A "Relational Green Card" Supporting Data Modeling in IS 2002

Leslie J. Waguespack, Jr., Ph.D.
LWaguespack@Bentley.edu
Computer information Systems Department,
Bentley University
Waltham, Massachusetts 02154-4705, USA

Abstract

No individual subject area in IS 2002 impacts more aspects of computing theory or professional preparation than data modeling. For more than four decades the bedrock of data modeling has been the relational data model. There are numerous extensions, variations and implementations of this theory but its core remains the central anchor in the practice of data-driven analysis and design. Like most theoretical foundations that have spawned application development tools and methodologies much of the pure theory of the relational model is obscured by necessary choices of syntax and implementation features that in many cases complicate if not defy a student's grasp of the theory. This is compounded by the progression from one tool or syntax to another as students traverse their computing curricula. This is a distillation of the relational data model compact enough to be easily committed to memory and robust enough to serve as the consistent reference to the relational paradigm spanning IS 2002.P0 through IS 2002.7 and IS 2002.8 for computing majors, minors and general education. In a format reminiscent of the IBM System/360 Principles of Operation Pocket Reference (the "Green Card"), this distillation fits nicely on two sides of a single sheet of 8.5" x 11" paper, hence a "Relational Green Card."

Keywords: relational data model, relational paradigm, data modeling, data-driven modeling, relational model quick reference, data modeling pedagogy

1. INTRODUCTION

No individual subject area in IS 2002 impacts more aspects of computing theory or professional preparation than data modeling. Relational database is among the first dozen learning units designated as prerequisite to the IS 2002 curriculum in IS 2002.P0 (Gorgone et. al 2002). Five of the model courses in IS 2002 explicitly identify database in the learning units including the first preprogram requirement of IS 2002. IS 2002.P0 is also frequently used as the model for computing in general education across all college curricula. Table 1 following lists all the model courses designated in IS 2002: number, title and prerequisites. In the last column those courses indicating required learning units in database are annotated with the number of learning units explicitly requiring database learning compared with the total number of required learning units designated in their descriptions.

For more than four decades the bedrock of data modeling that underpins database pedagogy has been the relational data model (Codd 1969, 1970). There are numerous extensions, variations and implementations of this theory but its core remains the central anchor in the practice of data-driven analysis and design (Chen 1976, Fagin 1981, Zaniolo 1982, Date 2004). Like most theoretical foundations that have spawned application development tools and methodologies much of the pure theory of the relational model is obscured by the necessities of syntax and implementation features that in many cases complicate if not defy a student's grasp of the theory. This is compounded by the progression from one tool or

syntax to another as students traverse their computing curricula.

C#	COURSE TITLE	PR	LU
P0	Personal Productivity With IS Technology		3/21
1	Fundamentals of Information Systems	P0	
2	Electronic Business Strategy, Architecture and Design	1	
3	Information Systems Theory and Practice	1	
4	Information Technology Hardware and System Software	1	
5	Programming, Data, File and Object Structures	1	2/20
6	Networks and Telecommunications	4	
7	Analysis & Logical Design	1	1/14
8	Physical Design and Implementation With DBMS	5,7	7/16
9	Physical Design and Implementation in Emerging Environments	2,8	
10	Project Management and Practice	7	1/11

Table 1
Database Content in IS 2002 Courses

This paper presents a distillation of the relational data model upon which is based an undergraduate and graduate data modeling pedagogy. It is compact enough to be easily committed to memory and robust enough to serve as the consistent reference to the relational paradigm spanning IS 2002.P0 through IS 2002.7 and IS 2002.8. In a format reminiscent of the IBM System/360 Fundamentals of Operation Pocket Reference (the "Green Card"), this distillation fits nicely on two sides of a single sheet of 8.5" x 11" paper, hence a "Relational Green Card."

2. THE RELATIONAL PARADIGM WITHOUT LANGUAGE OR SYNTAX

Every *language* that is invented to express concepts carries with it the understanding and the biases of the inventor. Depending on his/her purpose(s) those biases simplify certain tasks performed with the language but, may obscure underlying concepts.

