# Hypothetical Reasoning Over Databases



**Lei Dang**
*University of Manchester, UK*

**Suzanne M. Embury**
*University of Manchester, UK*

## INTRODUCTION

In recent years, the term *agile organisation* has been coined to denote an organisation which is able to change its working practices quickly in order to adapt to changing external pressures or to take advantage of new market opportunities that may arise (Perez-Bustamante, 1999). A key element of such agility is fast and reliable decision making; that is, the ability to determine *what* the new organisational behaviour shall be, as well as *how* the changes to it should be affected. In general, an organisation will be faced with several possible strategies for change, each of which has competing strengths and weaknesses. Management must then evaluate and compare each of these to determine which has the best forecasted outcome. This process is often referred to as "what if?" analysis (WiA), since it tries to answer the question: what will the outcome be if we adopt change X? (Codd, Codd, & Salley, 1993)

## BACKGROUND

Perhaps the most familiar WiA tool is the spreadsheet, in which the user can enter different values for the parameters of a mathematical model and observe the changing results. For example, a manager might experiment with different discount rates on forecasted sales, to see how profit levels are affected. The applicability of this technique, however, is limited to situations that can be accurately characterised by a simple set of mathematical equations. A far more common context for such decision making is provided by a company database (or data warehouse), which records historical information about the performance of the organisation.

In order to perform WiA in this context, we would like to be able to update the database, to reflect the proposed changes in organisational behaviour, and then to query it, in order to determine the properties of the new state. Since we obviously cannot risk modifying the live data itself in this way, we need to find alternative methods of making these hypothetical changes to data and querying their composite effect. If the proposed change is sufficiently simple, for example, both the change and the query can be made inside a transaction that is aborted once the query result has been obtained. This will preserve the integrity of the live data, but it can have a performance impact on other business processing and it does not allow easy comparison of different scenarios.

An alternative approach that is commonly adopted is to make a copy of the data involved in the WiA scenario and to make the necessary changes to that, instead of to the live data. This is a much better solution if the change or the queries are large/complex, and it also allows a more *ad hoc* style of analysis, in which changes and queries are interleaved, and determined based on the results of the previous set of queries. If data sets are large, however, or if many different strategies are to be compared, then the disk space required can make this approach impractical. The effect of the data extraction on the performance and availability of the live database must also be considered.

Researchers have therefore considered the possibility of embedding facilities for this form of WiA within the DBMS itself, to provide the capability to perform *hypothetical reasoning* over the contents of the database without impacting the integrity or performance of the database as a whole. Such facilities must support both *hypothetical updates* (HUs), which create a pseudostate (called a *hypothetical state*, or HS) based upon the real database state but modified according to the updates requested, and hypothetical queries, which derive properties from hypothetical states.

A range of possibilities exist for implementing such hypothetical reasoning facilities, depending on how the updates and queries are specified, and to what degree the hypothetical states are materialised or not. In general, the proposals made in the literature so far can be divided into two broad categories: those which support extensional representations of HUs and those which support intensional representations of HUs. The former approach is based around the notion of a hypothetical relation (HR), while the second is focussed on the provision of hypothetical querying facilities. In the remainder of this article, we will present an overview of the principal contributions

in both categories, followed by a brief discussion of some additional approaches that might be investigated in the future.

## HYPOTHETICAL RELATIONS

The earliest proposal for hypothetical reasoning facilities for databases was based upon the notion of a HR (Stonebraker & Keller, 1980). An HR appears to the query system to be a normal relation, but its contents are in fact defined in terms of the contents of another relation (typically an actual stored relation), with some user-specified additions and deletions. The HR itself does not store data, but instead contains the additions and deletions that have been hypothetically made to the underlying database. Thus, if $Add_R$ is the set of tuples that are to be hypothetically added to a relation $R$, and $Del_R$ is the set of tuples that are to be hypothetically deleted from $R$, then the tuples present in the new hypothetical version of $R$ are exactly those given by the query:

($R$ UNION $Add_R$) DIFFERENCE $Del_R$

Further queries can be posed against the HR (or a mixture of as many hypothetical and actual relations as are required) in order to understand the ramifications of a particular change in working practices.

Various methods of implementing HRs have been proposed, each with different performance and disk space characteristics. However, the basic concept is to store the details of the HUs in a special auxiliary relation (or relations), for use later in reconstructing the HR itself. In the earliest proposals (Stonebraker & Keller, 1980), a special relation called a *differential file* (DF) is created whenever a new HR is requested by the user (i.e. by making a HU to another relation). The columns of the DF are exactly the same as those of the relation that is being hypothetically updated, with the addition of a special *TupleID* column, which is used to match deleted tuples in the DF with their counterparts in the underlying relation, and a column indicating whether the tuple is an addition to or a deletion from the main relation.

The simplest method of querying an HR represented as a DF is to use a query rewriting approach (Stonebraker, 1981), in which any query involving HRs is transformed into an equivalent query that references only actual relations and DFs. This is done by replacing any references to an HR with a query similar to that given above to describe the contents of an HR. However, this approach (in conjunction with this simple DF design) has several disadvantages. The DF tracks all the updates made to a particular HR, and it may therefore be the case that a tuple in the underlying (actual) relation may appear several times in the DF, if (for example) it has been inserted and deleted many times. This means that the DF can grow much larger than the underlying relation itself, if HUs are numerous and frequent. A second disadvantage is that tuples which are deleted and then re-inserted into the same HR will not be picked up by the query mechanism, because all hypothetical deletions are always enacted after all hypothetical insertions. A later revision of the algorithm was proposed, which uses timestamps on tuples in the DF in order to allow the most recent changes to be visible in the HR (Woodfill & Stonebraker, 1983).

Later, these ideas were developed and extended to produce a more sophisticated form of HR, called an *Independently Updated View* (IUV) (Ramirez, Kulkarni, & Moser, 1991). IUVs are virtual relations defined over a conventional relational view (Kulkarni & Ramirez, 1997). As with HRs, HUs made to an IUV will not affect its *parental views* (i.e. the views or relations from which the IUV is built). The storage mechanism for HUs to IUVs is similar to the DF approach, except that IUVs can be materialised for higher querying performance, or stored in a separate auxiliary relation (called the *differential table*, or DT) for greater flexibility and speed of update. A further difference is that only accumulated net effects of updates are stored in the DT, so that IUVs require much less storage than HRs in the case of frequent repeated updates.

An additional problem which is present in HRs but resolved in IUVs is that of overlapping updates. These occur when the parent relations of an IUV (or HR) are updated normally (i.e. not through the hypothetical update facility). For example, suppose we have hypothetically added a new employee called Fred to a *Workers* relation, and after this the relation is actually updated to contain an employee called Fred. Which of the new Fred tuples should the HR contain? In general, the answer to this question will be application dependent. In some cases, we will wish to give priority to the real updates, while in other cases we will wish to block out any further changes and concentrate only on the HUs. In order to allow this, the designers of the IUV approach provide two options for hypothetical querying, an IUV-prioritised approach, which ignores changes to the parental views, and a Parental-View-prioritised approach, which forces a re-evaluation of the IUV in the light of the updates to the parental view.

## HYPOTHETICAL QUERIES

The HR approach concentrates on the problems of representing and processing hypothetical updates in the form of sets of tuples to be added or deleted. However, many forms of WiA are more conveniently expressed using an