

A Data Model Validation Approach for Relational Database Design Courses

Kevin R. Parker

Idaho State University, Pocatello, ID, USA

parkerkr@isu.edu

Abstract

This paper presents an instructional method for validating a relational database design. Data model validation is often overlooked in course projects involving relational database design, in part because while most database texts stress the importance of validation, few provide an instructional method for performing validation. Validation is a critical step, especially for students. A flawed data model may omit non-key attributes or even the foreign keys required to join tables. This can make the design of SQL queries, forms, and reports a frustrating experience. This approach requires the designer to determine which attributes account for the field values on forms and reports, which entities are associated with those attributes, and how those entities are linked to an integral or primary entity. Such an approach serves to validate the completeness of the data model.

Keywords: data model validation, instructional method, database, relational database, validation, verification.

Introduction

Several critical steps make up the database life cycle, but data model validation is often overlooked in course projects that teach relational database design. This is unsurprising, because while most database texts stress the importance of validation, few provide an instructional method for performing validation. If a flawed data model is not validated and corrected, SQL queries and the design of forms and reports can be frustrating or even hopeless. This is especially the case if students have failed to include one or more foreign keys, making it impossible to correctly link tables using their data model.

When students are assigned a relational database design project they are generally directed to follow a specific approach for design and normalization. They first identify objects and relations, using techniques such as analysis of report requirements, functional analysis, transaction analysis, or scenario analysis. They next develop an entity-relationship diagram, and finally they normalize the resulting design. Before implementing the design through table creation, the model should be validated to insure that the design is complete. The validation should confirm that all of the data items that appear on a form or report correspond to attributes in the entities, and that the entities are correctly linked.

Material published as part of this journal, either on-line or in print, is copyrighted by Informing Science. Permission to make digital or paper copy of part or all of these works for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage AND that copies 1) bear this notice in full and 2) give the full citation on the first page. It is permissible to abstract these works so long as credit is given. To copy in all other cases or to republish or to post on a server or to redistribute to lists requires specific permission from the publisher at Publisher@InformingScience.org

This paper presents an instructional method for validating a relational database design in order to insure that the database includes a complete attribute set as well as the requisite links between tables. The method also helps students to better understand how data are retrieved from multiple tables so

that they are able to write more complex queries.

Literature Review

An extensive literature review uncovered few, if any, useful instructional methods for data model validation. Wilson (1986) describes the structure of a database design course, but makes no mention of data model validation. Zamperoni and Löhr-Richter (1993) present a complex validation model based on a quantitative dependency graph, but such an approach is too arcane for most students. Haerder (1978) describes the structure of relationships between entities, but makes no mention of data model validation to insure that all relationships are accounted for. Teorey and Fry (1980) uses a processing matrix in which each transaction and report is represented in terms of the attributes required. It may be possible to adapt this approach for validation, but it is not presented as such.

Balouch (1999) makes a brief reference to logical access maps (LAMs). He cites an unnamed source that states that a logical access map “graphically depicts the route of a sequence of actions that act upon the entities and relationships of a data design.” It further states that a LAM “provides a useful check that all the attributes required for a particular transaction are present and can be accessed.” While the concept sounds similar to the one upon which this paper is based, differing definitions of LAMs exist in the literature and no similar definition could be found. Chundur (2001) not only recognizes the need for validation, but also suggests an approach. He notes that designers must insure that the logical data model supports the transactions required by the user view, and those transactions can be determined from the user’s requirements specification. He states that given the ER diagram, the data dictionary and the primary key/foreign entity links, the designer can perform a manual validation. However, he provides no details on how to go about performing such a validation

