cup \pi_{B_1,\ldots,B_n} S$
Join	$R_1 \bowtie_C R_2$
Lookup	$R \leftarrow \pi_{A_1,\ldots,A_n}(R_1 \bowtie_C R_2)$
Sort	$\tau_A(R)$
Rename	$\rho_{A_1 \leftarrow B_1,\ldots,A_n \leftarrow B_n}(R)$
Update Column	$\mathcal{U}_{A_1=\mathsf{Expr}_1,\ldots,A_n=\mathsf{Expr}_n\mid C}(R)$
Update Data	$R \leftarrow \mathcal{U}(R)_{A_1=\mathsf{Expr}_1,\ldots,A_n=\mathsf{Expr}_n\mid R.A_1=S.B_1\wedge,\ldots,}(S)$

Fig. 4. Translation of BPMN4ETL tasks to RA operations.

5 Relational Algebra Specification for Type 2 SCDs with Dependencies

The implementation in RA of the process described in the running example, is shown in Fig. 5. 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, only the part concerning SCDs will be explained (Eqs. 13 through 24). Equations 13 and 14 obtain the Sk_CustomerID of the current customer tuple in DimCustomer, and rename it to Sk_CustomerIDOLD, to keep the current surrogate key of DimCustomer in the flow, for the reasons already explained. Equations 15–16 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. 17). Then, the remaining columns needed are added (Eq. 18), and the tuple is inserted into DimCustomer (Eq. 19). After this, all current accounts of the customer are obtained (Eq. 20) and "deleted" (by setting, e.g., IsCurrent as 'False', in Eq. 21). 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. 22). Finally, Eq. 23 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. 24).

6 Performance Evaluation

This section briefly describes and reports results of the experimental evaluation on the TPC-DI benchmark described in Sect. 1. The benchmark contains one historical load and two identical incremental loads. Data sources are of different formats (xml, csv, txt, and so on). The experiments implement the historical and incremental loads in two ways: (a) Using PDI[1], translating the BPMN4ETL

[1] https://github.com/pentaho/pentaho-kettle.

$$\text{Temp1} \leftarrow \sigma_{\text{FetchCursorRow()}}(\text{Temp0}) \quad (3)$$

$$\text{UCTemp1} \leftarrow \sigma_{\text{ActionType = 'UPDCUST'}}(\text{Temp1}) \quad (4)$$

$$\text{UCTemp2} \leftarrow \mathcal{E}_{\text{Phone1 = GetPhone1(),..., Phone3 = GetPhone3()}}(\text{UCTemp1}) \quad (5)$$

$$\text{UCTemp3} \leftarrow \pi_{\ldots,\text{Addressline1},\ldots,\text{Email2}}(\text{UCTemp2} \bowtie_{\text{C\_ID = CusomerID} \wedge \text{IsCurrnet = 'True'}} \text{DimCustomer}) \quad (6)$$

$$\text{UCTemp4} \leftarrow \sigma_{\text{Addressline1} \neq \text{C\_ADLINE1} \vee, \ldots, \vee \text{Email2} \neq \text{C\_ALT\_EMAIL}}(\text{UCTemp3}) \quad (7)$$

$$\text{UCTemp5} \leftarrow \sigma_{\text{Addressline1} = \text{C\_ADLINE1} \vee, \ldots, \vee \text{Email2} = \text{C\_ALT\_EMAIL}}(\text{UCTemp4}) \quad (8)$$

$$\text{UCTemp6} \leftarrow \mathcal{U}_{\text{Addressline1} = \text{C\_ADLINE1}, \ldots, \text{Email2} = \text{C\_ALT\_EMAIL}}(\text{UCTemp4}) \quad (9)$$

$$\text{UCTemp7} \leftarrow \text{UCTemp5} \cup \text{UCTemp6} \quad (10)$$

$$\text{UCTemp8} \leftarrow \pi_{\ldots,\text{AgencyID},\text{Age},\ldots}(\text{UCTemp7} \bowtie_{\text{C\_LName},\ldots,\text{C\_ZipCode} = \text{LastName},\ldots\text{PostCode}} \text{Prospect}) \quad (11)$$

$$\text{UCTemp9} \leftarrow \mathcal{E}_{\text{MarketingNamePlate = SetMarketingNamePlate(Age,CreditRating,...)}}(\text{UCTemp8}) \quad (12)$$

