

Database Access Control for E-Business – A case study

Santosh Dwivedi

Bernard Menezes

Ashish Singh

Kanwal Rekhi School of Information Technology

Indian Institute of Technology Bombay

Powai, Mumbai, IN

{santosh, bernard, ashishc}@it.iitb.ac.in

ABSTRACT

In a multi-tiered E-Business application, database access control can be implemented in the application tier or in the database tier or in both. We describe the advantages of Application Service Provider-based Supply Chain Management and present a much simplified database schema for this application. Common user queries which provide input to decision making systems relevant to supply chain optimization are listed. Security requirements in the form of database access control rules are then spelled out. We investigate the use and shortcomings of Oracle's Virtual Private Database as a component of a database tier firewall.

1. INTRODUCTION

Security is critical in many non-trivial web applications such as internet banking or on-line commerce. Features of communication security such as authentication, message integrity, privacy and non-repudiation have been extensively researched [6], [7]. In addition to secure data communications, an important related issue is authorization. Having been authenticated, what permissions to on-site resources does a principal have and how are these determined? These permissions could be as varied as access rights to storage or "execute" rights to methods of individual classes. The focus of this paper is on the specification and enforcement of access rights to a shared database.

Many enterprise applications are structured as a sequence of tiers so as to logically separate and isolate distinct concerns. For example, business rules and logic are abstracted out in the application tier while database access, caching, indexing and query optimization are confined to the database or storage layer. There is also need for a web tier in keeping with the principle of separating the model (domain objects) and the view. End-to-end security, so crucial in web-based enterprise applications, can only be provided by securing each individual tier.

Security at the web tier involves authenticating a principal through the verification of its credentials. Passwords, digital certificates and biometric techniques possibly in conjunction with smart cards may be used here. These credentials can then be maintained in a "session context", [7] for the duration of a login session. The idea of a context exists in the web and application

tiers [5], [13] as well as in database tier [24] implementations. In the database tier, they serve as input to access control policy functions which in turn limit the data that is accessible to the principal.

This paper is principally about providing for secure access to a shared database in a web-based application with multiple business partners. We use as a case study centralized Supply Chain Management (SCM). Here, the inventory levels, orders, shipment details, etc. of all business partners are stored at a site owned and operated by an Application Service Provider (ASP). Because information sharing in this application has long been proposed to reduce costs, such a set-up makes sense. However, trust relationships between entities in the supply chain must be respected motivating the current work.

The application software for ASP-based SCM can be provided by the ASP or it can permit the use of custom software implemented by the different supply chain partners. In the latter case especially, database access control should be implemented in the database tier itself. For this purpose, we highlight the main functions of a database tier firewall. We study the use of Oracle's Virtual Private Database (VPD) as a component of the firewall. This involves writing policy file functions which when executed modify an application query by appending to it a where clause. The specific predicates appended are a function of the access constraints specified by the domain expert and captured in policy functions. We consider a core set of queries useful in providing input to a decision support system which in turn is used for supply chain optimization.

The organization of this paper is as follows. In Section 2, we introduce the important E-Business application of ASP (Application Service Provider)-based SCM. Section 3 discusses related work. We present our SCM database schema design and Virtual Private Database (VPD) basics in Section 4. In Section 5, we present a number of relevant user queries and explain the rationale for them. We state specific constraints on database access. We then present the modified queries upon application of the appropriate policy file functions. Section 6 contains a discussion of relevant issues and a conclusion.

2. ASP-BASED SCM

B2B partners usually form a chain for providing a service or manufacturing a product. The partners along the chain are either inventory buffer points or value adding points - they push the product nearer to completion/sale or customize it as per the requirements of the customers. This chain is referred to as a supply chain or supply web [21]. The network of suppliers and consumers is modeled as a graph with nodes representing entities such as suppliers, manufacturers, wholesalers, retailers, etc. (Fig. 1(a)). A supplier-consumer relationship is represented as an arc from the supplier to the consumer. The graph is a multi-stage or multi-partite graph with each stage comprising entities of a given type. Goods or products flow from a node to one connected to it

and downstream from it while orders and payments flow in the reverse direction. We assume that the buyer-seller relationships are static— a buyer always buys a product from the same seller though he may buy another product from a different seller. In Fig. 1(a), the arc labels represent the set of products or items supplied from the upstream to the downstream node.