As noted previously, few texts provide a method for data model validation. Kroenke (2003) notes that the most common form of validation is to show the data model to the users and obtain their feedback. While that may accurately reflect real-world practices, textbook cases do not provide real users, leading students to skip validation if no alternate approach is provided. Kroenke emphasizes that it is exceedingly important to verify that all of the data and relationships necessary to support the use cases are present and accurately represented in the data model. He explains that data model validation is exceedingly important because it is much easier and cheaper to correct errors at this early stage than it is to correct them once the database has been implemented. Connoelly and Begg (1999) briefly point out that data models must be validated against the transactions that they are required to support, and they note the critical role of user involvement in validation. Like Kroenke, however, they provide no instructional method. Kroenke (2004) devotes a bit more time to validation, stating that if the data model is incorrect, then the database will be incorrect. In addition, “forms, reports, and other application elements will then either be wrong, or their design will conflict with the design of the database and it will be very difficult and expensive to construct them.” Like the approach presented here, Kroenke indicates that the data model must incorporate all user descriptions of their data needs, and must support every form and report in the requirements.

Rob and Coronel (2001) define data model verification as verifying the E-R model against the proposed system processes in order to corroborate that the database model can support the intended processes. They include the concept of a central entity, but define it differently than the primary entity described in this paper. They propose a method of verification, but it is based on the concept of database modules and their cohesion and coupling with other modules.

Hoffer, Prescott, and McFadden, (2002) concur that the most common way of evaluating a data model is through a series of reviews. The review team insures that it will be possible to construct a schema to support every form and report in the requirements. Such an approach assumes that

forms and reports are available from an existing system that is being modified, or that prototypes of forms and reports have been constructed. Although it is not used in conjunction with validation, Hoffer, et al., use a type of relational map that shows various entities and the relationships between them. It might be possible to draw such a figure for each form, report, etc., in order to see if that figure is a subset of or can be derived from a similar relational map for the whole database. Then it might be possible to determine if the needed data and relationships are accessible. This is conjecture, because the text does not use relational maps for the purpose of validation.

To sum it up, an examination of the literature was unable to find a well-defined instruction method for data model validation. Most textbooks agree that when validation is performed in a real-world setting it should heavily involve the users, and may use existing or mock-up forms and reports as a basis. However, no one provides an instructional method for validating a data model. This method was developed to fill that gap.

An Instructional Method for Data Model Validation

This approach assumes that forms and reports are available from an existing system that is being modified, or that mock-ups of forms and reports have been developed. When performing the validation, the first step is to determine the primary entity from which the form or report is derived, and then show how the attributes, including those from other entities, are obtained through foreign keys, composite keys, etc. The primary entity is the entity from which the most important (qualitatively speaking) form or report information comes from. For example, a rental contract might have information from a contract entity, a customer entity, and a rental item entity, but the primary entity would be the contract entity. If one of the items on the form or report corresponds to an attribute in some other entity, that entity is referred to here as the source or source entity.

Each form or report must be validated, and each validation adheres to the following format:

```
report/form/process:  <report name/form name/process name>
  This involves <x> entities and composite entities:
    <entity1>
    <entity2>
    :
    <entityn>

The primary entity is <entityprimary>

The data items are accounted for in the E-R Diagram as follows:
```

At this point, each data item must be traced from its source back to the primary entity, and the links between intermediate entities are provided. Tracing can be done using a textual format, a graphical format, or both. There are several cases that must be taken into account. The possible cases are enumerated below, and a detailed explanation with formatting guidelines and examples follows.

The possible cases are

- Case 1: The data item is an attribute of a non-primary entity, which is not linked directly to the primary entity.
- Case 2: The data item is an attribute of the primary entity.

- Case 3: The data item is an attribute of a non-primary entity, which is linked directly to the primary entity. The data item is an attribute that is both the primary key of the non-primary entity and the foreign key of the primary entity or, depending on connectivity, both the primary key of the primary entity and the foreign key of the non-primary entity.
- Case 4: The data item is an attribute of a non-primary entity, which is linked to the primary entity through a composite entity.
- Case 5: The data item is an attribute of a composite entity that is not linked directly to the primary entity.
- Case 6: The data item is an attribute of a composite entity that is linked directly to the primary entity.
- Case 7: The data item is derived from attributes in a non-primary entity through a calculation or a series of calculations.
- Case 8: The data item is derived from previously validated data items through a calculation or a series of calculations.
- Case 9: The data item is obtained from system resources.

