[FC-06-012] Structured Query Language (SQL) and attribute queries

Abstract

The structured query language (SQL) for database interrogation is presented and illustrated with a few examples using attribute tables one might find in a common GIS database. A short background is presented on the history and goals that the creators of the SQL language hoped to achieve, followed by a review of SQL utility for data query, editing, and definition. While the SQL language is rich in content and breadth, this article attempts to build on a simple SQL and then iteratively add additional complexity to highlight the power that SQL affords to the GIS professional who has limited programming capabilities. The reader is asked to consider how minor modifications to SQL syntax can add complexity and even create more dynamic mathematical models with simple English-like command statements. Finally, the reader is challenged to consider how terse SQL statements may be used to replace relatively long and laborious command sequences required by a GIS GUI approach.

Keywords: joining, query languages, query operations, tables

Author & citation

Lembo, A. J. (2020). Structured Query Language (SQL) and Attribute Queries. The Geographic Information Science & Technology Body of Knowledge (4th Quarter 2020 Edition), John P. Wilson (Ed.). DOI: 10.22224/gistbok/2020.4.3.

This Topic is also available in the following editions:

DiBiase, D., DeMers, M., Johnson, A., Kemp, K., Luck, A. T., Plewe, B., and Wentz, E. (2006). Structured Query Language (SQL) and attribute queries. The Geographic Information Science & Technology Body of Knowledge. Washington, DC: Association of American Geographers. (2nd Quarter 2016, first digital).

Explanation

1. Introduction and Background

From its inception, Structured Query Language (SQL) was intended to be used for data query, update, and data definition. It was also designed as a query language for those normally unfamiliar or intimidated by lower level programming languages. The developers of SQL hoped to create a language where with a little practice, users could learn to read queries ... almost as though they were English prose (Chamberlin, 2012). One of the early

publications on SQL titled ‘‘SEQUEL: A Structured English Query Language” identified the primary audience of SQL as a large class of users who, while they are not computer specialists, would be willing to learn to interact with a computer in a reasonably high-level, non-procedural query language...such users are accountants, engineers, architects (Chamberlin and Boyce, 1974). This same goal is especially useful today, as many GIS professionals wish to move beyond the restrictive nature of a GUI interaction with the system, yet feel reluctant to take on the challenge of using a robust computer programming language.

Using SQL as a higher level, English-like command language was made possible by the work of E.F. Codd in his relational model definition. Codd’s work provided a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other (Codd, 1970). The relational data model presented by Codd allowed developers to create an easy to use language, divorced from the complicated implementation on the computer. In other words, SQL implementation translated a declarative query statement into a detailed plan for processing the query within the compiler (Chamberlin, 2012). The user simply focused on the declarative nature of the query statements, and could ignore the technical details of the implementation.

Understanding the role of attribute queries within a relational database is based on Set Theory, Relational Database Management Systems, and SQL Languages for GIS. These first two topics provide both a mathematical and philosophical theory of data and their relationships, while the third topic describes the technical implementation of the declarative language of SQL. This discussion highlights the more practical nature of using SQL syntax to interrogate relational databases using the general principles defined in set theory.

2. An Example Database

For this discussion, we consider a hypothetical database with two separate tables (property and owner), linked by a common identifier (parcelkey). The property table shows basic information about land properties typically stored in GIS, and the ownership table shows the owner name and property value.

Property Table

	parcelkey	acres	address	sq_ft	propclass	yr_built
1	5007001030000...	1.44	615 MEADOW S...	11099.0	Commercial	1998
2	5028890240000...	8.12	132 DURFEE RD	1904.0	Residential	1850
3	5007000790000...	0.1	224 CLEVELAN...	1098.0	Residential	1860
4	5007000660000...	0.15	201 PEARL ST	1296.0	Residential	1957
5	5007000910000...	0.85	105 COTTAGE PL	2121.0	Residential	1910
6	5007000800000...	2.03	301 GENEVA ST S	25948.0	Commercial	1925
7	5007000800000...	0.09	207 CLINTON S...	1300.0	Residential	1920
8	5007000820000...	0.09	207 GILES ST	1770.0	Residential	1920
9	5007000950000...	17.54	500 MEADOW S...	123090.0	Commercial	1998
10	5007001280000...	0.40999	653 SPENCER RD	828.0	Residential	1941

Table 1. Example of a database table typically found in a GIS that includes information on property characteristics such as size, address, classification, and year build. Source: author.

