[PD-05-017] SQL Languages for GIS

Abstract

SQL (Structured Query Language) is a declarative programming language that is closely linked to the relational database model. It is an accessible and widely adopted language used for query, data modification, and data definition—that is, defining data structures (tables) and other database objects. Important additions to the SQL standard include SQL/PSM, which adds control flow, local variables, and other procedural language features; and SQL/MM Part 3, which adds spatial support. Many complex geoprocessing workflows typically implemented in desktop GIS or scripting languages can easily be implemented in spatial SQL.

Keywords: procedural language, relational model, spatial database

Author & citation

Hachadoorian, L. (2019). SQL Languages for GIS. The Geographic Information Science & Technology Body of Knowledge (1st Quarter 2019 Edition), John P. Wilson (Ed).
DOI: 10.22224/gistbok/2019.1.7.

Explanation

1. Definitions

table: database object that stores data in rows representing entities and columns representing attributes of the entities.

2. Introduction

Structured Query Language or SQL (alternatively pronounced “sequel” or “ess queue ell”) is a declarative programming language that is closely linked to E.F. Codd’s relational database model (Codd 1970). Its English-like syntax was designed at the outset to provide a more accessible way than the symbolic logic then in use to create the relational algebra expressions necessary to query a relational database (Chamberlin 2012).

SQL is considered a declarative language because the programmer indicates the desired result, but does not specify how the result is to be achieved. In order to sort a set in a procedural language, a programmer would choose a specific sorting algorithm such as bubble sort or quicksort, which would specify the order in which elements of the set are

visited. The choice of algorithm will have a significant impact on the speed and efficiency of the sort. In core SQL, there are no control flow features that would allow a programmer to loop over a set to implement a specific sorting algorithm. Rather, the programmer indicates that the set is to be ordered by some attribute(s) of the elements, and the sorting algorithm is controlled by the query optimizer.

The query optimizer chooses among competing query plans. Figure 1 is a graphical representation of a query plan selected to find the distance from every large populated place in the United States to the five nearest populated places in any country (executed in PostgreSQL with graphical explain provided by pgAdmin III). The same SQL code may be executed using different query plans depending upon the size of the tables, the indexes that have been defined, how the data are stored on disk (itself controlled by the database engine rather than the programmer), and other factors.

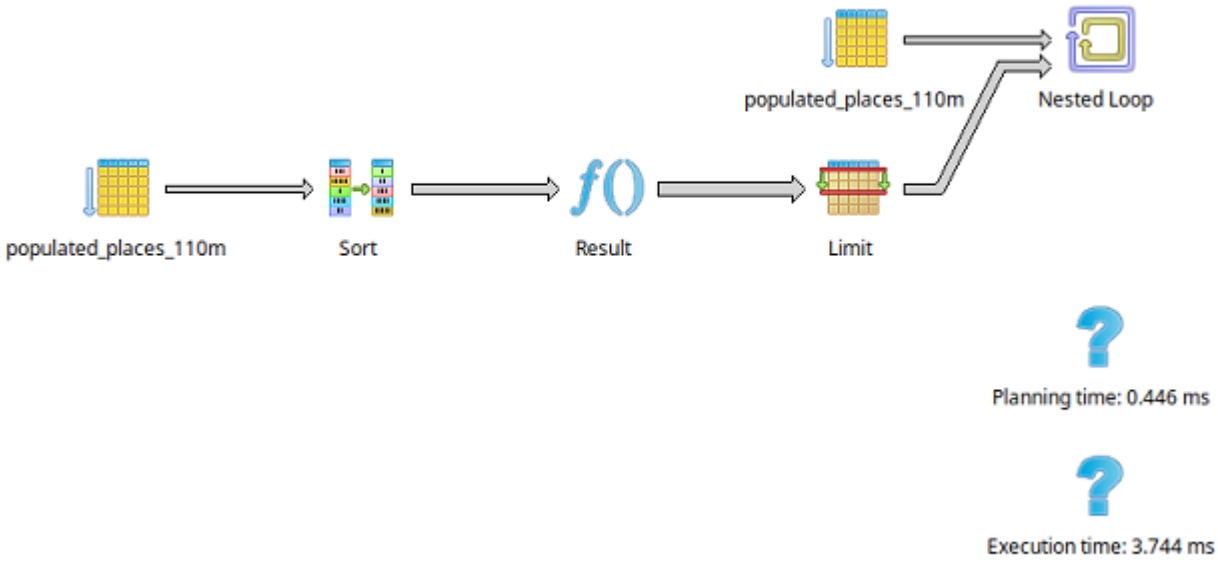

Figure 1. Example of a query plan selected to find the distance from every large populated place in the United States to the five nearest populated place in any country.

