SQL is comprised of two components, the data definition language and the data modification language. The DDL is responsible for creating and defining tables (and user defined types in SQL3). The DML is responsible for manipulating data, such as in INSERT commands. A SQL statement can be modified in a number of ways that are outlined in the text. The group by command can partition rows of tuples in various ways. The having clause is able to enforce that only certain groups are displayed, based on some precondition.

Extending SQL to handle spatial data is a vast undertaking that requires large modifications to many assumptions in relational databases, the most significant of which is that data types are no longer predefined. Although it is possible to store spatial data with BLOB types, much of the required spatial functionality is rendered infeasible. Instead, user defined abstract data types are the preferred method, which puts the burden of data definition on the developer. The OGIS spatial data standard is used in SQL3 to aid the developer by offering base classes which can represent most data types (in 2 dimensions). Because many spatial types share a similar base class, common operations can be performed upon them, such as calculations involving reference systems (ie. longitude and latitude), topological tests such as overlap and within, and general tests that are well defined, such as distance. This allows for most of the common spatial functionality but it still lacks a number of other functions representing spatial relationships, such as direction.

A number of interesting spatial functions are highlighted, such as the touch function, a topological operator which can determine if two objects share similar boundaries. The intersection function returns a new geometry type representing the amount of space covered by two geometries. This is separate from the intersect operator which only checks to see if two objects intersect. The buffer operation creates a geometry around an item of a certain distance, which can be used to perform further operations.

Because geometries can be highly irregular, users can create their own data types through the `CREATE TYPE` command. The command has the ability to specify how data is stored, as well as methods that operate on the data. These methods define the input and output data types. This format allows for a more object-oriented SQL syntax, where the name of the data type instance can be used with the dot operator to call its member function. Conspicuous in its absence though are instructions on how to implement functions and constructors. Although the SQL examples highlight how these are used, there is no indication how the database understands the syntax and implementation.
Implementing spatial searches is a challenging proposition due to the lack of support within traditional relational databases and the uniqueness of most spatial datasets. The authors discussed a few methods for storing and processing search queries for spatial data related to the SkyServer. These methods utilized the strengths (indexing, B-Tree like storage) of a commercial RDBMS in order to gain performance. Other optimizations were also made with the 3-D data, such as recognizing that spherical geometry did not need to use expensive sine and cosine operations. Instead, circles could be represented with vectors and point-polygon queries could be mapped to planes using dot products. Boundary zones could also be easily computed using vector form.

The authors primary goal was to perform search queries like point-proximity and polygon-polygon. With this in mind, three methods were used to efficiently process data. The hierarchical triangular mesh was a simple method for applying bounding boxes to regions and recursively sub-dividing the area until the actual point location was found. This had the property of being able to represent minute spaces, in a relatively small tree, with a simple array of numbers that indicated the triangle chosen at each level. Introducing HTM into a relational database was easily accomplished by adding the HTM-spine-schema, which defined precomputed HTM IDs as 64 bit quantities. The HTM Cover function was then implemented which returned all the points within a region. The problem the authors faced was that the HTM library existed outside of the database application and the linkage between the two caused a huge slowdown. When SQL server was running in virtual server mode though, this cost was mitigated somewhat. Ultimately, there was still a ceiling on processing capability. As a solution to the problem, the zone approach was introduced, which added another way to segment the celestial sphere. The difference was that by using normalized zones, stored in scalar form, the processing could be done within the database. This meant that indexes could be created to speed up processing. Zones were used to perform common queries like finding neighbors and nearby objects. For example, a view was created for each object in which its neighbors within an area were found. To get the objects, the zones to the north and south of the object were joined, meaning short zones processed faster than tall zones.

Representing points was done relatively simply with HTM and zones, but handling regions was more challenging. The basic building block was the convex area, which could be combined to represent complicated shapes. They were stored in a space efficient manner using a plane intersection and a point. A whole library was created to deal with these complex regions, which even included a way to automatically simplify regions when they became overly difficult to process. The authors describe a strained relational algebra for regions, allowing OR, AND and NOT operations which created a variety of new regions. Finally, zones were used to reduce the number of regions searched while performing queries. A zone pyramid was employed that adaptively found good size zones for the B-Tree-like searching.

The important point the paper made was that it is possible to implement spatial concepts within a relational database such that a SQL optimizer can perform tasks that would otherwise be difficult or slow.