A Taxonomy Of The Join Operations In The *REA* Data Model

Ting J. Wang (E-mail: tjwang@uwm.edu), University of Wisconsin-Milwaukee
Ping Liu (E-mail: kate0707@sina.com), East China Jiao Tong University

Abstract

The Resource-Event-Agent (REA) data model identifies these three categories of entities in business processes and establishes relationships among them based on the rules that underlay actual business practices. The model becomes more efficient when the principle of relational database design, i.e., normalization, is applied. However, the higher the level of normalization in the database, the higher will be the degree of information segregation. Therefore, to ensure the accuracy of the information retrieved, it is crucial to understand the database structure and apply queries with correct join operations. "Join" is one of the fundamental relational database query operations. Join handles the processes that determine how data from two tables will be merged and selected. In this paper, a taxonomy of the join operations applicable to the REA data model is presented: it classifies the combinations of the categorical components in the REA model, identifies the join operation, and links to AIS documents and reports.

1. Introduction

End-users are increasingly involved in querying databases to find and cope with the information needed to carry out their daily business activities (O'Donnell and David 2000). However, they are not equipped with the knowledge and skills that are essential for performing queries in relational database management systems (Bowen and Rohde 2002; Borthick et al. 2001; Chan et al. 1999). Database topics, such as database architecture, database management systems, and data modeling techniques, are covered to some extent in accounting education, but the textbooks written for the Accounting Information Systems (AIS) courses and covered by AIS educators often do not include sufficient coverage of information retrieval to provide the needed knowledge and skills (Bain, Blankley, and Smith 2002). Some studies provide basic information about database query, such as Structured Query Language (SQL) (Olsen 2000), and query guidelines (Pillsbury and Wang 2003). Nevertheless, a fundamental query language, "join", which specifies how data can be aggregated and selected, is never completely dealt with for AIS end-users.

The purpose of this paper is to provide AIS end-users and educators with knowledge about joins in relational database query, focusing on queries of information, such as accounting documents and financial reports in the *REA* model. We will provide a taxonomy of the join operations applicable to the *REA* data model to classify the combinations of the categorical components in the *REA* model, specify the type of join operation involved, and link to AIS documents and reports. We will demonstrate an *REA* data model by using a simple merchandise company case to illustrate applications of the join operations.

This paper is organized as follows: Section II reviews types of joins, Section III describes AIS applications of the join operations in the *REA* data model, and Section IV concludes the paper.

2. Types Of Joins

The *REA* data model has influenced industrial software and it plays a significant role in the AIS education (Weber 1986; Dunn and McCarthy 1997). The superiority of this data model over the traditional AIS architecture is documented in the literature for its ability to provide both financial and non-financial related information as well as improving data consistency and efficiency in storage (McCarthy 1982; Hollander, et al. 2000).

REA data modeling in relational databases results in related information being decomposed and stored in separate tables. For example, information about business activities are taken apart and stored in different tables according to the *Resources*, *Events*, and *Agents* categories (e.g., Product, Selling Event, and Customer, respectively). Normalization of relations (entities/tables) will further decompose related information stored in these categorical tables. Subsequently, queries are used to merge tables and retrieve related information from the relational database. The join operation is required when there is more than one table involved in a query, combining tuples (records) from two different tables based on some common information.

The join operation has been examined and discussed extensively in the CIS and MIS literature. Most of these studies relate to algorithms of the join operations used to improve efficiency and optimization (Morishita 1997; Mishra and Eich 1992). There are more than a few join types, for example, *equijoin*, *natural join*, *semijoin*, *outerjoin*, and *self-join* (Mishra and Eich 1992), and most database textbooks in CIS and MIS include coverage of them (e.g., Ramakrishnan and Gehrke 2000).

To illustrate join operations, let's first take a look at the Cartesian product of two tables since it is related to some of these operations. The Cartesian product of two tables combines each record of the first table with every record of the second table. In other words, the derived (virtual/result) table of such Cartesian product consists of all combinations of records (N x M) and attributes (data fields) from the two tables. For example, there are Customer and Zip Code tables as follows:

CUSTOMER TABLE	
Name	**Zip Code**
Adam Smith	53201
William King	53211

ZIP CODE TABLE		
ZipID	**City**	**State**
53201	Milwaukee	WI
53706	Madison	WI
54821	Cable	WI

The result table of the Cartesian product is as follows:

Name	**Zip Code**	**ZipID**	**City**	**State**
Adam Smith	53201	53201	Milwaukee	WI
Adam Smith	53201	53706	Madison	WI
Adam Smith	53201	54821	Cable	WI
William King	53211	53201	Milwaukee	WI
William King	53211	53706	Madison	WI
William King	53211	54821	Cable	WI

City and State give information about the customers, but they are stored in two different tables because they are violating the non-transitive dependency of the 3rd norm form (i.e., the Zip Code determines the City and the State attributes). As a result, City, State, and ZipID are discomposed from the Customer table. The data values of the Zip Code and ZipID in Customer and Zip Code tables, respectively, are set to be inconsistent in order to ease the demonstration of different join operations (of course, the inconsistency could also be a result of lack of enforcing *referential integrity*[1] between these two tables).

To perform a join operation, two data fields from the Cartesian product and a join condition must be specified in the query. Multiple conditions can be applied by using the logical connective *AND* (Desai 1990; El-Masri and Navathe 1989; Maier 1983). A general join is called a *theta-join* (or θ-join). A θ operator, such as =, ≠, >, <, ≥, or ≤, will be required in every condition. A join operation is called an *equijoin* when a θ operator, "=", is specified in the condition. For example, a join involving the Customer and the Zip Code tables based on the equality between Zip Code and ZipID, is an *equijoin*. In our previous example, the result table of the *equijoin* will contain the first record of the Cartesian product since that is the only row in which the join condition is met (i.e., the data values of Zip Code and ZipID equal to each other). If a "≠" is used as a θ operator in a general join, the result table will include all but the first records in our example.