SQL statements include clauses that must appear in a prescribed order (though many are optional), and statements are executed in the order presented without branching or looping. Nonetheless, major relational database management system (RDBMS) implementations include additional procedural languages that introduce control flow (looping, conditional evaluation, exception handling, etc.), local variables, and other features.

The SQL standard must be distinguished from an implementation of that standard. The standard is a reference document for the features and syntax of the SQL language. A set of features (“Core SQL”) are mandatory, and additional features are optional. An implementation may be non-conforming if it doesn’t provide core features, or implements

features using idiosyncratic syntax. In practice, most implementations provide features that goes beyond what the SQL standard requires. These extensions may be copied by other vendors, and as they come to be common among implementations, they may incorporated into the SQL standard. Thus, the standard evolves over time in response to the demands of industry. The first commercially available SQL database was Oracle V2 in 1979. The first SQL standard, SQL-86, was adopted (and named for the year of adoption) in 1986. SQL-92 (1992) was a major revision that saw wide adoption, while the current standard is SQL:2016 (International Organization for Standardization 2016).

Some implementations, such as PostgreSQL, strive for conformance and scrupulously document departures from the standard ("PostgreSQL: Documentation: 11: Appendix D. SQL Conformance," n.d.). Resources are available to compare conformance among major implementations (Winand, n.d.). Conformance among implementations is close enough that database developers sometimes strive to write SQL in an implementation-agnostic way. Such code is described as **portable**, and doing so can help an organization avoid the dangers of vendor lock-in. Minor differences in implementations, even in such basic features as the datetime data type or nulls, may make such endeavors challenging.

The relational model is particularly suited to representing vector data, with each table being a spatial layer and each row being a feature, or instance of the entity. This article will focus on vector examples, but it is worth noting that raster data can be stored in SQL databases, and projects such as PostGIS support a wide range of raster analysis operations including map algebra, reclassification, tiling, slope calculation etc. ("Chapter 9. Raster Reference" in PostGIS Development Group 2019).

3. Data Modification Language

As persistent stores of data, all databases implement CRUD (Create, Retrieve, Update, Delete) operations. Data retrieval ("query"), which can include transformation, aggregation, filtering, and ordering, is discussed in the forthcoming GIS&T Body of Knowledge entries on Structured Query Language (SQL) and Attribute Queries and Spatial Queries. The remaining operations (Create, Update, Delete) are collectively referred to as Data Modification Language, or DML. The SQL language provides DML through the INSERT, UPDATE, and DELETE statements.

INSERT is used to create new rows in an existing database table. The data for the new rows may be explicitly provided, or the result of a query on existing data:

INSERT INTO table_name (column_list)
VALUES (value_list);

Columns which are not nullable and do not have a default value **must** have values supplied. The list of column names may be omitted if values are supplied, in order, for all columns in the table. Explicitly specifying column names is highly recommended, in order to separate table structure from business logic.

A very common use case is the need to immediately retrieve the value of a primary key identifier automatically generated upon insertion of a new record. (Desktop GIS software

such as ArcGIS and QGIS require a unique integer identifier to maintain the link between a loaded layer and the backend database.) Major SQL implementations therefore provide special syntax to return the value of the newly generated key as part of the INSERT operation.

Many databases also provide **bulk copy** capability which allows the insertion of multiple rows from an external datasource (or the reverse). This allows for faster insertion of large datasets than can be achieved using INSERT statements, but implementations are vendor-specific.