Detailed Explanation

Consider a typical business data model in which a Customer places one or more Orders. Each order may include one or more Products. Each Order generates one or more Invoices, since there is a possibility of partial shipments and backordered Products. Each Invoice lists one or more Products. An entity-relationship diagram of this simple scenario appears in Figure 1.

The entity-relationship diagram is characterized by the following:

- Because a Customer can submit multiple orders, but an Order can be submitted only by a single Customer, the Customer table is linked to the Order table through a Customer# foreign key in the Order table.
- Because an Order can generate multiple Invoices, but an Invoice can be associated with at most a single Order, the Order table is linked to the Invoice table through an Order# foreign key in Invoice.
- A Product can be included on many Orders, and an Order can include many products, so the tables are linked through a composite entity (also called an associative entity), Order Includes, that has a composite

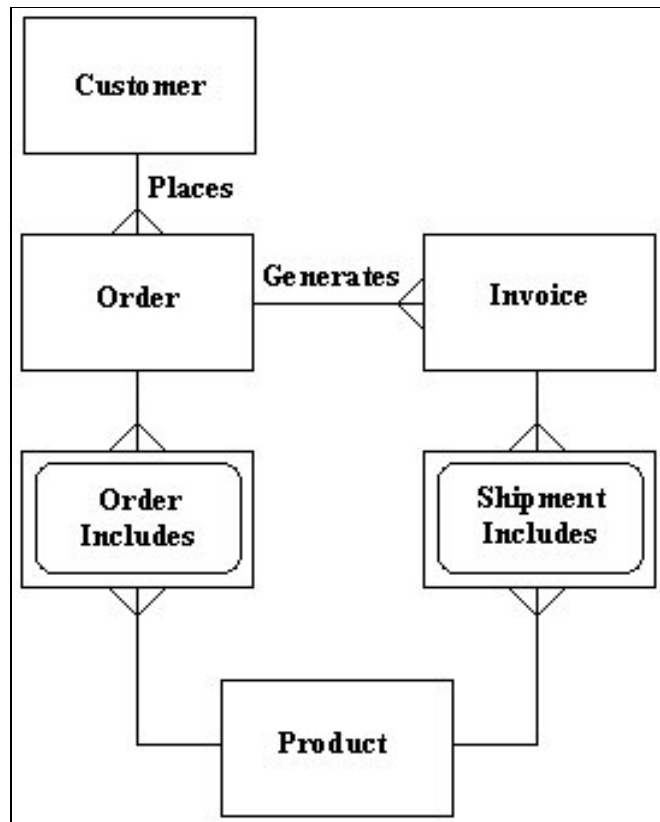


Figure 1: Customer Order Entity-Relationship Diagram.



key made up of Order# and Product#.

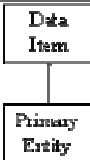

- Similarly, a Product can be included on many Invoices, and an Invoice can include many Products, so the tables are linked through the composite entity, Shipment Includes, that has a composite key made up of Invoice# and Product#.

The examples in the cases below are based on the above diagram, and will include the portion of the entity-relationship diagram that exemplifies the situation to which the case refers. Each case will explain the location of the data item that is being validated in relation to the location of the primary entity. It will then show a graphical depiction of the scenario through a small-scale entity-relationship diagram, followed by the syntax to be followed in the textual tracing. An example of a textual tracing for one of the attributes is then provided, accompanied by the portion of the example entity-relationship diagram or reference.

An alternative graphical tracing is also included. An informal survey of students indicates a slight preference for the graphical tracing, as it may be easier to interpret. The textual tracing, however, forces them to more carefully consider the way in which the entities are linked. Both alternatives are presented here in order to allow the instructor choose the method that best suits their teaching approach.