$$\text{UCTemp10} \leftarrow \pi_{\ldots,\text{Sk\_CustomerID}}(\text{UCTemp9} \bowtie_{\text{C\_ID=CustomerID} \wedge \text{IsCurrent = 'True'}} \text{DimCustomer}) \quad (13)$$

$$\text{UCTemp11} \leftarrow \rho_{\ldots,\text{Sk\_CustomerID} \leftarrow \text{Sk\_CustomerIDOLD}}(\text{UCTemp10}) \quad (14)$$

$$\text{UCTemp12} \leftarrow \sigma_{\text{Sk\_CustomerID = max(Sk\_CustomerID)} + 1}(\text{DimCustomer}) \quad (15)$$

$$\text{UCTemp13} \leftarrow \pi_{\ldots,\text{Sk\_CustomerID}}(\text{UCTemp11} \times \text{UCTemp12}) \quad (16)$$

$$\text{DimCustomer} \leftarrow \mathcal{U}(\text{DimCustomer})_{\text{EndDate = ActionTS, IsCurrent = False|CustomerID=C\_ID} \wedge \text{IsCurrent = 'True'}}(\text{UCTemp13}) \quad (17)$$

$$\text{UCTemp14} \leftarrow \mathcal{E}_{\text{Status = 'Active', IsCurrent = 'True', EffectiveDate = ActionTS, EndDate = '9999-12-30'}}(\text{UCTemp14}) \quad (18)$$

$$\text{DimCustomer} \leftarrow \text{DimCustomer} \cup (\pi_{\text{C\_ID},\ldots,\text{EndDate}}(\text{UCTemp14})) \quad (19)$$

$$\text{UCTemp15} \leftarrow \pi_{\text{AccountID, AccountDesc, TaxStatus}}(\text{UCTemp14} \bowtie_{\text{Sk\_CustomerIDOLD = Sk\_CustomerID} \wedge \text{IsCurrent = 'True'}} \text{DimAccount}) \quad (20)$$

$$\text{DimAccount} \leftarrow \mathcal{U}(\text{DimAccount})_{\text{EndDate = ActionTS, IsCurrent = False|AccountID=AccountID} \wedge \text{IsCurrent = 'True'} \wedge \text{EffectiveDate} \neq \text{ActionTS}}(\text{UCTemp15}) \quad (21)$$

$$\text{DimAccount} \leftarrow \mathcal{U}(\text{DimAccount})_{\text{Sk\_CustomerID = Sk\_CustomerID |AccountID=AccountID} \wedge \text{IsCurrent = 'True'} \wedge \text{EffectiveDate} = \text{ActionTS}}(\text{UCTemp15}) \quad (22)$$

$$\text{UCTemp16} \leftarrow \sigma_{\text{Sk\_AccountID = max(Sk\_AccountID) + rownumber()}}(\text{DimAccount}) \quad (23)$$

$$\text{DimAccount} \leftarrow \text{DimAccount} \cup (\pi_{\text{AccountID, SK\_BrokerID, Status,...,EndDate}}(\sigma_{\text{EffectiveDate} \neq \text{ActionTS}}\text{UCTemp16})) \quad (24)$$

Fig. 5. RA expressions to model the historical load for an updated customer

specification directly into PDI; (b) Translating the BPMN4ETL specification into RA, and then implementing the RA operations using Postgres PLSQL.

To optimize the performance of the PDI implementation, the PDI performance tuning tips were applied[2]. The PDI memory limit was increased from 2G to 4G in order to avoid java out of memory exceptions and improve performance. Both tests were run over an Intel i7 computer, with a RAM of 16 GB, running the Windows 10 Enterprise operating system, using the PostgresSQL database as the DW storage. The total execution times of the processes, for different scales factors, are reported in Table 1. 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. 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.

