Business Processes Meet Operational Business Intelligence

Umeshwar Dayal, Kevin Wilkinson, Alkis Simitsis, Malu Castellanos
HP Labs, Palo Alto, CA, USA

Abstract

As Business Intelligence architectures evolve from off-line strategic decision-making to on-line operational decision-making, the design of the backend Extract-Transform-Load (ETL) processes is becoming even more complex. We describe the challenges in ETL design and implementation, and the approach we are taking to meet these challenges. Our approach is centered around a layered methodology that starts with modeling the business processes of the enterprise, and their information requirements and service level objectives, and proceeds systematically through logical design to physical implementation. A key element of this approach is the explicit specification of a variety of quality objectives (we call these collectively the QoX objectives) at the business level, and the use of these objectives to drive the optimization of the design at the logical and physical levels.

1 Introduction

Today's Business Intelligence (BI) architecture typically consists of a data warehouse that consolidates data from several operational databases and serves a variety of querying, reporting, and analytic tools. The back-end of the architecture is a data integration pipeline for populating the data warehouse by extracting data from distributed and usually heterogeneous operational sources; cleansing, integrating, and transforming the data; and loading it into the data warehouse. The traditional data integration pipeline is a batch process, usually implemented by extract-transform-load (ETL) tools [1, 2]. Traditionally, BI systems are designed to support off-line, strategic "back-office" decision-making where information requirements are satisfied by periodic reporting and historical analysis queries. The operational business processes and analytic applications are kept separate: the former touch the OLTP databases; the latter run on the data warehouse; and ETL provides the mappings between them. We have learnt from discussions with consultants who specialize in BI projects that often 60-70% of the effort goes into ETL design and implementation. As enterprises become more automated, data-driven and real-time, the BI architecture must evolve to support *operational Business Intelligence*, that is, on-line, "front-office" decision-making integrated into the operational business processes of the enterprise [3]. This imposes even more challenging requirements on the integration pipeline. We describe some of these challenges and propose a new approach to ETL design to address them.

To motivate our approach, we use a simple, example workflow. Consider a hypothetical, on-line, retail enterprise and a business process for accepting a customer order, fulfilling and shipping the order and booking the revenue. Such an *Order-to-Revenue* process involves a number of steps, utilizing various operational (OLTP) databases and an enterprise data warehouse (Figure 1(a)). Assume a customer has been browsing the retailer web site and adding items to a shopping cart.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

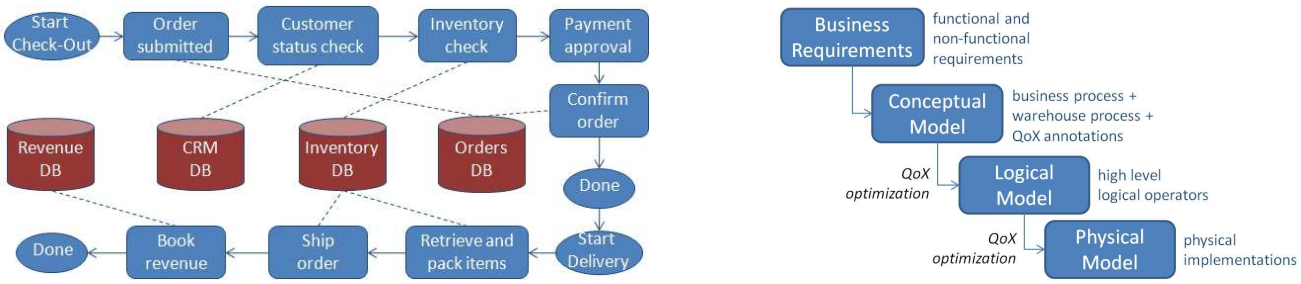


Figure 1: (a) Order-to-Revenue business process and (b) Layered methodology for ETL

Eventually, the customer is ready to make a purchase, which initiates the CheckOut process. This submits an entry to the order database and then the customer status is checked to validate the order. Next, the inventory database is checked to ensure the product is in stock. At this point, the order can be fulfilled so the customer payment is processed. Once confirmed, the Delivery process is initiated. The items are retrieved from inventory and packed. Finally, the order is shipped and the order revenue is added to the financial revenue database. In our example, revenue is not counted until the order is shipped.

Operational BI imposes new requirements on ETL that are difficult to meet using today's conventional approach. We describe three challenges.

