
GeomAttribute Documentation

Philip Whitten

Dec 11, 2018

CONTENTS:

1	About	3
2	Tools Included With This Plugin	5
2.1	Geometry Attribute Table	5
2.2	Load Data	6
3	A QGIS attribute table plugin to show empty and null shapes	7
3.1	Preamble	7
3.2	Introduction	7
3.3	Background	8
3.4	PROBLEM DEFINITION	23
3.5	Future Development	29
3.6	Bibliography	30
4	API	31
4.1	modelVectorLayers module	31
4.2	parseQGISGeometry module	31
4.3	zipGeomAttribute module	32
5	Changelog	33
5.1	Version 0.1.0	33
5.2	Version 0.1.1	33
6	Glossary	35
7	Indices and tables	37
	Bibliography	39
	Python Module Index	41

CHAPTER ONE

ABOUT

GeomAttribute version 0.1.0.

A QGIS 3 plugin created by Philip Whitten that is licensed under the GNU General Public License v3.0.

The plugin's repository is on [GitHub](#).



TOOLS INCLUDED WITH THIS PLUGIN

A QGIS 3 plugin that shows an attribute table which includes a geometry column for any vector layer.

This plugin has the following tools:

2.1 Geometry Attribute Table

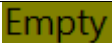









2.1.1 Description

An attribute table window which includes a geometry column for any vector layer.


The geometry column uses the icons shown in the table below to indicate the feature's geometry type in descending precedence: unknown (null); empty; point; multi-part point; line; multi-part line; polygon; and, multi-part polygon.

Table 2.1: Icons used to represent various geometries.


Icon	Geometry
	Empty geometry
	Unknown geometry
	Point
	Multi-Part Point
	Line
	Multi-Part Line
	Polygon
	Multi-Part Polygon

2.1.2 Use

This tool works on the current active layer within QGIS. The current layer needs to be a vector layer. This tool will not work on a raster layer.

Click on the vector layer in the *Layers Panel* then click on the *Geometry Attribute Table* icon  or select *Geometry Attribute Table* from the *Plugins* menu.

2.2 Load Data

The  *Load Data* function creates the following QGS memory vector layers:

- Multi-part Lines
- Multi-part Points

These layers are ideal to observe the utility of the *Geometry Attribute Table*.

The methods that create these layers are contained within the *modelVectorLayers module*.

A QGIS ATTRIBUTE TABLE PLUGIN TO SHOW EMPTY AND NULL SHAPES

DRAFT

3.1 Preamble

I am in the closing stages of a MGIS at Penn State and needed a topic for capstone project. What sort of topic?? I am gainfully employed as a GIS Officer at a local government bureau, but, my employer was not willing to offer a topic nor were they willing to offer open access to curious datasets. I have a passion for maps, cartography and spatial analysis, but, previous attempts to combine my passion with requirements for academia often ended in a feeling of futility. From my employment, I realised that my colleagues and I occasionally loose hours because some of our datasets were **corrupted** by *null* or empty geometries. I also realised that I was blissfully ignorant of the concept of *null* and empty geometries as they were not included in my GIS education. So, I wanted to learn a bit about *null* and empty geometries in the hope that I can be more capable in my employment and so that I can feed a curiosity.

The concept of a QGIS plugin that exposed *null* and empty shapes evolved quickly. A plugin is tangible and deliverable as opposed to curiosity which is instinct and always expanding. QGIS is the primary desktop GIS software where I work and I was keen to learn how to automate and develop QGIS for specific tasks. In my GIS degree we had used Python to automate tasks in ArcMap, and, C# to make apps in ArcMap, but, we had not dabbled in QGIS. So, this project was also an adventure in QGIS scripting.

What comes next? From this project I have learnt that much of the GIS software and the GIS community is GIS centric. After all, to create a new record in a dataset with either QGIS or ArcMap one creates the shape of a feature, and, subsequently populates the other attributes. To work in reverse and create the shape of a feature as the last step in a workflow is not only not conventional, but, also not trivial. But here's the point, whenever I have to create geospatial records I am always given a list of attributes and the shape is the last attribute to be created, the other attributes are already known and are re-entered. Furthermore, most of the major desktop GIS packages report set operations between datasets differently to established database conventions. I would like to work towards making QGIS and other software packages less GIS centric so that geospatial datasets become more ubiquitous and more open to other professional communities! To become less GIS centric you need to embrace the *null* and exploit the empty.

3.2 Introduction

A dataset is a collection of records, where each record has a defined number of elements and the data type of each element is also defined. The elements are commonly referred to as attributes. For example, a dataset

of street names may consist of records have the attribute heading “*Name*” of type variable character and the character heading “*Street Type*” of type street type. In this example, street types may consist of the values (*Lane*, *Road*, *Street*, ‘’). The first three of the values *Lane*, *Road* and *Street* are obvious – but what about the last value of ‘’? ‘’ **is a deliberate empty set of character values!** In our world there are many streets that have no type, and, indeed there are many that have no name. Here, the use of an empty set of characters, or ‘’, indicates that the Street Type attribute has no type, not that the type is unknown (null).

Geospatial datasets contain one or more values that refer to a location on earth. For the majority of geospatial datasets, the location consists of one or more points, lines and/or polygons that are referenced to a coordinate system that is a projection of the earth’s surface.

For the purposes of this project, any dataset element type that stores the geospatial shape with respect to a referenced coordinate system is referred to as a geometry. Any empty geometry element is simply a geometry that doesn’t have a shape.

In many enterprises geospatial datasets are contained within enterprise databases where frequently the same brand of enterprise database is used elsewhere within the same enterprise to contain non-spatial datasets. For example, a local government office may use one or more MicroSoft SQL Server installations (MS SQL) as a dataset repository for: a content management system; a customer relationship management system; a land management system; an asset management system; and, a GIS system.

QGIS [3] is a computer program that among other things is used to view, create and edit the geometry elements within geospatial datasets. For many dataformats QGIS QGIS does not: Pass null and empty geometry elements equivalently for different data storage formats; does not show directly show which records within a dataset have *null* or empty geometry elements; and, does not always process null and empty geometry elements as specified by standards.

This project aims to create a QGIS plugin that illuminates null and empty geometry elements in geospatial datasets.

3.3 Background

3.3.1 Relational Datasets

Set

A set is a collection of distinct objects. For example, a box of apples is a set of apples, and, the set of countries in North America consists of Canada, United States of America, and, Mexico. By convention, sets are symbolized by enclosing within curly brackets. Hence:

$$\textit{North American Countries} = \{\textit{Canada}, \textit{United States of America}, \textit{Mexico}\}$$

A dataset is any set where each element is restricted to one data type and where each element belongs to the same universal set.

Relational Dataset

A relational dataset is a collection of sets where:

1. The number of objects (elements) in each set is the same.
2. A one to one relationship exists between elements of different sets.

An example relational dataset showing the country name, country abbreviation and country population for the countries in North America is shown in Table 3.1. This relational dataset comprises of the three sets: a set of country names; a set of abbreviations; and, a set of country populations. A one to one relationship exists between the elements for each of these three sets. Hence, the country with the name of *Canada* has a one to one relationship with the country abbreviation *CAN* and the country population *3563000*.

Table 3.1: Example of a dataset of Countries in North America.

Country Name (letters and spaces)	Abbreviation (upper case letters)	Population (integer)
Canada	CAN	3563000
United States of America	USA	32663000
Mexico	MEX	12458000

Essential to any set is a definition or description of what type of objects can be a member. For example, an apple which is a valid type of the set of fruit can't be a member of a set of countries. For any dataset, both the data type and additional constraints are often used together define the universal set. For example, the data type for *Country Name* in Table 3.1 is any combination of letters and spaces, whilst the data type for *Abbreviation* is any combination of 3 upper case letters.

Although a *relational dataset* consists of multiple sets of data where the elements of each set are related, it is ubiquitously referred to as a **dataset**.