UPDATE is used to change values in an existing record. Often, a client application will allow modification of existing rows/features, which will then be written to the row's column values. The new values can also be transformations of the record's own values. For example, in a table of administrative areas with columns population and area, a pop_density column can be updated with the result of population divided by area.

In order to update a row based on values in another table, the SQL standard requires using a subquery. But many major SQL implementations instead allow the same FROM <table> JOIN ON <expression> syntax as the SELECT statement. SQLite/SpatiaLite stands out in **not** offering this capability. Many SQL implementations also provide a statement, alternatively called MERGE or UPSERT, that allows the client to submit a record which will be updated if it already exists in the table, or inserted if it does not yet exist.

The DELETE statement removes rows from a table, usually with the WHERE clause providing the criteria for which rows are to be deleted. It is common for the database engine to merely mark a deleted row as "dead", making it invisible to queries, with the disk space not actually recovered until a cleanup takes place (which may be manual or automatic, depending upon the database engine and configuration parameters).

Desktop GIS often holds modified data in an "edit" session, to be applied as a batch update at a later time. Row inserts, updates, or deletes will not be committed to the database until explicitly committed by the user.

4. Spatial Analysis

Most spatial methods can be grouped into four categories: (1) data format conversion; (2) retrieval of geometry properties (e.g. vector type, dimensions) or measures (length, area); (3) spatial relationship between geometries; (4) generating new geometries from others (Stolze 2003). The SQL/MM 3 standard introduces functions to accomplish all of these needs (International Organization for Standardization 2016). Functions are prefixed with ST_ (originally intended for spatial and temporal analysis, though the standard only encompasses spatial data types and analyses).

Analysts new to working with SQL should be aware of terminology with different meanings in the domains of spatial analysis and relational databases. For example, "union" in GIS terminology means to combine spatial layers so that the result contains the locations covered by all features in the input layers (whether overlapping or non-overlapping) while preserving the attributes of the features at the covered the locations; "union" in the relational database context refers to set theoretic union of table rows, which with some

with some differences is roughly the equivalent to GIS “merge” (combining features of the same kind of entity into a larger set). SQL/MM 3 (see below) implements ST_Union which is neither of these, and actually the equivalent of what GIS analysts call “dissolve”. A GIS union would be performed with a somewhat complex spatial query making use of ST_Intersection (returning the regions that overlap) and ST_Difference (returning the regions that do not overlap).

4.1 Data Format Conversion

External tools, such as ArcCatalog or GDAL/OGR, are commonly used to convert between the internal database format and external formats such as shapefile, GeoTIFF, or another spatial database. A small number of SQL functions such as ST_AsGeoJSON and ST_AsKML allow conversion of a geometry to the specified representation. This is, however, merely a geometry, which would have to be added to a file with required headers for use in other applications.

4.2 Retrieval of Geometry Properties

Geometry properties can be accessed through straightforward functions in SQL, such as ST_GeometryType to return the type and dimensions of a vector geometry. To find the area of all features in a polygon layer:

```
SELECT name, ST_Area(geom) AS area
FROM spatial_polygon_table;
```

To output coordinates of a point geometry:

```
SELECT name, ST_X(geom) AS x_coord, ST_Y(geom) AS y_coord
FROM spatial_point_table;
```

The SQL/MM specification includes a number of functions for more complex evaluation of spatial characteristics of a geometry. For example:

- ST_IsClosed determines whether a linestring’s start and end points are the same.
- ST_IsSimple determines whether a linestring has no self-intersections or points of self-tangency.
- ST_Valid determines whether a (multi)polygon is valid, which tests for, among other things, self-intersection (for example a bowtie shape is not valid) or line segments with no area (“spikes”) extending from the polygon.

An interesting PostGIS extension is the use of XYM coordinates of linestrings to create trajectories (animal paths or flight paths for example) by storing time in the M coordinate. The ST_IsValidTrajectory function determines whether the M coordinates are strictly increasing along the vertices that define the linestring.

4.3 Spatial Relationships between Geometries

A large number of SQL/MM functions determine spatial relationships between geometries, such as intersection, containment, touching, crossing, etc. These are explored in more detail in Spatial Query (forthcoming).