Case 1:	The data item is an attribute of a non-primary entity, which is not linked directly to the primary entity
ER Illustration:	
Format:	<p><data item>: attribute <attribute_n> of entry <entity_n></p> <ul style="list-style-type: none"> • <entity_n> is linked to <entity_m> through the <attribute_n> foreign key in <entity_m> • <entity_m> is linked to <entity_{primary}> through the <entity_m primary key> foreign key in <entity_{primary}>
Example ERI Segment:	
Example – textual tracing	<p>CUSTOMER NO: attribute C# of entity CUSTOMER</p> <ul style="list-style-type: none"> • CUSTOMER is linked to ORDER via the C# foreign key in ORDER • ORDER is linked to INVOICE through the O# foreign key in INVOICE
Example – graphical tracing	<p>CUSTOMER NO: attribute C# of entity CUSTOMER</p> <p style="text-align: center;"> <u>CUSTOMER</u> ←→ <u>ORDER</u> ←→ <u>INVOICE</u> C# C# C# C# </p>

Case 2	The data item is an attribute of the primary entity. E-R Illustration: 
Format <data item> attribute <attribute _n > of entity <entity _{primary} >	
Example ERD Segment: 	
Example – textual tracing: INVOICE NO attribute I# entity INVOICE.	
Example – graphical tracing: INVOICE NO: attribute I# entity INVOICE.	
<p><u>INVOICE</u></p> <p>I#</p>	

Case 3	The data item is an attribute of a non-primary entity, <entity _n >, which is linked directly to the primary entity, <entity _{primary} >. The data item is an attribute that is both the primary key of <entity _n > and foreign key of <entity _{primary} > or, depending on connectivity, both the primary key of <entity _{primary} > and foreign key of <entity _n >
E-R Illustration:	
Format: <data item> attribute <attribute _n > of entity <entity _{primary} > and entity <entity _n > • <entity _n > is linked to <entity _{primary} > through the <attribute _n > foreign key in <entity _{primary} >	
Example ERD Segment	
Example – textual tracing: ORDER NO: attribute O# of entity INVOICE and entity ORDER. • ORDER is linked to INVOICE through the O# foreign key in INVOICE	
Example – graphical tracing: ORDER NO: attribute O# of entity INVOICE and entity ORDER	
<p>ORDER ↔ INVOICE</p> <p>O# O#</p>	

Case 4: The data item is an attribute of a non-primary entity, which is linked to the primary entity through a composite entity.

E-R Illustration:

The diagram shows a vertical hierarchy. At the top is a box labeled 'Data Item'. A line connects it to a composite entity box (a rectangle with a double border) in the middle. Another line connects the composite entity box to a box labeled 'Primary Entity' at the bottom.

Format:

<data item>: attribute <attribute_n> of entity <entity_n>
 <entity_n> is linked to <entity_{primary}> through the (<entity_n primary key>, <entity_{primary} primary key>) composite primary key in composite entity <entity_{composite}>

Example ERD Segment:

The diagram shows three entities: 'Invoice' at the top, 'Shipment Includes' in the middle, and 'Product' at the bottom. 'Invoice' is connected to 'Shipment Includes' with a line ending in a crow's foot notation (a vertical line with a triangle and a vertical tick). 'Product' is connected to 'Shipment Includes' with a line ending in a crow's foot notation (a vertical line with a triangle and a vertical tick).

Example – textual tracing:
 PRODUCT NO: attribute P# of entity PRODUCT.

- PRODUCT is linked to SHIPMENT INCLUDES through the P# portion of the P#, I# composite primary key in the composite entity SHIPMENT INCLUDES.
- SHIPMENT INCLUDES is linked to INVOICE through the I# portion of the P#, I# composite primary key in SHIPMENT INCLUDES.

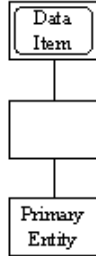
Example – graphical tracing:
 PRODUCT NO: attribute P# of entity PRODUCT.

