;login:
THE MAGAZINE OF USENIX & SAGE
December 2001 • Volume 26 • Number 8

inside:

PROGRAMMING

A Quick Introduction to Database Systems

By Richard Leyton

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

a quick introduction to database systems

Richard Leyton

Richard Leyton is a senior consultant at Paremus Limited, a newly established technology consultancy company (*http://www.paremus.com*). He has over 10 years experience with UNIX and has worked with various database system installations in and around finance and the dot-com industry.

richard@leyton.org

Introduction

One important area of computing systems management is often overlooked by system administrators but accounts for some of the biggest, most complex, and frequently most important systems for which we are responsible. That area is databases, and they're overlooked because they're perceived by many to be boring, concerned with old technology, unreliable, and the cause of many headaches. To top it all, too often they don't really do very much that's noticeable or interesting.

It's my belief that none of these perceptions have any validity; databases are an interesting, challenging, and evolving area of technology, which, if implemented and supported well, can bring a perceptible benefit to the system administrator, the system itself, and, of course, the users of the systems and of the institution.

What Is a Database?

In its very simplest form, a database can be viewed as a "repository for data." Tautological as it sounds, this repository is tasked with maintaining and presenting the data in a consistent and efficient fashion to the applications, and the users of such applications, that use it. It is these factors which complicate the matter.

Before databases appeared as a separate technology, data was stored in a variety of ways, often proprietary and specific to the implementation in question. Data couldn't be shared and couldn't be utilized outside of the application in which it resided. This clearly proved problematic – a company had the data, but couldn't do more with it — as new requirements came about.

Databases evolved to take responsibility for the data away from the application, and, most importantly, enable it to be shared. As applications grew and new applications appeared, a single data repository evolved, a repository that all applications could access (in an agreed format and model, of course).

Of the many forms possible, today's databases are usually "relational databases." This is not the only variety but has gained ascendancy because it is simple and effective. Older models include the "hierarchical" and "network" (N.B., not like the Internet) models, which can still be found in legacy mainframe environments. These models lost favor because they focused on storage issues rather than data issues. Newer and increasingly popular data models include object-oriented and object-relational, which can in certain circumstances map nicely to object-oriented systems.

But relational databases continue to form the bulk of database systems and are the focus of most books on database design and implementations. Relational databases became popular because they stripped away the machine-specific storage mechanics of the older models so developers no longer needed to worry about how the data was stored and how to retrieve it; they could focus on the data itself and concentrate on building functionality-rich applications.

Oracle (producer of one of the first commercial relational database implementations) was formed out of the research work undertaken at IBM on their System/R research work. The rest, as they say, is history. Now IBM, Sybase, Computer Associates, and various others have established very mature, stable products (though they are not market leaders). And new companies are entering the market all of the time: Clustra, RedHat,

Versant, and the GNU project are all producing database systems that offer something new and innovative, keeping the established players on their toes.

What Must a Database Product Provide?

- Consistency: It must ensure that the data itself is not only consistently stored but can be retrieved efficiently. This is even more critical when changes to the data occur without warning.
- Concurrency: It must enable multiple users and systems to all retrieve the data at the same time and to do so logically and consistently. Concurrency problems will be familiar to many readers, but in a database environment, concurrency is further complicated by the necessity to undo changes made in certain circumstances (e.g., deadlocks and aborted transactions).
- Performance: Users will be very demanding if faced with long response times. Scaling to cope with large numbers of users, all with demands on the resources, can become complex (but it's not rocket science). The database administrator can help by reviewing the access strategies to the data (indexes, caching and compute resources). Sometimes, a very simple change to some or all of these components can significantly improve performance (and sometimes decrease it elsewhere).
- Standard adherence: Most people have heard of SQL (Structured Query Language). It was envisaged by the original researchers at IBM as a query language designed specifically for the relational model to enable programmers to specify how the data should be extracted from the database in an easy way that is independent of the programming language being used. Most databases support the ANSI-ratified SQL92. Additionally, connectivity standards are required. Two of them – ODBC and JDBC – provide common APIs to the database, which gives developers a greater degree of flexibility over which underlying platform they use.
- Security: A database that provides access to any data for any user and also allows them to change it is not really suitable to many business applications. Database systems solve this through access permissions (much like files at the operating-system level) and specific database mechanisms such as triggers.
- Reliability: Of course, databases must keep their stored data intact. Additionally, coping well when things go awry is often a good indicator of the strength of a system administrator and the strength of a database system. A database must, if set up properly, be able to recover to a known consistent point. The use of write-ahead logs (transaction logs) facilitates this but can introduce performance bottlenecks. Needless to say, after repairing a faulty disk array, the very very last thing an administrator wants to deal with is a corrupted or unusable database.