The *natural join* can be defined as an *equijoin*, joining on same data fields in both tables and projecting on either one of data fields. In most cases, especially in the *REA* data model, one of the data fields is a key of one table and the other a foreign key created to link the tables, and the *referential integrity* is enforced on a 1-to-Many relationship. Our previous example of the Cartesian product does not fit in, as most cases do, because the *referential integrity* is not enforced (i.e., data values in Zip Code and ZipID are not consistent). Therefore, to demonstrate the *natural join*, suppose that the *referential integrity* were enforced (that is, a Zip Code value, 53211, is referenced in the Zip Code table with 53211, Shorewood, WI), the result table of the *natural join* will include both records from the Customer table with all the data fields from two tables *except* that one of the common data fields is (Zip Code or ZipID) dropped, as shown below.

Name	Zip Code	City	State
Adam Smith	53201	Milwaukee	WI
William King	53211	Shorewood	WI

If the *referential integrity* is enforced between tables (which is not required in the *natural join*) with a 1-to-Many relationship, the information obtained in the result table from the *natural join* will be based on the table that has the "Many" side of the relationship (i.e., the numbers of records are exactly the same). The *natural join* takes the table that has the "Many" side of the relationship and merges data from the table that has the "1" side of the relationship, based on the referenced values in the common data fields. In our modified example, there is a 1-to-Many relationship between the Zip Code and the Customer tables with the referential integrity enforced (i.e., all of the data values in the Zip Code, a foreign key in the Customer table, are referenced in the ZipID, a primary key in the Zip Code table). In the *natural join*, the result table will include all of the information from the Customer table (i.e., the basis of the information from the table with the "Many" side of the relationship) and then merge information about City and State from the Zip Code table based on the referenced values stored in the Zip Code and ZipID. In a case in which the *referential integrity* is not enforced, the basis of the information will still remain in the table that has the "Many" side of relationship, but the non-participative records will be excluded.

The *semijoin* operation is designed to exclude data fields of one table in the result table. It is similar to any join that projects only on data fields from one table. *Semijoin* reduces the size of the second table that is participating in the join operation and improves efficiency (Kambayashi 1985; Perrizo et al. 1989; Yoo and Lafortune 1989). For example, an *equijoin* joins on Zip Code and ZipID from Customer and Zip Code tables, respectively, and projects on Name and Zip Code data fields of the Customer table. It first drops the City and State data fields of the Zip Code table from the join operation and then selects records from the Cartesian product based on the equality (or any other θ operator) condition of Zip Code and ZipID. It will then exclude the joining data field, ZipID, before the result table is finalized.

The *outerjoin* operation (also called the *external join*) handles non-participative (or dangling) records from either or both tables. In any of the before-mentioned joins, records that are not included in the result table are called "non-participative records". For example, the second records in the Customer table, William King and 53211, and the second and the third records in the Zip Code table, 53706 and 54821, are non-participative records in the *equijoin* of the two tables on Zip Code and ZipID. There are three kinds of *outerjoins*: *left-outerjoin*, *right-outerjoin*, and *full-outerjoin*. In the result table, in addition to all participative records, the *left-outerjoin* includes the non-participative records from the left side of the table and the *right-outerjoin* includes the non-participative records from the right side of the table when the tables are placed next to one another in the query (e.g., SQL statement or Query-By-Example [QBE] screen). The *full-outerjoin* includes non-participative (in addition to participative) records from both tables. In our example, the *left-outerjoin* of Customer and Zip Code tables on Zip Code and ZipID for equality contains the first record from both tables and all non-participative records in the Customer table (as shown below), assuming the Customer table is on the left side and the Zip Code is on the right side. The non-participative records will contain null vales on the joining data fields from the table on the right.

Name	Zip Code	ZipID	City	State
Adam Smith	53201	53201	Milwaukee	WI
William King	53211	NULL	NULL	NULL

The *self-join* operation involves joining a table with the table itself. That is, a Cartesian product will be formed consisting of the combination of each record with all records of the same table. The selection of the records that takes place next is based on the data field(s) and the join condition using any of the θ operators. The *self-join* is performed when the relationship between records on a set of specific data fields in that table is of interest (e.g., we may be interested in the customer's purchasing behavior after they have purchased a specific product.)

Some of these join operations are required to retrieve disaggregated information from the database due to the *R-E-A* categories and normalization, and some are used to manipulate the data for various purposes. However, the most commonly used join operations in the *REA* data model are *natural* and *outer joins* because the relationships between tables, if there are any, are pre-defined and pre-established in the data modeling. In the next section, we will use a simple merchandise company case to demonstrate the applications of these two join operations in the *REA* data model to retrieve accounting information, such as accounting documents and financial reports.

3. AIS Applications Of Join Operations In The *REA* Model

We created a simplified merchandise case similar to those in McCarthy (1979) and Wang, Du, and Lee (2002) in order to illustrate the taxonomy of the join operations in the REA data model. Table I shows the entities recognized within each type of business processes for a simplified merchandise company using the *R-E-A* categories in the *REA* data model. There is one *Resources* (Product), four consecutive *Events* (Purchase Ordering, Goods Receiving, Purchase Invoicing, and Purchase Payment Disbursing), and two *Agents* (Vendor and Employee) recognized in the Acquisition and Payment business processes, and one *Resource* (Product), four consecutive *Events* (Sales Ordering, Goods Shipping, Sales Invoicing, and Sales Collecting), and four *Agents* (Salesperson, Customer, Carrier, and Employee) in the Sales and Collection business processes. We eliminate the Cash *Resource* by assuming that there is only one cash account involved in the case. To simplify the case further, suppose that the purchase-invoicing event is an optional entity in the database since most of its data can be obtained from the matched records between the goods receiving and purchase ordering entities.

Figure 1 depicts the *REA* data model, based on the entities recognized in Table I, and the cardinalities of relationships based on a generic merchandising case. We will also assume the types of relationships between entities in the data model, especially for those involving with Many-to-Many type (we will demonstrate the join operation dealing with a Many-to-Many relationship later in the paper). The layout of the *REA* data model in Figure 1 is organized using the types of business processes and the *R-E-A* categories of the entities.

Table II shows the taxonomy of join operations in the *REA* data model by listing the entity categories involved, entity focused, join type based, origin of the information, and documents and reports generated in a query. There are many combinations of the three *R-E-A* categories. However, in our example, there are two common types of combinations applicable to the accounting information systems, such as *R-E-A* and *Event-Event* (*E-E*). In the *REA* data model, most of the relationships are pre-defined and pre-established through the *REA* data modeling. As a result, the most commonly used join operations are *Natural* and *Outer*. Typical documents and reports often used by accounting users in their daily business activities, such as Purchase Order, Receiving Report, Purchase Journal, and Cash Disbursement Journal, are retrieved using these join operations.