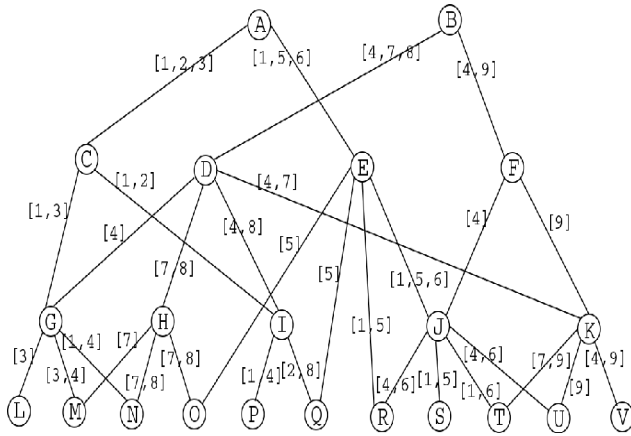


Figure 1(a). A Supply Web

A node X is an indirect supplier of Y for product P if there is a sequence of nodes N1, N2, ... Nm such that X is a direct supplier to N1, N1 is a direct supplier to N2, ... Nm is a direct supplier to Y for product P. Even though there may be multiple disjoint paths from a node to a given downstream node in the complete supply graph, the subgraph linking suppliers and customers for a specific product, P, is a directed tree. This follows from our assumption that, for a given customer, a product will always be sourced from the same supplier. We refer to this directed tree as a supply tree. Fig. 1(b) shows a supply tree rooted at Manufacturer B for Product 4.

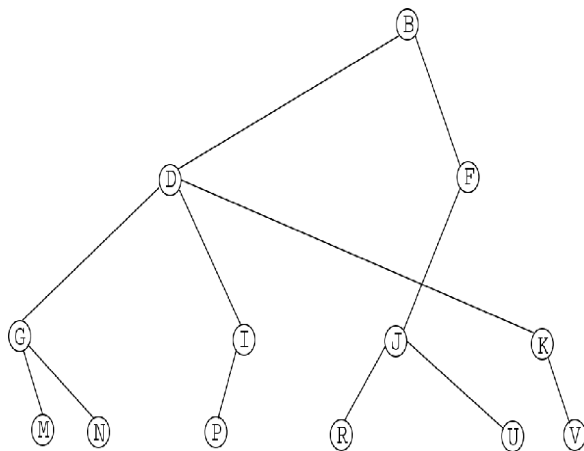


Figure 1(b). A Supply Tree for Product 4 Rooted at B

Our approach to providing support to the business processes in SCM exploits the pervasive internet by hosting the SCM software on a single remote site provided by an Application Service Provider (ASP) [22], [23]. This is often employed when the client is a small or medium enterprise which lacks the resources or manpower to support such an application. ASPs

remove technology risks (including obsolescence) out of investment decisions and insulate users from the pain of licensing, installation, upgrades, troubleshooting and other perfunctory tasks such as performing back-ups, recovery and maintenance.

Intelligent software on the ASP can provide a host of other services for partners in the supply chain such as:

- Demand/Sale forecasts at any node in the chain by analyzing long and short-term trends, seasonality, cyclicity, etc.
- Triggers and alerts which inform entities of sudden surges or slumps in sales of different items
- Computation of re-order points and quantities based on updated POS (point of sale) and lead time information [9]
- Payment gateway functionality to effect funds transfer from buyer to seller account by having direct connection to the banking network
- Facilities to mine information on sales to identify, for example, co-relations between sales of related items, the effect of promotions/discounts on sales volume/profits, etc.

3. RELATED WORK

The ASP-based approach has received much attention in the literature as has the outsourcing of data management which is considered an attractive option for many organizations [12][3]. It relieves the clients from the responsibility of building in-house data management infrastructure [18]. Partitioning data among two or more service providers in order to preserve privacy so that exposure of the contents of any one partition does not result in privacy violation has been discussed in [3]. Oracle's Virtual Private Database (VPD) uses query rewriting – a predicate is appended to an application's SQL query [24]. An alternative is to use authorization views [19].

Many proposals have evolved with encryption at the core - client side encryption is performed and then data is stored remotely [10][11][2][12]. Moreover some security issues in encrypted databases have also been mentioned[14]. Hippocratic databases have been proposed recently [1]. These advocate a set of ten principles that a database server must follow in order to advertise its "hippocracy". The rules that hippocratic databases must follow are designed as per the various privacy bills in the U.S. and elsewhere.

The specification of policies plays a significant role in any access control security mechanism. W3C has developed a specification for the Platform of Privacy Preferences (P3P)[17], which encapsulates a standard format of encoding privacy policies. However, it has been judged to be too complex to be used efficiently by some [15] and too limiting by others [20].

The above approaches used for database privacy such as replication and encryption have substantial associated costs. The cost may be justified especially when data confidentiality from database administrators at the ASP site is an important concern. Here, we consider the less general problem of specifying and enforcing database access restrictions among all valid subscribers (in our case the supply chain partners).