Ownership Table

	parcelkey	owner	value
1	5007000570000...	MOORE, DERRICK	$130,400.00
2	5007000610000...	TOMPKINS TRUST COMPANY	$600,000.00
3	5007000930000...	KILEY, DARBY K	$139,000.00
4	5007000610000...	MCCUNE FAMILY PROP, LLC	$307,500.00
5	5007000560000...	TUMBAR, TUDORITA	$190,000.00
6	5007000670000...	ANDERSON, ELIZABETH J	$205,000.00
7	5007000670000...	THRASH, BRIAN L	$395,000.00
8	5028890240000...	ABELE, ROBERT J	$176,000.00
9	5007000560000...	SPRAGUE, JUDY S	$75,000.00
10	5007000690000...	BORG, THOMAS	$160,000.00

Table 2. Example of a database table typically found in a GIS that includes information on property ownership such as the owner name and property value. Source: author.

3. The SQL Syntax

As discussed in SQL Languages for GIS, SQL is a declarative programming language linked to the relational database model and is a widely adopted language used for query, data modification, and data definition. The declarative nature of SQL makes it readily achievable for the average GIS user to gain proficiency in issuing moderate to complex queries of database tables. One of the benefits of the SQL standard is that it allows the user to issue English-like commands, while the technical details for achieving a result are hidden from the user. For instance, a user may simply issue a construct such as ORDER BY to perform a sort operation, while not having to be distracted by the query engine's implementation of a particular sorting algorithm, storing variables, or iterating through lists. Other constructs like GROUP BY (discussed later) allow for a simple two-word statement to bypass the necessary back-end implementation of if/then, for, and while statements to achieve the necessary results. Therefore, the user may ignore sophisticated back-end processing and simply concentrate on the terse, English-like syntax while at the same time accessing rather powerful capabilities.

As a model for query, modification, and data definition, SQL facilitates standard database operations within an easily understood high level language. Query operations work directly on tables to retrieve data elements while modification operations allow the user to change data in terms of updates, deletions, or insertions. Data definition operations allow users to create or modify the structure of a database table.

3.1 The Basic Form of SQL Syntax

Like any language, SQL exposes a vast number of ways to achieve a solution. However, for most beginning users, SQL queries follow as particular order in form of:

SELECT < some criteria >
FROM < some data table >
WHERE < some condition is satisfied >

while modification of tables typically follow the form of

UPDATE < some table >
SET < some field with a particular value(s) >
WHERE < some condition is satisfied >

Data definition syntax follows the form of

ALTER < some table >, ALTER < some field >
SET < some definition >

Although modifications to the SQL syntax do exist, it is important to note that they generally follow the structure listed above, and thus are easily adapted to consider different criteria. The remainder of this document will primarily focus on the query capabilities of SQL, and only briefly discuss modification and definition capabilities.

3.2 SQL Functionality

SQL queries include general query functions, calculations, and aggregate functions. And,

as previously stated, when the SQL function follows the basic form of an SQL query they may be modified or have greater complexity added to them. While hundreds of SQL commands exist, we will focus on just a few for illustrative purposes.

3.3 Query Operations

Using the general form of the SQL queries defined earlier, one can interrogate the property table with an easy to use query such as:

```
SELECT *
FROM property
```

(where * is shorthand for selecting every column in a table) to obtain each record in the property table. However, building upon this base query, one can select specific columns simply by replacing the * with column names:

```
SELECT address, propclass, acres
FROM property
```

	acres	address	sq_ft	propclass	yr_built
1	1.44	615 MEADOW S...	11099.0	Commercial	1998.0
2	17.54	500 MEADOW S...	123090.0	Commercial	1998.0
3	5.74	614 MEADOW S...	58296.0	Commercial	2002.0
4	1.41	400 MEADOW S...	23720.0	Commercial	1985.0
5	1.16	138 OAKWOOD...	2089.0	Residential	1975.0
6	1.45	127 TAYLOR PL	1088.0	Residential	2008.0
7	0.21999	311 WOOD ST	1844.0	Residential	1956.0
8	0.67	101 WILLIAMS ...	1288.0	Residential	1952.0
9	1.54	220 CHERRY ST	2120.0	Industrial	2006.0
10	0.89999	1 WESTWOOD ...	2424.0	Residential	1962.0

Table 3. Example from a query that returns only certain fields. Source: author.

