WHAT A DATABASE *REALLY* IS: PREDICATES AND PROPOSITIONS

An open letter to Open University database students

Hugh Darwen

ABSTRACT

Despite the name, a database is best thought of as a repository not just for data, but rather for *facts*—that is, for *true propositions*. The article that follows explains this remark in lay terms and begins to explore some of its many implications.

PREFACE

This article is an essay by Hugh Darwen, addressed to the students he tutors on a database course offered by the Open University in the UK. When he showed it to me for review, I was very pleased, for I had long felt such an article was sorely needed; indeed, I had thought of trying to write one myself. Hugh's explanation for writing it intrigued me:

> "Almost every year, about halfway through the course, some student or other asks, in a manner that suggests the question has been burning: *What's a predicate?* I'm always so pleased that anybody has even realised this might be an important question that I take pains over my careful but *ad hoc* answer. And then I think: Why didn't we get this out of the way right at the very beginning? This year, I'm giving it a try."

Well, I hope it works for Hugh's students, for I sincerely believe that if only more people—especially DBMS implementers—thought this way about databases, then we might stand a chance of seeing the emergence of respectable database systems that people could enjoy and not have to fight with.

—Chris Date

AUTHOR'S NOTE

The essay was originally written in 1997 in connection with the Open University's course M357, *Data Models and Databases,* that was M358's predecessor. This 2005 version is identical to the original apart from some obvious minor revisions to bring it up to date (such as in the first paragraph of the following introduction).

INTRODUCTION

Hello. Let me introduce myself. My name is Hugh Darwen, and I am the tutor (staff number 44525) in region 4 for certain students on course M358, which is entitled *Relational Databases*.

The foregoing paragraph contains three sentences, and those sentences are different in kind as well as in content:

- The first, "Hello," is a mere signal, establishing contact.

- The second is in the form of an imperative, demanding something of you, the reader (though of course we all know that it's really just a common courtesy in this particular instance).
- The third is a plain statement of fact.

Each of these three kinds of sentence has an analogue in communications between people and computers. We often have to do something special to *establish contact* with a computer; we often use the imperative style to give *commands* to the computer; and sometimes such a command includes a *statement of fact* that we want the computer to remember (because that fact is both true and relevant), or forget (because it is now either false or irrelevant). I focus for the rest of this article on this "statement of fact" kind of sentence.

STATEMENTS OF FACT

How do we distinguish statements of fact from other kinds of sentences, such as greetings, imperatives, and questions? Well, here's another example that might help:

> I am writing this article in my study at home in Warwickshire on February 9th, 1997.

Now, you cannot tell whether what I've just told you is a true statement of fact or a false one, but you do know, from its very form, that it *is* either true or false. By contrast, we cannot say of utterances such as "Hello," "Let me introduce myself," or "What's the time?" that they must be either true or false.

By the way, you will observe that I do entertain the notion that a "statement of fact" might be false. If you think that the very term *statement of fact* connotes undeniable truth, please don't worry too much—I could have chosen an alternative term, such as *assertion* or *declaration*. As always, the concept is more important than the terminology, and sometimes it's difficult to choose the most appropriate term in everyday speech to match a concept one is trying to communicate, especially when that concept is a very precise one. I'll continue to use *statement of fact* in this article, but I'm about to introduce an alternative term also, one that is conventionally used to mean precisely the concept I'm trying to convey. That term is **proposition**.

PROPOSITIONS

The term *proposition* is the one logicians use for the "statement of fact" concept. Aristotle (384-322 BC) understood the importance of propositions, and he worked out a formal system of reasoning whereby, from an assumption of the truth of certain given propositions, the truth of certain other derived propositions can be concluded. The given propositions are called **axioms** and the method of reasoning is called **logic**. The axioms and the *derived* propositions concluded from those axioms are collectively called **theorems**. For example, given certain propositions already discussed in this article, you could use logic to obtain the following *logical consequence* (i.e., derived proposition):

> The home of the Open University tutor identified by staff number 44525 is in Warwickshire.