4. DATABASE SCHEMA DESIGN AND ACCESS CONTROL

In this section we first present a highly simplified version of the database schema for centralized SCM. We investigate when access control should be enforced in the database tier and propose the database tier firewall. Finally, we study the use of Oracle's VPD as a component of the firewall.

4.1 Software Development Models and Schema Design

There are at least two software development models in the context of ASP-based SCM.

Model 1: All application software deployed on behalf of the different business partners is implemented by the ASP.

Model 2: Each client designs and implements customized application software with its own specific needs and features in mind.

In the second model, application code should not be trusted to provide access control. Application tier programmers could be provided with a standard set of APIs they can use for data access. This is, of course, in addition to the ever popular SQL queries. All application programmers see the same E-R or UML class diagram - they use the same name to refer to a table of purchase orders. In reality, they see different views of the database but these are *unnamed* views. In fact, the user is not explicitly aware of them. The application program on behalf of client X need not be aware that the orders placed by one of its grandchildren, Z, is in a table called *orders_Z* or in a view named *orders_X.Y.Z*. Internally, the Orders table may actually be split (for performance or security reasons) into *Orders_A*, *Orders_B*, etc. where *Orders_A* contains all orders placed by A. However, application programs need not be burdened with such details. Thus, not only is the programmers' abstraction of the data stored the same but the vocabulary of relational schema instances and attributes is identical.

A simplified database schema for centralized SCM is shown in the UML class diagram of Fig. 2. Note that in this *abstract schema*, orders placed by all nodes are stored in the same table. Each order is made up of possibly multiple items - information on each of these is contained in the *Order_Lines* table.

In keeping with recent trends in SCM such as Vendor Managed Inventory (VMI) [21], decisions on order quantity and delivery schedule are often made collaboratively by both supplier and customer. The digital signature fields in an *Orders* object are used to satisfy legal requirements pertaining to approval by both sides of the terms of the order including pricing. Even after an order has been signed by both parties, it may be modified by one party provided that the other party consents to the modification within a specified time limit. Modifications to an order are saved in tables *Order_Modification_Log* and *Order_Modification_Details*.

There is also a single table for shipments - regardless of where a shipment originates and to which node or nodes it is destined, a shipment is stored in the same *Shipments* table. Fulfillment of a single order may be made in different shipments. Also, a single shipment may carry goods connected with more than one order and to possibly multiple destinations. Thus the relationship between Order and Shipment is many-to-many.

In this paper, we assume static chains, i.e., a customer will always source a specific item from the same supplier - this information is stored in the *Business_Topology* schema. Finally, *Point_Of_Sale* contains sales volumes of each product stocked by a retail outlet over different time intervals.

4.2 Access Control Issues

There are a number of basic issues to be addressed in connection with database access control:

1. Who specifies the access control rules, how are they expressed and how and where are they enforced?

2. What is the burden placed on the application developer and the system administrator in supporting access control and can this be reduced?
3. What are the performance implications of any strategy or mechanism we devise for access control?

The access control rules for this application are distilled from experts in the area of collaborative supply chains. Examples of such constraints are presented in Section 5. These constraints are expressed in a high-level, human-readable form maintained by the system. They can be periodically reviewed and modified to reflect changing subscriber requirements. They can be enforced in the application tier, the database tier or both tiers could share some responsibility for access control.

In the first model of software development, the responsibility of enforcing access control can be placed in the application or database tier. For example, Sun's J2EE specification uses the container architecture to support security. The application designer can grant authenticated clients access rights to individual methods of different beans. These access rights can be specified in a mapping between role and permissions in the deployment descriptor. Each call to a bean method is intercepted by the container and the rights of the principal calling the method are checked from the deployment descriptor. In addition, instance-based access control is supported programmatically within the bean i.e. the bean makes a determination whether another bean method may be called. This is often done by invoking methods in the application context to obtain the role of the principal involved.

Use of the above methods of database access control in the application layer in Model 2 are fraught with danger since each client is allowed to deploy its own code on the ASP. It is especially critical in this case that database access control mechanisms be deployed outside the reach of the application code and possibly in the database tier itself.

4.3 The Database Tier Firewall

The database tier firewall, as envisaged by us, is a security mechanism that validates and then transforms an application generated SQL query to produce a query understood by the database and in keeping with the access restrictions implied by the security policy. For this purpose it uses abstract-to-logical schema mappings, internally created view definitions, security policy files and information from the session context. To facilitate ease of programming, it offers the programmer a set of APIs that complement the provision for expressing queries in SQL.

