Some ideas about a new Data Model

Vladimir Odrljin
New York City, NY USA

Email: vldm10@yahoo.com
Posted: September 17, 2005

1. Ordered pair (Conceptual Model, Logical Model)

A Conceptual Model is a domain in which a part of the Real World associated with subject knowledge is represented.
This model, aside from Entities, Relationship, Attributes and Attributes' values, has Events. There are only two kinds of events in the Real World:
i) An event which causes new information
ii) An event which causes some existing information to not be valid after this event. We will also say this event closes information.
Here information is the meaning of the event in the Conceptual Model. A Conceptual Model has events which correspond to events from the Real World. Here they can have only two values: N and C, they are abbreviations for "new" and "close".

1.1 Construction of Conceptual Model

We determine the Conceptual Model so that every entity and every relationship has only one attribute, all of whose values are distinct. So this attribute doesn't have two of the same values. We will call this attribute the Identifier of the state of an entity or relationship. We will denote this attribute by the symbol Ack. All other attributes can have values which are the same for some different members of an entity set or a relationship set. Besides Ack, every entity has an attribute which is the Identifier of the entity or can provide identification of the entity. This Identifier has one value for all the states of one entity or relationship.
Like the Logical Model, here we will use the Relational Model, although the above is not limited to the Relational Model.
An entity set and a relational set are mapped into relations of the Relation Model. Attributes from an entity or relationship are mapped to the relation's attributes. The events from Conceptual Model are mapped to corresponding "new" and "close" events from the Relational Model.
Let's denote by Ark the attribute in relation R which corresponds to the Ack. All the values of the attribute Ark are unique.

1.2 Definition of key

The key K for relation R is the attribute Ark such that:
1. The key K uniquely determines a tuple in relation R and is the primary key.
2. One particular value of K provides identification for one state of the corresponding entity (or the relationship)
3. Two or more of the key's values from relation R, which are related to one corresponding entity (relationship), provide better identification and meaning to the entity (relationship).

This construction of entities and relationships enables corresponding relations to be almost normalized. We don't have compound keys, we have only one candidate key, and generally speaking, all the other values that are not Ark's can be repeated any number of times.

1.3 Example

Given the table Car

CarKey	CarID	Maker	Color	…
…				
23	vin1	Buick	silver	…
24	vin1	Buick	blue	...
25	vin1	Buick	red	…
26	vin2	Honda	silver	…
27	vin3	Ford	black	...
…				

Here the CarKey is the Identifier of the state of the entity Car and this is the only column of the table which has unique values. So the attribute CarKey is the primary key.
Car ID is an Identifier of the entity Car. We use VIN values for this attribute. Maker, Color,… are attributes of the table Car. In this example the CarKey's values 23, 24 and 25 denote three states of the one car identified with the CarID = vin1. These states are determined with the car's color.
We can see that the above definition of the key is satisfied:
1. Every value of the key is from the attribute CarKey. The CarKey is, in fact, the column Akr and because of that it has unique values.
2. One particular value of the key is one state of the car. Value 23 defines the car as silver, and value 24 describes the car's color as blue.
3. The three values 23, 24, and 25 describe the car much better than one value. For some purposes these three values are more meaningful (for example an investigation of a stolen car). We can identify every car in the real World with the value of the CarID. Now, let Person be the following table:

PersonKey	PersonID	PersonName	…
…			
208	ssn1	Mary Jones	…
209	ssn1	Mary Adams	…
210	ssn2	John Stewart	…
…			

Where PersonKey is an Identifier of the state of the entity Person, PersonID is the Identifier of the entity Person, PersonName is the name of the person. Here Mrs. Mary Jones changed her last name because she had gotten married to Mr. Adams.
Let's form the relationship Owner which corresponds to a person who is the owner of a car: This relationship can have the following values

OwnerKey	Person Key	CarKey	Year
…			
54	210	26	2003
55	210	24	2004
56	210	26	2005
…			

where OwnerKey is the Identifier of the state of the relationship Owner,
PersonKey is the Identifier of the state of the entity Person,
CarKey is the Identifier of the state of the entity Car, and year is the year of ownership.
Here Mr. John Stewart bought a Honda in 2003 and then he sold it to his friend.
He bought a Buick in 2004. In 2005 he again bought his old Honda from his friend.