In these examples, no criteria was requested. We can easily add criteria using the WHERE clause to provide greater specification (finding properties that are over 2 acres in size) in the query:

```
SELECT address, propclass, acres
FROM property
WHERE acres > 2
```

	acres	address	sq_ft	propclass	yr_built
1	17.54	500 MEADOW S...	123090.0	Commercial	1998.0
2	5.74	614 MEADOW S...	58296.0	Commercial	2002.0
3	4.15	120 MAPLE AVE	46296.0	Community Srv...	1980.0
4	2.17	102 OAKWOOD...	1648.0	Residential	1952.0
5	10.59	725-45 WILLO...	40850.0	Community Srv...	1990.0
6	2.32	1112-16 HECTO...	1062.0	Residential	1952.0
7	2.8	103 NEEDHAM ...	2159.0	Residential	1954.0
8	2.93	515 CAMPBELL ...	1376.0	Residential	1970.0
9	2.29	222 CAYUGA ST S	54378.0	Commercial	1973.0
10	3.34	529 MEADOW S...	19968.0	Commercial	1961.0

Table 4. Example from a query that returns certain fields and those properties meeting the condition of being over 2 acres in size. Source: author.

Those familiar with computer programming will understand the complexity for traversing a list or array using a for..each and if..then criteria, even for a relatively straightforward query to identify three columns in a two-dimensional list for elements that have more than 2 acres. In the case of SQL, all the technical details for achieving the results are hidden, and the user simply issues a very straight-forward, English-like command.

Recalling the definition of subsets (⊆), union (∪), intersection (∩), in Set Theory, SQL allows a user to implement the set criteria through the clauses of AND, OR, and NOT, again, with the technical details hidden from the user:

```
SELECT address, propclass, acres
FROM property
WHERE acres > 2
AND propclass = 'Commercial' {one may replace the AND with OR, or NOT}
```

	acres	address	sq_ft	propclass	yr_built
1	17.54	500 MEADOW S...	123090.0	Commercial	1998.0
2	5.74	614 MEADOW S...	58296.0	Commercial	2002.0
3	2.29	222 CAYUGA ST S	54378.0	Commercial	1973.0
4	3.34	529 MEADOW S...	19968.0	Commercial	1961.0
5	5.93	THIRD ST	400.0	Commercial	2006.0
6	4.94	376 ELMIRA RD	14800.0	Commercial	2007.0
7	15.6	WEST VILLAGE ...	74900.0	Commercial	1971.0
8	15.6	WEST VILLAGE ...	74900.0	Commercial	1971.0
9	15.6	WEST VILLAGE ...	74900.0	Commercial	1971.0
10	14.53	710-734 MEAD...	65275.0	Commercial	2002.0

Table 5. Example from a query that returns certain fields and those properties meeting the condition of being over 2 acres in size with a property classification of commercial. Source: author.

The query implements the subset (⊆) defined within set theory to select records that are both over 2 acres and with a commercial classification. Intersection (∩) and union (∪) will be addressed later when evaluating multiple tables in a single query.

3.4 Calculations in Queries

SQL allows calculations in queries. Virtually any mathematical function may be performed within a query both as a way to return a new column value or as an expression within a selection criteria. For example, we can find all properties who's building square footage is over 25% of the land area, as:

SELECT address, propclass, acres, (**(sq_ft / 43560) / acres) AS percent_structure_area**
FROM property
WHERE **(sq_ft / 43560) / acres** > .25
AND propclass = 'Commercial'
ORDER BY percent_structure_area DESC
LIMIT 10

	address	propclass	acres	percent_structure_area
1	801 BUFFALO ST W, UNIT ...	Commercial	0.00999	28.9660643195997
2	120 S AURORA ST	Commercial	0.18999	12.1719953266111
3	102 CLINTON ST E	Commercial	0.92	6.29783507006827
4	314-20 STATE ST E	Commercial	0.11999	5.45270416244513
5	307 COLLEGE AVE	Commercial	0.28999	3.47681473833159
6	312 COLLEGE AVE	Commercial	0.75999	3.39499899900814
7	312 COLLEGE AVE	Commercial	0.75999	3.39499899900814
8	118 DRYDEN RD	Commercial	0.50999	3.24102935740597
9	311 GREEN ST E	Commercial	0.31999	3.18098496592711
10	113 DRYDEN RD	Commercial	0.51999	2.98625683467

Table 6. Example from a query that returns a result that calculates the proportion of the building square footage to the property size. Source: author.