End-to-end operational views of the enterprise. In the conventional BI architecture, the data warehouse provides an historical view of the enterprise; e.g., it can be used to provide reports on weekly sales, the top selling items, seasonal trends or to build customer segmentation models. The architecture may even incorporate an operational data store to provide a near-real time view of transactional data in the OLTP databases. Still, it does not provide the integrated, near real-time view of the entire (end-to-end) enterprise needed by the new breed of operational BI applications. For example, suppose we want to make special offers to particular customers based on their purchasing history, recent browsing actions, today's revenue, and current inventory. This operation requires data from the OLTP databases (current inventory, customer's recent browsing actions), the data warehouse (customer segment, purchasing history), and in-flight data (today's revenue, including orders that haven't yet shipped) that may be in staging areas on its way to the data warehouse. Such enterprise views are very complicated to design, implement, and maintain, and current BI tools provide little support for them.

Design by objective. The focus for ETL so far has been on correct functionality and adequate performance, i.e., the functional mappings from data sources to warehouse must be correct and their execution must complete within a certain time window. However, a focus on just functionality and performance misses other important business objectives (e.g., recoverability, maintainability, reliability) that, while harder to quantify, are needed for a successful ETL deployment. This is especially true for operational BI where there may be a wide range of competing objectives. Fraud detection may require a high degree of provenance for certain parts of the ETL flow. High reliability may be needed for parts of the flow related to revenue, e.g., the loss of click-stream data is acceptable whereas the loss of payment is not. Consequently, what is needed is a more general approach where the ETL design is driven by objectives and can be optimized considering their tradeoffs.

Design evolution. A typical ETL engagement consumes many months starting with business requirements and design objectives, infrastructure surveys, conceptual and logical design, and culminating in a physical design and implementation. In an ideal world, the requirements never change. In the real world and especially in operational BI, requirements change rapidly as the business evolves and grows. For example, assume the order-to-revenue process was implemented with an expected latency for the (external) payment approval process, but months later the credit agency doubles the latency. This affects the entire downstream ETL pipeline and perhaps substantial redesign to maintain service level objectives. A methodology that requires many additional months to adapt to such a change would not be useful in operational BI.

In [4], we describe a layered methodology that proceeds in successive, stepwise refinements from high-

level business requirements, through several levels of more concrete specifications, down to execution models (Figure 1(b)). At each level of design, different qualities (QoX objectives) are introduced or refined from higher levels [5]. This layered approach presents opportunities for QoX-driven optimization at each successive level. By connecting the designs and objectives at successive levels of refinement we are better able to track objectives and rapidly generate new designs as the business evolves.

An important feature of our design methodology is the use of business process models for the conceptual (high level) design. This has several advantages. It provides a unified formalism for modeling both production (operational) processes such as Order-to-Revenue as well as the processes that populate the data warehouse and intermediate enterprise states. It enables ETL design starting from a business view that hides the low-level implementation details and therefore facilitates the specification of SLAs (Service Level Agreements) and metrics by business analysts.

In brief, our approach leverages business process models to enable operational business intelligence. It captures end-to-end views of enterprise data and associates them with high-level design objectives, which are used to optimize the ETL processes and implementation. In the following sections, we elaborate on these ideas.