PRODUCT ↔ SHIPMENT_INCLUDES ↔ INVOICE

P# P#, I# I#

Case 5: The data item is an attribute of a composite entity that is not linked directly to the primary entity.

E-R Illustration:

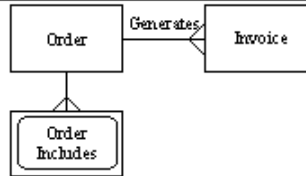


Format:

<data item>: attribute <attribute_n> of composite entity <entity_{composite}>

- <entity_{composite}> is linked to <entity_n> through the <entity_n primary key> portion of the composite primary key in <entity_{composite}>
- <entity_n> is linked to <entity_{primary}> through the <attribute_n> foreign key in <entity_{primary}>

Example ERD Segment:



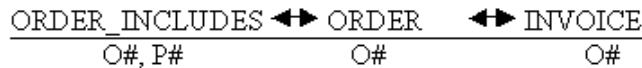
Example – textual tracing:

QTY ORDERED: attribute ORDER_PRODUCT_QTY of composite entity ORDER INCLUDES.

- ORDER INCLUDES is linked to ORDER through the O# portion of the O#, P# composite primary key in ORDER INCLUDES.
- ORDER is linked to INVOICE through the O# foreign key in INVOICE.

Example – graphical tracing:

QTY ORDERED: attribute ORDER_PRODUCT_QTY of composite entity ORDER INCLUDES.



Case 8:	The data item is derived from previously validated data items through a calculation or a series of calculations.
E-R Illustration:	none
Format:	<data item>: obtained from the calculation <data item x> <op> <data item y>
Example ERD Segment:	none
Example – textual tracing:	TOTAL PRICE: obtained by multiplying QTY SHIPPED by UNIT PRICE.
Example – graphical tracing:	TOTAL PRICE: obtained by multiplying QTY SHIPPED by UNIT PRICE.

Case 9:	The data item is obtained from system resources.
E-R Illustration:	none
Format:	<data item>: obtained from system <time/date/etc.>
Example ERD Segment:	none
Example – textual tracing:	CURRENT DATE: obtained from system date.
Example – graphical tracing:	CURRENT DATE: obtained from system date.

Example Validation of the Customer Invoice

The following validation is based on the E-R Diagram presented previously, and is a validation of the customer invoice that appears in Figure 2:

CUSTOMER NO.: <u>12223</u>			INVOICE NO.: <u>06275</u>			
NAME: <u>Clarence Walters</u>			DATE: <u>2/5/04</u>			
ADDRESS: <u>133 6th ST.</u>			ORDER NO.: <u>41284</u>			
<u>Austin, TX 28384</u>						
PRODUCT NO.	DESCRIPTION	QTY. ORD.	QTY. SHIP.	QTY. BACK.	UNIT PRICE	TOTAL PRICE
D331	Computer Desk	3	3		350.00	1050.00
B118	Bookcase	5	3	2	200.00	600.00
L221	Table Lamp	1	1		125.00	125.00
					TOTAL AMOUNT	1775.00
					5.0% DISCOUNT	88.75
					AMOUNT DUE	1686.20

Figure 2: Sample Customer Invoice

Report: Customer Invoice

The Customer Invoice involves six entities and composite entities,

CUSTOMER

ORDER

ORDER INCLUDES

PRODUCT

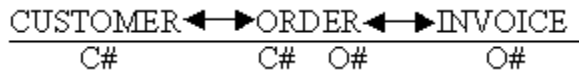
INVOICE

SHIPMENT INCLUDES

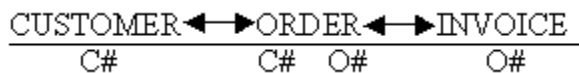
The primary entity is INVOICE.