As a special case programming language design is further complicated by the need for feasibility of automated translation and interoperability with other programming languages and operating systems. Designers must consider upward, downward, and cross-compatibility within versions of a programming language. Compromises and assumptions are chosen to make the resulting language efficient, effective and marketable but not to clarify the underlying theory!

The goal of this description of the relational paradigm is to strip away the extraneous facets that programming language or tool design must use to achieve their "practical" product requirements; and in so doing to succinctly make the underlying relational data model concepts evident and understandable. This approach follows the success of an analogous effort to present the core concepts of the object-oriented paradigm (Waguespack 2009). It provides a knowledge-base that both teacher and student can carry from one data modeling tool or application to another exposing how they treat relational paradigm concepts alike and/or how they treat them differently in practice.

3. ONTOLOGY OF THE RELATIONAL PARADIGM

Computer science and information science categorize a domain of concepts as 1) individuals, 2) attributes, 3) relationships and 4) classes. Following that discipline this ontology of the relational paradigm attempts to eschew the vestiges of implementation languages and development methodologies in order to expose the core nature and value of the relational concepts. The relational ontology is arranged as follows (and is depicted graphically in the map in Figure 1 below while an illustration of a two-page rendering of the "Green Card" is found in appendix A):

Individuals
- Tuple

Attributes
- Data Attributes

Classes
- Relation

Relationships
- Behavioral Relationships
  - -- Functional Dependency
    - --- Entity Integrity
  - -- Association
    - --- Relational Operations
    - --- Join Compatibility
    - --- Referential Integrity

-- Normalization
--- First Normal Form
--- Second Normal Form
--- Third Normal Form

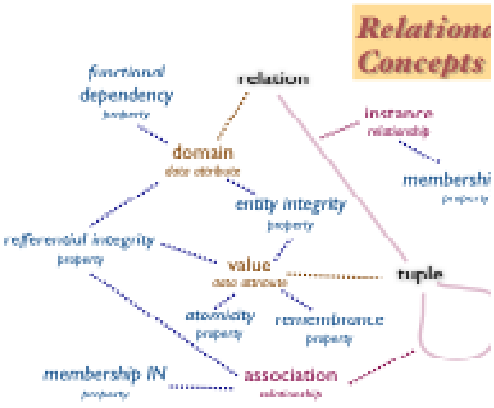


Figure 1 – Relational Concept Map

Table 2
Ontology of the Relational Paradigm

Individuals – The most concrete concept in the relational paradigm is the *tuple*.

Tuple – A tuple corresponds 1-1 with a single concept of reality that it represents. A tuple collects the facts that identify it as a single concept and the facts most closely identified with it.

Attributes – *Attributes* are those characteristics (facts) that describe a tuple. In the relational paradigm attributes define data characteristics - each of which has a static and dynamic form. A prescribed set of attributes defines what is called the structure of a tuple. From inception to extinction the structure of a tuple is immutable. The number of attributes in a tuple is called its *degree*.

Data Attributes – Data attributes store information (data) in the tuple and implement the property of *remembrance*. Remembrance is manifest in each attribute dynamically as "what is remembered," a particular *data attribute value* particular to each tuple derived from a *data attribute domain* that statically defines "what can be remembered," the possible values of the attribute.

Classes – The relational paradigm groups individuals into a collection called a *relation*. The relation corresponds directly with its mathematical antecedent where attribute values within each tuple reflect a correspondence with the coincidence of facts in the "real world," a correspondence (attribute relationship) that is shared by every tuple in that relation.

Relation – The relation concept combines both a definition of structure and the collection of tuple(s) based on that structure. A relation is defined as a fixed set of data attribute domains. Every tuple is an instance of a specific relation and shares the same static structure defined by that relation with every other tuple of that relation. The relation concept thereby fuses the existence of the tuples to that of their relation; tuples cannot exist independent of their defining relation. Tuples are said to be *members of* their relation. Tuples are added to or deleted from their relation. The order of attributes in a relation is insignificant except that the order is consistent for all tuples. A relation is also commonly called a *table* and each of its tuples or instances, a *row*. The collection of data attribute value(s) for a particular data attribute from every row in a table is called a *column*.