Geospatial Dataset

A geospatial dataset refers to any dataset where one or more of the composite sets refer to a location on the earth's surface. This project is only concerned with those geospatial datasets where the location on the earth's surface is represented by one or more points, lines or polygons that are located by coordinates and stored as vectors. These points, lines and polygons are collectively referred to as shapes or geometries⁵. Datasets that include one or more sets of geospatial vectors are referred to as *Vector Datasets* by the GIS community.

A vector geospatial dataset is a subtype of geospatial dataset where the geospatial sets can be located graphically on the earth's surface by the application of coordinates. This project is only concerned with vector geospatial datasets.

3.3.2 Geometry data types

All datasets contain some restriction on the type of data each constituent set may contain. From a software perspective, a restriction of type is essential for minimizing both the storage size of the dataset and the response time for a dataset query. Analogous to specific data types for storage of numbers, text or dates there are data types specifically used for the storage of geospatial geometries [5]. Similarly, just as there are often specific data types of signed and unsigned integers, float, and, decimal numbers there are also specific data types for different types of geometries, with the type often referring to how the geometry is constructed.

⁵ For SQL relational databases, the term geometry is restricted to those shapes that are located by cartesian coordinates.

For any dataset software the geometry data types that are available for use can be shown schematically as a hierarchy like the one shown in Fig. 3.1. Within this hierarchy, the possible data types are described by the labels in the boxes. Essential to all such hierarchy's, a set of data of a declared type may consist of any type below it on the hierarchy. Hence, if a set of data has a declared type of *Geometry Collection* then any data element within it may consist of *Geometry Collection*, *Multi-Point*, *Multi-Curve**, *Multi-Line*, *Multi-Surface*, and, *Multi-Polygon*. Similarly, if a set of data has a declared type of *Point* then it may not contain a *Polygon* nor a *Line* as neither are below the sub-type *Point* on the hierarchy.

The single part constrained geometry subtypes in the lower part of Fig. 3.1 are referred to as *Primitive Types* and must contain only one single part geometry per set element. In contrast, the Multiple Part geometries may consist of one *or* many parts per feature. For example, a feature of the “*Multi-Point*” geometry sub type may have one point, no points or multiple points. Another characteristic of the single part primitive types is that the *Line* and *Polygon* subtypes may only exist of straight line segments between coordinates.

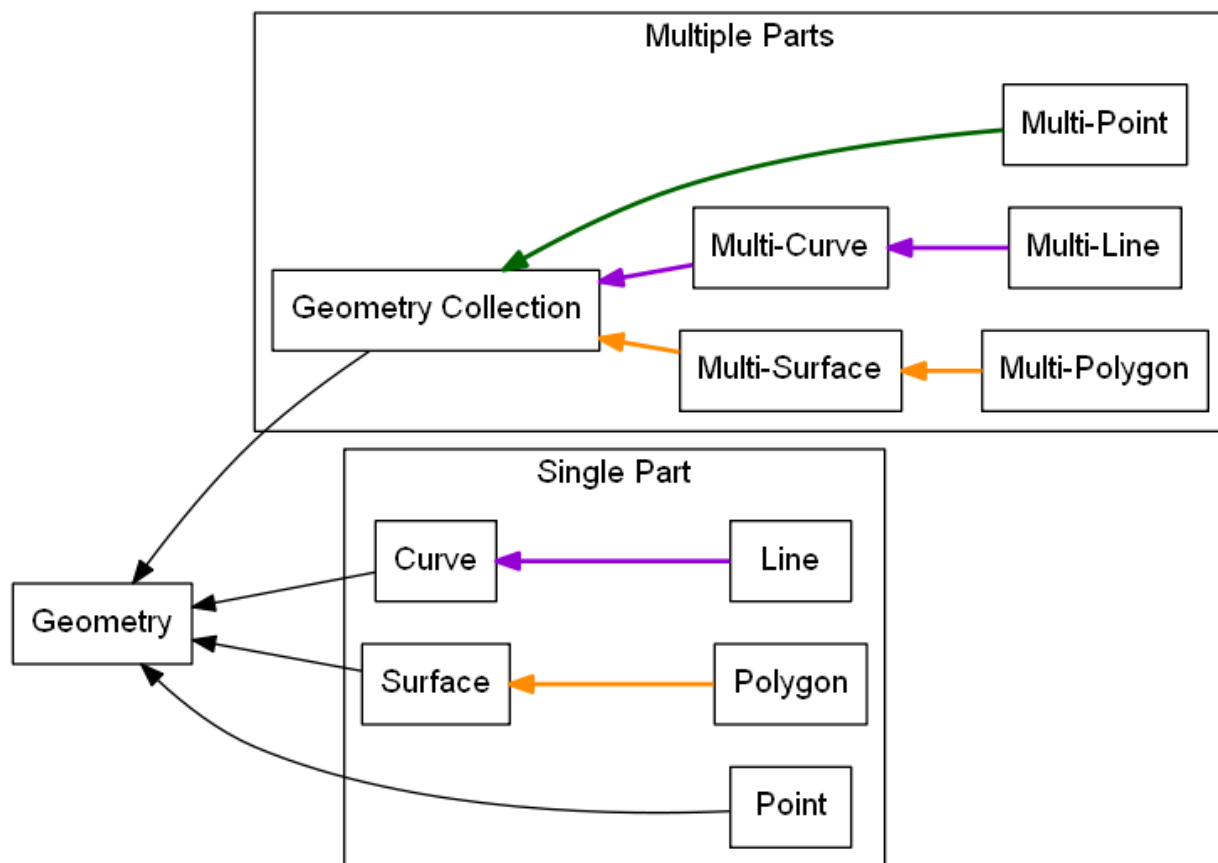


Fig. 3.1: GIS Geometry subtype hierarchy. Adapted from [5]. The more conventional term “*LineString*” that is used in the QGIS API and [5] is replaced here with “*Line*” for clarity.

In reality there may be many more geometry subtypes than the simplified hierarchy shown in Fig. 3.1. For example, some common additional subtypes for datasets are created for sets of geometries that incorporate elevation, or, for lines that are constructed from curves as opposed to straight line segments.

Many GIS data format standards, and, many GIS software have a geometry subtype hierarchy that is **similar** with Fig. 3.1. Within a GIS dataset, geometry objects there are several geometry subtypes, with the

main ones without elevation are shown in Fig. 3.1. Schematically, this hierarchy of geometry subtypes is replicated by the [inheritance diagram](#) for `QgsAbstractGeometry` [12].

3.3.3 Geometry data values

For any data type there exists a universal set of valid values. For example, a set of birthday dates must be restricted to valid dates. For example, a birthday on the 30th of February is not valid as the 30th of February is not part of the universal set of dates. Similarly, a valid geometry needs to be located within the boundaries of the coordinate system that it is referenced to. **Empty** and **null** are two values that may be part of any set of data and could be fairly described as being:

1. Controversial
2. Miss-understood
3. Best avoided

Empty

A box of apples can be described as a set of apples. A empty Apple box is an empty set of apples. An empty geometry element is a geometry that has no coordinates. Whether an empty element is a valid member of a set depends on the context. For example, if a study of chickens hatching from eggs recorded the date that each chicken hatches for a set of 10 eggs than the hatch date element is of the hatch date set is empty before the chicken hatches. It is *known* that the chicken has not hatched.