The principal building blocks of the database tier firewall (Fig. 3) include:

Query Validator - It checks whether the input query submitted by the application is valid with respect to the abstract schema i.e. whether the table objects and attributes referred in the query are all contained in the abstract schema visible to the application developer. It also contains a parser which checks whether the given query conforms to standard SQL syntax.

Query Generator - The application program has the liberty of intimating its queries to the database in the form of API calls as published by the database firewall. Whenever the firewall receives such a request, it transforms the request into an appropriate query statement involving tables in the abstract schema and delivers the query to the query transformer.

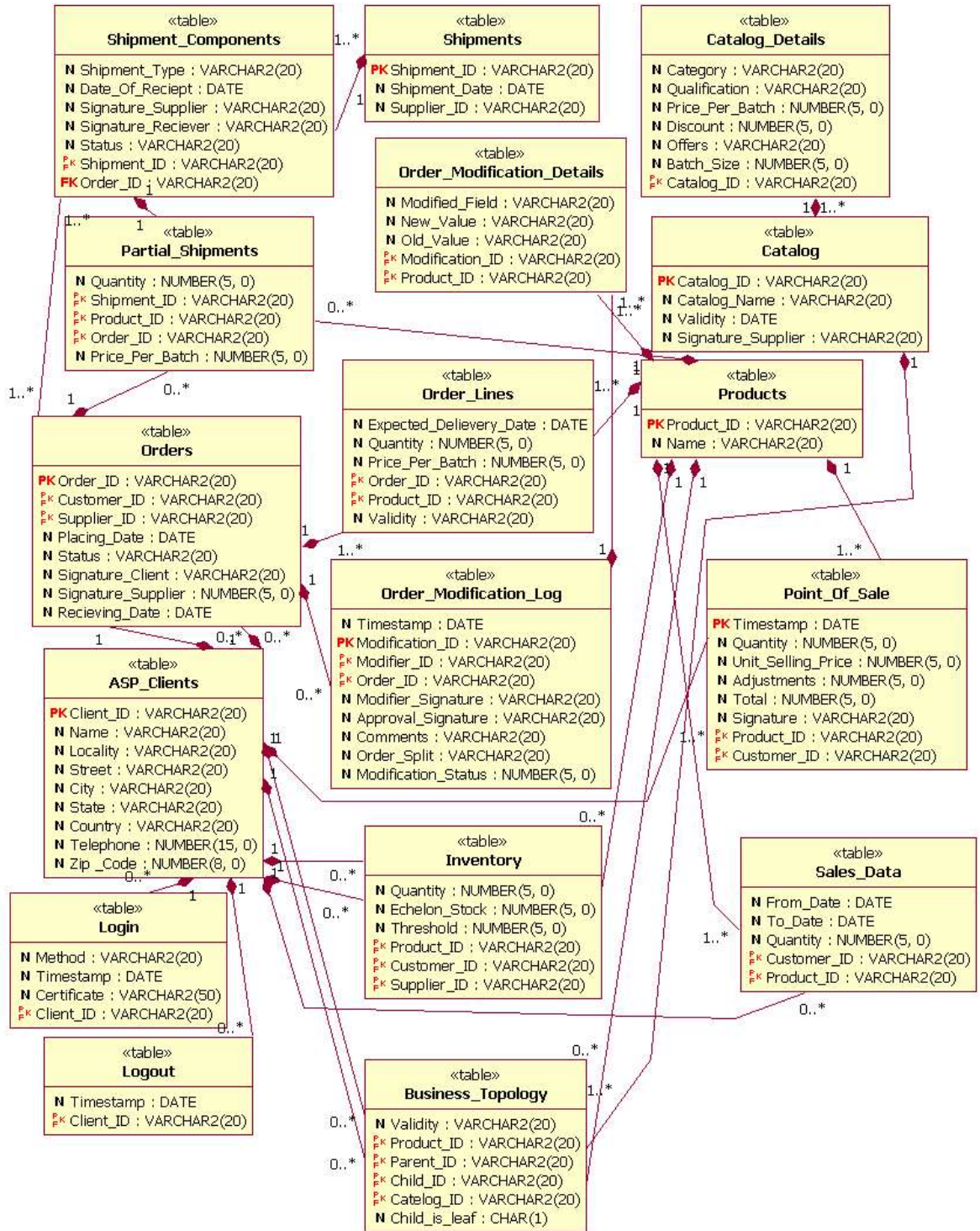


Figure 2. Database Schema for ASP Based SCM

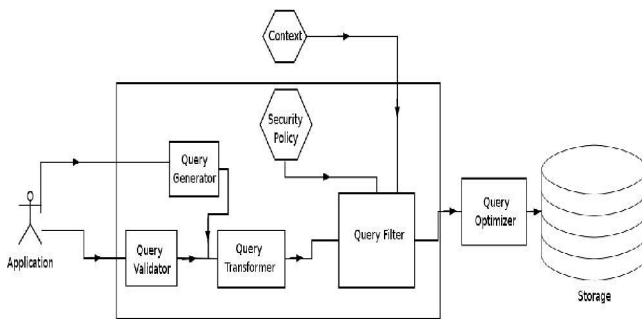


Figure 3. The Database Tier Firewall

Query Transformer – This block transforms a query on the abstract-schema into an equivalent query containing internally defined view names or tables in the logical-schema. For example, abstract schema may specify a single table for all orders - thus an application will reference this table in a query involving orders. The logical schema might have as many order tables as there are suppliers - *OrdersX* containing all orders placed to the same supplier, X. The query transformer would then substitute one or more order tables in the logical schema for the single orders table in the abstract schema.

Query Filter – This is the most security-aware component in the firewall, performing operations such as appending a predicate to a query and projecting out only the columns that the current user is authorized to see. For this purpose, it consults the security policies administered by the security manager. In addition, it may also consult attributes in the session or application context. In effect, this block does both row-level and column-level filtering.

We have conceptualized the database tier firewall which is one of the critical components in a prototype of ASP-based SC – our longer term goal. Rather than reinvent the wheel, we decided to examine the use of existing products in building the Database Tier Firewall. One such product is Oracle’s Virtual Private Database (VPD).

4.4 Role of the Virtual Private database (VPD)

One or more security policies may be attached to each table and view in the database – each policy is specified in the form of a set of functions coded in PL/SQL, C or JAVA. A user directly or indirectly accessing a table or view having a security policy associated with it causes the server to dynamically modify the query statement by appending a predicate at the end, transparently to the user. This predicate is returned by the policy function. The policy function is highly flexible and it can return different predicates depending on the values of attributes contained in the application context. Some attributes are captured from the login session of the user with help of a login trigger (such as login credentials) while others are given a default value at login time and the responsibility of changing them with a particular submitted query lies with the application. An example of one such attribute is discussed later in this section.