The first major type of entity combination involves all three *R-E-A* categories and has *Event* as the origin of the information that will require a *Natural* join in the query (see Table II). This type of join focuses on the *Event* and retrieves its related information from the *Resources* and *Agents* tables. The origin of *Event*-type information includes events such as Purchase Ordering, Goods Receiving, Purchase Disbursing, Sales Ordering, Goods Shipping, Sales Invoicing, and Sales Collecting. Documents are directly associated with *Events* since they are used to capture and/or process information involved in the *Events*. Reports are used to provide summarized information to users about the *Events*. Related events-documents-reports include Purchase Ordering-Purchase Order-Purchase Journal, Goods Receiving-Receiving Report-Receiving Summary, Purchase Disbursing-Check-Cash Disbursement Journal, etc.

For example, to obtain the Sales Journal, *natural joins* are performed involving Product (tblProduct), Sales Invoicing (tblSalesInvoicing), and Customer (tblCustomer) tables (see Figure 1 as well). A relationship table[2] is created for the implementation of a Many-to-Many relationship between tblProduct and tblSalesInvoicing in relational database management systems. The join operation is straightforward in a 1-to-Many situation because it is not involved with an additional table. Figure 2 shows the process of the *natural joins*. In the joins, related information from tblProduct and tblCustomer[3] are first aggregated with the information in the tblSalesInvoicing based on common information stored in the *foreign* keys[4] (e.g., Customer# in tblSalesInvoicing, and Product# and Inv# individually in the relationship table, tblSIProd) and then included in the result table. (See the note in Figure 2 for detailed descriptions of the join operations.) Figure 3 illustrates *natural joins* performed using both QBE and SQL. QBE has a graphical user interface that allows users to create queries by using example tables on the screen; it is often used as a more intuitive user-interface for simpler queries while SQL is utilized for more complex queries.

In the join operations of the *R-E-A* categories in the *REA* data model, the focus also can be placed on the *Resources* (see Table II). In our example, the focus can only be on the Product table since it is the only resource in the simplified case. We can obtain Inventory Subsidiary information using the *outer join* operation on the Product table because we want to show all activities--receiving and/or shipping out, or inactivity--for each of the items in the Product table. However, to obtain Inventory Subsidiary, we must combine information from the database about goods receiving and shipping out. First, a *natural join* will be applied to tblProduct and tblGoodsShipping to obtain information from the database about goods shipped out (See Figure 1 as well). The *natural join* is used because the company can only ship whatever they have in the tblProduct. Since there is a Many-to-Many relationship between tblProduct and tblGoodsShipping, a relationship table is thus created. This relationship table will serve as the basis of the information for the result table in the *natural join* since it is the "Many" side of the relationships to both tblProduct and tblGoodsShipping. Second, an *outer join* operation will be applied to tblProduct and tblGoodsReceiving, based on tblProduct, in order to include all records, both participative and non-participative (i.e., received and non-received), from the tblProduct. It would be a left (or right) *outer join* if tblProduct is on the left (or right). Finally, a *union* operation will be used to combine the results of these two join operations. QBE, as well as SQL, can be used to accomplish these tasks. (The join processes with detailed descriptions, are shown in Figure 4). For example, using QBE, we can first create a Make-Table query for the *natural join* operation after we create a permanent table object (e.g., InventorySubsidiary) to store goods shipped-out-information in the Inventory Subsidiary, and then create another Append query for the *outer join* operation to combine both goods shipped-out- and received-information. The Inventory Subsidiary is retrieved after the result table is sorted, based on the items in the tblProduct and the date.

The focus can also be placed on the *Agents* in the join operations of the *R-E-A* categories (see Table II). The origin of *Agent*-type information includes Salesperson, Customer, Carrier, Employee, and Vendor. Examples of reports about this type of information are performance evaluations-related. The join process in such joins is similar to the *outer join* mentioned above, but because of space limitations, we will not go through any examples.

The second major type of entity combination involves two sequential events, *E-E*, with an emphasis on the previously occurred event. This combination requires an *outer join* on the previously occurred event in the query and produces a result table containing all records, both participative and non-participative, from the previously-occurred event table. The purpose of performing this *outer join* is to obtain non-participative records as well as the participative records that do not meet a specific condition (e.g., quantity received does not equal quantity ordered). As a result, the participative records that do satisfy the condition will be screened out in the query operation. The origin of the *Event→Event*-type information includes Sales Ordering→Goods Shipping, Sales Invoicing→Sales Collecting, Purchase Ordering→Goods Receiving, and Goods Receiving→Purchase Disbursing. Reports that result from the use of this type of information are unfilled sale orders, A/R subsidiary, unfilled purchase orders, and A/P subsidiary.

The processes of the join operations for A/R Subsidiary are similar to those for Inventory Subsidiary (i.e., using Union in Figure 4). However, Figure 5 illustrates an example of A/R aging report by invoice instead. In this example, an *outer join* is performed on two result tables: one for the Invoice Summary information (involving *natural joins* with tables such as tblCustomer, tblSalesInvoicing, tblSIProd, and tblProduct, shown in Figure 2), and another for the Collection Summary information (involving a *natural join* on tblSalesCollecting and tblCustomer) based on the Invoice Summary which contains information about all of the invoices issued. Information contained in both result tables will be aggregated (i.e., GROUP BY and SUM) based on the Invoice# before the *outer join* operation. A query condition will be set on the inequality between InvTotal and CkAmt (i.e., a balance remains on the invoice) to screen out invoices that have been paid in full (See the note in Figure 5 for detailed descriptions of the join operations.) This process also can be used to obtain inventory on-hand information.

4. Conclusions And Future Research

There are many other types of entity combinations that can be applied to AIS using the *REA* data model, such as *E-A*, *R-E*, or *R-A* (these may exist in some cases, but they are not covered in the case used here, nor are many other needed accounting documents and reports). A presentation of the join operations involved in a complete case is definitely of interest but is not within the scope of this paper. We hope that the taxonomy of join operations presented will provide AIS end-users and educators with the needed general knowledge and understanding of join operations and their accounting applications in relational database queries.