All empty sets including an empty geometry value are place holders for when it is *known* that an element does not exist [19]. For example, consider the intersection of the Blue Crosses and the Red Circles with the two squares shown in Fig. 3.2. Both of the Blue Crosses B1 and B2 intersect the Left square, and, the Blue Cross B3 intersects the Right square. The intersections of the Squares and Blue Crosses, and the Squares with Red Circles are summarized by the datasets shown in Table 3.2 and Table 3.3. As shown in Table 3.2 the *Left Square* intersects with the *Blue Crosses* B1 and B2 which are represented as a subset $\{B1, B2\}$. Similarly, it is reported in Table 3.3 that the *Left square* intersects the subset of *Red Circles* $\{R1\}$. In contrast, also in Table 3.3 it is shown that the *Left square* does not intersect with any *Red Circles* as shown by the empty set $\{ \}$. Here the empty set $\{ \}$ shows that it is known that no intersection occurs. The reporting of those combinations where intersections are known to not occur as shown Table 3.3 follows the convention used by most SQL type relational databases for all set intersections regardless of whether they are geospatial or not. In contrast, the convention for many GIS desktop software including QGIS and ArcGIS is to only show those combinations where intersections are known to occur (are True), hence, Table 3.3 is reported as Table 3.4 by QGIS.

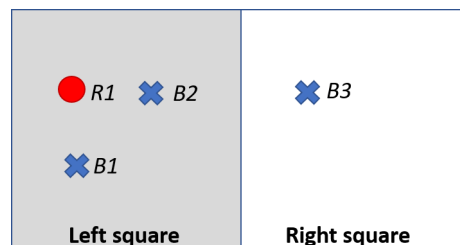


Fig. 3.2: The location of blue crosses and red circles in the “Left Square” and the “Right Square”.

Table 3.2: The intersection of the the *Squares* and the *Blue Crosses*.

Square	Blue Crosses
Left square	{B1, B2}
Right square	{B3}

Table 3.3: The intersection of the the *Squares* and the *Red Circles*.

Square	Red Circles
Left square	{R1}
Right square	{ }

Table 3.4: The intersection of the the *Squares* and the *Red Circles* as shown by QGIS.

Square	Red Circles
Left square	{R1}

The real utility of empty geometry values is realised when the intersection of all the squares and both types of points (*Red Circles* and *Blue Crosses*) are collated in one dataset as shown in [Table 3.5](#) as opposed to [Table 3.6](#). By using the empty set { } as a place holder for the known non-intersection of *Red Circles* with the *Right square* the sets of *Blue Crosses* and *Red Circles* are maintained as separate columns. Although this approach is efficient and intuitive it is not suitable when there is a large number of point types as the number of columns has a linear relationship to the numbs of points.

Table 3.5: The intersection of all point types and the squares. Note that the sets each point type are still maintained as individual columns. This approach is not feasible for a large number of point types as there would be too many columns.

Square	Point type	
	Blue Crosses	Red Circles
Left square	{B1, B2}	{R1}
Right square	{B3}	{ }

Table 3.6: The intersection of all point types and the squares.

Square	Point type	Geometry
Left square	Blue Crosses	{B1, B2}
Right square	Blue Crosses	{B3}
Left square	Red Circles	{R1}

Datasets with multiple geometry sets

Much of the GIS community work with the restriction of a single geometry set for each dataset (a single geometry column within a table). It is difficult to have multiple geometry attributes without also allowing

empty geometry attributes. Next, I will examine the advantages and disadvantages of multiple geometry attributes.

The fundamental advantage of multiple geometry attributes is they facilitate topology. Topology refers to how the constituent parts of a system are interrelated or arranged. The location of points within squares shown schematically in [Fig. 3.2](#) is an example of topology as it shows how the points are related to squares. [Table 3.5](#) shows the topological association of point type by square type, but, uses two geometry columns to do so. [Table 3.7](#) shows the same data as [Table 3.5](#) using only one geometry column. Examination of [Table 3.7](#) reveals that the relationship between a type of point (e.g. Blue Crosses) and the Square type (e.g. Left Square or Right Square) has to be reported as two separate relationships (two separate records) and a user is left with the task of mentally connecting these two relationships. Clearly, without using multiple geometry columns showing topological relationships is less intuitive.

Table 3.7: Another way to represent the data in [Table 3.5](#) that uses only one geometry column.

Square	Point type
Left square	Blue crosses $\{B1, B2\}$
Left square	Red circles $\{R1\}$
Right square	Blue crosses $\{B3\}$
Red circles	red circles $\{Empty\}$

The major disadvantage of multiple geometry columns is that they are not supported by many pieces of GIS software or GIS file formats. For example, ArcGIS does not support multiple geometry columns in any capacity, QGIS treats each geometry column as an unrelated dataset, and, the ubiquitous shapefile can only contain one geometry column. So, by adopting multiple geometry columns you are isolating yourself from a large portion of the GIS community.

null

null is the most common value (element) recorded many disciplines and software formats for *unknown* data values⁶. For example, if a study of chickens hatching from eggs recorded the hatch date of each chicken hatching for a set of 10 eggs than the hatch date element of the hatch date set is *null* (unknown) if the hatch date was not recorded. Strictly speaking a *null* hatch date can be any value from the universal set of hatch dates including *Empty* allowing for eggs that never hatched.

The most useful feature of *null* values is that they enable incomplete datasets. For example, consider the Blue Crosses dataset shown in [Table 3.8](#) where the coordinates for B4 are unknown. Datasets like [Table 3.8](#) can stem from requests to georeference existing datasets where the georeferencing is incomplete.

⁶ Python uses *None* instead of *null*, but, PyQt uses *NULL* as a QVariant, so, PyQGIS script may have a mixture of *None* and *NULL* depending on the origin of the classes in use.

Table 3.8: The age, size and coordinates for the Blue Crosses.

Blue Cross	Age (years)	Size	Coordinates
B1	2	Big	(1, 1)
B2	2	Small	(2, 2)
B3	3	Small	(4, 2)
B4	8	Big	<i>null</i>

Using joins to eliminate null

Many GIS datasets do not allow *null* geometries. Having a dataset constraint that prevents *null* geometries does not imply that the geometries are all known, it only means that the dataset can't have a *null* geometry. The prevention of *null* geometries without knowing all of the geometries is achieved by using multiple datasets that include a geometry only dataset that has a relationship with a non-geometry dataset as shown in Fig. 3.3. The relationship is typically achieved by the use of a **key** that is used in all related datasets to distinguish each relationship across the datasets. Joins refer the process of forming a new dataset from multiple datasets by the use of a relationship. The dataset shown in Table 3.8 can be created from the datasets shown in Fig. 3.3 by application of an *outer join*.

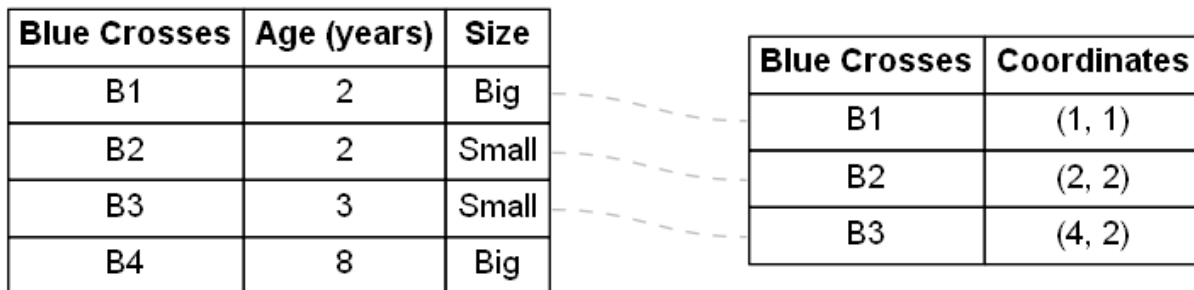


Fig. 3.3: Table 3.8 presented as two separate datasets where *null Coordinates* data values are not permitted. The *Blue Crosses* key is used to register relationships between the two datasets that is symbolised by the grey dashed lines.

Reasons for preventing null geometries

The reasons for preventing null geometries include: it's keeps GIS feature creation simple: one can't map *null*; they permit the geometry value to be a variable; and, it keeps logic based algorithms simple.