The mapping of policy to a database table for a specific operation (select, insert, delete, update) is maintained by the RLS (Row Level Security) package in the DBMS_RLS table. More than one policy could be maintained on each table - this

strengthens the access constraints on that table since the final predicate is the logical AND of all individual predicates returned by individual security policies. A security policy can be enabled or disabled by setting a boolean variable in the DBMS_RLS table. Whenever a query is submitted for execution, DBMS_RLS is consulted for any security policies attached to the table(s) which is/are mentioned in the query. If any policies are found attached to those tables, then all such policies are executed in order to get a predicate. This predicate is appended to the end of the query and then the query resumes its execution in the normal fashion.

An example of a policy function is shown in the Appendix.

Consider a query from user U asking for all orders it has placed. The application program, acting on behalf of U would present the following SQL query to the database.

```
Select * from orders where customer_id =U;
```

An untrustworthy application component, to obtain unauthorized order information about its competitor, C, may substitute C for U. Its attempt would be thwarted by the VPD if the security policy is programmed to append the predicate

```
customer_id = SYS_CONTEXT('APPCTX','USER')
```

for any query involving access to the Orders relation. The value of attribute named “USER” in the application context (which in the present case is U) is used in the predicate for the above query. The conjunction of the predicate supplied by the application and that supplied by the policy procedure will evaluate to false and hence no tuples in the Orders relation will be returned.

Thus U should simply submit the query *Select * from orders*. However, there are two interpretations of this query – “Does U wish to see all orders that it has placed or the orders that it has received?” This ambiguity can be resolved by introducing an attribute in the application context which we call ROLE. This can be set by the application and read and interpreted by the security policy. Before submitting a query to read all orders U has placed, it sets ROLE to “customer”. When U wishes to see all orders placed from it, it sets ROLE to “supplier”. On receiving the query from U, the security policy inspects the ROLE attribute and respectively appends one of the predicates,

```
Select * from orders where customer_id = SYS_CONTEXT('APPCTX','USER');
```

or

```
Select * from orders where supplier_id = SYS_CONTEXT('APPCTX','USER');
```

Thus the process of query modification is entirely transparent to the user. The user’s query is relatively simple with the onus of ensuring secure access resting squarely with the author of the security policy.

5. ACCESS CONSTRAINTS IN CENTRALIZED SCM

In order to make the supply chain more efficient and responsive, pertinent and up-to-date information should be provided to authorized parties which in turn is used as input to their decision making systems. In this section, we specify the database access constraints and present the predicates generated by our policy files for a core set of queries used to provide such information.

