RELATIONAL DATABASES: Concepts, Selection and Implementation

RELATIONAL DATABASES: Concepts, Selection and Implementation

A. J. Page

SIGMA PRESS – Wilmslow, United Kingdom

First published in 1990 by

Sigma Press, 1 South Oak Lane, Wilmslow, Cheshire SK9 6AR, England.

British Library Cataloguing in Publication Data

A CIP catalogue record for this book is available from the British Library.

ISBN: 1-85058-140-1

Typesetting and design by

Sigma Hi-Tech Services Ltd

Printed in Malta by

Interprint Ltd.

Distributed by

John Wiley & Sons Ltd., Baffins Lane, Chichester, West Sussex, England.

Preface

I remember vividly, when sitting a board interview with the Civil Service many years ago, being asked to describe a Relational Database Of course at that time I didn't know how to reply to the question, so it is a little ironic that I now come to write this book, which above all seeks to answer just this question. Between that time and now, I have gained extensive experience of three Relational products, namely Ingres, Oracle and perhaps a less known, but equally significant product, the DBC/1012 Database Computer and have often presented or demonstrated these products to potential purchasers. It was in fact these presentations and conversations with people interested in investing in relational technology, that made me decide to write this book. It seemed to me to that, although great amounts of time, effort and care were taken in evaluating different relational products, very rarely was I convinced that the purchasers either knew the complete story or were basing decisions on reality rather than theory.

The book is thus born out of my own experience and focuses on three themes. Firstly on relational concepts including both some history and possible future developments. Secondly it details some of the more important facilities that a Relational Database Management System (RDBMS) should provide and, lastly, real-life examples and viewpoints illustrate the practicalities of using an RDBMS.

There are two major parts. Part 1 seeks to define both what databases are and how they are used. It describes the differences between Relational Databases and their predecessors, and details the relational model by describing its close affinity with data normalisation techniques. Chapter 5 describes the different the various database architectures used to support current products, and in this I rely heavily on my experiences with Oracle, Ingres and the Teradata DBC/1012. I realise that in time my descriptions of these products will become out of date, but this is unavoidable.

A logical step from database architectures takes us into a chapter devoted to On-Line Transaction Processing and to complete Part 1, I have illustrated the 12 rules that were formulated by E. F. Codd to define a Relational Database.

Part 2 is devoted to describing some of the more common and important facilities that should be provided in an (RDBMS). I have used the three aforementioned products to illustrate these facilities, and again I lay myself open to the possibility (indeed certainty), that my descriptions will become out of date. I have not attempted to cover all the facilities that should be on offer, and perhaps the two most noteworthy that I have omitted are the use of Structured Query Language (SQL) from inside Third Generation Languages (embedded SQL), and the close affinity of relational systems with Fourth Generation Languages (4GL). Both of these subjects deserved a greater degree of discussion than I can manage in a book of this nature.

Topics that are covered in Part 2 include a discussion on SQL (Chapter 8) followed by Chapters 9 and 10 covering the topics of indexing and optimisation. Following these performance-oriented chapters, the next two are concerned with transaction management and detail both the concept of a transaction and the required locking mechanisms that can make them work. Next I have included a chapter covering security and follow this with a description of integrity constraints. Journalling, Auditing and Dictionaries are each dealt with in separate chapters. Finally, I have brought the book to a close with a chapter on the subject of the Distributed Relational Database which may be of use to people involved in the evaluation of such architectures.

I have started each chapter with an indicator of its subject matter, and have tried at the end of each, to summarise the major points to be found within the chapter and points that should be considered in more depth when reviewing that particular subject.

Finally, let me say that writing this book was a very educational and enjoyable experience for me. I thank all who helped me and those who simple put up with me during those long hours in front of a PC, and I hope that all who read this book will learn something, no matter how small. I also thank the people in Oracle, Ingres and Teradata who have helped me directly or otherwise, in my understanding of their products – I hope I have done them justice in this book.

Jon Page

Contents

Introduction

The world of the programmer today stands at something of a crossroad. Rarely before has there been such a wide variety of specialisms awaiting mastery and never before have the rewards for those that have mastered them been so disproportionately scaled against those that have not. Changes occur every day and the industry has gone through many times of uncertainty. There have been battles between the Mainframe diehards and the micro computer innovators. Mini computers have often been used to avoid decision making and many of our large blue chip industries have changed their strategic hardware and software policies so often that lack of decision making becomes a real productivity problem.

Now however, there are even more choices to be made. In addition to those that previously demanded our attention, we must now consider Relational Database software, Fourth Generation Languages and Expert Systems to name just a few. We must also understand that these items, coupled with hardware, are all likely to be interrelated by some overall company requirement. What is the role of these Relational Databases, Fourth Generation Languages and Expert Systems and is technology at last advancing at a faster rate than we can assimilate? These are all valid issues and in this book I have attempted to supply some of the answers regarding the use of Relational Databases.