For QGIS and many desktop GIS systems, records are created by drawing the geometry and subsequently entering the other data values. This geospatial geometry centered approach intuitively keeps the related computer programming simple in comparison to any approach that allows a user to enter any of the data values including the geometry in the sequence that they choose.

There is no accepted universal approach to mapping a *null* geometry. It is obvious that if a geometry is *null* then using a defined symbol at a single location is a miss-representation. There is active research into approaches for mapping the unknown.

The use of a geometry specific dataset enables geometry to be a variable. For example, take the Blue Cross B1. This Blue Cross may represent a boat in a sea. Hence, at different points in time, B1 may have different

coordinates (Table 3.9). Most geospatial datasets have geometries that are variable as our technology for recording and knowing location is improving. For example, an allotment of land may be static as it is relative to reference points, but, the mapped location and hence the recorded geometry for the allotment of land will change as the location of the reference points is refined to a higher accuracy. Whether a dataset owner should track changes to a geometry is a question that is beyond the scope of this discussion.

Table 3.9: The coordinates of the Blue Crosses for yesterday and today.

Blue Cross	Coordinates	
	Yesterday	Today
B1	(1, 1)	(2, 2)
B2	(2, 2)	(1, 2)
B3	(4, 2)	(4, 1)

Whether a dataset allows *null* values or not directly affects the type of logic applied to the dataset for set operations. Boolean logic, also referred to as two value logic, allows only for True or False answers to set operations. Boolean logic can't be applied when the answer is unknown. When *null* values are permitted, three value logic is required for set operations. The intersection of the squares with the two subsets of blue crosses {B1, B2, B3} and {B1, B2, B3, B4} that are described in Table 3.8 is shown in Table 3.10. For {B1, B2} it is True that they intersect the Left square, whilst it is also True that {B3} does not intersect the same square, however, it is null (unknown) whether {B4} intersects the Left Square. Compounding the implementation of Three value logic is the fact that different database platforms implement three value logic differently leading to widespread avoidance of three value logic regardless of whether the data type is geometry or something more generic like an integer. In summary, even when *null* values are permitted in datasets, the records associated with them are typically excluded from set operations.

Table 3.10: The intersection of the squares with the subsets {B1, B2, B3} and {B1, B2, B3, and B4} as shown in Table 3.8.

Square	Intersection with the subset	
	{B1, B2, B3}	{B1, B2, B3, B4}
	Boolean logic	Three value logic
Left square	{B1, B2}	{B1, B2, null-B4}
Right square	{B3}	{B3, null-B4}

Records with false Boolean logic

Many GIS software packages force the user to perform a selection whenever they perform a set operation like a union, intersection or difference. The results of an intersection without a selection of the points and squares shown in Fig. 3.2 includes those records that don't intersect (Table 3.7). To achieve the more common output shown in Table 3.11 a selection must be performed on the data to include only those records that do intersect. Hence, the logic performed by most GIS software packages including ArcGIS and QGIS is: 1. Select those records where intersect is True; 2. Perform intersection.

Table 3.11: Another way to represent the data in 5 where records where no intersection occurs are also shown.

Square	Point type
Left square	Blue crosses $\{P1, P2\}$
Left square	Red circles $\{P3\}$
Right square	Blue crosses $\{P4\}$

Being forced to do a selection in addition to an intersection is an algorithmic solution that eliminates the need for empty geometry values. Forcing a selection as part of a set operation introduces the assumption that records missing from the results are empty (e.g. do not intersect) and hence prohibits the ability to also record unknown (*null*) set operations. Another problem of forced selection is that problem solving of erroneous set operations is hindered as one needs to undertake additional steps to confirm those records that were explicitly excluded by the forced selection. For logic and critical thinking analysing the negative results can be more fruitful than analysing the positive results!

Although beyond the scope of this project, it is noted that although Boolean logic is applied the same in the majority of relational databases there is known diversity in the application for three value logic.

Reasons for allowing null geometries

Although the majority of GIS systems do not promote the use of null geometry data values, there are several reasons why they should be adopted: they force the adoption of geospatial datasets by enterprise databases; they illuminate the unknown; and, they facilitate a simpler dataset structure.

The forced adoption of geospatial datasets by enterprise databases is the most compelling reason to adopt null geometry values. For example, consider a commercial database used for rates at a local government. The current approach of most databases for rates is to mimic [Fig. 3.3](#) where the tables for the land registry are maintained within a land registry database, and, the tables for the georeferenced land allotments are in a GIS database and these databases are joined. This approach facilitates bureaucracies where separate teams of people maintain each database, and, where the land registry database does not adopt spatial functionality as the data is located in a separate database. Whilst using a separate GIS database allows the local government to develop a custom GIS system, it comes at a cost of minimal inbuilt spatial capability.

Allowing *null* has the potential to expose the unknown. By exposing the unknown, it is evident where further data capture is required, and, it is more likely that any spatial analysis will also establish the degree of dataset completeness.

A disadvantage of the lookup table approach is the requirement for the documentation of database schema for users to maintain the lookup key and to perform database queries that include spatial and non-spatial attributes. Without lookup tables the database structure is simpler leading to fewer errors and fewer joins when constructing queries.

3.3.4 Parsing data by QGIS data Providers

A component of software that allows reading and editing of datasets without the user importing to or exporting from another data format is called a data provider. QGIS parses data to and from many data formats without requiring importing or exporting to or from a common data format. The algorithms for this data parsing are contained within 1 of 18 different QGIS data providers. A data providers is specific for one or

more data formats. The data providers are scripted within C++ without python handles. To function, the relevant data provider need to read and write each data type and the appropriate values for each set (column) included in the dataset. The intent of this data parsing is that a user may read, write and analyse data stored in different formats with a common QGIS user interface.

The parsing of datasets by QGIS data providers is silent and can change both geometry sub-type and the geometry data values of *null* and *Empty* (Table 3.12). The change of geometry sub-type from line to multi-line for Geopackage, Shapefile and Spatialite is necessary as none of these data formats permits the mixing of primitive geometry types within multi-part geometry sub-types. Except for the PostGis format, either the parsing of the known geometry data value of *Empty* is changed to the unknown value of *null*, or, visa versa. The replacement of known with unknown, or unknown with known will cause erroneous analysis and interpretation. Without experience errors may be introduced into datasets by the parsing of data by QGIS's data providers.

Table 3.12: Appending of non-empty single part multi-line, non-empty line, empty line and null geometry records by QGIS to 5 popular data formats.

Original	Geopackage	Shapefile	Spatialite ⁹	PostGis ¹⁰	MS SQL ¹⁰
Multi-Line	Multi-Line	Multi-Line	Multi-Line	Multi-Line	Multi-Line
Line	<i>Multi-Line</i>	<i>Multi-Line</i>	<i>Multi-Line</i>	Line	Line
Empty	Empty	<i>null</i>	Empty	Empty	<i>null</i>
<i>null</i>	Empty	<i>null</i>	Empty	<i>null</i>	<i>null</i>

Parsing Data Types

Although different GIS data formats have **similar** geometry subtype hierarchy's, they are not the same. The most typical deviations between different hierarchy's are the selection of sub-types that are included, and, which sub-types can be directly instantiated.

Set of Geometry Sub-Types

Most data formats have a unique set of geometry subtypes available. It is the data provider's task to:

- associate sub-type within an external data format with a QGIS sub-type
- provide an algorithm to parse data to and from the external data's sub type to the QGIS sub-type
- for many data formats this association and parsing is per record, not per dataset.

The number and type of geometry Sub-Types are different for many data formats. The data providers needs to associate each geometry subtype in an external format with an equivalent or similar QGIS sub-type. For example, the hierarchy for the ubiquitous Shapefile shown in Fig. 3.4 is vastly different to the conventional geometry hierarchy shown in Fig. 3.1. For the purpose of constructing a single part line, QGIS has the four geometry subtypes of "*Multi-Line*", "*Circular-String*", "*Compound-Curve*", and, "*Line*", whereas Shapefile has only the single subtype of "*PolyLine*". Hence, in an QGIS editing session, a user may create a line using any of QGIS's four line subtypes, but, the line will only be recorded as a "*PolyLine*" and it is the

⁹ SpatialLite table has a *Multi-Line* geometry data type.

