Microsoft offers three very successful database products, FoxPro, Access and SQL Server. While SQL Server excels in multi-user transaction performance, Access and its underlying Jet engine excel as end-user and development tool for desktop and client-server applications. One of our top priorities is to improve the integration of these two products. With respect to their query processing capabilities, we plan on combining the strengths of SQL Server with those of Access. SQL Server's strengths are focused on management of very large tables, server-side cursors, and the use of stored procedures as scripts and as triggers. Access' strengths are queries over multiple servers, updatable query results, and bit-mapped processing.

In our next generation of products, SQL Server will employ new query processing technology. Both optimization and execution will be based on an extensible set of operators. At this time, we are focusing on the relational algebra augmented with a few operators such as the "top" operator found in Access, and suitable loops-, index-, sort-, hash-, and bitmap-based execution algorithms. Moreover, we are planning on executing these algorithms both sequentially and in parallel.

We have four design goals, namely functionality, performance, scalability, and extensibility. In the first release of the new technology, we focus on matching and surpassing existing query functionality of the two products. By providing a variety of algorithms and choosing among them very carefully, we hope to achieve excellent performance. Scalability from desktops to superservers in our goal; rather than essentially recompiling existing code on new platforms, we are re-designing and re-implementing a substantial component of our product. Finally, by architecting the new code base for extensibility, we not only integrate the latest query technology today but enable rapid innovation for years to come. It is really the last two goals that require the careful yet radical re-design of the query component that we are currently undertaking.

The algebraic optimizer is based on the Cascades optimizer framework first developed outside of Microsoft in collaboration with Tandem. It employs a rule-based rewrite engine, guided by anticipated execution costs, heuristic guidance, branch-and-bound pruning, and heuristic pruning. By use of heuristics, the search engine integrates into one what is called query rewriting, join optimization, and algorithm selection. The information or properties carried up and down a query tree, in addition to costs, includes cardinality (captured as minimum, maximum, and expected values), degree (number of columns), column information (type, constraints, histograms), relational keys and functional dependencies, referential integrity constraints, sort order, columns present in memory, location, and partitioning. The information is divided into logical information, which does not change through an equivalence transformation of the query, and physical information, which is specific to a query plan. Query operators, rewrite rules, properties, and costs are all modelled as classes that can easily be augmented with new subclasses and new instances.

The algebraic execution engine employs a uniform, demand-driven interface between algorithms, which we call iterators and which may be divided into work, enforcer, and control operations. For example, a merge- or hash-join truly modifies the data content, whereas a sort operation (without aggregation, duplicate removal, top, etc.) does not modify the data but enforces a particular representation. Special enforcer operations translate back and forth between bitmap representations and typical iterators over record sets. Control operations typically schedule execution within a single query, e.g., an operation at the root of a common
subexpression that loads data into a spool file eagerly or lazily. By insisting on a uniform interface, the task of the query optimizer is tremendously simplified, as well as enables future rapid addition of new algorithms. Alternative representations of intermediate results are modelled as physical properties.

Among the directions in which we hope to extend the query processor, three stand out. First, we plan on integrating update processing into the framework, including constraint checking and index maintenance. Second, we plan on integrating utilities such as consistency checking, index creation, data import, and data export. Third, we will depend on this framework to address many of the challenges in the new SQL standards.