2. Creating, representing and implementing knowledge in a database

(Posted September 22, 2005)

2.1 Categorizing of data

We will divide data into two categories

1. Derived data

Simply said, these data are not written into a database. They are obtained from existing data in the database. For example, these are data which we can get from a report, display, view, or query, as well as data which we can get by applying operations on existing data in the database.

2. Data which get written into the database; they are new and can't be derived from existing data in the database. In this case the database will be changed. There are two kinds of these data:

i) Data which are our own information and which we maintain.

ii) Data which are somebody else's information and which somebody else sometimes maintains. But we get these data from them and store the data into the database. These kinds of data are, for example SSN, Address etc. It means that in this case there are multiple sources of knowledge about a data. Similarly, pictures and multimedia information which are made by certain devices have specifics sources of knowledge. For example a picture is made by a camera and stored directly into a database.

The user can write in (enter) only two kinds of data into the database:

i) new data i.e. data which is new in the database;

ii) data which have existed in the database as valid and which now have been announced invalid from some point in the time i.e. these data are not current or existing anymore from some point of time. We will call this – closing the current data or simply, closing.

Here the term data refers to a value that is stored in the database. We create the new data using the Constructor. We close the data (closing) with the ClosingConstructor. These two constructors in some way correspond to the Constructor and Destructor from OOP. The difference is that ClosingConstructor doesn't delete or destroy data; it just says that the data is not valid from some point in the time. The second difference is that these constructors can be applied only to one data. The third difference is that these two constructors are related (initiated) to events. There are only two events in the database: event N and C. As it is earlier said, the first one is creating new data in the database and the second is closing current data. Events N and C usually correspond to events in the Real World. The forth Difference is that, using Constructor and ClosingConstructor, we create keys and knowledge in the database. Finally, these two constructors are user created i.e. they can vary from case to case. The event in the Real World defines the state of an entity or relationship following this event. The amount of knowledge which we will apply depends on us.

2.2 Knowledge related to the data and its representation in the database.

1. We will distinguish our knowledge from another's and we will separately represent these two in the database.

2. We will distinguish knowledge related to the data of the Real World from knowledge related to the data of the Logical Model and we will separately represent these two in the database.

Knowledge about one particular data, i.e. about an attribute's values, is represented by a set F of facts about that data.

$F=\{F1,\ldots,Fm\}$ … (1)

where F1, F2,…, Fm are facts about one particular data. This total knowledge about data consists of knowledge related to the Real World and of knowledge related to the Logical Model.

2.3 The representation of knowledge in the Conceptual Model

As stated in 1.1 one entity (similarly with a relationship) takes the form

(P,E, A1, … , An) … (2)

Where P is the Identifier of the state of the entity (relationship)
E is the Identifier of the entity
A1,…,An are attributes of an entity (relationship)
Each attribute, including E and P, can have different sets of knowledge F associated to them. Thus
P has knowledge Fp1, Fp2,…,Fpi
E has knowledge Fe1, Fe2,…,Fej
A1 has knowledge F11, F12,…,F1k … (3)
.
.
.
An has knowledge Fn1,Fn2,…,Fns

Here, in the Conceptual Model we represent knowledge related to the Real World. Because the Conceptual Model is an interpretation of the Real World, and because the Conceptual Model is mapped to the Logical Model, here there is no knowledge related to data of the Logical Model.

2.4 The representation of knowledge in the Relational Model

In the Relational Model knowledge is represented by columns. An entity (2) can be represented in two ways in the Relational Model.
1. As a relation R, plus additional columns of knowledge, all in one tuple for one member of an entity (relationship) set.

or

2. As a set of the following relations
i) K-relations
(P,A1, F11,…,F1k,L11,…,L1p) … (4)
.
.
.

(P, An, Fn1,…,Fns, Ln1,…,Lnq)

ii) E-relation

(P, E, Fe1,…,Fej, Le1,…,Ler) … (5)

iii) S-relation

(P, E, Fp1,…,Fpi, Lp1,…,Lps) … (6)

Here we have added knowledge related to the Logical Model and it is denoted by Lij. We

also include knowledge from Conceptual Model, here it is denoted by Fij.