We have demonstrated the join applications applicable to the *REA* data model, but this approach can be applied to any *E-R* data model using categories such as Objects, Events, Persons, Places, and Concepts (McFadden, Hoffer, and Prescott 1999). A similar taxonomy consisting of combinations of the categorical elements, focused entity, origin of the information, and documents and reports, can be created accordingly. As a matter of fact, the results in this case may be similar since the categories in the *REA* and the *E-R* are interchangeable (e.g., *Resources*=Objects, *Events*=Events, and *Agents*=Persons).

Footnotes

1 Referential integrity ensures that the foreign key (i.e., Zip Code) of the Customer table is a reference to the primary key (i.e., ZipID) of the Zip Code table.

2 Since a Many-to-Many relationship cannot be directly implemented in relational database management systems, a relationship/bridge table is created to bridge this relationship. As a result, most of the relationships (if not all) in relational database will be One-to-Many.

3 A Zip Code table consisting of City and State may also be involved in the joins if they are needed in the Sales Journal. To simplify the demonstration, however, this particular table is excluded. Some customer and product information is also excluded in the result table.

4 Foreign keys are included to establish relationships between two tables.

References

1. Bain, C. E., A. I. Blankley, and L. M. Smith. 2002. "An examination of topical coverage for the first accounting information systems course". *Journal of Information Systems* (Fall): 143-164.
2. Borthick, A. F., P. L. Bowen, S. Liew, and F. H. Rohde. 2001. "The effects of normalization on end user query errors: an experimental evaluation". *International Journal of Accounting Information Systems* (December): 195-223.
3. Bowen, P. L., and F. H. Rohde. 2002. "Further evidence of the effects of normalization on end-user query errors: an experimental evaluation". *International Journal of Accounting Information Systems* (December): 255-290.
4. Chan, H. C., B. C. Wei, and K. K. Wei. 1999. "Three important determinants of user performance for database retrieval". *International Journal of Human-Computer Study* (November): 895-918.
5. Desai, B. C. 1990. *An Introduction to Database Systems*. St. Paul, MN: West Publishing Co.
6. Dunn, C. L., and W. E. McCarthy. 1997. "The REA accounting model: intellectual heritage and prospects for progress". *Journal of Information Systems* (Spring): 31-51.
7. El-Masri, R., and S. B. Navathe, 1989. *Fundamentals of Database Systems*. Menlo Park, CA: Benjamin/Cummings.
8. Hollander, A. S., E. L. Denna, and J. O. Cherrington. 2000. *Accounting, Information Technology, and Business Solutions*. 2nd ed., Burr Ridge, IL: Irwin/McGraw-Hill.

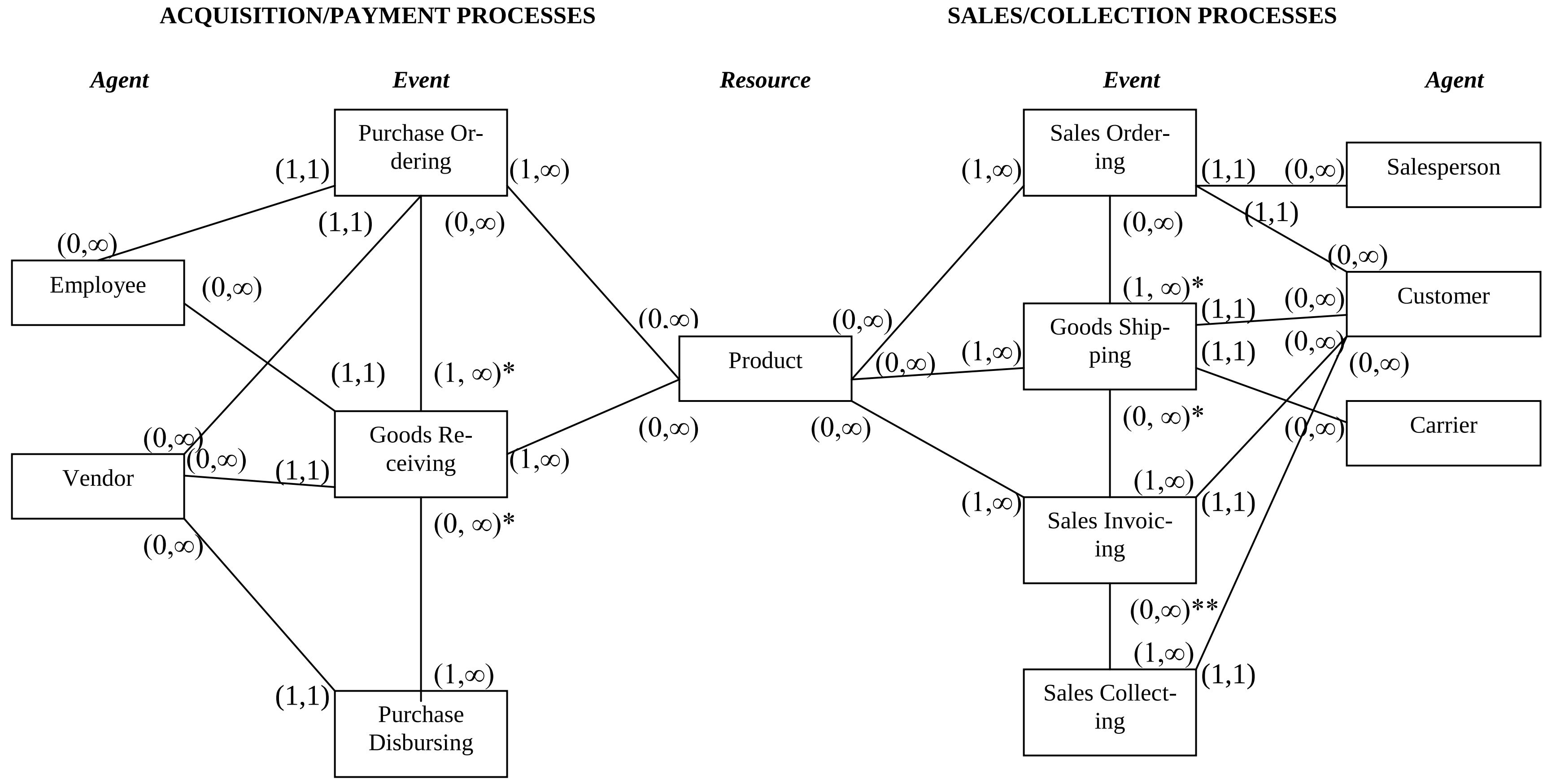
9. Kambayashi, Y. 1985. "Processing cyclic queries". In *Query Processing in Database Systems*, edited by W. Kim, D. S. Reiner, and D. S. Batory, 62-78. New York, NY: Springer-Verlag.
10. McFadden, F. R., J. A. Hoffer, and M. B. Prescott. 1999. *Modern Database Management*. Reading, MA: Addison-Wesley.
11. Maier, D. 1983. *The Theory of Relational Databases*. Rockville, MD Computer Science Press.
12. McCarthy, W. E. 1979. "An entity-relationship view of accounting models". *The Accounting Review* (October): 667-686.
13. ____. 1982. "The REA accounting model: a generalized framework for accounting systems in a shared data environment". *The Accounting Review* (July): 554-578.
14. Mishra, P. and M. H. Eich. 1992. "Join processing in relational databases". *ACM Computing Surveys* (March): 63-113.
15. Morishita, S. 1997. "Avoiding cartesian products for multiple joins". *Journal of the ACM* (January): 57-85.
16. O'Donnell E., and J. S. David. 2000. "How information systems influence user decisions: a research framework and literature review". *International Journal of Accounting Information Systems* (December): 178-203.
17. Olsen, D. H. 2000. "Accounting database design and SQL implementation revisited". *The Review of Accounting Information Systems* (Winter): 53-58.
18. Perrizo, W., J. Y. Lin, and W. Hoffman. 1989. "Algorithms for distributed query processing in broadcast local area networks". *IEEE Transactions on Knowledge Data Engineering* (June): 215-225.
19. Pillsbury, C. M., and T. J. Wang. 2003. "More on supplemental materials for database management system knowledge and skills in the accounting information system course". *The Review of Business Information Systems* (Winter): 11-22.
20. Ramakrishnan, R., and J. Gehrke. 2000. *Database Management Systems*. Boston, MA: McGraw Hill.
21. Wang, T. J., H. Du, and H. Lee. 2002. "A user-oriented approach to data modeling: a blueprint for generating financial statements and other accounting-related documents and reports". *The Review of Business Information Systems* (Fall): 77-92.
22. Weber, R. 1986. "Data models research in accounting: an evaluation of wholesale distribution software". *The Accounting Review* (July): 498-519.
23. Yoo, H., and S. Lafortune. 1989. "An intelligent search method for query optimization by semi-joins". *IEEE Transactions on Knowledge Data Engineering* (June): 226-237.