From the view point of the DP professional there is little benefit in today's climate of either standing the middle ground or of being the 'jack of all trades'. The third generation of computer languages, and those that swear by them, find themselves in the middle of the road whereas yesterday they not only built the road but also decided in what direction it was to go. In the blink of an eye, instead of three generations of programming language there are now at least five, and the people who invested everything in their third generation expertise, find themselves severely out in the cold. These people must now learn new skills to survive, or perhaps revert to older ones long forgotten.

Not to be outdone by the great progress being made in the development of programming languages, advance in the design of data storage methods has not been slow, and to keep a certain symmetry, I have tried to define five generations of database structure to match the well-documented five generations of programming language. Whilst there is certainly some relationship in the way the languages and databases have grown up, this relationship is not always clear, as you will see.

By way of introducing the subject of this book, I would like to pause for a while and examine in broad terms the software evolution that we have witnessed in the last few decades.

It is common, and in fact I already have done so, to refer to the evolution of computer languages in terms of 'generations', the first of which, not surprisingly enough, called the 'first generation'. This refers back to the days when all computer instructions were coded; and often toggled into the machine in binary. It was indeed a very primitive and labour intensive activity, and one certainly prone to many errors. Thankfully, and very quickly, it was discovered to be a lot easier to devise codes for these long-winded binary instructions and to feed these codes to the machine to be assembled into the binary 'machine instructions' required. This represented the jump from first to second generation programming languages.

It was a large and revolutionary step. Instead of pandering to the computer and having to spoon feed it in its own 'yes or no' language, the programmer could write in codes and symbols more understandable to himself and others, and let a software tool, the assembler, convert these instructions into the explicit bit configurations understood by the hardware. As the instructions became easier to write and understand so productivity increased, a factor which benefited further from the fact that single 'assembler language' statements could represent many instructions at the 'machine code' level.

Just as assembler programs could thus become more sophisticated and procedural; a progression in the idea of such programs manipulating data dawned, and although at this stage such data tended to be simple in structure and low in volume, rapid advancement was awaiting round the corner.

Let me take this opportunity of stating that, for the purpose of this introduction, my idea of a database is simply that of a store of data, in any format and of any volume. At the lowest level of programming, databases took the form of simple areas of storage, defined as variables or arrays within programs. The contents of such 'databases' were 'hard coded' into the program, or otherwise created, destroyed or manipulated without reference to any datastore outside the context of that program. As such, data existed as combinations of lists, arrays, variables and literals and even at this stage the limitations of such data storage techniques were easily recognisable; a fact that led quickly to the separation of the data (database) and code (programs) as two separate entities. Indeed, this simple action represented a major advancement in the concept of data storage and management, which took the sophistication of the database itself from its first to its second generation.

It is a recognisable phenomenon within the industry that advancement in design comes with a continued and increasing abstraction from lower levels of detail. Just as assembler languages separated the programmer to a great extent from the physical machine, whilst still within themselves remaining essentially machine specific, the third generation of languages provided tools that went three stages further, to bring dramatic increases in productivity. Firstly, they enable the generation of many

assembler level instructions from brief and simple commands, which were themselves designed with much greater thought and consideration to the human interface. Secondly, they sought to implement standard languages that were usable across different types of machine. Thus whilst the assembler programmer must learn largely different assembler languages, if he should switch from IBM to ICL machines, and indeed often between systems manufactured by the same supplier, he would find surprisingly little difference in the dialects of COBOL in use, because all the low level differences are ironed out by the respective COBOL compilers. The third major enhancement saw the inclusion of standard methods, within the languages, to structure and access data stores, which at this stage are more commonly referred to as files. Whilst third generation languages still support the first generation of database, by facilitating the definition and storage of data structures and values entirely within the program, there was however, a rapid movement to hold all data externally in files. These data files then became accessible to many different programs simultaneously. It must be said that in achieving this, languages were extended with both a mishmash of operating-system-specific commands, and those that were destined to become industry standard in the field of file handling.

Very quickly dawned the realisation that the act of storing, structuring and accessing data could be optimised and that many fundamental issues arising from the inherent independence of data, such as security, currency and access control, could be centralised. This led to the creation of the third generation database in the form of the all encompassing Database Management System, supported originally by both specialised software and extension to the current third generation languages.