Further, if the given propositions are in fact true, then you can be sure the logical consequence (the conclusion) is true, too.

Note: Why might you bother to obtain a derived proposition (or logical consequence) such as the one just shown? I suggest that you would be very unlikely to do so unless somebody asked you a question to which that derived proposition would be an appropriate answer.

A DATABASE IS A SET OF TRUE PROPOSITIONS

It's useful to view a database as representing a set of propositions (assumed to be true ones) concerning some enterprise of which the database is supposed to provide some kind of record (I'm using the term *record* here in its generic sense, not the special sense in which it's often used in computer contexts—in database contexts in particular). If we do take that view, there are some important questions that immediately arise:

- How do we choose which propositions should be stated to form the record of our enterprise?
- In what form should those propositions be stated?
- How can we instruct the computer to remember or forget a given proposition or set of propositions?
- Can we get the computer to prevent us from stating propositions that are ridiculous or contradictory? A ridiculous proposition might be one that states that a certain person is 200 years old; contradictory ones might state that a person is both male and female, or that a tutor in region 4 is—contrary to Open University rules—tutoring a student in region 5.
- In what form can we present a question (or "query") to the computer, the response to which would be a proposition or a set of propositions derived by logic from a given database? And in what form should we expect to find that response?

Course M358 answers these questions. Indeed, there's little in that course that's not related to at least one of them, though you might note that I didn't bother to mention certain subsidiary matters, such as who's allowed to access a database, how the computer checks authorizations, how databases might be protected from accidental loss or damage, and so on.

PREDICATES

There's a word in my title that I haven't used yet, and I come to it now: **predicate**. The concept of predicates is very important, for an understanding of them could underpin everything you will be asked to learn on course M358.

Consider two things:

- First, logicians from Aristotle onwards found that reasoning based just on the notion of propositions had certain severe limitations, which they eventually overcame by studying certain *generalized forms* of such propositions. They found that, when several propositions are of the same generalized form, various impressive shortcuts could be taken by reasoning in terms of those general forms instead of in terms of individual propositions *per se*.
- Second, commercial databases can contain billions of propositions; if propositions of the same general form cannot somehow be lumped together, such databases will surely be unmanageable and unusable.

Logicians use the term *predicate* for the "general form" in question. (Some authorities use the alternative term **propositional function**, which does have a certain attraction to it for reasons too deep for the present article.) It's predicates that have made databases, database management, and database queries tractable to computerization. Consider once again the proposition from my opening paragraph, which I'll now restate in a very slightly different way:

> Hugh Darwen is the name of a tutor (staff number 44525) in region 4 for certain students on course M358, which is entitled *Relational Databases*.

It's easy to see that this statement has a certain form that could be common to a whole set of propositions one might wish to state in some record of the enterprise called the Open University. For example, we could replace the course number M358 by M355, thereby obtaining a proposition which makes the same kind of sense as the original one, and might even be true. (In fact it's not true, for two reasons: First, I'm not a tutor for any students on course M355; second, course M355 is not entitled *Relational Databases*.)

Here's what is probably the most general form of the original proposition that we might all agree on:

> ... is the name of a tutor (staff number ...) in region ... for certain students on course ... , which is entitled ... .

And *that's* a predicate!

Note: In the database literature you'll sometimes find the term *predicate* used in some more restrictive sense; you might even find it used in some subtly different sense. For example, quite often it's used where the term *condition* would be appropriate. I prefer to keep *predicate* for the sense agreed by logicians.