Example 2.5

How to create, implement and represent knowledge related to the data in the database.
We will show this with the example of the data Amount from a Savings table. We will suppose that it is possible to make a deposit by sending a check by mail. The table Savings has only two attributes:
Amount and AmountKey. The second attribute is the primary key. The table is simplified for the purpose paying attention to creating knowledge. Usually we create a table like this:

AmountKey	Amount
…	
116	$2000.00
…	

However if we want to implement knowledge on the level of a value of the attribute Amount, then we can, for example add six new columns in this table: Date1, Date2, Operater1, Date3, Date4, Operater2. Table Saving now can look like:

AmountKey	Amount	Date1	Date2	Operator1	Date3	Date4	Operator2
…							
116	$2000.00	5/Oct/04	1	John Mayell	6/Oct/04	1	Paul Jones
117	$2000.00	5/Oct/04	28/Oct/04	Mick Smith	6/Oct/04	29/Oct/04	Lee Evans
118	$2500.00	28/Oct/)4	1	Mick Smith	29/Oct/)4	1	Lee Evans
…							

These six columns are newly created knowledge about the Amount data. We could add more or less than six columns; it depends on which level of knowledge representation we want. The first three columns form a logical whole (unit) and are related to an event in the Real World regarding Amount.
The second three columns are also a logical whole but they are related to a corresponding event in the database. Say that on 5/Oct/04 somebody deposits $2000.00 (by mailing a check) into the Savings account, and John Mayell receives the check and confirm this. The next day, 6/Oct/04, Paul Jones enters these data into the database. On 28/Oct/04 somebody deposits $500 dollars into the account. Therefore, we will “close” the data about $2000.00 and proclaim it not valid. In this case this is done by entering 28/Oct/04 in Date2, that is, on this day in the Real World this data ceased to exist as current (valid). This information was entered into the database the next day 29/Oct/04 by Lee Evans, so this data is no longer valid in the database. The new value of $2500.00 in column Amount is now valid, starting from 29/Oct/04. Here the “1” (or any other specific value) in Date2 means that the data is current in the Real World. If “1” is in Date4 then the data is valid (current) in the Database.

The rows with key values of 116 and 118 were made by Constructor, while the row with the key value of 117 was made by the ClosingConstructor. The values in Date3, Date4, and Operator2 are created by the system. In fact, constructors get these values from the system and store them into the database.
Thus, here we have an example in which knowledge from two sources is associated to one value of the attribute Amount. One of the sources comes from an operator in the Real World and the other one from the system.

We see that the key has two roles: it is the primary key for the table Savings, that is, it makes

differing rows in the table. Secondly, out of the database, i.e. in the Real World, multiple values of this key help us to determine the states, meanings, and knowledge of one entity's instance in the Real World.
Let us assume that now we have a table with more than two attributes. Now we can assign a different number of columns of knowledge to each attribute. These columns can also be different, concerning what they represent.

2.6 Solutions for two significant problems

This technique of implementing knowledge enables us to make solutions for certain problems of a general character. Let's look at following two problems.

Problem 1.
We need a solution such that it can, in a formal way, recognize who created the data and how it was created, (for all its data). We can make this problem more complex. We can create a kind of game with the end user in which the user's goal is to "break" the database solution Let us allow the user to enter in the database what he/she wants and how many times he/she wants, without any limitations. The end users utilize corresponding data entry screens. The solution should recognize all actions which were done and show exactly who did it and how it was done, without the help of programmer. For example, a manager who doesn't know programming can control many aspects of data on a his screen. This is important in case we need to know how a fault was created and who is responsible for it. This is also a solution for malpractice of the database. It also enables us to sell the database solution and maintain it easily. The above Example 2.5 is a good way to solve this problem.

Problem 2.
To solve problems with TransRelational Model.
In this model columns of relation are stored separately. If we want to access more then one attribute of a given record we have to know how to rebuild the "record". To make this problem more general, we can try to solve it using an entity with all its states. This problem has a very simple solution that uses a query, knowledge columns and the E-Relation. We can also display a "record" in many different shapes and meanings. We can also display one entity's instance as all of its states.