Table 1
Entity Types for Acquisition/Payment and Sales/Collection Processes of a Simple Merchandise Company

Types of Business Processes	**Entity Types Based on *REA* Data Model**		
	Resource[#]	**Event**	**Agent**
Acquisition/ Payment	Product (tblProduct)	Purchase Ordering (tblPO) Goods Receiving (tblGoodsReceiving) Purchase Invoicing (tblPurchaseInvoice)* Purchase Payment Disbursing (tblPayment)	Vendor (tblVendor) Employee (tblEmployee)
Sales/Collection	Product (tblProduct)	Sales Ordering (tblSO) Goods Shipping (tblGoodsShipping) Sales Invoicing (tblSalesInvoicing) Sales Collecting (tblCashReceiving)	Salesperson (tblSalesperson) Customer (tblCustomer) Carrier (tblCarrier) Employee (tblEmployee)

* This is an optional entity since the information can be derived from tblPO and tblGoodsReceiving (i.e., the matched records).
# To simplify the illustrations, suppose there is only one Cash account involved in the case. As a result, there is no need to create a Cash entity. The cash related information can be gathered from Purchase Payment Disbursing (outflow) and Sales Collecting (inflow) events.

Figure 1
A Simplified *REA* Data Model For A Simple Merchandise Company



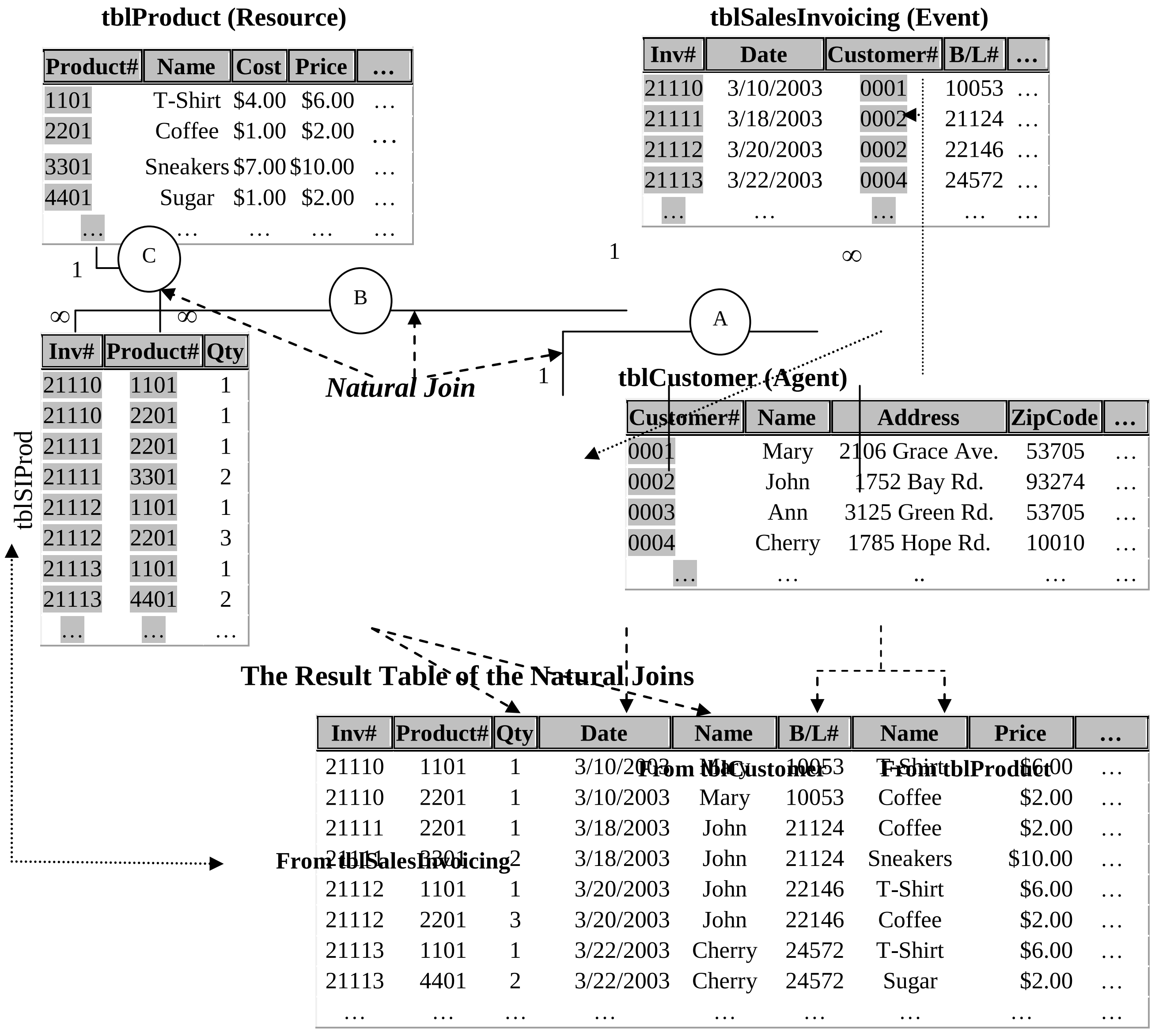
* The maximum cardinality may be 1 in some other cases.
** Customer may have a choice to pay partial payment on the invoice.