In fact, the penetration of the Database Management System (DBMS) is now almost complete within data processing environment, and nearly all computer manufacturers supply both file handling capability and DBMSs accessible to all languages supported by their systems. However, in many ways this has represented a step back from true advancement, because the DBMSs that were built, were designed initially to support specific types of application on specific hardware platforms, and so did not truly progress the concept of the independent data repository. It is true also that the methods of accessing these structures ensured that the languages used, once again became machine dependent simply because the DBMSs they referenced were themselves not portable between machines or operating systems. Of course, this problem should be seen in context. If the majority of commercial computers in the world are IBM, the fact that, as an IBM programmer, you know IBM's major third generation database offering, IMS and the methods required to access it, puts you in a very strong position compared to the person that understands well the COBOL extensions required to access the Data General INFOS system. Portability is a worthwhile goal only if combined with profitability, and indeed at the corporate level perceptions change. The company that has locked its data into an IDMS database is going to find it very difficult and expensive to change its computer strategy to incorporate increasingly smaller, distributed machines from a different manufacturer. The potential, however, for such changes, and the financial implications of them, are becoming more significant as every day goes by.

The requirement at this stage for beneficial progress, was for a simplification of the way data was seen to be stored and was available to be accessed; in short, a further level of abstraction from the third generation database and language.

It was a simple realisation that if the entire industry all used the same simple logical way of representing, storing and accessing data, then in one sweep, enormous productivity benefits would arise, as data, code and skills became portable, and the science became more easily understood and accessible to the people it serves.

It's pleasing to note that by and large this realisation has been accepted, and this logical view of data which the new generation of both languages and databases supports, is defined on mathematically proven set theory. The term relational has arrived, and has given opportunities to all and sundry to build new products, or advance older ones, so that they store data logically, in terms of flat, two-dimensional tables or relations. Similarly, a growth industry lies in providing the tools that can manipulate this data through the new database access language SQL, and to provide new human interfaces whose ease of use reflects the enormous strides the new culture had made into simplification. We can therefore see, that the fourth generation of both language and database arose together and as mutual partners.

Probably at this level, we see the clearest and most direct relationship between a generation of languages and its equivalent generation of databases. It is certainly true, and has been a necessity in achieving upgrade paths, that most third generation languages can access fourth generation databases (relational), and likewise many fourth generation languages (4GLs) can access many third generation databases, but in truth there is a very close kinship between the fourth generation products. It is hard to imagine 4GLs being successful without the inherent simplicity of the relational database, and conversely it would seem futile to plough effort into developing more simple database products if this simplicity could not be reflected in the programming tools that support them. There is certainly great room for improvement here yet, and so today there is both a huge presence of 3GL products, and an increased penetration of 4GL products and concepts. So what then for the future and the fifth generation database in particular?

It's probably the case, and will continue to be for some time, that the programmer's life is destined to be spent providing applications specific to very concise and often complicated business problems, and we have sought, or been forced, to put several barriers in between the data and the person that wishes to derive information from it. Typically these barriers have been in the form of hardware, software and people whose task, in simple terms, has been to change data into such information.

The role of the fifth generation, both language and database, should, I believe, seek to cut away this barrier, and allow the end-user to extract information directly from data quickly, with flexibility, and based on volumes unthought of at present.

My view therefore of fifth generation database products may be fulfilled when these databases can make better use of specialised hardware, perhaps in the form of general

purpose database machines. I would like to see them lowered in status to that of a simple utility, much perhaps as communications functions have been off-loaded from mainframes. We now commonly accept that many communications functions are executed in dedicated 'front-end' processors, and perhaps database activity should also be 'black boxed' as a 'back-end' processor in its fifth generation. Certainly such databases should interface to many different host computers, and many different software packages, and be served with many different development tools. It must be relational in the logical sense, and be capable, both of holding enormous quantities of data, and of delivering the necessary through-put. It will be fault tolerant, available 24 hours a day, and almost certainly built in such a fashion as to provide linear processing expandability. Perhaps as important as any, will be its ability to exhibit some degree of intelligence in providing support for an improved version of SQL. This 'new SQL' must have much greater awareness itself of data structures, so it can relieve the end user of this particular and tiresome burden. To illustrate this last point let's remember that whilst it is indeed a huge improvement to write, for example, in SQL:

```
SELECT * FROM SALARIES WHERE NAME = 'Jon'
```

than write the 50 or so lines needed to do the same in COBOL, when we can say:

```
GIVE ME Jon's SALARY
```

and have a quick response, we will possibly have managed the move from fourth generation environments and be preparing for the fifth.

However, we are not quite at this heady level as yet, and although there are products, or rather combinations of products, that can achieve the above to some extent today, through the use of specialised hardware (and indeed one of these, namely Teradata's DBC/1012 database machine, is used to a small degree to illustrate various points made within this book), there is still a long way to go.