5.1 Common Queries on the SCM Database

In the most restrictive scenario, the only information shared between two parties would be orders placed by a customer to its supplier. However, numerous studies [8] have shown that it is precisely the lack of information sharing that is responsible for the Bullwhip effect and the concomitant increase in chain-wide costs. Essentially, the Bullwhip effect [16] refers to the phenomenon where orders to the suppliers tend to have larger variance than sales to the end customer and this distortion propagates upstream in an amplified form. The distortion of demand information implies that the manufacturer who only observes its immediate order data will be misled by the amplified demand patterns and this has serious cost implications. Sell-through data and information on inventory status is the key to improving channel coordination and dampening the bullwhip effect [21].

The response to the following query will provide information that can be used to mitigate the Bullwhip effect.

Query 1: A node wishes to see the graph of the total sales made by all of its leaf descendants for item x between dates $d1$ and $d2$.

Other queries of relevance in improving chain efficiencies are:

Query 2: The root node (manufacturer) wishes to know the sales volume of item x at a particular retail outlet, R , between dates $d1$ and $d2$.

Query 3: The root node (manufacturer) wishes to know the quantity of item x ordered by a particular wholesaler, W , between dates $d1$ and $d2$.

Queries 2 and 3 above are especially relevant for products that have a small cycle time and become obsolete soon. In that case, the manufacturer would prefer to study sales of item x in specific stores where, by experience, shopping trends are first detected. The sales reported by Retail store R of item x may be a harbinger of sales to come elsewhere. Also, wholesaler W may be shrewd in spotting trends and the quantity ordered by him may be indicative of near-term sales of item x . A surge or slump in sales may aid the manufacturer in making tactical decisions regarding whether to ramp up production or to scale back.

An important operation in supply chains is the management of inventory. An effective way of managing inventory at a non-leaf node, N , is to consider the *echelon stock* – the inventory at N plus the total of in-stock and in-transit inventory at all descendants of N [8]. The ordering decision at N is based on its *echelon inventory position* which is its echelon stock plus the items ordered by N that have not yet arrived. To implement such an ordering policy, node N would make the following query.

Query 4: Node N wishes to know its echelon stock for product x .

5.2 Access Constraints on the Supply Chain Database

Answering the above queries is straightforward given the schema in Fig. 2. However, users may have several restrictions concerning access to their data. Some of these are enumerated below.

- A node requires that its sales and inventory levels of item x be visible to only its direct or indirect suppliers of item x .
- A node requires that the orders placed by it of item x be visible to only its direct or indirect suppliers of item x . However, the price information should be visible only to its direct supplier of x .
- Certain fields are write-once only.

With the first two constraints in place, a node would be authorized to receive inventory, sales or order information only about a node to which it is a direct or indirect supplier for that item. The third constraint prevents a node from unilaterally changing the terms of an order after both sides have signed it. We next present the predicates generated by our policy functions in connection with Queries 1-4 and subject to the constraints enumerated above.

5.3 Policy Files and Access Control

5.3.1 Aggregate Sales of all retail outlets in the Supply Tree rooted at N (for item x)

Query 1 in Section 5.1 needs access to only the Sales_Data table. The latter maintains the aggregate sales of all leaf descendants in the supply tree of each node N for item x . This value is updated by application software on receiving fresh point-of-sale information for x from any retail outlet in its supply tree. Alternatively, a database trigger can be written to update Sales_data for all nodes that are direct or indirect suppliers of the retail outlet for item, x . Execution of the security policy in this case would simply result in appending the predicate

where Customer_ID=N

to the original query from the application

Select quantity, from_date, to_date from sales_data where product_id = x and d1 <= From_Date and d2 <= To_Date;

5.3.2 Sales/Order quantities at specific points in a Supply Tree

In Query 2, a node N requires the sales of item, x at only a specific retailer, R . This information is contained in the Point_Of_Sale table and should be accessible by N only if R is in the supply tree rooted at N for x . The security policy must generate the following predicate:

customer_id in (select customer_id from business_topology where child_is_leaf = 'yes' start with parent_id = SYS_CONTEXT('APPCTX', 'USER') connect by prior child_id = parent_id and product_id = x)

Query 3, fired by the manufacturer of item x , requires order information from a wholesaler to its distributor. This information can be easily obtained by performing a join of tables Orders and Order_Lines. However, the manufacturer is not permitted to see the pricing information in the orders. For this purpose, a view, Order_Lines_Ancessor is created out of Order_Lines by projecting out the fields other than Price. The query fired by the manufacturer is