Relationships – Relationships in the relational paradigm are based on the property of remembrance and the juxtaposition of data attribute values in one or more tuples in the same or across relations.

Behavioral Relationships – The behavioral relationships are all based upon the data attribute value(s) and which values are permitted to coexist in and across tuples and relations.

Functional Dependency – In a relation a data attribute is *functionally dependent* when its data attribute value is always the same in any tuple for a given value in a second data attribute. In other words, the value of the first data attribute is *determined by* the value of the second; the second attribute is sometimes called the *determinant*. Functional dependency expresses the informational integrity of relations.

Entity Integrity – *Entity integrity* defines the two-fold quality of tuple uniqueness in a relation: a) every tuple in a relation is distinct in some data attribute value(s) from every other tuple in that relation or symmetrically, b) there is a designated subset of data attributes (column(s)) called the *primary key* such that the data attribute value(s) of those data attribute(s) in that relation is distinct for all tuples and no values among them may be *null* (a value which is unknown and incomparable to any other value). There may be more than one subset of data attributes with the value characteristics of the primary key (each called a *candidate key*) but only one is designated as the primary key.

Association – An *association* is a relationship between tuples in the same or different relations. Tuples are intrinsically separable by way of entity integrity. At the same time, humans are compelled to categorize their experience of things in the physical world by superimposing groupings that collect tuples into sets. Tuples become members in a group based upon data attribute value(s). This property is called *membership IN*. This property also permits humans to identify a tuple that is not in a set (i.e. discrimination). (Membership IN an association is distinct from membership OF a relation that is intrinsic by way of instance relationship.)

Relational Operations – Membership IN is realized through relational operations keying on relation structure and values. Each relational operation produces a real or virtual relation as its result. The *selection* operation retrieves tuple(s) based upon a selection predicate testing data attribute value(s) to determine whether each tuple is or is not in the set. Selection predicates are based on any boolean comparison including constant values or values referenced in data attribute value(s). The *projection* operation copies all the data attribute value(s) for a particular column(s). Association between relations (or a relation and itself) is based upon relating (matching) data attribute values in tuples of one relation with those of another. The *join* operation pairs every combination of tuples from one relation with those of another relation and copies the data attribute values from the pairs where the pairing satisfies a selection predicate. This relational operation is called join because facts from two sources are joined in the result.

Join Compatibility – *Join compatibility* requires that the values involved in comparisons (i.e. selection predicates) whether constants or data attribute values derive from the same data attribute domain.

Referential Integrity – When relations are devised such that a tuple in one relation predisposes the existence of (owns) tuple(s) in another, the data attribute(s) of the second required to join the relations is called a *foreign key*. *Referential integrity* asserts that any value found in the data attribute(s) of a foreign key must appear in a tuple of the first relation as the value of a candidate key or itself be null.

Normalization – Relational model consistency depends on the semantic concurrence of the behavioral relationships and the objectives of the database modeler, the intension, (rather than the accident of a relation's contents at any particular instant, its extension). The integrity properties defined above enable the database modeler to devise a structure and behavior of relations that avoid semantic discord called *anomalies*, the unintended loss or modification of information by relational operations. Relations designed to avoid certain kinds of anomalies are said to be *normalized* or in *normal form*. *Normalization* is the arrangement of data attributes and their relationships among relation structures to prevent particular anomalies.

First Normal Form – *First Normal Form* asserts that every data attribute value is atomic, indivisible in value or form and may not be operated upon except as a whole and single value.

Second Normal Form – *Second Normal Form* is first normal form and asserts that every data attribute value not in the primary key is *fully functionally dependent* upon the primary key. ("Fully" means applying to every data attribute

of the primary key.)

Third Normal Form – *Third Normal Form* presupposes first and second normal forms and asserts that no data attribute outside the primary key is *transitively dependent* upon the primary key. ("Transitively" means an attribute(s) functionally dependent upon an attribute functionally dependent upon an attribute [. . .] functionally dependent on the primary key.)

4. DISCUSSION

As with any formal theory this relational ontology as a distillation forms the basis not a complete pedagogy for teaching/learning data-driven modeling and analysis. Pedagogical completeness is not the intention. An effective pedagogy built upon this ontology also depends upon the academic maturity of the audience, the curricular context of the coursework and the expository style of the teacher.