3. About Some explanations and notes related to chapters 1 and 2

(Posted January 9, 2006)

I wrote these explanations and notes during the discussion which was about the data model presented in chapters 1 and 2. The discussion took place on September / October 2005 in the user group Comp.Databases.Theory. I believe that these notes can be useful for practical usage in applications and as additional explanations for some topics in this data model.

3.1 Regarding the Conceptual Model I defined it as a product of both objectivity and subjectivity , i.e., the Conceptual Model is related to both the world and to the subject who interprets (a part of) the world. So concepts deal with the real world and with a subject's view of it. Knowledge also is involved in this functionality.

3.2 During the construction of the Conceptual Model two things among others should be considered:
(i) The identifier of an entity (relationship)
(ii) The identifier of the state of an entity (relationship).

Every time an attribute of an entity or relationship is changed we will create a new identifier of the state of an entity or relationship. But this working definition which uses the term "change of the attributes" is not real and it is not precise. We will determine the event from the real world which causes the change of an attribute's value i.e., change of the corresponding state of the entity (relationship). So in the case that an entity (relationship) is changed we will create a new identifier of the state which corresponds to the real world event, which causes this change.
This technique can be very useful if we want to associate knowledge with data. As a database is more complex this model becomes more effective. For example, if a water supply is closed twice for three hours during one day, we would create two new identifiers of state. They correspond to the two mentioned events of this relationship. An appropriate office or department will add necessary documentation about these events. In this example our key is related to real world events and they are externally verifiable by the documentation.
(If we use the usual solutions then we need a compound key with the following "attributes": address of building, address of the water meter (water meter can be in another street), date and time. We can notice that date has three "subfields" - day, month, and year. The time also has three "subfields". An address has five "subfields" and we have two addresses.
Is date an attribute of some entity or relationship? We should be aware that a date is based on the ratio of two different paths of Earth, with some corrections. Time corresponds to the earth's path around itself)

3.3 The knowledge about data is defined as a set of facts about one particular data. Here the data is an attribute's value. An entity or relationship also can have their own facts.
The identifier of the state of an entity (relationship) also has the purpose of connecting knowledge about a data with the data itself and its states. Whenever the knowledge about a data is changed, we will assign a new identifier of the state of an entity or relationship. Example 2.5 shows how to connect the knowledge to corresponding data using the identifier of the state of an entity (relationship).

3.4. The knowledge, the identifier of the state of an entity (relationship) and the identifier of an entity (relationship) are optional. Their application in a particular database project depends on the complexity of the database, its organization and business rules. If, for example, there are no changes of the attributes' values then the identifier of the state of an entity (relationship) is equal to identifier of an entity (relationship). The use of knowledge depends on an application's needs. Sometimes we don't need any knowledge; sometimes the use of knowledge can be complex involving relations and logic among facts.

3.5 Regarding knowledge, I started by defining knowledge about an attribute's value as a set of facts about this data. Here facts are:

(i) Permanent, i.e., they are memorized in a database;
(ii) They are associated with one particular entity's attribute value in the database;
(iii) The facts about one data can be from different sources;
(iv) The facts can be true, false or can be mistakes. The source of the facts can purposely give false facts.
(v) The set F of facts about a data, which is recorded in the database, can be a subset of a set G of the facts, which has some additional facts about the data which are not recorded in the database.

Here we are speaking only about the facts which we have in a database and which are the knowledge about one particular attribute's value from the database. In a similar way we can define knowledge about an entity and relationship.

3.6 Regarding the definition of a key, the following should be satisfied:
(i) Key determines a tuple. This means that the Key value in Relation Model enables all the rows in a table to be different. This property of Key is related to the Relational Model.
(ii) Key provides identification of an entity or relationship. Key does not always directly identify an entity or relationship. In example 3.2 beside Key, we will use the documentation or maybe some additional attributes and knowledge to identify the relationship. Thus, generally speaking Key does not identify an entity (relationship), rather we will say that Key provides an identification of an entity (relationship). This property of Key is related to the real world.
(iii) In the case that we have more states of one entity or relationship, Key enables us to treat this set as one entity or relationship. This property of Key helps to identify the entity in the real world and enables it a better meaning.

4. What conditions must be satisfied in order for relational schema R to be equal to the join of its corresponding binary schemas?