Fifth generation products are not my prime concern at this present time however, when I witness how poorly we are coming to terms with the fourth generation, and it is indeed at this level that this book is directed. Although, as I have illustrated, the progression to 4GLs and Relational Database Management Systems is largely a natural one, prompted very much by that of applications backlogs and the need for increased simplicity, it seems to me that there are a great many installations of these products that are not benefiting fully from their presence .

This may be due largely to a lack of clear direction when evaluating, buying using and supporting the products. I think it is fair to say that the relational model of data as defined by E Codd several years ago now, and explained later in this book, is proving its practical worth, and now is implementable in a variety of software products which have themselves proved their excellence. Indeed when combining the rediscovered concepts of analysis and design, such as Third Normal Form (TNF) and Entity Relationship modelling, with the excellence of some of the current RDBMSs

available, and such automated development techniques as Computer Aided Systems Engineering (CASE), DP managers should be revelling in a flurry of software development progress.

However, this scenario is very rarely achieved in the real world, and it seems likely that the reason why this is true, is that the scale and scope of the modern day Relational Database System is not appreciated by purchasers, and is not emphasised by the vendors. This is indeed an unfortunate combination that often ensures that the products under-achieve in terms of their true capabilities. Let us expand this idea and throw some light on current RDBMS usage.

In the recent past, the word database has often conjured up pictures of large installations with large machines, enormous amounts of data, and thousands of on-line terminals. Such a database, more often than not, belongs in my third generation and would typically be built around an IMS or IDMS type product, supported by teams of system programmers and a Database Administration team responsible for the 'corporate data'. The database held data, structured as in the 'corporate data' model, defined by an autonomous design group – the Data Administration. It is a fairly obvious reality that an entity of this size represents a large investment, in many senses of the word, and is of limited flexibility. In the time frame required to build such database applications, the business requirements to be met by them invariably change. Unfortunately, the architecture which the data structures were founded upon were not easily changed, in either physical or logical terms, and certainly the programming tools used to access these structures were complex and procedural. Even worse, the structure of the data was directly reflected in the applications that used that data, and so a picture emerges of a three-fold problem:

- huge investment
- limited flexibility
- poor programming tools giving low productivity

As an aside, and when considering these problems, it is tempting to forget what I believe to be the ultimate fallacy – that of the 'corporate database', but I fear that here is not the time or the place to elaborate on this concept.

The picture illustrated above is still in major evidence now, and the realisation of such problems in some large IBM installations for example, is responsible for a growing move to the strategic relational offering from IBM: DB2. This illustrates indeed, the paradox of the RDBMS. In an IBM world, DB2 will replace (co-existing initially with) IMS and other large DBMSs because it will be seen as their equivalent, or greater, in all areas of functionality including speed. For this reason DB2 installations are likely to be successful, ambitious, beneficial and productive (given of course that the product itself proves to be usable), because its perceived importance as a strategic tool will ensure the presence of staff, and commitment to make it work, both from the user and IBM itself.

In the non-IBM world, it is rare that a RDBMSs will be seen in the same light, and it is common to find them being used as slightly superior file handlers, losing many of the advantages of the products, and thus down-grading their contribution and status to something a good deal less than they deserve. Thus, we have a fairly common scenario: in order to solve some easily perceivable problems – the applications backlog for example – senior management is suffering from muddled thinking, by envisaging an RDBMS as a quick and easy solution to systems building. In fact this is a false idea, and is fundamental to the reason why realised productivity gains are lower than expected in most cases. The answer really, and it does work when managed realistically, is that the products can be used to help clear application backlogs, but to be successful requires not only fourth generation products, but a methodology ensuring their beneficial use, based on a clearly defined relational database strategy. This should be defined in just as much detail, and with just as much commitment, as one might assume from any other more traditional large-scale database implementation.

In order to conclude this introduction therefore, and to throw some further light on the subject, I believe that in the long-term it is almost certain that the relational world will split into two fragments – DB2 and the rest, and that this is not altogether a bad thing. I think it unlikely that products, such as Oracle and Ingres, will establish and maintain a large presence in the IBM world, and in fact history tells me that DB2 will only have to be somewhere "as good as" other products to win enormous market share. Perhaps the main threat to DB2 will be from the specialist database machines, which will almost certainly remain quicker and more capable of handling enormous quantities of data. These are finding their market niche right now.

The non-IBM world is currently looking no further than a handful of products, including the well-known Ingres and Oracle systems, but with fair competition arising from all quarters. All of these products will be offering the same types of facilities, so whilst in this book I have illustrated facilities and architectures, often with reference to Oracle and Ingres, I have made strenuous efforts to avoid bias to any particular product. Indeed I have often illustrated certain facilities, knowing full well that they will be out of date by the time this book is published, but I feel that it is often the background to relational products that is so interesting, and sheds light on their current state.