Although the ontology is succinct it has proven effective as a teaching vehicle for presenting concepts at varying levels of detail. It is elemental and defines the essential vocabulary to frame any discussion of data modeling. Issues of data integrity can be addressed at either the operational level as with the relational operations and their mechanics or at a conceptual level as with the intention of the data modeler in their representation of "reality" through normalization. The richness of the paradigm is thereby preserved and crystallized.

The relational ontology provides both a framework for organizing pedagogy and a discipline for choosing pedagogical instruments that consistently ground the teacher and student to the theoretical roots regardless of which application domain or problem solving tool is chosen.

5. SUMMARY

This is a very short presentation of a succinct, compact description of the relational paradigm without the embellishments or compromises often necessary to support computer-based translation (as in a query language such as SQL or QBE) or a graphically augmented representation such as Entity-Relationship diagrams. The ontology derives from the very earliest of conceptions of the relational paradigm at a time before there was competition for commercial-dominance, language or methodology standardization.

The primary value of this approach in explaining the relational paradigm is two-fold. First, absent the accidents of implementation that accompany all programming languages, both the student and teacher of the relational data model have a basis for discriminating between those features that are essential to the paradigm and those that are accidental to an implementation of it (Brooks 1987). Second, it also facilitates assessing the relational data model's role in more advanced applications of the paradigm (e.g. query languages, embedded data languages and application programming interfaces).

Data modeling can be likened to a religion with its saints, zealots and heretics. For that reason and the fact that at its core it is a framework or pattern for creating abstractions, conceptions in the human mind, it may not be possible to find a uniquely perfect, universally accessible depiction of the paradigm itself. As with all models, this explanatory model for the relational paradigm cannot be judged as perfect, but perhaps it may be judged as useful.

6. ACKNOWLEDGEMENTS

Special thanks are due my colleagues in Computer Information Systems at Bentley University for their insightful discussions and comments on several earlier drafts of these ideas and to Bentley's administration for the continuing support of this exploration. Thanks to the Bentley students who suffered through earlier versions of the "Green Card" and motivated clarifications and crystallizations of the text. Thanks to the ISECON community (organizers, editors, reviewers and participants) for continuing to support a venue where ideas and discussions of information systems education can be aired and debated for the benefit of all our students and our disciplines.

7. References

Brooks, Frederick P. (1987) "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, Vol. 20, No. 4 (April) pp. 10-19.

Chen, Peter (1976), "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems*, Vol. 1, No. 1, March, pp. 1-36.

Codd, E. F. (1969) "Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks," IBM Research Report

Codd, E. F. (1970) "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, Vol. 13, No. 6 [June 1970]: pp. 377-387.

Date, C. J. (2004) Introduction to Database Systems, Boston, Pearson/Addison Wesley.

Fagin, Ronald (1981), "A Normal Form for Relational Databases That is Based on Domains and Keys," *Communications of the ACM*, Vol. 6, pp. 387-415.

Gorgone, John T., Gordon B. Davis, Joseph S. Valacich, Heikki Topi, David L. Feinstein, and Herbert E. Longenecker, Jr. (2002), Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems, Association for Computing Machinery (ACM), Association for Information Systems (AIS), Association of Information Technology Professionals (AITP).

Waguespack, L. J. (2009), "A Two-Page 'OO Green Card' for Students and Teachers," Information Systems Education Journal, 7 (61), http://isedj.org/7/61/. ISSN: 1545-679X. (A preliminary version appears in The Proceedings of ISECON 2007: §3743. ISSN: 1542-7382.)

Zaniolo, Carlo (1982), "A New Normal Form for the Design of Relational Database Schemata," *ACM Transactions on Database Systems*, Vol. 7, No. 3, September

Appendix A

Green Card Illustration

The Relational Green Card may be effectively reproduced as the front and back of a single 8.5" x 11" sheet of paper. Terms used with special meaning are italicized. Those initially defined are also bolded.