¹⁰ The geometry type saved by PostGis and MS SQL Server depends on geometry constraints within the databases, the use of a *Geometry columns* lookup table, and, what geometry types already exist within the respective datasets.

Provider’s task to inform QGIS of this requirement (Fig. 3.5). In comparison to a shapefile, SpatiaLite has the “Multi-Line” and “Line” subtypes and QGIS has to count the number of geometry parts (the number of separate lines) if the “Line” subtype is specified for SpatiaLite (Fig. 3.5). Although QGIS’s data providers are able to instruct QGIS to change a geometry sub-type upon saving the edit, they do not stop a user from creating a geometry sub-type that is incompatible with the stored data format. For example, with QGIS in an edit session, one may create a line with multiple parts for a single part “Line” dataset, and, it is only when the user attempts to commit the edits by saving the dataset that QGIS throws an error.

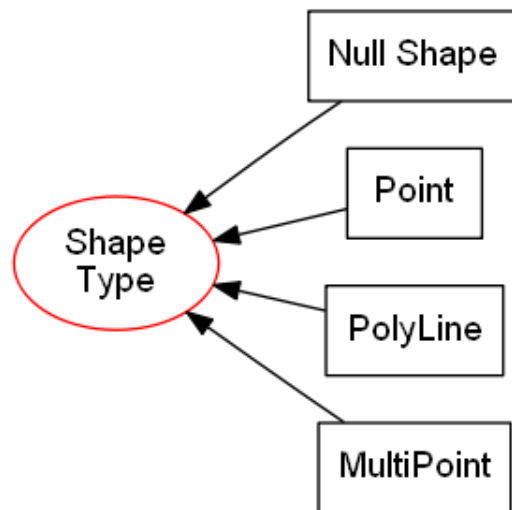


Fig. 3.4: The hierarchy of geometry types for a shapefile excluding those that include elevation or measurements (adapted from [17]). The *Shape Type* can’t be instantiated directly and is included for consistency.

SpatiaLite and Shapefiles have a single geometry sub-type defined for a dataset⁸ which is simpler than formats like MS SQL Server and PostGIS where each record maybe a different sub-type.

Abstract Geometry Sub-Types

Regardless of data-type, those data types that can’t be instantiated directly are referred to as *Abstract*. For example, without imposing constraints, PostGIS and MS SQL Server by default have a *Geometry* type⁴ for all geometry datasets. Consequently, PostGIS or MS SQL Server the geometry sub-type can vary for each record. For example, a single geometry set in PostGIS can contain point, line and Multi-Surface sub-types, thereby mixing primitive and multi-part branches, and, mixing geometry types of different dimensionality. In comparison, the *Geometry* subtype equivalent in the QGIS data type inheritance diagram [12] has the title “*QgsAbstractGeometry*” where the *Abstract* keyword implies that this datatype can’t be created by itself, only it’s children (sub-types) can be created.

For several dataproviders including QGIS memory layer, PostGIS and MicroSoft SQL Server QGIS employs a subtype hierarchy shown by Fig. 3.6. Hence for some dataproviders, by default, QGIS creates a new one part line in a dataset of the “*Multi-Line*” subtype as a Line and not a “*Multi-Line*”. The insertion of a “*Line*” geometry subtype into a “*Multi-Line*” geometry subtype dataset is readily demonstrated by python

⁸ Technically in a Shapefile the geometry sub-type is recorded for each record, but, the technical specifications state that “All non-null shapes must be of the same shape type” [17].

⁴ In many QSL databases the hierarchy separates all geometry types that employ cartesian coordinates from those that employ geographic coordinates.

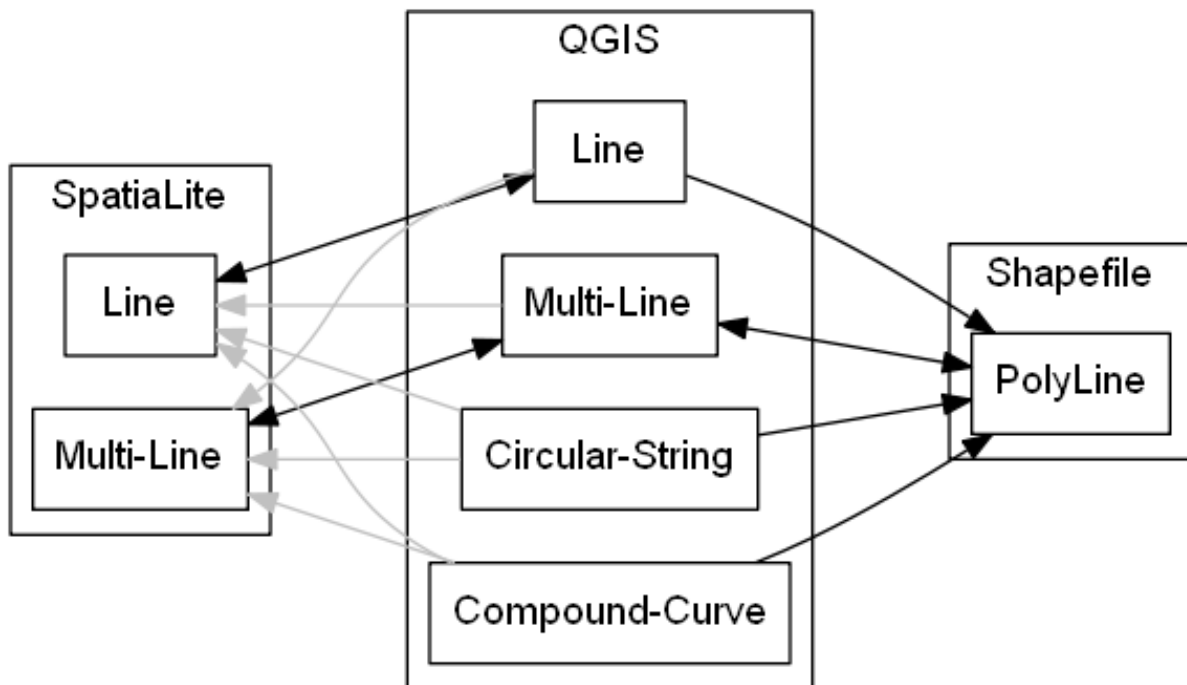


Fig. 3.5: The association of various geometry line sub-types for SpatiaLite, QGIS and Shapefile: grey arrows refer to changes in geometry sub-type within QGIS prior to committing data; black arrows indicate data parsing between QGIS and the external formats.

script using the QGIS API:

```
>>> from qgis.core import *
>>> layerMulti=QgsVectorLayer('MultiLineString?crs=epsg:4326&field=ID:string',
↪ 'a', "memory")
>>> providerMulti=layerMulti.dataProvider()
>>> recordWrite = QgsFeature()
>>> recordWrite.setAttributes(['1'])
>>> recordWrite.setGeometry(QgsGeometry.fromWkt('LINESTRING (1 1, 6 1)'))
>>> providerMulti.addFeature(recordWrite)
True
>>> recordRead = layerMulti.getFeature(1)
>>> print(QgsWkbTypes.displayString(recordRead.geometry().wkbType()))
LineString
>>> print(QgsWkbTypes.displayString(layerMulti.dataProvider().wkbType()))
MultiLineString
```