Table 2
Taxonomy Of Join Operation In The *REA* Data Model

Entity Categories Involved	Entity Focused	Join Type Based†	Origin of the Information	Documents	Reports
R-E-A	R-E*-A	Natural Join	Purchase Ordering	Purchase Order	Purchase Order Summary
			Goods Receiving	Receiving Report	Receiving Summary
			Purchase Disbursing	Check	Cash Disbursement Journal
			Sales Ordering	Sales Order	Sales Order Summary
			Goods Shipping	Bill of Lading	Shipping Summary
			Sales Invoicing	Sales Invoice	Sales Journal
			Sales Collecting	Cash Receipts	Cash Receipt Journal
	(A-E-)R*-E-A	Outer & Natural Joins	Product	Price List	Inventory Subsidiary
	R-E-A*	Outer & Natural Joins	Salesperson		Performance Evaluation
			Customer		
			Carrier		
			Employee		
			Vendor		
E-E	E_1^*-E_2	Outer & Natural Joins	Sales Ordering & Goods Shipping		Unfilled Sales Orders
			Sales Invoicing & Collecting		A/R Subsidiary A/R Aging Report
			Purchase Ordering & Goods Receiving		Unfilled Purchase Orders
			Goods Receiving & Purchase Disbursing		A/P Subsidiary
	E_1-E_2^*	Outer & Natural Joins	Purchase Ordering & Goods Receiving		Purchase Journal

* Denotes that it is the focus of the join in the query.
† Indicates the join operation based on the focused entity (although there may be other join operations involved in the query).

Figure 2
Natural Join: A Sales Journal Example



tblProduct (Resource)

Product#	Name	Cost	Price	…
1101	T-Shirt	$4.00	$6.00	…
2201	Coffee	$1.00	$2.00	…
3301	Sneakers	$7.00	$10.00	…
4401	Sugar	$1.00	$2.00	…
…	…	…	…	…

tblSalesInvoicing (Event)

Inv#	Date	Customer#	B/L#	…
21110	3/10/2003	0001	10053	…
21111	3/18/2003	0002	21124	…
21112	3/20/2003	0002	22146	…
21113	3/22/2003	0004	24572	…
…	…	…	…	…

tblSIProd

Inv#	Product#	Qty
21110	1101	1
21110	2201	1
21111	2201	1
21111	3301	2
21112	1101	1
21112	2201	3
21113	1101	1
21113	4401	2
…	…	…

tblCustomer (Agent)

Customer#	Name	Address	ZipCode	…
0001	Mary	2106 Grace Ave.	53705	…
0002	John	1752 Bay Rd.	93274	…
0003	Ann	3125 Green Rd.	53705	…
0004	Cherry	1785 Hope Rd.	10010	…
…	…	..	…	…

The Result Table of the Natural Joins

Inv#	Product#	Qty	Date	Name	B/L#	Name	Price	…
21110	1101	1	3/10/2003	Mary	10053	T-Shirt	$6.00	…
21110	2201	1	3/10/2003	Mary	10053	Coffee	$2.00	…
21111	2201	1	3/18/2003	John	21124	Coffee	$2.00	…
21111	3301	2	3/18/2003	John	21124	Sneakers	$10.00	…
21112	1101	1	3/20/2003	John	22146	T-Shirt	$6.00	…
21112	2201	3	3/20/2003	John	22146	Coffee	$2.00	…
21113	1101	1	3/22/2003	Cherry	24572	T-Shirt	$6.00	…
21113	4401	2	3/22/2003	Cherry	24572	Sugar	$2.00	…
…	…	…	…	…	…	…	…	…

Note:

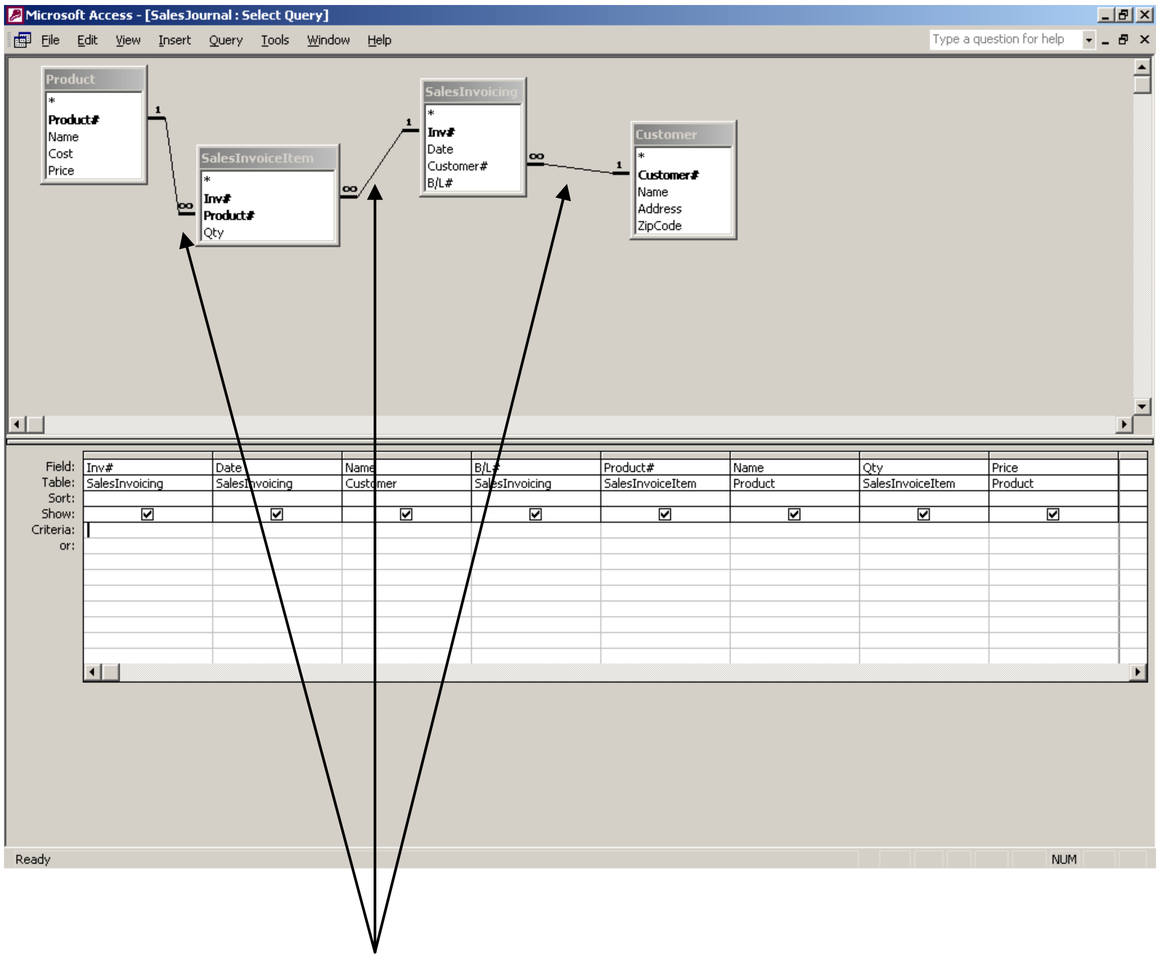
Three join operations are required to obtain information on Sales Journal from four tables, such as tblSalesInvoicing, tblProduct, tblSIProd (a relationship/bridge table created to implement Many-to-Many relationship between tblSalesInvoicing and tblProduct), and tblCustomer.

A *natural join*, marked as **A** above, is performed on tblCustomer and tblSalesInvoicing. The information of Name, Address, and ZipCode from tblCustomer is merged with the information in tblSalesInvoicing based on the referenced values stored in Customer#.

Two *natural joins*, marked as **B** and **C** above, are performed between tblSIProd and tblSalesInvoicing and between tblSIProd and tblProduct, respectively. The contents of the tblSIProd serve as the basis for the information in the result table of the joins because it is the "Many" side of the relationships. Information from tblProduct and the information from the previous result table of the natural join **A** is merged based on the referenced values stored in Product# and Inv#, respectively.

Figure 3
Natural Join: A Sales Journal Example Using QBE and SQL

A. Query-By-Example (QBE)



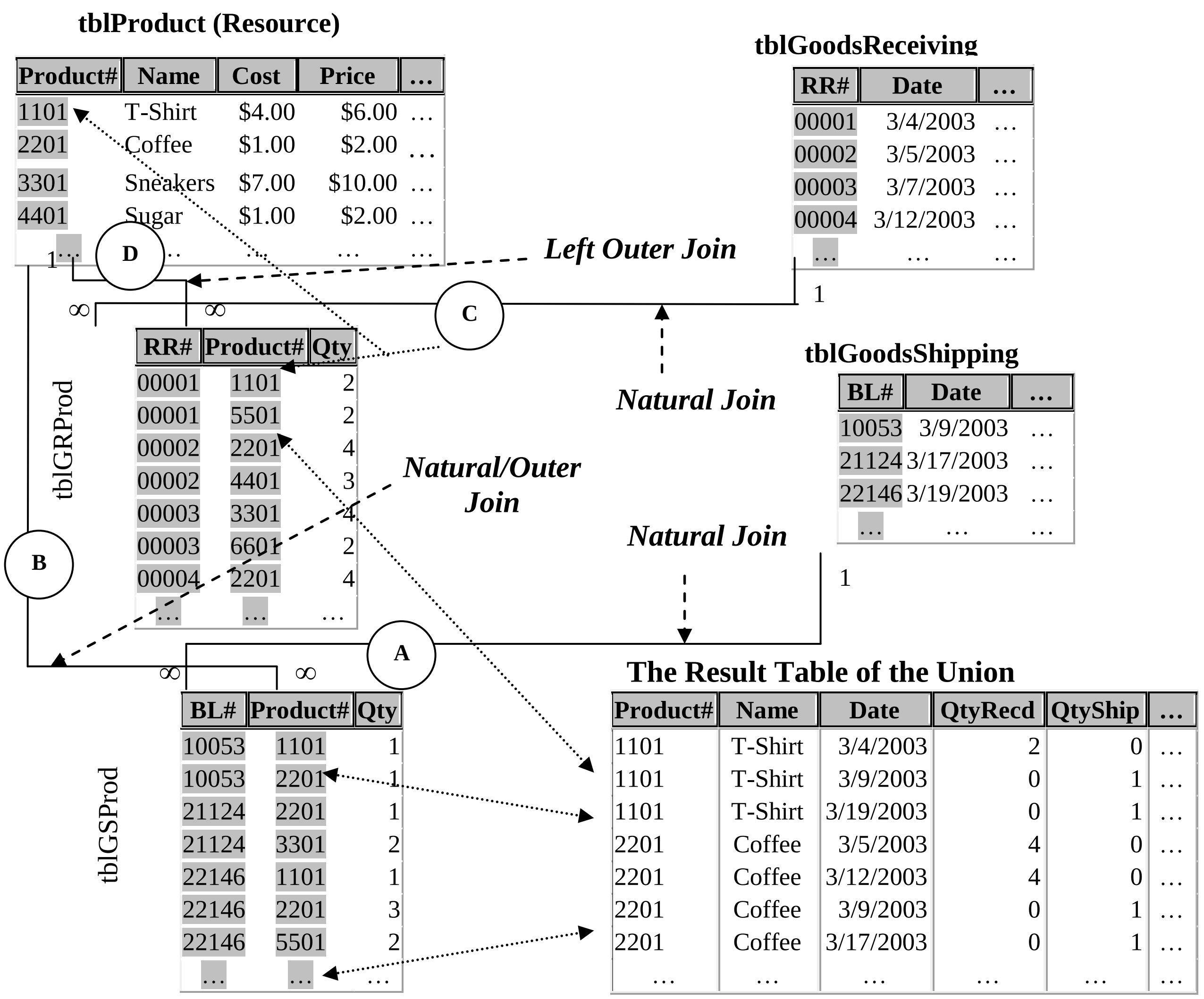
Field:	Inv#	Date	Name	B/L#	Product#	Name	Qty	Price
Table:	SalesInvoicing	SalesInvoicing	Customer	SalesInvoicing	SalesInvoiceItem	Product	SalesInvoiceItem	Product
Sort:								
Show:	☑	☑	☑	☑	☑	☑	☑	☑
Criteria:								
or:								