The data items are accounted for in the E-R Diagram as follows:

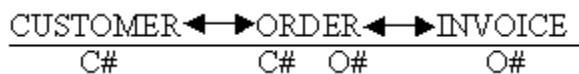
- CUSTOMER NO: attribute C# of entity CUSTOMER.
 - CUSTOMER is linked to ORDER via the C# foreign key in ORDER.
 - ORDER is linked to INVOICE through the O# foreign key in INVOICE



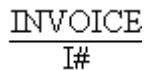
- NAME: attribute C_NAME of entity CUSTOMER.
 - CUSTOMER is linked to ORDER via the C# foreign key in ORDER.
 - ORDER is linked to INVOICE through the O# foreign key in INVOICE.



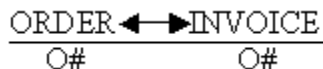
- ADDRESS: attribute C_ADDR of entity CUSTOMER.
 - CUSTOMER is linked to ORDER via the C# foreign key in ORDER.
 - ORDER is linked to INVOICE through the O# foreign key in INVOICE.



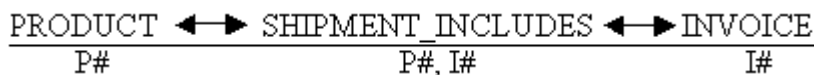
- INVOICE NO: attribute I# entity INVOICE.



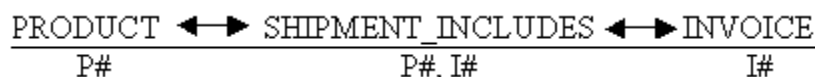
- DATE: obtained from system date.
- ORDER NO: attribute O# of entity INVOICE and entity ORDER
 - ORDER is linked to INVOICE through the O# foreign key in INVOICE.



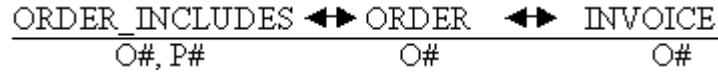
- PRODUCT NO: attribute P# of entity PRODUCT.
 - PRODUCT is linked to INVOICE through the P#, I# composite primary key in composite entity SHIPMENT INCLUDES.



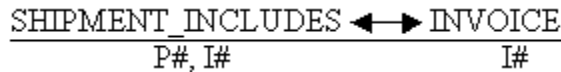
- DESCRIPTION: attribute P_DESCR of entity PRODUCT.
 - PRODUCT is linked to INVOICE through the P#, I# composite primary key in composite entity SHIPMENT INCLUDES.



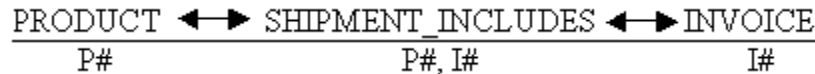
- QTY ORD: attribute ORDER_PRODUCT_QTY of composite entity ORDER INCLUDES.
 - ORDER INCLUDES is linked to ORDER through the O# portion of the O#, P# composite primary key in ORDER INCLUDES.
 - ORDER is linked to INVOICE through the O# foreign key in INVOICE.



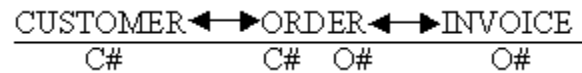
- QTY SHIP: attribute SHIPMENT_PRODUCT_QTY of composite entity SHIPMENT INCLUDES.
 - SHIPMENT INCLUDES is linked to INVOICE through the I# portion of the composite primary key in SHIPMENT INCLUDES.



- QTY BACK: obtained from the calculation ORDER_PRODUCT_QTY minus SHIPMENT_PRODUCT_QTY (see previous two items to trace links).
- UNIT PRICE: attribute P_PRICE of entity PRODUCT.
 - PRODUCT is linked to INVOICE through the P#, I# composite primary key in composite entity SHIPMENT INCLUDES.