THE RELATIONAL "GREEN CARD" NOVEMBER 11, 2008

The Relational Paradigm

Without a Language or Syntax!

What is the relational world all about?

The Relational Ontology

This ontology is consistent with the practice in computer science and information science categorizing a domain of concepts (i.e. individuals, attributes, classes and relationships). This ontology of the relational paradigm of data modeling minimizes the vestiges of implementation languages and methodologies to expose the core nature of relational concepts.

1. Individuals

The most concrete concept in the relational paradigm is the ***tuple***.

1.1. Tuple

A *tuple* corresponds 1-1 with a single concept of reality that it represents. A *tuple* collects the facts that identify it as a single concept and the facts most closely identified with it.

2. Attributes

Attributes are those characteristics (facts) that describe a *tuple*. In the relational paradigm *attributes* define data characteristics - each of which has a static and dynamic form. A prescribed set of *attributes* defines what is called the ***structure*** of a *tuple*. From inception to extinction the *structure* of a *tuple* is immutable. The number of *attributes* in a *tuple* is called its ***degree***.

2.1. Data Attribute

Data attributes store information (data) in the *tuple* and implement the property of ***remembrance***. *Remembrance* is manifest in each *attribute* dynamically as "what *is* remembered," a particular ***data attribute value*** for each *tuple* derived from a ***data attribute domain*** that statically defines "what *can* be remembered," the possible values of the *attribute*.

3. Classes

The relational paradigm groups individuals into a collection called a ***relation***. The *relation* corresponds directly with its mathematical antecedent where *attribute* values within each *tuple* reflect a correspondence with the coincidence of facts in the "real world," a correspondence (*attribute* relationship) that is shared by every *tuple* in that *relation*.

3.1. Relation

The ***relation*** concept combines both a definition of *structure* and the collection of *tuple(s)* based on that *structure*. A *relation* is defined as a fixed set of *data attributes*. Every *tuple* is an ***instance*** of a specific *relation* and shares the same static *structure* defined by that *relation* with every other *tuple* of that *relation*. The *relation* concept thereby fuses the existence of the *tuples* to that of their *relation*; *tuples* cannot exist independent of their defining *relation*. *Tuples* are said to be ***members of*** their *relation*. *Tuples* are added to or deleted from their *relation*. The order of attributes in a *relation* is insignificant except that the order is consistent for all *tuples*. *A relation* is also commonly called a ***table*** and each of its *instances*, a ***row***. The collection of every *data attribute value(s)* for a particular *data attribute* in a *table* is called a ***column***.

4. Relationships

Relationships in the *relational* paradigm are based on the property of *remembrance* and the juxtaposition of *data attribute values* in one or more tuples in the same or across *relations*.

4.1. Behavioral Relationships

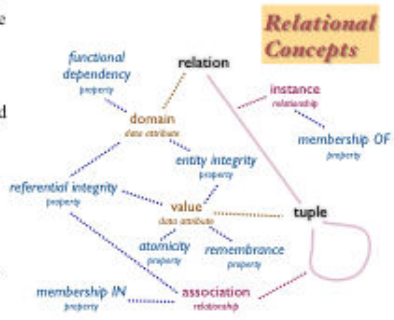
The behavioral relationships are all based upon the *data attribute value(s)* and which values are permitted to coexist in and across *tuples* and *relations*.

4.1.1. Functional Dependency

In a *relation* a *data attribute* is ***functionally dependent*** when its *data attribute value* is always the same in any *tuple* for a given value in a second *data attribute*. In other words, the value of the first *data attribute* ***is determined by*** the value of the second (called the ***determinant***). *Functional dependency* expresses the informational integrity of *relations*.

4.1.1.1. Entity Integrity

Entity integrity defines the two-fold quality of *tuple* uniqueness in a *relation*: a) every *tuple* in a *relation* is distinct in some *data attribute*

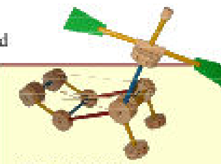


 PAGE 1

value(s) from every other *tuple* in that *relation* or symmetrically, b) there is a designated subset of *data attributes* (*column(s)*) called the ***primary key*** such that the *data attribute value(s)* in that *relation* is distinct for all *tuples* and no values may be ***null*** (a value which is unknown and incomparable to any other value). There may be more than one subset of *data attributes* with the value characteristics of the *primary key* (each called a ***candidate key***) but only one is designated as the *primary key*.