(Posted May 15, 2006)

I will introduce several of well-known terms:
- A key is simple if it consists of a single attribute.
- Two or more attributes are mutually independent if none of them is functionally dependent on any combination of the others.

Now, let R (K, A1, A2,…,An) be a relation schema, where
(a) key K is simple
(b) A1, A2,…, An are the nonkey attributes which are mutually independent
(c) R1 (K, A1), R2 (K, A2), …, Rn (K, An) are the corresponding binary schemas.
We will say that relational schema R is equal to join of its corresponding binary schemas and denote it as
R (K, A1, A2, …,An) = R1 (K, A1) join R2 (K, A2), join … join Rn (K, An)
if and only if every relation that is a legal value for R is equal to the join of its corresponding binary relations.
(d) R has a set of the associated integrity constraints.

4.1 Definition.
Relation schema R (K, A1, A2,…,An) is in Simple Form if R satisfies:
R (K, A1, A2, …,An) = R1 (K, A1) join R2 (K, A2), join … join Rn (K, An)
if and only if
1. Key K is simple
2. A1, A2,… , An are mutually independent.

(In definition 4.1 the relations are joined using common column K. Schema = (Sig, C) where Sig is a schema signature and C is a set of integrity constraint expressed as sentences. Relational schema R is related to one relation from the RM)
Simple Form says what we should to do in order to decompose a relation schema into the join of its binary relation schemas. It says how to effectively make the column-based representation of the relation (i.e. How to do this).
In fact Simple Form suggests that a “good” design starts at the conceptual level. The design of an entity (relationship) should satisfy two conditions:
(a) The construction of the key so that the key is simple.
(b) The attributes of an entity (relationship) should to be mutually independent.
Of course this second condition is natural. The attributes of an entity (relationship) in the real

world are not mutually dependent.
The conditions for Simple Form, that the key is simple and that the attributes are mutually independent, in fact mean that relation schema R is in 2NF, 3NF, BCNF, 5NF (PJ/NF) and that the relation is equal to the join of its binary relations.

4.2 The construction that makes every key simple

Now we have to solve the main problem: how to design the construction of the simple key in order that this construction is universally applicable to every entity (relationship)? This construction is shown in chapters 1 and 2, the key should be constructed according to definition 1.2.

In chapters 1 and 2 all databases are divided into the following two groups:
(A) The databases in which changes to the entities (relationships) are maintained.
Every attribute of every entity (relationship) can be changed and the database solution can maintain all states of the entities (relationships). These databases can be very complex. They are solved by applying the identifier of the state of an entity (relationship), the identifier of an entity (relationship) and according to definition 1.2.
(B) The databases in which changes to the entities (relationships) are not maintained. These databases exist in two forms.
(i) All the entities (relationships) are unchangeable i.e. The entities and relationships are same as the first time they were seen.
(ii) There are the changes of an entity or relationship, but we keep only the most recent updates, so they are same as the last time they were seen.
These (B) databases usually are simple but frequent in business. For these databases we assume that the identifier of the state of an entity (relationship) is equal to the identifier of an entity (relationship). In (B) case we will construct the simple key that is in fact the identifier of the entity (relationship).

4.3 We can apply another approach to the binary relation schemas. We can use the set of the mappings from entity E1 (K, A1,…,An) to binary relational schemas R1 (K, A1), R2 (K, A2), …, Rn (K, An). Now we do not have relational schema for R (K, A1, A2,…,An), rather we have binary relational schemas R1 (K, A1), R2 (K, A2) , …, Rn (K, An) in Relational Model. Here we can derive every relation r, as the join of the corresponding binary relations. We can do this using only one (join) operator and key K. We can do this for all entities and relationships from Conceptual Model.

4.4 It is important to notice that E-relation or S-relation from 2.4 eliminates the transitive functional dependencies. If we decompose a relation R into E-relation and relation R1 (A1, … ,An-1), then R1 (A1, …¸An-1) does not have the transitive dependencies, by its construction. E-relation (from 2.4) has great importance.
It defines the relationship between the identifier of the state of the entity (relationship) and the identifier of the entity (relationship) i.e. It defines the relationship between the identified entity (relationship) and its changes. The E-relation also improves the meaning its intention and extension, involving the changes and knowledge of the related and identified entity (relationship).