In this query, two calculations are performed: converting the building square footage to acres (by dividing by 43,560), and then dividing that quotient by the property acres to determine the proportion of the building square footage to the property size. In the example, the calculations are performed twice; one time to create the criteria for calculation, and another time to present a new column that represents that actual percentage of the property that the building occupies. The final results present the top 10 properties in descending order by leveraging the ORDER BY and LIMIT commands. Once again, the English-like syntax is very achievable for a novice user who might be completely overwhelmed when attempting to write a computer program to achieve the same results.

3.5 Aggregate Functions in Queries

Both string and mathematical aggregate functions exist within the SQL language, and are available for interrogation or analysis of data elements within a table. Aggregate functions perform calculations on a set of values and return a single value. These functions are numerous but most often follow similar functions one would find in a modern spreadsheet (i.e. min, max, sum, average). For example, we could determine the minimum, maximum, average, and standard deviation of square footage of buildings in the property table:

```
SELECT min(sq_ft), max(sq_ft),  avg(sq_ft) as Avg_sqft, stddev(sq_ft)
FROM property
WHERE propclass = 'Residential'
```

	min	max	avg_sqft	stddev
1	208.0	13481.0	1813.17196939186	710.339478875653

Table 7. Example from a query performing aggregate calculations on numeric fields in a database. Source: author.

3.6 Grouping Aggregate Queries

The previous example performed an aggregate function on every record in the table that met the criteria for residential properties. SQL also offers the ability to aggregate the mathematical operations on defined subsets of data using the GROUP BY function. Often, the GROUP BY function is used in conjunction with aggregate queries to separate aggregate calculations into their unique groups. For instance, instead of evaluating just the residential properties, one can calculate the same aggregate queries and present (or GROUP) the results for each unique property classification:

SELECT **propclass**, min(sq_ft), max(sq_ft), avg(sq_ft) as Avg_sqft, stddev(sq_ft),
stddev(sq_ft) / avg(sq_ft) AS coeff_of_variation
FROM property
GROUP BY propclass
ORDER by **coeff_of_variation** DESC

	propclass	min	max	avg_sqft	stddev	coeff_of_variation
1	Vacant	576.0	189860.0	4544.42028985507	22701.2893618051	4.99542029870857
2	Community Srv...	285.0	4252863.0	138654.196335079	633545.457921424	4.56924835069807
3	Public Services	240.0	121150.0	6962.36842105263	19947.51506223	2.86504733100783
4	Commercial	260.0	622822.0	8436.89855072464	22142.3593530339	2.62446670656389
5	Industrial	432.0	462839.0	63981.7534246575	130261.706101365	2.03591960409081
6	Recreation	540.0	31440.0	6693.11428571429	6966.04952306284	1.04077851142197
7	Agriculture	468.0	31517.0	2378.77022653722	2179.55956968649	0.916254771213...
8	Forest	900.0	5492.0	2735.75	1521.89075733862	0.55629745310742
9	Residential	208.0	13481.0	1813.17196939186	710.339478875653	0.391766192543...

Table 8. Example from a query that performs aggregate calculations on numeric fields in a database, and also calculates a statistical formula for the coefficient of variation. Source: author.

By simply adding the GROUP BY clause and an in-query calculation called the coefficient of variation (CV) (McGrew, et. al., 2014), the query created a simple model of variability

among land use types for building size. The resulting table shows a relative stability of building sizes for residential properties (CV = .39), and a larger CV for vacant (4.99) and community service buildings (4.5). Again, in these queries, the technical details for sorting, grouping, and summarizing data are hidden from the user, allowing the user to focus on relatively simplistic commands.

3.7 Joining Tables

SQL syntax may simultaneously perform queries on multiple tables, using a common identifier between the tables. In our example, the property and owner table share a common identifier: parcelkey. A simple query can obtain records from both tables by using the common identifier:

```
SELECT ownership.owner, ownership.value, property.propclass
FROM property, ownership
WHERE property.parcelkey = ownership.parcelkey
```

	value	owner	propclass
1	130400.0	MOORE, DERRI...	Residential
2	600000.0	TOMPKINS TRU...	Commercial
3	139000.0	KILEY, DARBY K	Residential
4	307500.0	MCCUNE FAMI...	Commercial
5	190000.0	TUMBAR, TUD...	Residential
6	205000.0	ANDERSON, ELI...	Commercial
7	395000.0	THRASH, BRIA...	Residential
8	176000.0	ABELE, ROBERT J	Residential
9	75000.0	SPRAGUE, JUDY S	Residential
10	160000.0	BORG, THOMAS	Commercial

Table 9. Example from a query that queries and presents records from two separate tables, joined by a common column. Source: author.