Here then are some important points to note and questions to be asked:

1. The predicate as shown can be broken down in various ways into smaller pieces, each of which is a predicate in turn. For example, "... is the name of a tutor" is a predicate, and so are "... is the name of a tutor (staff number ...)" and "course ... is entitled ..." (and so on).
2. It might be useful to give the predicate a name, such as TUTOR_INFO. Such names are used a great deal in database designs, as you will discover in course M358. Indeed, many of the names used in databases are really predicate names, even if they aren't often referred to as such.
3. The holes marked by "..." are known as **placeholders**. It might be useful to give them names, too. In fact, predicates are often written using such names. For example:

   > TUTOR is the name of a tutor (staff number STAFF#) in region REGION# for certain students on course COURSE#, which is entitled TITLE.

4. Notice that the placeholder names are often accompanied by text indicating the kind of thing they stand for: "TUTOR is the name," "staff number STAFF#," "region REGION#," and so on. Staff number and name here are both **common nouns**, standing for anything or everything of the kind indicated. Region is perhaps a little sloppy, considering that what follows is really a region *number,* not a region *per se,* but "the region identified by region number REGION#" seemed just a little heavy-handed for my present purpose.

   Now, if those "indicators of kind" are common nouns, then the accompanying placeholders themselves can be thought of as **pronouns**. For example, consider the statement "He is her father." This statement contains two pronouns, *he* and *her; he* stands for some unspecified person who is the father of some other unspecified person, *her*. Now, in normal discourse the context would provide referents for these pronouns, and we would know precisely who is being asserted to be whose father. Because there are no referents here, we can't tell which people they actually stand for. However, imagine a context in which the referents are Tom and Jane, respectively. Then we will understand that we need to substitute Tom and Jane for the pronouns to obtain "Tom is Jane's father." In a like manner, when we substitute an appropriate name or *proper noun* for each placeholder in a predicate, we obtain a *proposition*. For example, if we substitute Hugh Darwen for TUTOR and 44525 for STAFF# (and so on) in the TUTOR_INFO predicate, we obtain once again the proposition:

   > Hugh Darwen is the name of a tutor (staff number 44525) in region 4 for certain students on course M358, which is entitled *Relational Databases*.

In general, if there are *n* placeholders and we substitute a proper noun for one of them, we obtain a predicate with *n*-1 placeholders (and so on). When there are no placeholders left at all, the predicate degenerates to a pure proposition—it is now true or false, unequivocally.

5. The presence of at least one placeholder in a predicate means it cannot be the kind of statement of which we can say categorically that "it is true" or "it is false." Although we can make propositions out of predicates, a predicate is not, in general, a proposition (the exception is the degenerate case of a predicate with no placeholders at all).
6. It's important to agree on what proper nouns, in general, are appropriate for each placeholder. For example, we might not wish to form propositions such as:

   3.14159 is the name of a tutor (staff number Camembert) in region CV35 7AN for certain students on course Aintree, which is entitled Jurassic Park.

   Note: I hope you'll bear with my suggestion to treat, e.g., 3.14159 and CV35 7AN as proper nouns—logically, they are.

   Indeed, the proper nouns agreed upon will almost certainly bear a close relationship to the kind of thing the placeholder represents, often indicated by the presence of a common noun in the predicate (as discussed in point 4).
7. The connectives *and* and *or* can be used with predicates, to make longer predicates, just as they can be used with propositions to make longer propositions. *Not* can be used, too.
8. Would it actually be a good idea to use the suggested form, TUTOR_INFO, to hold information about tutors in the Open University database? Can you think of any problems that might arise if we did?
9. The form of some predicate might also be used to formulate a question ("query") to be presented to the computer. The answer to that question would be the set of all propositions that (a) can be formed by substitution of proper nouns for placeholders in the form and (b) can be shown to be true. Of course, if the database itself uses the form of that same predicate to hold the original given statements of fact, then "showing to be true" will be, for the computer, a trivial task of mere regurgitation.

QUANTIFICATION

In point 4 in the previous section I showed how to make a proposition out of a predicate by substitution of a proper noun for each placeholder. However, there's another way of disposing of a placeholder; it goes by the name of **quantification**, meaning "saying how many." Consider for example the simple predicate:

ARTIST painted a portrait of PERSON.