Applied the default join operation, natural join.

B. Structured Query Language (SQL)

SELECT SalesInvoicing.[Inv#], SalesInvoicing.Date, Customer.Name,
SalesInvoicing.[B/L#], SalesInvoiceItem.[Product#], Product.Name,
SalesInvoiceItem.Qty, Product.Price
FROM (Customer INNER JOIN SalesInvoicing ON Customer.[Customer#] =
SalesInvoicing.[Customer#]) INNER JOIN (Product INNER JOIN
SalesInvoiceItem ON Product.[Product#] = SalesInvoiceItem.[Product#])
ON SalesInvoicing.[Inv#] = SalesInvoiceItem.[Inv#];

Figure 4
Outer and Natural Joins: An Inventory Subsidiary Example



tblProduct (Resource)

Product#	Name	Cost	Price	…
1101	T-Shirt	$4.00	$6.00	…
2201	Coffee	$1.00	$2.00	…
3301	Sneakers	$7.00	$10.00	…
4401	Sugar	$1.00	$2.00	…
…	…	…	…	…

tblGoodsReceiving

RR#	Date	…
00001	3/4/2003	…
00002	3/5/2003	…
00003	3/7/2003	…
00004	3/12/2003	…
…	…	…

tblGRProd

RR#	Product#	Qty
00001	1101	2
00001	5501	2
00002	2201	4
00002	4401	3
00003	3301	4
00003	6601	2
00004	2201	4
…	…	…

tblGoodsShipping

BL#	Date	…
10053	3/9/2003	…
21124	3/17/2003	…
22146	3/19/2003	…
…	…	…

tblGSProd

BL#	Product#	Qty
10053	1101	1
10053	2201	1
21124	2201	1
21124	3301	2
22146	1101	1
22146	2201	3
22146	5501	2
…	…	…

The Result Table of the Union

Product#	Name	Date	QtyRecd	QtyShip	…
1101	T-Shirt	3/4/2003	2	0	…
1101	T-Shirt	3/9/2003	0	1	…
1101	T-Shirt	3/19/2003	0	1	…
2201	Coffee	3/5/2003	4	0	…
2201	Coffee	3/12/2003	4	0	…
2201	Coffee	3/9/2003	0	1	…
2201	Coffee	3/17/2003	0	1	…
…	…	…	…	…	…

Note:

A *Union* of two result tables from four join operations is required to obtain information of Inventory Subsidiary from five tables, such as tblGoodsReceiving, tbleProduct, tblGRProd, tblGoodsShipping, and tblGSProd. Both tblGRProd and tblGSProd are relationship tables.

A *natural join*, marked as **A**, is performed on tblGoodsShipping and tblGSProd. The information of shipping date and others from tblGoodsShipping is merged with the information in tblGSProd based on the referenced values stored in BL#. A *natural join* (or an *outer join* based on tblProduct), marked as **B**, is performed on tblProduct and the result table from the previous join based on the referenced values stored in Product#. The result table will contain information about goods shipping.

A *natural join*, marked as **C**, is performed on tblGoodsReceiving and tblGRProd. The information of receiving date and others from tblGoodsReceiving is merged with the information in tblGRProd based on the referenced values stored in RR#. An *outer join* based on tblProduct, marked as **D**, is performed on tblProduct and the result table from the previous join based on the referenced values stored in Product#. The result table will contain information of ALL goods receiving or not. Finally, a *Union* operation is performed to combine both result tables.

Figure 5
Outer and Natural Joins: A/R Aging Report Example

The Result Table of Invoice Summary Information

Invoice#	Name	Date	InvTotal	…
21110	Mary	3/10/2003	8.00	…
21111	John	3/18/2003	22.00	…
21112	John	3/20/2003	12.00	…
21113	Cherry	3/22/2003	10.00	…
21114	Cherry	3/25/2003	6.00	…
…		…	…	…

The Result Table of Collection Summary Information

Invoice#	CkAmt	…
21110	6.00	…
21111	22.00	…
21112	6.00	…
21114	6.00	…
…		…

Left Outer Join

The Result Table of the Left Outer Join

Invoice#	Name	Date	InvTotal	CkAmt	…
21110	Mary	3/10/2003	8.00	6.00	…
21111	John	3/18/2003	22.00	22.00	…
21112	John	3/20/2003	12.00	6.00	…
21113	Cherry	3/22/2003	10.00		…
21114	Cherry	3/25/2003	6.00	6.00	…
…	…	…	…	…	…

Note:

The result table of Invoice Summary information was obtained using aggregate functions (GROUP BY and SUM) based on the Invoice#, on the result of three *natural joins* on tblCustomer, tblSalesInvoicing, tblSIProd, and tblProduct (shown in Figure 2).

The result table of Collection Summary information was obtained using aggregate functions based on the Invoice#, on the result of a natural join on tblCustomer and tblSalesCollecting.

The A/R aging report by invoice is generated using an *outer join* on the previous result tables with the Invoice Summary table serving as the basis for the join since it contains all invoices issued so far. All invoices in the Invoice Summary are included, whether it is participative or non-participative in the join with the Collection Summary table, in the final result table.

Notes