- TOTAL PRICE: obtained by multiplying QTY SHIP by UNIT PRICE.
- TOTAL AMOUNT: obtained by summing the TOTAL PRICE column.
- DISCOUNT: obtained from the calculation attribute C_DISCOUNT of entity CUSTOMER multiplied by TOTAL AMOUNT.
 - CUSTOMER is linked to ORDER via the C# foreign key in ORDER.
 - ORDER is linked to INVOICE through the O# foreign key in INVOICE.



- AMOUNT DUE: obtained by subtracting DISCOUNT from TOTAL AMOUNT.

Conclusion

This approach arose from a personal need to provide students with a formal approach to validate their data models. A database designer must insure that all necessary forms and reports can be generated from the available entities and attributes, because a design is of little or no use if the final deliverable is incapable of satisfying all specified query and reporting requirements. As no instructional validation techniques were available, this approach was developed and has evolved over time. Over several years of use it has proved to be useful in not only validating student models, but also in enhancing student understanding of how tables are joined to form relationships. Further, experience has shown that students who perform a thorough validation experience fewer problems when developing queries that involve multiple tables, and this in turn makes the development of forms and reports much less stressful. While this technique may or may not be practical in a real-world situation, it has proven to be very useful as an instructional method. Various

approaches such as textual or graphical tracing can be utilized, but the underlying concept of determining which attributes account for the field values on forms and reports, which entities are associated with those attributes, and how those entities are linked to an integral or primary entity serves to validate the completeness of a data model. It is hoped that by sharing this technique others who teach relational database design will finally have a method with which to better explain the concept of data model validation and its importance in the database life cycle.

References

- Balouch, M.Q. (1999). Information system and database for foreign department office. Retrieved December 10, 2003 from <http://www.geocities.com/rehanaq/bachelor/lam.html>
- Chundur, S. (1991). Database design methodology summary. Retrieved December 10, 2003 from <http://homepages.uc.edu/~chundusa/databasemethodology.html>
- Connolly, T., & Begg, C. (1998). *Database systems: a practical approach to design implementation and management* (2nd ed.). Reading, Massachusetts: Addison-Wesley.
- Haerder, T. (1978). Implementing a generalized access path structure for a relational database system. *ACM Transactions on Database Systems (TODS)*, 3 (3), 285–298.
- Hoffer, J.A., Prescott, M.B., & McFadden, F.R. (2002). *Modern database management* (6th ed.). Upper Saddle River, New Jersey: Prentice-Hall, 2002.
- Kroenke, D.M. (2003). *Database concepts*. Upper Saddle River, New Jersey: Prentice-Hall.
- Kroenke, D.M. (2004). *Database processing: Fundamentals, design, and implementation* (9th ed.). Upper Saddle River, New Jersey: Prentice-Hall.
- Rob, P. & Coronel, C. (2001). *Database systems: Design, implementation, and management* (5th ed.). Boston: Course Technology.
- Teorey, T.J., & Fry, J.P. (1980). The logical record access approach to database design. *ACM Computing Surveys (CSUR)*, 12 (2), 179–211.
- Wilson, J.D. (1986). Problems teaching database design with information complexity to information systems undergraduates. *ACM SIGCSE Bulletin, Proceedings of the Seventeenth SIGCSE Technical Symposium on Computer Science Education*, 18 (1), 2-7.
- Zamperoni, A., & Lohr-Richter, P. (1993). Enhancing the quality of conceptual database specifications through validation. Technical Report 93-17, University of Leiden, The Netherlands, Retrieved December 10, 2003 from <http://citeseer.nj.nec.com/zamperoni93enhancing.html>

Biography

Kevin R. Parker is an Associate Professor of Computer Information Systems at Idaho State University. He has taught courses in both computer science and information systems over the course of his thirteen years in academia. Dr. Parker's research interests include e-commerce marketing, competitive intelligence, knowledge management, and information filtering. He has published several papers in these areas including publications in *Marketing Intelligence and Planning* and the *Journal of Information Systems Education*. Dr. Parker's teaching interests include programming languages, data structures, and database management systems. Dr. Parker holds a Ph.D. in Management Information Systems from Texas Tech University