Instead of just substituting appropriate names for both ARTIST and PERSON (as in, e.g., "Holbein painted a portrait of Henry VIII"), I can obtain propositions by saying *how many* artists painted a portrait of a certain person, or *how many* people had their portraits painted by a certain artist.

Now, there's one particular form of quantification that is both fundamental and very common: It's called **existential quantification**, and it involves replacing the placeholder in question by a phrase involving something like "at least one" or "some" or "there exists." For example, the following are all propositions that can be obtained from the predicate "ARTIST painted a portrait of PERSON":

Holbein painted a portrait of some person.

Some artist painted a portrait of Henry VIII.

Some artist painted a portrait of some person.

Aristotle studied propositions of a certain form that includes quantifiers. He realized that if propositions of the form "*a* is *x*" and "*a* is not *x*" are interesting, then we might also want to

consider the truth of "Some *a* is *x*," "Every *a* is *x*," and "No *a* is *x*." "Some" (as we have seen) is the existential quantifier. "Every" is what we now call the **universal quantifier**. "No" is a *negated* form of the existential quantifier, for "No *a* is *x*" clearly means the same as "It is not the case that some *a* is *x*." However, while these observations might justify a claim that Aristotle started the study of predicates, that study did not come to fruition until the late 19th century, with the contributions of Frege, Boole, Peirce, and others.

CONCLUDING REMARKS

The M358 course material does use the term *predicate* a little, but not a lot, for there are other ways of saying what's going on in databases—ways that are often more appealing, in their own special contexts, than always talking in terms of predicates. However, if you have difficulty understanding those other ways, try referring them back to the predicates and simple logic that really underpin the whole subject.

To end, here are three observations to give you a feel for that universal underpinning:

- In *Blocks I* and *IV* of the course we study a method of analysis known as "Entity-Relationship modelling", this activity being a common preliminary step in database design. Deciding what entity types to describe and what types of relationship might exist among instances of those entity types is really just deciding what kinds of statements of fact one would like to use to make a formal record of the enterprise being modelled. Kinds of statements of fact, as we have seen, are otherwise known as predicates. The concept known as **attribute** in this modelling method is just *placeholder* by another name.
- In *Block II* we study a theory called "The Relational Model of Data" and a simple computer language called RAS based on that theory. We will discover how the ideas of predicate names and placeholder names are used in the formulation of such languages, and we will discover how queries can be presented to the computer by constructing predicates from predicates, using substitution, quantification, and the "connectives" *and, or,* and *not*. Incidentally, in this theory we will discover that the mathematical term *relation* is used to refer collectively both to the general form of a predicate and to the set of true propositions that can be formed from it.
- RAS is not a commercially used language — it has been designed especially for the Open University for tutorial purposes. In *Block III* we will study the industry's most widely accepted attempt to implement the theory we learned in Block II, a database language called SQL. SQL has become so prevalent since its first commercial appearance in 1979 as to have been characterized by one rather flamboyant authority as "intergalactic dataspeak." Alas, the industry's most widely accepted attempt is not a very good one, as we shall see, but a good understanding of predicates will help us to use SQL wisely in spite of its traps and shortcomings.

ACKNOWLEDGMENTS

Thanks to Adrian Larner, who first taught me to think this way, and for the observation that a database is not really a model of an enterprise, but a record or *account* of it; thanks also to Mike Newton of the Open University, Chris Date, and Adrian Larner again, for reviews of early drafts of this article and valuable suggestions. Further thanks to Chris Date for the preface, which he wrote for the essay's appearance in the book mentioned below.

(Originally published in *Database Programming & Design 11,* No. 7, July 1998. Also appears in *Relational Database Writings 1994-1997* by C.J. Date, as Part II, Chapter 1. Addison-Wesley Longman Limited 1998. ISBN: 0-201-39814-1.)