4.4 The conclusion

In chapters 1 and 2 it is shown how to effectively decompose arbitrary relation into its binary relations and how to implement this procedure. In chapter 4 it is shown which conditions a relation should to satisfy so that it can be represented as join of its binary relations. These conditions also suggest the design of an entity (relationship) and this conditions are in Simple Form. Chapter 4 shows the decomposition without the knowledge columns. Chapter 5 will consider decomposition that includes the knowledge columns.

5 The attributes, the facts and the states

In 2.2.2 knowledge about one particular data i.e. About one attribute's value was defined as follows:

5.1 Definition.

The knowledge about one attribute's value is based on:
(i) The set of the facts F = {F1,F2, … ,Fij).
(ii) These facts are related to the relationships between one attribute and an entity (relationship).

Now we will define an attribute of an entity (relationship).

5.2 Definition.

The attribute is represented by knowledge about an property. This knowledge is based on:
(i) One fact.
(ii) It is about one entity's property.

5.3 We will call knowledge about attribute - primitive, because:
(i) It is based on one fact.
(ii) It is not derived from anything, but from which others are derived.
(iii) It represents one particular property from the actual world.

We can notice the difference between knowledge about attribute and knowledge about the attribute's value. (in definitions 5.1 and 5.2)
(i) Knowledge about the attribute's value is based on the set of facts, while knowledge about the entity's property is based on only one fact.
(ii) The first definition is about one attribute's value while the second definition is about one entity's property.

5.4 We also have the data about the entity as a whole. These data are common for all attributes of one entity. They are based on the set of the facts.

5.5 Facts from 5.1 are related to one particular attribute. They are not the attributes or the properties of the entity. The others concepts should be treated in the same way. For example, time and date.

5.6 The State of an entity or relationship

The states are derived from the attributes and from knowledge about the attribute's values. We also defined the attributes as a kind of knowledge, which we named primitive. Because of this we have the following definition:

5.7 Definition.

A state of an entity (relationship) is knowledge about this entity (relationship).

5.8 Definition.

A change of a state of an entity (relationship) is a change of the actual knowledge about the corresponding entity (relationship). This change of the state of one entity (relationship) is represented with the identifier of the state of an entity (relationship)

5.9 The decomposition of relation schema R that has the knowledge columns.

Let R (K, A1,F11,…,F1i,…,An,Fn1,…,Fnk) be a relation schema , where key K is simple. A1,…,An, F11,…,Fnk are mutually independent attributes and the corresponding knowledge

columns are also mutually independent by their construction. However Fij are not primitive. Then R = R1 (K, A1, F11,…,F1i) join,…,join Rn (K, An, Fn1,…,Fnk) because K is simple. This join is in the sense of 4/c and 4.1. E-relation or S-relation defined in 2.4 eliminates the possibilities that a key be composite. If we decompose a relation R into E-relation and relation R0, then R0 doesn't have a key that is composite. This is by the construction of R. We can further apply this decomposition to the relations R1,…,Rn - if we need it.

6 The Examples

(Posted November 22, 2007)

The examples are from my posts on user group comp.database.theory where I discussed the ideas presented in this paper.
They are an addition for the explanation and help better understand what was written in the paper.

6.1 Example

In the following example I will use the table Car, slightly altered from example 1.3, and explain the construction of Simple Form, i.e. the construction of binary relations.

Step1

Table Car

CarKey	CarID	Maker	Type	Color	Datefrom	Dateto
…						
23	vin1	Buick	sedan	silver	1.1.2000.	12. 31. 2000
24	vin1	Buick	sedan	blue	1.1.2001	8.1.2001
25	vin1	Buick	sedan	red	8.2. 2001	1.1.2005
26	vin1	Buick	sedan	silver	1.2. 2005	999999
27	vin2	Honda	sedan	silver	3.15.2006	999999
28	vin3	Ford	sedan	black	3.15.2006	999999
…						

999999 – represents the maximal date in the used software and means that the corresponding data is current.
In this table, the columns Datefrom and Dateto are strictly related to one attribute from the column Color. Datefrom and Dateto are not related to the entity Car. Datefrom and Dateto are also not attributes. They are a part of our actual knowledge about one particular attribute from the column Color.