4.4 Generating New Geometries from Others

The incorporation of geometry processing functions in SQL/MM 3 allows SQL to be used for many GIS analytical purposes that would typically be accomplished in desktop GIS or a scripting language. Common operations such as buffering, intersection, and convex hull construction are included in the standard. Additional functions such as feature simplification, concave hulls, Voronoi polygons, and tessellation are supported by some vendors, often through importing of libraries in the open source geospatial stack (such as GEOS or SFCGAL).

Single geometries can be buffered with ST_Buffer(geom, distance), where the distance is understood as units in the spatial reference system of the geometry, or meters if the geometry is stored in WGS84. One example of the power of using a SQL database as a backend is that a buffer query can be stored as a spatial view. In a file-based spatial format (such as shapefiles or KML), any change to the input layer would entail repeating the buffering operation and storing the new buffers in a new output file. Putting the buffer function in a database view ensures that the buffer layer is refreshed when requested, and will show the buffers based on the current data without any intervention on the part of the analyst.

As an example of a more complex operation, take the common use case of areal interpolation of a uniformly distributed phenomenon (Goodchild and Lam 1980). Given population in census enumeration units, SQL/MM 3 functions can be used to estimate the population for a non-coextensive service geography:

SELECT service_area.id, SUM(census.totalpop * ST_Area(ST_Intersection(service_area.geom, census.geom)) / ST_Area(census.geom)) **AS** population
FROM service_area **JOIN** census **ON** ST_Intersects(service_area.geom, census_enum.geom)
GROUP BY service_area.id;

This spatial query demonstrates the use of several spatial functions and the processing order of the SQL language.

1. The population in the intersections of the Census geometries and service area geometries is calculated:
    1. The areas of the Census geometries are calculated using ST_Area(census.geom).
    2. The intersection of the Census geometries and service area geometries is created using ST_Intersection(service_area.geom, census.geom).
    3. The areas of the intersections are calculated using ST_Area(...).
    4. The ratios of these areas are calculated by dividing the result of **1.c** by the result of **1**.**a**.
    5. These ratios are multiplied by the population of the Census area, census.totalpop.
2. The query rows with population allocated to each intersection is grouped by service area ID, and the SUM of populations that fall in the same service area is computed.

To again draw a distinction between SQL and procedural languages, in a procedural language the analyst would have to use a nested loop over two sets of geometries, construct the intersection, calculate the area-weighted population, and keep a running total

of the population by service area.

5. Data Definition Language

One of the goals of SQL was to create a language which could be used to build data structures as well as manipulate and query the data stored in those structures (Chamberlin 2012). Data Definition Language (DDL) refers to the part of the SQL language which creates and modifies database objects such as tables, indices, and views. (Prior to the creation of SQL, it would not be unusual for query and DML, which were intended for database users, to use a different language than DDL, which was intended for database developers.)

New tables are created using the CREATE TABLE statement. Column data types must be specified. Columns will be nullable, unless NOT NULL is specified. Features may be specified using geometry (Euclidean) or geography (spherical) data types. They may be subtyped as points, lines, or polygons, and have the coordinates system specified using a Spatial Reference Identifier (SRID).

From the point of view of the database, geometry is just another data type, while from the point of view of a desktop GIS, the presence of a geometry column turns the table into a spatial layer. Since a geometry column is just another column, the database engine will be fine with defining a table with two geometry columns—for example, to store both detailed and cartographically generalized versions of a river. This requires the desktop GIS to be able to interpret the table as two “different” spatial layers with the same attribute table, which not all desktop GIS can do. (Desktop GIS which rely on GDAL for data access can do so.) Additionally, table rows may become large enough to impact performance during certain operations (Obe and Hsu 2014).

In order to display and possibly modify the data, desktop GIS typically requires a spatial layer to have an integer primary key. A typical statement to create a spatial layer might be:

```
CREATE TABLE country (
    gid serial PRIMARY KEY,
    name varchar NOT NULL,
    iso3 varchar(3) NOT NULL,
    geom geometry(MultiPolygon, 4326)
);
```

Often a spatial index will be created as part of the table creation process. In order to speed up bulk loading of large datasets, index creation may be delayed (or removed from an existing table) and recreated after the data load.