Beyond the Basics

Once the idea of databases was established, databases could store and retrieve data efficiently, effectively, and reliably. Then vendors began to add features to enhance this basic functionality and give them a competitive edge. Some of the extensions have included the following.

Programming Languages to Manage the Data

Oracle has PL/SQL; Sybase/Microsoft have T-SQL. These languages go beyond the de facto standard "SQL" and add functionality (iterative loops, variables, procedures and

A database that provides access to any data for any user and also allows them to change it is not really suitable to many business applications.

Data is really only useful if it has some meaning.

mathematical functions) you'd normally find in more commonly known programming languages. This helps users manage data effectively but also reduces portability.

Maintenance of Data Integrity

Data is really only useful if it has some meaning (i.e., data in the "employee" table that is only employee information and not corrupted by, say, supermarket prices). When data is inserted, deleted, or modified in the database, implicit meaning can be (or might need to be) associated with that data. By using mechanisms known as "triggers" (code that is executed on such events), databases can maintain, introduce, or enforce the meaning. For example, when adding an employee to a database, checks are made to ensure that their social security number is stored and that their manager is defined.

Connectivity

A client application must be able to communicate effectively with the database. Vendors often produce native drivers/libraries for client programs in order to enable efficient connections and queries. However, in this time of open standards, several new bridging and connectivity standards enable programmers to program independently of the actual underlying database: ODBC (the Microsoft-instigated Open Database Connectivity), JDBC (Java Database Connectivity), and Roguewave's DBTools are the three most widely known.

Unfortunately, database independence too often comes at a cost, as it often becomes difficult to avoid using a vendor's features as a quick solution for a complex problem. This can place a greater burden on the application developer as the client or application server might need to undertake more work. Furthermore, performance can also decline since only SQL92 standard queries can be used. The matching (impedance, if you will) between a set of standard function calls and the vendor's calls (especially in older database client libraries) can incur a client-side penalty.

Redundancy/Reliability/Recovery

Over the last few years, highly available systems have been demanded, with downtime of, at worst, minutes per month (five minutes per month is 99.99% reliability) rather than hours per month (99.7% reliability is just two hours per month of downtime).

Database vendors have been somewhat slow to recognize this, but products and solutions are now widely available. Many of them take the approach that high availability needs to be offered in conjunction with operating system vendor cluster/high-availability solutions. Others take the approach that operating systems can't be trusted in this regard and implement a distributed redundant approach themselves as an integral part of the product.

Recovery of a failed system (or resorting to a known-safe point in time) is crucial, but backups of huge systems can take a correspondingly huge amount of time, even with the best backup system in the world. Incremental dumps are vital too. Being able to restore a system to a particular point in time is important, especially when dealing with time-sensitive data or situations. By dumping out the transaction/activity logs, many database vendors have been able to offer acceptable backup solutions to a very fine level of granularity.

Getting Started

It would be foolish to assume that a short article like this can cover the entire subject area of database systems. But hopefully it has presented some of the basics. There are, of course, plenty of books that serve as good, comprehensive introductions to databases, which the interested reader might wish to consider.

C.J. Date's *An Introduction to Database Systems* is widely considered to be the best all around, in-depth book.

Theory and Practice of Relational Databases, by Stefan Stanczyk, Bob Champion, and Richard Leyton ably initiates the reader into both the theory and practice issues of databases. For more information, visit *http://www.theorypractice.org*.

There are plenty of resources online, too. Here are two of the best:

http://directory.google.com/Top/Computers/Software/Databases/
http://uk.dir.yahoo.com/Computers_and_Internet/Software/Databases/

Next Up

In upcoming issues, some of the following areas will be covered in more depth:

- Recent developments and innovations in database technology
- Open source databases vs. closed source databases
- Performance tuning database installations
- Improving reliability of database installations
- Integrating databases in corporate environments

References

The paper that started the relational model: *http://www.acm.org/classics/nov95/*.