The relation Car also represents knowledge about a particular attribute from the column Color. Knowledge related to one particular attribute is defined in 2.3 and 2.4. Therefore, besides columns which represent the attributes, the relation Car also has columns which represent knowledge about attributes.
This construction of a relation which includes knowledge columns is determined and defined by that which is stated in 1, 2.3 and 2.4.

Step2

Now from the table Car I will construct the following four tables:

Table1

CarKey	CarID
…	
23	vin1
24	vin1
25	vin1
26	vin1
27	vin2
28	vin3
…	

Table2

CarKey	Maker
…	
23	Buick
24	Buick
25	Buick
26	Buick
27	Honda
28	Ford
…	

Table3

CarKey	Type
…	
23	sedan
24	sedan
25	sedan
26	sedan
27	sedan
28	sedan
…	

Table4

CarKey	Color	DateFrom	DateTo
…			
23	silver	1.1.2000	12.31.2000
24	blue	1.1.2001	8.1.2001
25	red	8.2.2001	1.1.2005
26	silver	1.2.2005	999999
27	silver	3.15.2006	999999
28	black	3.15.2006	999999
…			

Basically, here in Step2 I constructed four “column-based” or “attribute-based” relations from the relation represented by the table Car in Step1. The first three tables in fact have one column and key.
Table4, in addition, has knowledge about the property Color which is represented by two columns (Datefrom and Dateto).
One can add some other “knowledge-columns” related to Color. Now in Step2 we have the relation Car from table1 represented in Simple Form.

6.1.1 How a Relation Should Be Constructed in the General Case

Now, considering the above example, we can define the construction of a relation in the process of the design of a database. The following description is for complex databases whose entities and corresponding relations are defined in 2.3 and 2.4.

1. First, we will construct the entity. The construction of entity / relationship should have:
 (i) A simple key
 (ii) Mutually independent attributes
 (iii) The attributes only from one entity / relationship, i.e. only its own attributes.
 This condition can be derived from an entity’s definition and from 5.2.

2. Now that we have the entity, we will directly construct the corresponding binary relations. The construction of the binary relations should be as it is done in Step2 in 6.1 example. So we do not need Step1. We can get the relation from Step1 using a “join” of the corresponding binary relations. This construction of binary relations is based on 2.3, 2.4 and on the set of mappings mentioned in 4.3. These mappings are between set of entitis and the corresponding set of the binary relations. This set of mappings gives the links between an entity represented as the set of attributes i.e. the entity as the whole, on one side and the binary relations between one entity’s attribute and the entity’s identifier on the other side.

Briefly in 4.3 there are links among an entity, the binary relations, the entity's, attributes, the states and the identifiers.
Now, when we have binary relations and their entities, we can define mapping from binary relations to the entity's instances. This mapping is determined by key K.

We can notice that this design is based on the immediate construction of binary relations rather than on the decomposition of a relation.

6.1.2 The Identifiers

In example 6.1 CarID is an identifier of the entity Car, and it is a VIN number. CarKey is the identifier of the state of the entity Car.

- The identifier of an entity or relationship identifies the abstraction: entity, or the abstraction: relationship. This identifier should also be used to identify the corresponding real world object or relationship which we abstract.

- Similar to this is the identifier of the state of an entity or relationship – this identifier identifies the abstraction: state of an entity, or the abstraction: state of the relationship. The identifier of the state should also be used to identify the state of the corresponding object (or relationship) from the real world.

- The group of identifiers of the state of an entity or relationship can be used to provide better identification and meaning to the entity or relationship. This is explained in 3.6.

The identifier of the state of an entity or relationship is not created arbitrarily. It is always initiated by a real world event, as is defined in 3.2. This connection with a real world event enables a company great possibility to create its own technology. For instance, in the above example 6.1, a company can establish additional paper documentation for any painting of a car, with customer signed agreement and many other options – all this associated with the identifier of the state of the entity Car. The identifier of the state of an entity or relationship always goes with the identifier of an entity or relationship. In the above example the identifier 26 is tied with VIN1, so it is not arbitrary at all.

Obviously, the different states correspond to the different identifiers of the state, and vice versa. There are no two same states of one entity / relationship.