*Select * from Orders, Order_Lines_Ancessor where customer_id = 'W' and placing_date between(d1, d2) and product_id = x.*

The predicate appended by the policy is

customer_id in (select child_id from business_topology start with parent_id = SYS_CONTEXT('APPCTX','USER') connect by prior child_id = parent_id and product_id = x)

5.3.3 Echelon Stock of complete Supply Tree

Query 4 requires the echelon stock at node N for item, x. This quantity will increase if fresh shipments of x arrive at N and will decrease if the leaves of this supply tree sell items of x. The echelon stock is included in the table *Inventory* and can be updated either by the application program itself (in response to the two types of events mentioned above) or by a trigger. A node can see its own echelon stock, so the simple policy file that satisfies this constraint will append the following predicate to the raw query:

customer_id = SYS_CONTEXT('APPCTX','USER')

6. DISCUSSION AND CONCLUSIONS

One deficiency of VPDs is that it provides row level security, not column level security (since policy functions may only specify what predicate to append but not which columns to project out.). Another problem is with predicates generated by policy functions on tables that cross-reference each other. For example, the predicates returned by the policy functions corresponding to tables *Orders* and *Order_Lines* in response to SQL queries fired by "ABC" as a grandparent when he wished to see the orders received by his grandchildren are respectively

order_id in (Select order_id from order_lines where product_id in (select product_id from business_topology start with parent_id = "ABC" connect by prior child_id =parent_id))

and

product_id in (Select product_id from business_topology start with parent_id = "ABC" connect by prior child_id =parent_id) and order_id in (Select order_id from orders where supplier_id in (select child_id from business_topology start with parent_id = "ABC" connect by prior child_id =parent_id))

Now the predicate to be appended to an SQL query containing the variable *orders* references the table *order_lines* and the predicate to be appended to an SQL query containing the variable *order_lines* references the table *orders*, the resulting predicate generated by a query that contains either of these tables is invalid.

If customized application tier implementations by third parties is permitted, access control should be implemented in the database tier. The database tier firewall makes it possible to present a simpler, uniform view of the database to the application programmer while at the same time providing for more secure access control. The application developer now has merely to set appropriate attributes in the application context and issue relatively simple queries. Database access control is partially implemented by security views that are not known to the application. These views are known and maintained by the DBA. In addition, he or the security manager (not the VPD) writes and maintains the policy functions that generate predicates that get appended to application queries.

We have considered a small set of potentially useful queries that provide input to help optimize the operations of a supply chain. We believe that in a complete system, which would

include logistics, forecasting, etc., the security requirements would increase commensurately making the job of the DBA or security manager quite challenging. In real world supply chains, supplier-customer relationships would be dynamic making policies hard to maintain manually. Also, requirements of specific clients may vary. For example, X may trust its immediate supplier but not its supplier's supplier. All of these issues make it imperative that we design and implement tools that help to automate the creation and maintenance of views and policy files.

7. REFERENCES

- [1] R. Agarwal, Jerry Kierman, R. Srikant, Y. Xu, "Hippocratic Databases," *Proc. of 28th International Conference on Very Large Databases*, (2002). 143-154.
- [2] R. Agarwal, Jerry Kierman, R. Srikant, Y. Xu, "Order Preserving Encryption of Numeric Data," *Proc. SIGMOD* (2004).563-574.
- [3] G. Aggarwal, M. Bawa, P.Ganeshan, H. Garcia-Molina, K.Kenthapadi, R.Motwani, U. Srivastava, D.Thomas, Y. Xu, "Two can keep a secret: A Distributed Architecture for Secure Database Services," *Technical Report, Stanford University* (2004)
- [4] D. Alur, J. Crupi, D. Malks, "Core J2EE Patterns," *Prentice Hall PTR*(2001).
- [5] Mark Artiges et al, "BEA Weblogic Server 8.1 Unleashed," *Sams Publication* (2003)
- [6] R.Bragg, M. Rhodes-Ousley, K.Strassberg, "Network Security – The Complete Reference," *Tata McGraw Hill* (2004).
- [7] Brian Buege, Randy Layman, Art Taylor, "Hacking Exposed J2EE & JAVA:Developing Secure Applications with JAVA Technology," *Tata McGraw-Hill* (2002)
- [8] F. Chen, "Echelon Reorder Points, Installation Reorder Points, and the Value of Centralized Demand Information," *Management Science*, Vol. 44, No. 12, S (1998) 221-234.
- [9] G.Gangadharan, L.Khandelwal, and B.L.Menezes, "Exchange and Use of critical information in ASP-based Supply Chain Management", *International Conference on E-Governance, New Delhi*, (Dec, 2003).
- [10]H. Hacigumus, B.Iyer, C. Li and S Mehrotra, " Executing SQL over encrypted data in the database service provider model," *Proc. SIGMOD* (2002). 216-227.
- [11]H. Hacigumus, B.Iyer, S. Mehrotra, "Efficient Execution of Aggregation Queries Over Encrypted relational databases," *Proc. DASFAA* (2004) 125-136.
- [12]H. Hacigumus, B.Iyer, S. Mehrotra, "Providing database as a service," *Proc. ICDE* (2002). 29-32.
- [13]Rashmi Jain, B.L. Menezes, Prashant Rajoria, "Security for E-business applications", *Proc. of INFOSEC 2004*, Mumbai, May 2004, 41 – 47.
- [14]M. Kanratcioglu, Chris Clifton, "Security Issues in Querying Encrypted Data," *Technical Report TR04-013, Purdue University*, (2004)