2 QoX-driven integrated business views

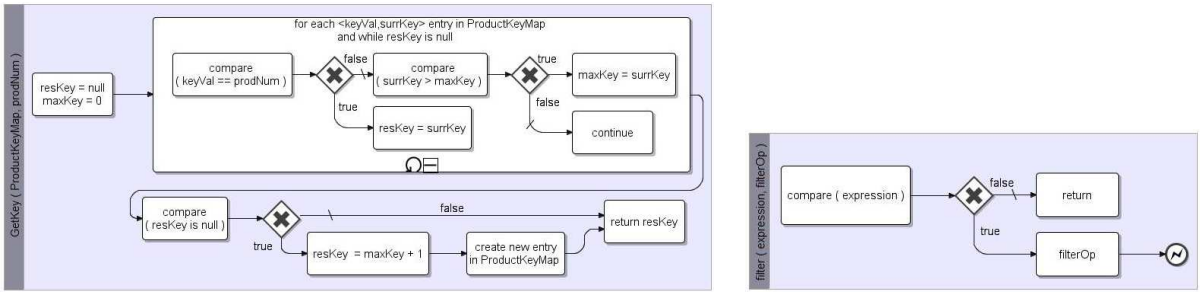
This section discusses our approach to obtain an integrated, end-to-end view of the enterprise at the conceptual level and, from that, how to get a logical ETL design. The facts in a data warehouse define the business objects and events of interest for decision-making. The data sources for these facts are objects in the OLTP databases manipulated by operational business processes, e.g., CheckOut, Delivery. ETL flows define the mappings between the source objects and the facts in the warehouse. However, ETL tools today do not support the modeling of these mappings at the conceptual level (i.e., in terms of business objects). Rather, they support only logical ETL design at the level of objects such as tables, indexes, files, communication links. We believe that modeling ETL at a conceptual level can benefit operational BI in a number of ways. First, it enables users of the warehouse to see the provenance of the warehouse data in business terms they understand. Second, it provides an up-to-date view of the enterprise by exposing the intermediate state of the ETL pipeline, the data under transformation before it is loaded to the warehouse. This intermediate view creates new opportunities for real-time operational applications in that it can be used at any time for operational decision-making, avoiding a wait for the warehouse to be refreshed. Third, this conceptual model can be used to derive the logical model for ETL.

Our approach is to use BPMN[1] for the conceptual model. Since BPMN can also be used to model operational business processes, this provides a common formalism to model the complete information supply chain for the enterprise. For each fact (and dimension, view, etc.) object in the warehouse, there is a corresponding BPMN *business fact process* that shows how that object is created out of the operational business processes. Probes inserted in the operational business process are used to send messages to all fact processes that need data from that point in the process. But, there are three challenges with using BPMN.

The first challenge is that to derive the logical ETL flow, we need a way to map fragments of a BPMN fact process to the corresponding logical ETL operators. To do this, we employ three techniques: (1) an expression language to specify method invocation in the nodes of a fact process; expressions can be easily mapped to logical ETL operators; (2) macro expansion; e.g., Figure 2(a) illustrates an example for the frequent ETL operation of surrogate key generation; and (3) templates for mapping specific patterns to ETL operations; e.g., a compare method followed by a true/false branch where one branch terminates the flow is recognized as an ETL filter operator (see Figure 2(b)). These examples for macro expansion and templates are discussed later. Note that a legend for BPMN notation is provided in Figure 3.

The second challenge is that BPMN models process flow, but ETL is fundamentally a data flow. So, we need to augment our BPMN diagrams to convey the necessary data flow information, in particular, the input,

[1] Business Process Modeling Notation, http://www.bpmn.org/



(a) (b)

Figure 2: Example (a) macro expansion for a SK operator and (b) template for filters

output, and parameters of ETL operators. To do this, we augment our BPMN fact process with input, output and parameter schemas as follows: we assume that each BPMN message is described by an XML schema; when a message is received by a method, it is included as part of the node's input schema. We use the BPMN annotation capability to annotate methods with XML schemas for other inputs, outputs and any other parameters. In going from process flows to data flows, we also have to consider that a business process typically creates a new process instance per business event or transaction, whereas an ETL process typically uses a single data flow to process a batch of records. Consequently, to create a batch of objects, we introduce a spool method that inputs a sequence of objects and outputs a set of objects.

The third challenge derives from our ultimate goal of producing an optimized physical ETL implementation. For this purpose, we must also incorporate the QoX objectives into the BPMN fact processes and the derived logical ETL flows.

Example. We illustrate our ideas by presenting the BPMN diagram[2] for a *DailyRevenue* fact process (see Figure 3). This process computes the revenue for each product sold per day. We assume that the business requirements include a QoX measure for freshness. This specifies how frequently the warehouse should be updated (e.g., daily for high freshness, monthly for low freshness, etc.). Thus, the DailyRevenue process is initiated once per freshness interval. Recall that revenue is counted when a product is shipped. Thus, a probe must be added to the Delivery business process to send the order details to the DailyRevenue process. The spool method separates order details into an order summary and its constituent lineitems and accumulates this data for the freshness period. Afterward, the set of spooled lineitems is forwarded to the partitioning method which groups lineitems by date and product number. For each group, it creates an instance of the DailyRevenue process and sends the process its lineitems.

