Spatial databases are becoming more prevalent while growing in quantity. Most DBMS are not yet capable of supporting such datasets in a manner that is easily employed. The need for such specialized databases is thus a paramount concern within different communities, including specialists who analyze geographic information as well as common users interacting with the data through user-friendly interfaces.

The common framework for specifying database schema has generally not included spatial information types, making storing information such as rectangles difficult. As a solution, abstract data types (ADTs), defined by users, have begun to appear that have taken their cue from object oriented programming. Unfortunately, both object oriented databases are not widely deployed and the current SQL specification does not code for spatial queries. Instead of waiting for standards to catch up, a middle layer that handles ADTs as well as spatial data queries, spatial indexing and efficient algorithms has been proposed. This middle layer can usually be plugged into the user level application and allows seamless integration with an existing DBMS.

In terms of implementing a spatial database, the language and processing methodology are both important. Since SQL is the de-facto norm for all database interaction today, any additions have to be incorporated within SQL. SQL-3 aims to support ATDs which will give developers more freedom. Query processing is another field in need of attention. As background, there are two possible types of queries: single scan and multi-scan queries. A multi-scan queries has to touch each tuple more than once in memory. One example is a spatial-join where two tables have to be combined and then different ADTs have to be compared, such as a determining if a rectangle is within another rectangle. Such multi-scan queries require a filtering stage where candidates are eliminated. For spatial joins this usually involves determining rectangle intersection by comparing corner positions. The aim is to reduce the number of actual geometries that have to be examined, since rectangles are much easier to handle than various and non-uniform shapes.

Most databases use B-Trees to store data in order sets, which allows a log n time retrieval even for very large datasets. Unfortunately, this does not map well to spatial data. The need thus led to the creation of R-Trees that used minimum bounding rectangles to encapsulate child nodes, while keeping a meaningful sort order. The indexing used though is hard to choose optimally for spatial data, as it is not immediately clear how the data will be processed or searched. For example, sorting by area is of a different nature than searching for a point within an object.

Spatial data has become even more important as people realize its applicability to data mining. In terms of improving database utility, spatial data is the next frontier.
The SkyServer is a massive database combined with a friendly user interface that aims to help scientists and students explore real astronomical data. The design of the different aspects of the system highlight the tradeoffs and needs of the community. While a significant portion of the paper is devoted towards discussing web design issues, the important lessons are how the system developers worked with scientists to come to terms with the best means for aiding research.

The developers began by asking scientists for twenty questions they would like to be able to easily and quickly answer by the SkyServer. This focused the design and ensured the system remained useful for those actually using it. As it turned out, most queries, although they would have taken days for the astronomers to handle, were solved with simple SQL commands that had at most minute run times. While solving these problems, the need for handling a variety of data arose. Raw data is processed extensively and broken down into various objects that have different attributes. This worked well in the original design, using an object-oriented database, but caused problems when translating to a more traditional RDBMS. Views, foreign keys and SQL joins were able to achieve a level of complexity suitable for the application. The OODBMS also employed tags which acted as a way to speed queries for attributes that were accessed frequently. In the RDBMS, only indexes were available and thus had to be pre-defined, leading to a large number of them.

An interesting challenge was in handling spatial data, a key component in astronomical queries. The hierarchical triangular mesh was added to the SQL server. Instead of traditional bounding boxes, the HTM envisions the celestial sphere as an octahedron where each point within the sphere can be mapped on to the octahedron. Each face of the octahedron is then recursively subdivided until it represents a minute sliver of the sky. The coordinates for an object can then be encoded with integers and stored in a B-Tree. Extra functions needed to be added though to give users more flexibility in their processing, but the end result allowed for spatial joins that were efficiently processed.

An odd decision was made in that the developers wanted to use an out of the box solution with no extra tuning. To that extent, Microsoft SQL server was deployed on massive RAID01 arrays. Surely, higher performance could have been achieved with a modified database suited to storing and indexing astronomical data. On the other hand, the system has proved that readily available databases can be used to aid research efforts without much modification.