A database view is a stored query that can itself be queried like a table. Views allow persistence of common queries, and can also be used to implement security, for example by only exposing certain columns. Very simple views—usually those that select from a single table—may be updatable. Some databases also support materialized views, a query whose result set is written to disk and refreshed periodically, providing faster access to the results of a complex query.

Often we need to work with geospatial data in a transformed coordinate system. Views

provide a way to expose a spatial layer in commonly needed coordinate systems without storing multiple copies of the features and their attributes. If the underlying geometry or attributes change, the view will dynamically display the correct data. In order to speed up spatial operations, a functional index can be created on the transformed geometry (Obe and Hsu 2014).

6. Procedural Languages

Early SQL databases offered **prepared statements** (also known as **parameterized queries**) as a way to achieve performance gains by compiling the statement once and reusing it within the same session. In addition, prepared statements provide some defense against SQL injection attacks. Prepared statements do not persist (i.e., they are only reusable within a SQL session), and offer the greatest gains for queries with complex plans, such as multi-table joins.

Stored procedures offer similar benefits but are server-side code that persists beyond the session and can be accessed by multiple clients. Additionally, they provide procedural programming elements such as local variables and control flow. Oracle was the first vendor to implement a procedural language, PL/SQL. In response to widespread emulation by other database vendors, SQL Part 4: Persistent Stored Modules (SQL/PSM) was added to the SQL standard (International Organization for Standardization 1996). Among major SQL implementations, some (such as MySQL) implement SQL/PSM, some implement a vendor-specific procedural language (such as SQL Server's Transact-SQL or Oracle's PL/SQL), and some implement a large number of languages and allow developers to create new ones (such as PostgreSQL). Of particular interest to analysts is PL/R, a PostgreSQL procedural language that allows the creation of user-defined functions in the statistical language R (Conway 2016).

While prepared statements can return only one result set (and associated messages and warnings), stored procedures can return multiple result sets, as well as variables which have been declared as output parameters. Because of control flow elements, stored procedures can perform complex database operations which touch multiple tables. Stored procedures are invoked using a keyword such as CALL (Oracle) or EXECUTE (SQL Server).

Many SQL implementations, such as SQL Server and Oracle, distinguish between stored procedures and functions. User-defined functions have return values and can appear anywhere an expression can appear in a SQL statement. Functions are understood to not change the database state and to always have a return value (even if that value is void or null). Procedures are understood to be able to change the database state (i.e. alter persistent data using typical CRUD operations) and can commit transactions in batches. They do not have return values, but may have output parameters. In practice, the difference has to do with programming practice rather than the capabilities of the procedural language, and implementations such as PostgreSQL do away with the distinction. In PostgreSQL, everything is a function, and functions can have both output parameters and return values, can return result sets, and can alter the database state as the programmer sees fit. However, PostgreSQL functions cannot commit transactions, which makes them less than ideal for batch processing. PostgreSQL 11 has implemented stored procedures in response.

Procedural languages can be used to create database triggers, which are functions or procedures that execute when some other database statement occurs, possibly altering the result of that statement. A common use case is an INSERT or UPDATE trigger used to update derived columns or check data validity. In a polygon layer, an area column can be updated to reflect the area of the polygon whenever a row is added or the polygon geometry is changed. Data to be inserted to a line layer can be checked for validity (e.g., avoiding self-intersection) and discarded or diverted to a different table for manual review.

Procedural languages can be used to construct and execute DDL operations, creating and populating new database objects. As an example, in a retail orders database, recent records may be the most heavily accessed, and older records may be very infrequently accessed. A typical solution would be to implement table partitioning, so that each quarter (or other period) of data appears in its own table, inheriting from a parent orders table. The DDL to create the newest table, inheriting from the parent and named for the current quarter (e.g. orders_2018q1), could be constructed and executed in a stored procedure. Triggers would be used to insert new records into the appropriate table based on the date of the order. The creation of the new partition could itself be triggered by the attempt to insert a record for a quarter that lacked a target table.

