How do you know?

Jim Gray
Microsoft Research
January 2003
(a sidebar for an NRC study fundamental challenges in CS and what motivates and excites researchers)

How can knowledge be represented so that algorithms can make new inferences from the knowledge base? This problem has challenged philosophers for millennia. There has been progress. Euclid axiomitized Geometry and proved its basic theorems. Euclid implicitly demonstrated mechanical reasoning from first principles. George Boole's *Laws of Thought* created a predicate calculus and Laplace's work on probability was a first start on statistical inference.

Each of these threads, proofs, predicate calculus, and statistical inference were major advances; but each requires substantial human creativity to fit new problems to the solution. Wouldn't it be nice if we could just put all the books and journals in a library that would automatically organize them and start producing new answers?

There are huge gaps between our current tools and the goal of a self-organizing library; but Computer Scientists are trying to fill the gaps with better algorithms and better ways of representing knowledge. Databases are one branch of this effort to represent information and reason about it. The database community has taken a bottom-up approach working with simple data representations and developing a calculus for asking and answering questions about the database.

The fundamental approach of data base researchers is to insist that the information must be *schematized* – the information must be represented in a predefined schema that assigns a meaning to each value. The author-title-subject-abstract schema of a library system is a typical example of this approach. The schema is used to both organize the data and to make it easy to express questions about the database.

Database researchers have labored to make it easy to define the schema, easy to add data to the database, and easy to pose questions to the database. Early database systems were dreadfully difficult to use – largely because we lacked the algorithms to automatically index huge databases and lacked powerful query tools. Today there are good tools to define schemas, and graphical tools that make it easy to explore and analyze the contents of a database.

This has required invention at all levels of the problem. At the lowest levels we had to discover efficient algorithms to sort, index, and organize numeric, text, temporal, and spatial information so that higher level software could just pick from a wide variety of organizations and algorithms. These low level algorithms mask data placement so that it can be spread among hundreds or thousands of disks, they mask concurrency so that the higher-level software can view a consistent data snapshot, even though the data is in flux, and the low-level software includes enough redundancy so that once data is placed in the database, it is safe to assume that the data will never be lost. The main focus of my research was the theory and algorithms to automatically provide these concurrency-reliability guarantees.

Text, spatial, and temporal databases have always posed special challenges. Certainly there have been huge advances in indexing these databases, but researchers still have many more problems to solve. The advent of image, video, and sound databases raises new issues. In particular, we are now able to extract a huge number of features from images and sounds, but we have no really good ways to index these features. This is just another aspect of the "curse of dimensionality" faced by database systems in the data mining and data analysis area. When each object has more than a dozen attributes, traditional indexing techniques give little help in reducing the approximate search space.

So, there are still many unsolved research challenges for the low-level database "plumbers."

The higher level software that uses this plumbing has been a huge success. Early on, the research community embraced the relational data model championed by Ted Codd. Codd advocated the use of non-procedural set-oriented programming to define schemas, and to pose queries. After a decade of

experimentation , these research ideas evolved into the SQL database language. Having this high level language non-procedural language was a boon both to application programmers and to database implementers. Application programmers could write much simpler programs. The database implementers faced the challenge of optimizing and executing SQL. Because it is so high level (SQL is a non-procedural functional dataflow language), SQL allows data to be distributed across many computers and disks. Because the programs do not mention any physical structures, the implementer is free to use whatever "plumbing" is available. And because the language is functional, it can be executed in parallel.

Techniques for implementing the relational data model and algorithms for efficiently executing database queries remain a core part of the database research agenda. Over the last decade, the traditional database systems have grown to include analytics (data cubes), and also data mining algorithms borrowed from the machine learning and statistics communities. There is increasing interest in solving information retrieval and multimedia database issues.