6. 2 Example

In the following example I will consider more then two knowledge-columns related to property Color.
Here I modified Table4 from 6.1 and added the six knowledge-column related to the property Color: Datefrom1, Dateto1, Datefrom2, Dateto2, Operator1, Operator2. So Table4 can have for example the following data:

Table 4

CarKey	Color	Datefrom1	Dateto1	Operator1	Datefrom2	Dateto2	Operator2
…							
23	silver	1.1.2000	999999	John	1.2.2000	999999	Mike
24	silver	1.1.2000	12.31.2000	Paul	1.2.2000	1.2.2001	Bill
25	blue	1.8.2001	999999	Bill	1.8.2001	999999	Bill

…

These six added columns have same role as it is in example 2.5.
However here in example 6.2. is shown link between whole relation Car and the relation represented with Table4. Regarding 2.3 and 2.4 we can associate arbitrary number of knowledge-columns to every binary relation, including E-relation.
Off course when we add knowledge-columns to a binary relation, we transform this relation to n-ary relation. However this n-ary relation still has only one attribute and Simple key.

6.2.1 The construction of information or data

Problem1 which is presented in 2.6 sets the following question:
How to recognize in formal way who or which procedure created the information.
Although there are the different tries to solve this problem, example 6.2 shows that the binary relation in combination with knowledge-column can be powerful solution for this problem. In 6.2 we have the construction where one information is created by two sources.

Generally speaking the binary relation enables the construction of information.
So we don't have a data, rather we have the construction of a data and associated knowledge about this construction of data.

6.3 About some terms in this paper
(posted December 4, 2007)

In this paper special attention is devoted to constructions which enable us to solve and design the databases in which the attributes of the entities (or relationship) are changeable or knowledge about the entities (or relationship) is changeable.
Basic constructions are entities (relationships) and states of entities (relationships). An entity (relationship) has its states (an entity possesses states). Formally it can be said that the identifier of the entity (or relationship) determines one set of its identifiers of state. Of course the same identifier of an entity (relationship) corresponds to all its identifiers of state. The entity is defined by its attributes like in the E/R model. In literature the same term is in used for both the real world entity and for the entity's abstraction. Besides these two, I will use the term entity also for abstract entities which don't have the corresponding real world entities. It will be always defined which form of the term is applied.

The identifier of an entity (relationship) can identify the following:
1) an entity in the real world or
2) an abstraction of the entity or
3) both, i.e. a real world entity and its abstraction
It is also possible that an entity doesn't have an identifier of entity.

Database design determines which of these cases we will apply. Of course we should be able to identify real world entities, abstractions or both. These three possibilities enable many combinations of database design in practice.

7 The Semantic Model and Database Design

7.1. Identifying the Plurality

In the process of identifying a plurality there are two constructions. I will first consider the construction of the entities:

Construction1 (or first step)

(i) We will construct an entity which is distinguishable from any other entity. To do this we will use the corresponding entities attributes. If the entities attributes can't establish the unique entity then we will add to the entity a new attribute. The new attribute will later become an identifier of the entity. This construction is done first as an abstraction that is represented by a scheme of the corresponding entity set.

(ii) For more complex databases, instead of an entity we will use an entity's state. The state of an entity is uniquely determined by the entity's attributes and by knowledge about the entity attributes.

Constructuin2 (or second step)

Here we will construct identifiers.

Regarding Construction1:

(i) For each unique entity we will construct an identifier of the entity

(ii) For each unique state of the entity we will construct an identifier of the state of the entity.

Similar to the above is the construction of relationships.

Now we will define the following principles:

The Principle of Distinction

This is construction of the unique entity OR the construction of the unique state of the entity as described in Construction1

Principle of Identification

This is the construction of the unique identifiers as described in Construction 2

Example:

During a phone call, we would never say the following: "May I speak with the 5 foot tall and has blue eyes and has brown hair and …"

Rather, we will say: "May I speak with John?"

The aforementioned identifiers can exist in the real world and they are simultaneously the identifiers of our abstraction. In real life we often make mistakes by not distinguishing between the aforementioned two principles.

The state of an entity has two identifiers: the identifier of entity and the identifier of the state of the entity.