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

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

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

These results are partially explained by the poor performance of PDI when it comes to implementing loops, needed for updates of the SCDs with dependencies. Except for the 'NEW' action type, which is loaded in batch form, all other action types are loaded using loops, one row at a time. PDI handles loops with the Copy Rows to Result step, that stores the rows in memory and retrieves them one row at a time. It takes PDI about 15 h out of the total running time for the historical load of scale factor 5, to finish running the DimCustomer and DimAccount dimension tables due to this loop. The same applies to the DimCompany and DimSecurity tables. Also, in spite of the tips applied to improve performance mentioned above, certain steps that cause slow execution could not be avoided. For example Merge Join Steps required input data to be sorted in advance in PDI. This slows the ETL flow since the complete results of the sorted input data had to be obtained prior the execution of the update process.

To conclude, the results reported here suggest that the alternative of implementing ETL SQL-based processes based on a translation from a RA specification is plausible and competitive.

7 Related Work

Several different strategies have been proposed to model ETL processes. The conceptual model proposed in [12] analyzes the structure and data of the data sources and their mapping to a target DW. The conceptual model proposed in [10], uses UML to design ETL processes, where each ETL process is represented by a stereotyped class. This model is refined in [4]. The work in [1,2] proposes a vendor-independent conceptual metamodel for designing ETL processes based on BPMN, which combines two perspectives, a control process view, and a data 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. Relational algebra was first used to model ETL processes in [7–9]. ETL processes for slowly changing dimensions specifications [7], data quality enforcement tasks [8], and ETL conciliation tasks [9] were modelled with relational algebra and applied to a real-world ETL scenario. Finally, relevant to

the work presented here, a research reported in [5] presents a framework for ETL development based on writing software code, instead of specifying the process using commercial tools like PDI, Integration Services, etc., and discusses the advantages of this approach.

8 Conclusion

This paper proposed RA as a language to specify ETL processes at the logical level. To illustrate the proposal, and show the plausibility of the approach in real-world scenarios, the TPC-DI benchmark was used in two ways: one, using RA to translate the BPMN4ETL specification of the benchmark into SQL. The other one implements the benchmark using BPMN4ETL, and translates this directly into the PDI tool. The experiments showed that the SQL implementation runs orders of magnitude faster than that of PDI. This work did not consider structural changes of the data sources, which will be addressed in future research.

Acknowledgments. Alejandro Vaisman was partially supported by PICT-2017 Project 1054 from the Argentinian Scientific Agency.

References

1. El Akkaoui, Z., Zimányi, E.: Defining ETL worfklows using BPMN and BPEL. In: Proceedings DOLAP, pp. 41–48. ACM (2009)
2. El Akkaoui, Z., Zimányi, E., Mazón, J.N., Trujillo, J.: A BPMN-based design and maintenance framework for ETL processes. Int. J. Data Warehouse. Min. (IJDWM) **9**(3), 46–72 (2013)
3. Kimball, R., Caserta, J.: The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. Wiley, New York (2011)
4. Muñoz, L., Mazón, J.-N., Pardillo, J., Trujillo, J.: Modelling ETL processes of data warehouses with UML activity diagrams. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2008. LNCS, vol. 5333, pp. 44–53. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88875-8_21
5. Pedersen, T.B.: Programmatic ETL. Business Intelligence and Big Data: 7th European Summer School, eBISS 2017, Bruxelles, Belgium, July 2–7, 2017, Tutorial Lectures 324, 21 (2018)
6. Poess, M., Rabl, T., Jacobsen, H.A., Caufield, B.: TPC-DI: the first industry benchmark for data integration. Proc. VLDB Endowment **7**(13), 1367–1378 (2014)
7. Santos, V., Belo, O.: Slowly changing dimensions specification a relational algebra approach. Int. J. Inf. Technol. **1**(3), 63–68 (2011)
8. Santos, V., Belo, O.: Modeling ETL data quality enforcement tasks using relational algebra operators. Procedia Technol. **9**, 442–450 (2013)
9. Santos, V., Belo, O.: Modelling ETL conciliation tasks using relational algebra operators. In: Proceedings of the 2014 European Modelling Symposium, pp. 275–280. IEEE, Pisa (2014)

10. Trujillo, J., Luján-Mora, S.: A UML based approach for modeling ETL processes in data warehouses. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, pp. 307–320. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39648-2_25
11. Vaisman, A., Zimányi, E.: Data Warehouse Systems. DSA. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54655-6
12. Vassiliadis, P., Simitsis, A., Skiadopoulos, S.: Conceptual modeling for ETL processes. In: Proceedings of DOLAP, pp. 14–21. ACM (2002)