7. Influence

Surveys of software developers show that SQL is an extremely popular “technology” (“Stack Overflow Developer Survey 2018” 2018). Whether it ranks as a popular or in-demand “programming language” turns on whether it is excluded from consideration (“TIOBE Index” 2018; Misirlakis 2017). It is common for job announcements seeking fully procedural programming languages to also require knowledge of SQL.

NoSQL refers to a variety of database management systems using non-relational storage models such as key-value stores, document stores, graph databases, and column-oriented (or wide table) databases (McCreary and Kelly 2014; Li 2018). NoSQL can refer both to the query language and the relational model it works with. NoSQL more specifically refers to a non-relational model with certain characteristics such as scalability, high throughput, and lack of joins. Some NoSQL models, such as key-value stores, provide key-based put, get, and delete operations, but no query language (Hecht and Jablonski 2011). For those with a query language, NoSQL does not mean that it’s query language is not SQL-like!

A NoSQL project may support SQL syntax or an extremely similar syntax, such as N1QL (Couchbase) and CQL (Cassandra). Others, such as SPARQL (semantic web), and MongoDB’s query language, are considerably different. However, the widespread knowledge of SQL syntax creates pressure for supporting a SQL query language. In fact, experience has shown that working with query languages which are almost but not quite the same as SQL “actually created more mental friction: engineers didn’t know what was supported and what wasn’t” (Kulkarni 2017). Google recently decided to implement a SQL query language for Google Spanner in order to leverage existing knowledge of SQL and create consistency among their products (Bacon et al. 2017). SQL may well be a “universal interface for data analysis” (Kulkarni 2017).

References

Bacon, D. F., Kogan, E., Lloyd, A., Melnik, S., Rao, R., Shue, D., Taylor, C., et al. 2017. Spanner: Becoming a SQL System. In Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17, 331–43. Chicago, Illinois, USA: ACM Press.

Chamberlin, D. D. (2012). Early History of SQL. IEEE Annals of the History of Computing, vol. 34, no. 4, pp. 78-82.

Codd, E. F. (1970). A Relational Data Model for Large Shared Data Banks. Communications of the ACM. 13(6), 377-387.

Conway, J.E. (2009) PL/R User's Guide-R Procedural Language.

Goodchild, M. F. and Lam, N. S.-N., (1980). Areal interpolation: a variant of the traditional spatial problem. Geo-Processing, 1, 297–312.

Hecht, R. and Jablonski, S. (2011). NoSQL Evaluation: A Use Case Oriented Survey. In 2011 International Conference on Cloud and Service Computing, 336–341. Hong Kong, China: IEEE.

International Organization for Standardization (ISO). (1996). "ISO/IEC 9075-4:1996 - Information Technology – Database Languages – SQL – Part 4: Persistent Stored Modules (SQL/PSM)."

ISO (International Standards Organization). (2016). Information Technology - Database Languages - SQL Multimedia and Application Packages - Part 3: Spatial. ISO/IEC 13249-3:2016.

Kulkarni, A. (2017). Why SQL Is Beating NoSQL, and What This Means for the Future of Data. Timescale.

Li, Z. (2018). NoSQL Databases. The Geographic Information Science & Technology Body of Knowledge (2nd Quarter 2018 Edition), John P. Wilson (Ed).

McCreary, D. G. and Kelly, A. M. (2014). Making Sense of NoSQL: A Guide for Managers and the Rest of Us. Shelter Island: Manning.

Misirlakis, S. (2017). The 7 Most in-Demand Programming Languages of 2018. Coding Dojo Blog.

Obe, R. and Hsu, L. S. (2021). PostGIS in Action. Simon and Schuster.

PostGIS Development Group, The. (2019). "PostGIS 2.5.2dev Manual." https://postgis.net/docs/.

PostgreSQL (n.d.). Appendix D. SQL Conformance. PostgreSQL Open Source Database.

Stack Overflow. (2018). 2018 Stack Overflow Developer Survey Results.

Stolze, K. (2003). SQL/MM Spatial: The Standard to Manage Spatial Data in Relational Database Systems. In BTW 2003: Database Systems for Business, Technology and Web. Leipzig, Germany.

TIOBE Index. (2018). TIOBE - the Software Quality Company.

Winand, M. (n.d.). Modern SQL website.