The DailyRevenue process does the work of creating one new fact. It iterates through the lineitems, aggregating their details. The GetKey method converts production keys to surrogate keys. Note that GetKey method is a macro expansion and the corresponding BPMN diagram is shown in Figure 2(a). Internal orders, denoted by a null shipping address, should not be counted as revenue so they are filtered out. The template that is used to recognize this pattern as a filter operation is shown in Figure 2(b). When all lineitems in the group are processed, the totals are added to the warehouse as a new fact.

Given the DailyRevenue process description along with annotations for the data flow, the logical ETL flow, DailyRevenue_ETL, can be generated using a relatively straightforward translation. The details are omitted in this paper. The logical ETL flow is depicted in Figure 4(a). Here we use the notation of an open source ETL tool (i.e., Pentaho's Kettle). Designing a tool-agnostic, logical ETL flow language is itself an interesting challenge.

Operational BI example. As discussed, the BPMN fact process enables new opportunities for real-time decision-making without waiting for warehouse refresh. As an example, suppose the on-line retailer wants to include special offers in the shipping package such as product rebates, free shipping or discounts on new

[2]Our BPMN diagrams are intended for presentation and are not necessarily entirely consistent with the specifications.

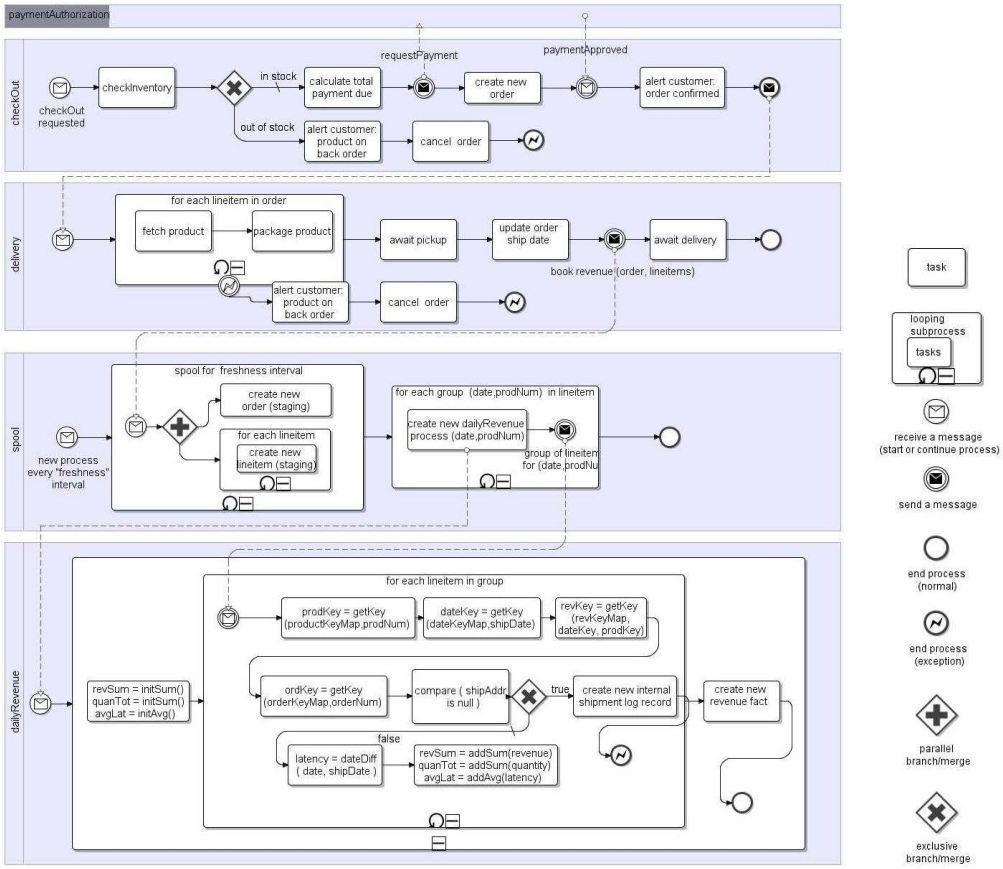


Figure 3: Example operational business processes (CheckOut, Delivery) and business fact process (DailyRevenue, spool)

products. And suppose these offers depend on today's current daily revenue. The current day's revenue is not available in the warehouse so the special offers process must access the intermediate state of the enterprise.