More worryingly, as shown in the next python script, the reverse is also possible. One may add a feature with a *Multi-Line* geometry sub-type to a *Line* dataset.

```
>>> from qgis.core import *
>>> layerSingle=QgsVectorLayer('LineString?crs=epsg:4326&field=ID:string', 'b
↪ ', "memory")
>>> providerSingle = layerSingle.dataProvider()
>>> recordWrite = QgsFeature()
>>> recordWrite.setAttributes(['1'])
>>> recordWrite.setGeometry(QgsGeometry.fromWkt('MULTILINESTRING ((1 1, 6 1))
↪ '))
>>> providerSingle.addFeature(recordWrite)
True
>>> recordRead = layerSingle.getFeature(1)
>>> print(QgsWkbTypes.displayString(recordRead.geometry().wkbType()))
MultiLineString
>>> print(QgsWkbTypes.displayString(layerSingle.dataProvider().wkbType()))
LineString
```

Fortunately with a QGIS memory layer you can't insert a *Point* into a *Line* dataset, or, otherwise mix geometry sub-types of different dimensionality.

```
>>> from qgis.core import *
>>> layerSingle=QgsVectorLayer('LineString?crs=epsg:4326&field=ID:string', 'b
↪ ', "memory")
>>> providerSingle = layerSingle.dataProvider()
>>> recordWrite = QgsFeature()
>>> recordWrite.setAttributes(['1'])
>>> recordWrite.setGeometry(QgsGeometry.fromWkt('POINT (1 1)'))
>>> providerSingle.addFeature(recordWrite)
False
```

QGIS mixes the primitive geometry types with the modern geometry types by default. The data providers typically handle this well, but, the pre-scripted decisions can have consequences.

For example, by default QGIS will permit a user to add a Multi-line feature to a MicroSoft SQL database

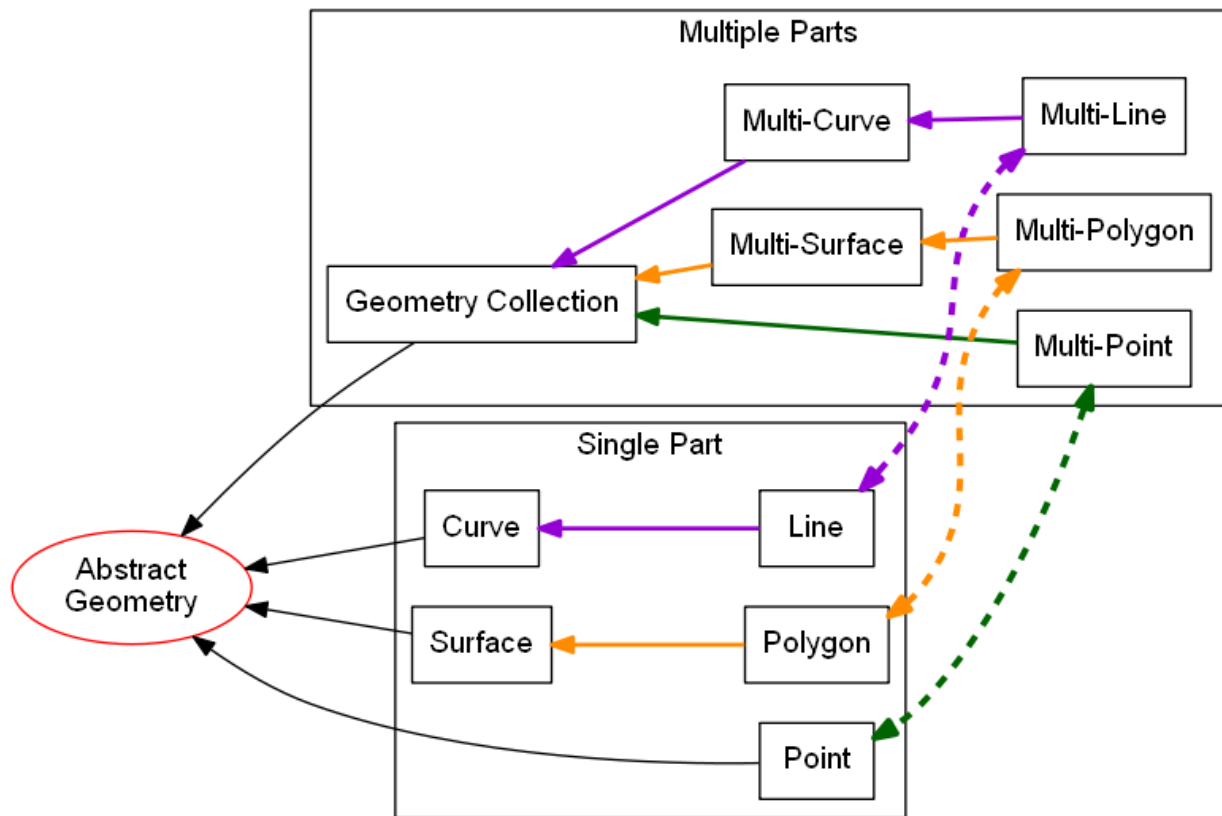


Fig. 3.6: The Geometry subtype hierarchy employed by QGIS by default for several data providers.

that only contains Line features.

3.3.5 GIS Centric And the Wall of Mystery

3.3.6 QGIS

QGIS is self-promoted as an “Open Source Geographic Information System” [3]. QGIS is used for creating, manipulating and publishing spatial data sets by many organisations. Some organisations use QGIS to edit or create geometries for geospatial datasets within enterprise databases in their native format as no commercial software has this capability. For example, [SMEC Pavement Management Software](#) uses a Microsoft SQL database to contain it’s pavement datasets. QGIS can browse and edit the geometries contained within this dataset without importing or exporting any dataset. In comparison, software like ArcMap requires a user to import, edit, then export the data to the SMEC Pavement Management Software even though both installation could have their data respective datasets within the same Microsoft SQL Server installation, and, both datasets are using the same datatype for the geometry values. By intentional design and function, the majority of QGIS users use other software packages or software formats developed by other organisations to store geospatial datasets. For example, the PostGIS, MS SQL and SpatiaLite databases, and, the esri shapefile format are all processed in their native format by QGIS.

QGIS history

QGIS was created by Gary Sherman in 2002 [15][22]. In 2007 it became a project of the Open Source Geospatial Foundation with version 1 being released in January 2009 [15]. The version of QGIS used in this project, Version 3, was released in February 2018 [15]. Version 2 of QGIS employed Python 2 for scripting and PyQt4 for the graphical user interface (GUI). Version 3 of QGIS employs Python 3 for scripting and PyQt5 for the GUI. QGIS version 3 is self-described as a “huge overhaul and cleanup” of QGIS version 2 [9]. Many of the python scripts configured for version 2 no longer work with version 3 with much of the legacy sub-version support dropped.

Vector datasets within QGIS

Within QGIS, the geometries for each record are contained within instances of `QgsAbstractGeometry` subclasses [7]. The manner in which QGIS stores empty geometries is not inherited from the `QgsAbstractGeometry` superclass, but rather is determined for each subclass separately. Although QGIS is capable to parse empty geometries for the majority of the `QgsAbstractGeometry` subclass there may be some that are unable to do so.

The variations in how `QgsAbstractGeometry` subclasses contain empty geometries is demonstrated in the following section by example. Using the QGIS API, empty geometries for several geometry types can be instantiated by instantiating the relevant `QgsAbstractGeometry` subclass without a set of vertices. For example, to test that a `QgsLineString()` is empty:

```
>>> from qgis.core import QgsLineString
>>> QgsLineString().isEmpty()
True
```

Although empty geometries can be created for most geometry types with the QGIS API by instantiation without a set of vertices, it is not currently possible to instantiate an empty point geometry (Table 3.13). As demonstrated below, the well known text representation of the call to instantiate an empty point reveals that

QGIS is wrongly adding a vertex with the coordinates of (0 0) [8]¹.

```
>>> from qgis.core import QgsLineString, QgsPoint
>>> print(QgsLineString().asWkt())
LineString ()
>>> print(QgsPoint().asWkt())
Point (0 0)
>>> print(QgsPoint().createEmptyWithSameType().asWkt())
Point (nan nan)
```

Table 3.13: Testing whether an empty geometry has been created by the instantiation of various types of QgsAbstractGeometry subclasses using the python Console in QGIS 3.0.3.