- [15] J. Kaufman, S. Edlund, D. Ford, and C. Powers, "The Social Contract Core," *Proc. International World Wide Web Conference, Honolulu, Hawaii*, (May 2002). 210-220.
- [16] H.L. Lee, V. Padmanabhan, S. Whang, "Information Distortion in a Supply Chain: The Bullwhip Effect," *Management Science*, 43-4 (1997) 546-558.
- [17] M. Marchiori, editor, "The Platform for Privacy Preferences 1.0 (P3P1.0) specification," *W3C Proposed recommendation*, (Jan, 2002)
- [18] "JP Morgan signs outsourcing deal with IBM," *Computer World*, (Dec, 2002)
- [19] Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, Prasan Roy, "Extending query rewriting for fine-grained access control," *SIGMOD* (2004), Paris, France. 551-562.
- [20] M. Rotenberg, "Fair Information practices and the architecture of privacy," *Stanford Technology Law Review*, 1, (2001)
- [21] D. Simchi-Levi, P. Kaminsky and E. Simchi-Levi, "Designing and managing the Supply Chain – Concepts, Strategies and Case Studies," *McGraw Hill*, (2000).
- [22] L. Tao, "Shifting Paradigms with the Application Service Provider Model," *IEEE Computer*, (2001) 32-39.
- [23] K. Walsh, "Analyzing the Application ASP Concept: Technologies, Economics and Strategies," *Communications of the ACM*, Vol. 46, No. 8, (2003) 103-107.
- [24] "The Virtual Private Database in Oracle 9i r2," An Oracle Technical White Paper <http://otn.oracle.com/deploy/-security/oracle9ir2/pdf/vpd9ir2twp.pdf>.

A. SAMPLE POLICY FILE

Policy File for Shipments table

Example Query: "Get the shipments that I am expecting within the next 48 hours"

```

CREATE OR REPLACE PACKAGE BODY
SHIPMENT_SECURITY AS
  FUNCTION SEC_SHIPMENTS(OWNER
  VARCHAR2, OBJECT VARCHAR2)
  RETURN VARCHAR2
  IS
  PREDICATE VARCHAR2(2000);
  FILEHANDLER UTL_FILE.FILE_TYPE;
  BEGIN
  FILEHANDLER := UTL_FILE.FOPEN
('DBDIR','PREDICATELOG.TXT','W');
  IF ( SYS_CONTEXT('APPCTX','ROLE')='CUSTOMER')
  THEN
  PREDICATE := 'SHIPMENT_ID IN ( SELECT
SHIPMENT_ID FROM SHIPMENT_COMPONENTS
WHERE ORDER_ID IN ( SELECT ORDER_ID FROM
ORDERS WHERE CUSTOMER_ID=SYS_CONTEXT
('APPCTX','USER')));
  UTL_FILE.PUT_LINE (FILEHANDLER, PREDICATE);
  UTL_FILE.FCLOSE(FILEHANDLER);
  RETURN PREDICATE;
  END IF;
  IF ( SYS_CONTEXT('APPCTX','ROLE')='SUPPLIER')
  THEN
  PREDICATE := 'SUPPLIER_ID = SYS_CONTEXT
('APPCTX','USER');
  UTL_FILE.PUT_LINE(FILEHANDLER, PREDICATE);
  UTL_FILE.FCLOSE(FILEHANDLER);
  RETURN PREDICATE;
  END IF;
  EXCEPTION
  WHEN utl_file.invalid_path THEN
  raise_application_error(-20000, 'ERROR: Invalid
path for file or path not in INIT.ORA. ');
  END SEC_SHIPMENTS;
END SHIPMENT_SECURITY;
/

```