To accomplish this, we need to link the RetrieveAndPack method in the Delivery process to a new process, ShippingInserts (Figure 4(b)). This new process returns a set of offers to include in the shipping package according to the business rules. In our example, we assume rebates are offered for orders that had an exceptional delay and free shipping is provided for orders that exceed twice the average order amount. We need to adjust the freshness interval to ensure that the DailyRevenue is updated hourly (or possibly more frequently) so that the running totals can be tracked. Note this requires a slight modification of the DailyRevenue fact process (not shown) to maintain a running total, i.e., it should process multiple groups from the spooler and only update the warehouse once in each refresh cycle.

3 QoX-driven optimization

After having captured the business requirements and produced an appropriate logical ETL design, the next step involves the optimization of the ETL design based on the QoX metrics. The challenges in doing this include the definition of cost models for evaluating the QoX metrics, definition of the design space, and algorithms for searching the design space to produce the optimal design. In [5], we showed how tradeoffs among the QoX objectives can lead to very different designs. Here, we summarize some of the optimization techniques and tradeoffs.

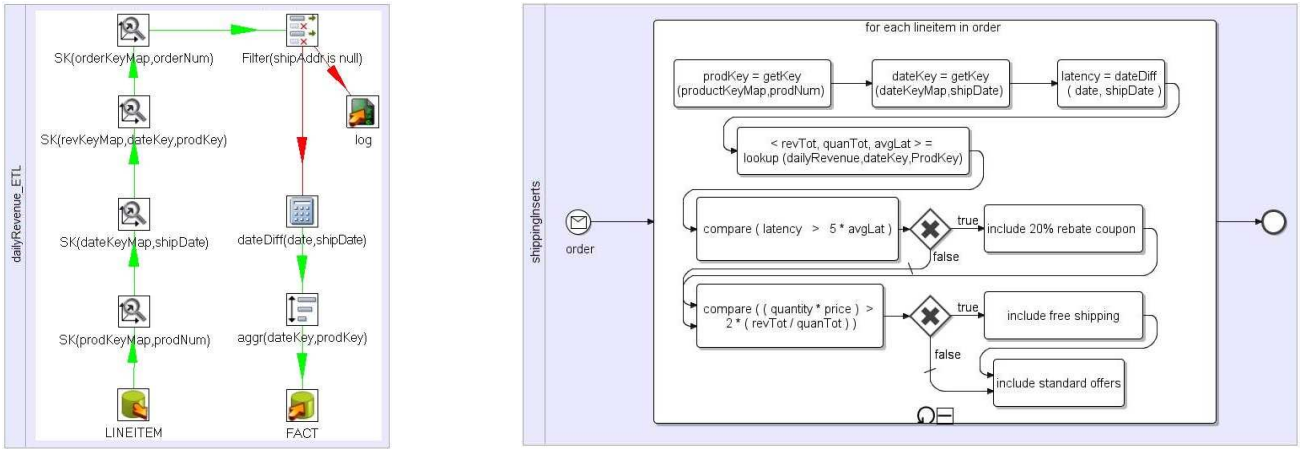


Figure 4: (a) Logical ETL flow and (b) Real-time offers

Optimizing for *performance* (i.e., improving the execution time of an ETL flow) typically exploits algebraic rewriting (e.g., postponing the getKey method for revKey until after the aggregation to decrease the amount of data) or flow restructuring (e.g., partitioning the flow for parallelization). The optimizer must select among many choices for rewriting and restructuring the flow (e.g., how and where to partition). An ETL workflow may fail due to operational or system errors. Designing for *recoverability* typically involves the addition of recovery points at several places in the workflow from which the ETL process resumes after a failure and continues its operation. However, I/O costs are incurred for maintaining recovery points, and hence there are tradeoffs between recoverability and performance. The optimizer must decide on the number and placement of recovery points. Sometimes, we cannot afford to use recovery points, as for example when high freshness is required. In such cases, it might be best to design for *fault-tolerance* through the use of redundancy (i.e., replication, failover, diversity). There are many challenges such as determining which parts of the workflow to replicate and achieving a balance between the use of recovery points and redundancy. *Freshness* is a critical requirement for operational BI, and designing for freshness is an important area of research [6, 7]. Alternative techniques here include the use of partitioned parallelism, the avoidance of blocking operations and recovery points, streaming implementations of transformation operators such as joins (e.g., [8]) or the loading phase (e.g., [9]). Also, scheduling of the ETL flows and execution order of transformations becomes crucial [10].