4.1.2. Association

An ***association*** is a relationship between *tuples* in the same or different *relations*. *Tuples* are intrinsically separable by way of *entity integrity*. At the same time, humans are compelled to categorize their experience of things in the physical world by superimposing groupings that collect *tuples* into sets. *Tuples* become members in a group based upon *data attribute value(s)*. This property is called ***membership IN***. This property also permits humans to identify a *tuple* that is not in a set (i.e. discrimination). (*Membership IN an association is distinct from membership OF a relation which is intrinsic by way of instance relationship.*)



Without syntax!

Every language that is invented to express concepts carries with it the understanding and the biases of the inventor. Depending on his/her purpose(s) those biases simplify certain tasks performed with the language but may obscure the underlying concepts.

Programming language design must deal with the feasibility of automated translation and interoperability with other programming languages and operating systems. Compromises and assumptions are chosen to make the resulting language efficient, effective and marketable.

The goal of this description of the entity-relationship paradigm is to succinctly make the concepts understandable - an ambitious task to say the least.

-Professor Waguespack

4.1.2.1. Relational Operations

Membership IN is realized through **relational operations** keying on relation structure and values. Each relational operation produces a real or virtual relation as its result. The **selection** operation retrieves *tuple*(s) based upon a **selection predicate** testing data attribute value(s) to determine whether each *tuple* is or is not in the set. Selection predicates are based on any boolean comparison including constant values or values referenced in *data attribute value(s)*. The **projection** operation copies all the *data attribute value(s)* for a particular *column(s)*.

Association between *relations* (or a *relation* and itself) is based upon relating (matching) *data attribute values* in *tuples* of one *relation* with those of another. The ***join*** operation pairs every combination of *tuples* from one *relation* with those of another *relation* and copies the *data attribute values* from the pairs where the pairing satisfies a *selection predicate*. This *relational operation* is called *join* because facts from two sources are joined in the result.

4.1.2.2. Join Compatibility

Join compatibility requires that the values involved in comparisons (i.e. *selection predicates*) whether constants or *data attribute values* derive from the same *data attribute domain*.

4.1.2.3. Referential Integrity

When *relations* are devised such that a *tuple* in one *relation* predisposes the existence of (***owns***) *tuple(s)* in another, the *data attribute(s)* of the second required to *join* the *relations* is called a ***foreign key***. Referential integrity asserts that any value found in the *data value attribute(s)* of a *foreign key* must appear in a *tuple* of the first *relation* as the value of a *candidate key* or itself be *null*.

4.1.3. Normalization

Relational model consistency depends on the semantic concurrence of the behavioral relationships and the objectives of the database modeler, the ***intension,*** (rather than the accident of a *relation's* contents at any particular instant, its ***extension***). The integrity properties defined above enable the database modeler to devise a structure and behavior of *relations* that avoid semantic discord called ***anomalies***, the unintended loss or modification of information by relational operations. *Relations* designed to avoid certain kinds of *anomalies* are said to be ***normalized*** or in ***normal form***. ***Normalization*** is the arrangement of *data attributes* and their relationships among *relation* structures to prevent particular *anomalies*.

4.1.3.1. First Normal Form

First Normal Form asserts that every *data attribute value* is ***atomic***, indivisible in value or form and may not be operated upon except as a whole and single value.

4.1.3.2. Second Normal Form

Second Normal Form is first normal form and asserts that every *data attribute value* not in the primary key is ***fully functionally dependent*** upon the *primary key*. ("Fully" means applying to every *data attribute* of the *primary key*.)

4.1.3.3. Third Normal Form

Third Normal Form presupposes first and second normal forms and asserts that no *data attribute* outside the *primary key* is ***transitively dependent*** upon the *primary key*. ("Transitively" means an attribute(s) *functionally dependent* upon an attribute *functionally dependent* upon an attribute (. . .) *functionally dependent* on the *primary key*.)