Further, intersection (∩) and union (∪) operations defined in set theory may be applied as:

```
SELECT parcelkey
FROM property
INTERSECT
SELECT parcelkey
FROM ownership
WHERE value > 50000
```

where the parcelkey field values are common to both tables. Also, one can use a simple join to obtain more fields as:

```
SELECT property.address, ownership.value, ownership.owner
FROM property, ownership
WHERE value > 50000
AND property.parcelkey = ownership.parcelkey
```

	address	value	owner
1	168 CHESTNUT ST	130400.0	MOORE, DERRICK
2	110-12 SENECA ST W	600000.0	TOMPKINS TRUST COMP...
3	310 TITUS AVE N	139000.0	KILEY, DARBY K
4	114 SENECA ST W	307500.0	MCCUNE FAMILY PROP, L...
5	205 RICHARD PL	190000.0	TUMBAR, TUDORITA
6	308 FAIRMOUNT AVE	205000.0	ANDERSON, ELIZABETH J
7	315 ELMWOOD AVE	395000.0	THRASH, BRIAN L
8	132 DURFEE RD	176000.0	ABELE, ROBERT J
9	204 WARREN PL	75000.0	SPRAGUE, JUDY S
10	431 SENECA ST E	160000.0	BORG, THOMAS

Table 10. Example from a query that queries and presents records from two separate tables, joined by a common column but has the added condition of restricting the result to records that meet a particular criteria. Source: author.

In this example, the address field is taken from the property table, while the value and owner field are taken from the ownership table. The tables themselves are linked with an additional criteria where the property.parcelkey = ownership.parcelkey. The query also obtains a common subset from both tables based on the criteria of property values being greater than $50,000.

3.8 Modification Operations

While most users will conduct query operations, other users can leverage SQL to modify data in a database. For example, one might update the owner name in the ownership table as:

```
UPDATE ownership
SET owner = 'LEMBO, ARTHUR'
WHERE parcelkey = '5060700532'
```

Or, mathematical operations may be performed, such as increasing all property values by 23%:

```
UPDATE ownership
```

```
SET value = value * 1.23
```

Similar to queries, tables may be joined to perform calculations based on criteria in another table. For instance, property values for only commercial properties could be increased by 23%:

```
UPDATE ownership
SET value = value * 1.23
FROM properties
WHERE property.parcelkey = parcelkey
AND property.propclass = 'Commercial'
```

3.9 Data Definition Operations

Finally, the structure of the data and tables themselves may be modified with SQL. Although this discussion focuses on attribute queries with SQL, it should be noted that SQL does in fact provide the opportunity to define the structure of the data itself. These modifications include creating new tables, adding, removing, or altering existing tables. For instance, one could add a new column to the ownership table to include an integer field for compliance in some zoning requirement. Typically, these types of queries utilize the ALTER statement.

```
ALTER TABLE ownership
ADD COLUMN compliance INTEGER
```

Similarly, SQL may be used to create the tables in this discussion

```
CREATE TABLE ownership (parcelkey varchar,
owner varchar, value numeric, compliance integer)
```

and finally populate values by adding records to the table

```
INSERT INTO ownership (parcelkey, owner, value)
VALUES ('5060700532', 'LEMBO, ARTHUR', 386000.00)
```

4. Conclusion

For almost a half century, the implementation of SQL within relational database management systems has clearly achieved its goal of allowing ordinary users to interact with a computer in a reasonably high-level, non-procedural query language to perform sophisticated data interrogation without the need to write complex computer code. This document illustrates some of the basic capabilities and syntax of using SQL for data query, modification, and definition. Readers should see the simplicity of the syntax and have confidence that with a moderate level of effort, they should be able to perform relatively sophisticated interrogation of their own databases.

References

Arlinghaus, S. L. (2019). Set Theory. The Geographic Information Science & Technology Body of Knowledge (2nd Quarter 2019 Edition), John P. Wilson (Ed.).

Chamberlain, D. and Boyce, R. F. (1974). SEQUEL: A structured English query language. SIGFIDET '74: Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control. Workshop, Ann Arbor, Michigan, May, 1974

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.

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

McGrew, J. C., Lembo, A. J., and Monroe, C. (2014). An Introduction to Statistical Program Solving in Geography, 3rd Edition. Waveland Press.