Optimizing for each of the QoX metrics is a challenge by itself because of the large design space. However, the main challenge is to consider these implementation alternatives together in order to optimize against a combination of QoX objectives specified by the business requirements. Figure 5 illustrates some of the tradeoffs in optimizing for freshness, performance, recoverability, and fault-tolerance for a specific flow [5]. The solid blue line represents the baseline performance of the original flow. For improving freshness (i.e., reducing the latency of an update at the target site - y axis), we need to increase the number of loads (x axis). In doing so, the best performance (i.e., lowest latency) may be achieved with parallelization (black dotted line). Using recovery points hurts freshness more or less depending on whether we use a high (green line with larger dashes) or a low (red line with smaller dashes) number of recovery points, respectively. The alternative of using triple modular redundancy (red line with larger dashes) for fault-tolerance achieves nearly the same level of freshness as the original design.

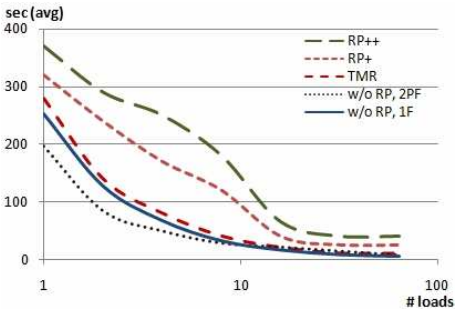


Figure 5: Example design space for QoX metrics [5]

4 Summary

We described a layered methodology for designing ETL processes in operational Business Intelligence systems. A key feature of this methodology is the use of a unified formalism for modeling the operational business processes of the enterprise as well as the processes for generating the end-to-end information views (e.g., business facts) required by operational decision-making. The methodology starts with a conceptual specification from which the logical definition and physical implementation are systematically derived. Included in the conceptual model is the specification of QoX objectives, which drive the design and optimization at the logical and physical levels.

Our ongoing research addresses the following problems: (1) Conceptual modeling formalism: We have illustrated our approach using BPMN. However, as we discussed, BPMN is not especially well suited to expressing the data flow mappings for constructing information views. Also, the modeling formalism must support annotation of the process and data flows with quality objectives. (2) Logical modeling formalism: This must include the typical operators required by the mappings, but must be agnostic to any specific implementation engine, and it must enable QoX-driven optimization. (3) Automatic derivation of the logical model from the conceptual model. (4) QoX-driven optimization: This includes a cost model for expressing the QoX metrics, and algorithms for optimizing against these metrics. (5) Techniques for validating the design against the business level specifications. (6) Techniques for evolving the design as business level requirements change.

References

[1] W. Inmon, *Building the Data Warehouse*. John Wiley, 1993.

[2] R. Kimball and J. Caserta, *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. John Wiley, 2004.

[3] C. White, "The Next Generation of Business Intelligence: Operational BI," DM Review Magazine, May 2005.

[4] U. Dayal, M. Castellanos, A. Simitsis, and K. Wilkinson, "Data Integration Flows for Business Intelligence," in *EDBT*, 2009, pp. 1–11.

[5] A. Simitsis, K. Wilkinson, M. Castellanos, and U. Dayal, "QoX-driven ETL Design: Reducing the Cost of ETL Consulting Engagements," in *SIGMOD Conference*, 2009, pp. 953–960.

[6] D. Agrawal, "The Reality of Real-time Business Intelligence," in *BIRTE (Informal Proceedings)*, 2008.

[7] P. Vassiliadis and A. Simitsis, *New Trends in Data Warehousing and Data Analysis*. Springer, 2008, ch. Near Real Time ETL, pp. 1–31.

[8] N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simitsis, and N.-E. Frantzell, "Supporting Streaming Updates in an Active Data Warehouse," in *ICDE*, 2007, pp. 476–485.

[9] C. Thomsen, T. B. Pedersen, and W. Lehner, "RiTE: Providing On-Demand Data for Right-Time Data Warehousing," in *ICDE*, 2008, pp. 456–465.

[10] L. Golab, T. Johnson, and V. Shkapenyuk, "Scheduling Updates in a Real-Time Stream Warehouse," in *ICDE*, 2009, pp. 1207–1210.