Input	Output
<code>QgsPoint().isEmpty()</code>	False
<code>QgsLineString().isEmpty()</code>	True
<code>QgsPolygon().isEmpty()</code>	True
<code>QgsGeometryCollection().isEmpty()</code>	True

3.4 PROBLEM DEFINITION

The problems that this project intends to address are:

1. Null and empty shapes are parsed differently by different data providers in QGIS.
2. Within QGIS, there is no published plugin or output that shows records within a dataset that have null or empty geometry values.
3. Many GIS professionals do not anticipate or are aware of datasets that have null or empty geometries.
4. Empty geometry values were not included in the set of valid values in the original QGIS or GDAL specifications.

3.4.1 PARSING OF NULL AND EMPTY SHAPES TO AND FROM EXTERNAL DATABASES BY QGIS

Using source specific data providers, QGIS processes data to and from third party databases without requiring constraints or additional tables in the third-party database. Each data provider has been created independently and these do treat the same data values differently. For example, as expected QGIS parses empty linestrings from PostGis databases as not null but it incorrectly parses empty linestrings from Microsoft SQL Server as null [23].

QGIS Data Providers

QGIS contains 18 data providers that it uses to read and edit datasets in a wide range of data formats [13]. Being open source, each of the data QGIS providers were created at different times for different purposes, are founded on different philosophies and have different levels of development. Consequently, even when different data formats are follow the same geospatial standards, equivalent geometries from those different

¹ a `Point (nan nan)` is also reported when an empty PostGis point is parsed by QGIS.

database packages may be read as different shapes due to variations or errors between these data providers. All of the QGIS data providers are scripted in C++ and unfortunately none are exposed in the QGIS API which inhibits modification of their function [13].

QGIS's data providers that allow it to read or write in native format without need for additional constraints or data tables is a key feature that makes it popular in large organisations. By reading and writing in the native format, QGIS can edit or create geospatial data within an enterprise database that is configured for another piece of software and without importing and exporting. By application of data providers to datasets in enterprise databases, QGIS has removed one of the barriers to integration of geospatial data within enterprise datasets. Geospatial data within enterprise datasets enables built in geospatial analysis.

In comparison to QGIS's approach of editing the data in it's native format, ESRI's ArcMap requires a user to import into a geodatabase and undertake the editing there— even though the enterprise database and the geodatabase may be using the same database server [14]. ESRI's approach often leads to a lookup table being used for geospatial data and then scripts run on those lookup tables to publish an integrated dataset. [LINK TO PART ON LOOKUP TABLES](#)

3.4.2 EXPOSURE OF NULL AND EMPTY SHAPE VALUES IN QGIS

Within QGIS, without using custom expressions or scripts there is only limited exposure to null shapes, and, empty shapes. Two locations within QGIS where you may find descriptive information of each record are the “*Attribute Table*” and the “*DB Manager*” plugin.

Without using expressions and creating new attributes, the “Attribute Table” does not contain any information about the record's shapes. The only component of the “Attribute Table” that exposes empty or null geometries is when a user selects “Zoom to Feature” for a specific record. In QGIS version 3.x, a warning message is shown on the map canvas that the shape is empty or doesn't exist depending on whether the shape is empty or *null* (Fig. 3.7).

The DB Manager in QGIS 3.x is a core plugin (it can't be uninstalled). The DB Manager plugin provides database specific information for data from a limited number of databases that includes PostGIS, but, excludes Microsoft SQL Server. The Table view within DB Manager shows all of the columns within the table including the geometry attribute. For the geometry column it gives the geometry type and exposes null shapes as “*NULL*” (Fig. 3.8).

3.4.3 GIS PROFESSIONAL AWARENESS

Many GIS professionals are educated and work within GIS dataset schema where both null and empty geometries are excluded. It is plausible that GIS professionals that are not familiar with null and empty shapes are ignorant of them when using a GIS where they are permitted. This is a situation that confronted the proponent of this project. Examples of wrong understanding of null and empty geometry values are widespread on the world wide web. For example, an article published by esri incorrectly states that an empty geometry exists for any geometry where the coordinates are unknown [2].

Microsoft SQL and PostGis both permit null and empty geometry values unless explicitly excluded by constraints or third party software. Even when a primary dataset contains no null or empty geometry values, processing of that dataset may produce empty null or empty geometry values. Performing set operations like intersections within the database, and, editing geometry of specific records within QGIS are two ways empty geometry values can be created.

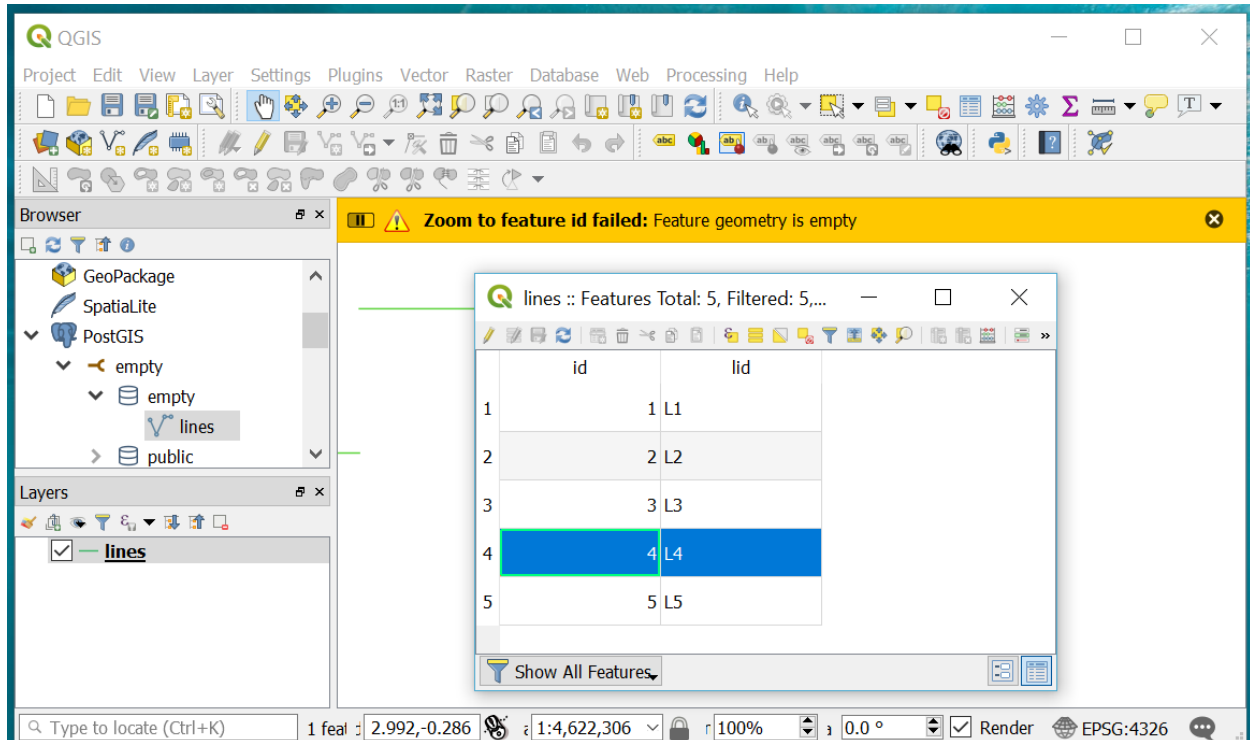


Fig. 3.7: A warning message is displayed in QGIS when a user attempts to zoom to a record with an empty geometry.

Set operations in Microsoft SQL or PostGIS can produce empty geometry values

There are several fundamental set operations that are used to construct new sets from existing sets regardless to what it is a set of [16]. Within a database, a set operation needs to be consistent for all data types. As introduced in section *Records with false Boolean logic* some databases generate empty geometry values for set operations. Microsoft SQL Server and PostGIS are examples of databases that can generate empty geometry values for set operations. Both of these databases developed from SQL conventions and their generation of empty values for geometries is consistent with set operations for other data types.

Intersection of two sets is an example of a set operation. Intersect refers to those locations where two or more objects meet, cross or cover each other [10]. Many different fields that employ set theory, Intersect of two datasets returns all records that exist in both datasets (Fig. 3.9). The intersect concept can be applied to many datatypes including characters, numbers, dates, and, geometries.

Consider the intersection of the points 'Pt1' and 'Pt2' with the circle shown in Fig. 3.10 which is an example of an intersection between two geometry datasets. 'Pt 1' lies within the circle, and, hence intersects the circle. 'Pt2' lies outside the circle and does not. In both QGIS v3.x and ArcGIS desktop, the intersection of a points dataset containing 'Pt1' and 'Pt2' and a dataset containing the circle will return only those records that intersect, hence, only 'Pt1' is returned. In contrast, in Microsoft SQL Server, the intersection returns both the records that do and don't intersect with a dataset containing both 'Pt1' and 'Pt2'.

There is no error with either the approach to intersection taken by QGIS 3.x and Microsoft SQL Server. As described in the section *Records with false Boolean logic* QGIS 3.x includes an additional selection logic step that removes those records that don't intersect. For 'Pt2', Microsoft SQL server is returning an empty

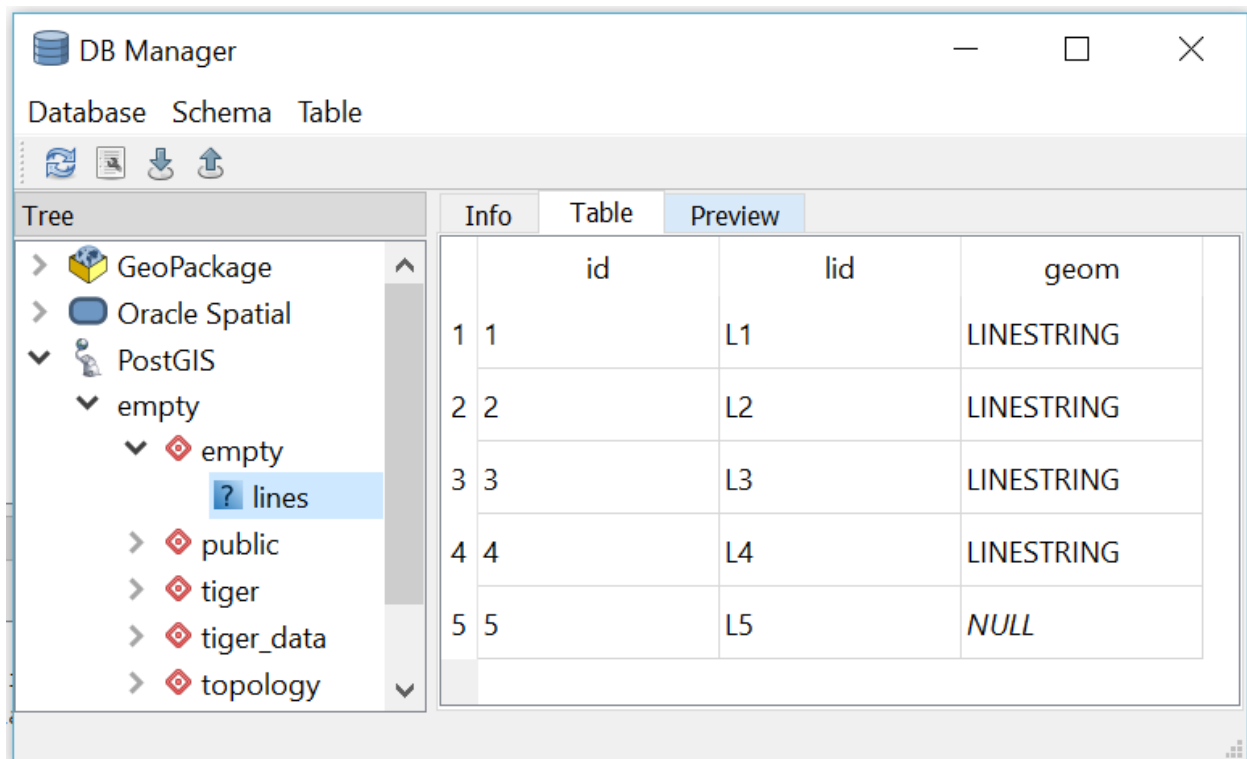


Fig. 3.8: The output from the DB Manager plugin in QGIS 3.x for a dataset that contains both empty geometry values and null geometry values.

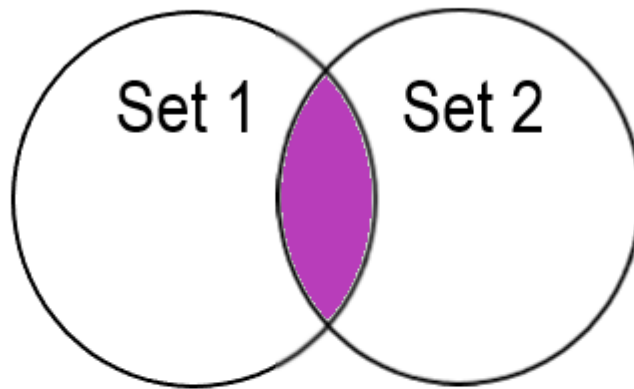


Fig. 3.9: The schematic intersection of Set 1 and Set 2 is purple.

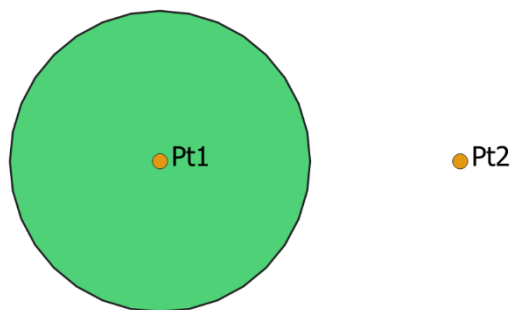


Fig. 3.10: A schematic of the intersection of points 'Pt1' and 'Pt2' with a circle.

geometry confirming that no intersection exists, as shown by the following Transact-SQL script²:

```
DECLARE @circle geometry = 'CURVEPOLYGON (CIRCULARSTRING (0 1, 1 2, 2 1, 1 0, ↵
↵0 1))';
DECLARE @Pt2 geometry = 'POINT (3 1)';
SELECT @Pt2.STIntersection(@circle).ToString();
GEOMETRYCOLLECTION EMPTY
```

This intersection example shows the ease with which one can inadvertently generate empty geometry values in Microsoft SQL Server, and, these empty geometry values will be passed to QGIS.

Inadvertently Creating Empty Geometries within QGIS

Within QGIS, an empty geometry is created by using the “Vertex Tool” to delete all vertices of an existing shape. A user unfamiliar with QGIS may incorrectly assume that deleting all the vertices of a shape also deletes the record. Hence, when editing a shape, a QGIS user may unintentionally create an empty shape when they are attempting to delete the shape.

3.4.4 Retrospective incorporation of empty and null values into Software

The Geospatial Data Abstraction Library (GDAL) is ubiquitous within most GIS Software to translate and process geospatial data. Handles for empty shapes were not part of the original GDAL specification, with empty shapes being treated as null. The retrospective incorporation of empty handles into GDAL has not been picked up by many applications that employ GDAL, and, even within GDAL there are many processing tools that do not allow for empty shapes.

Many of the current GDAL set operators continue to convert empty geometry values to null geometry values even when the set being operated on has successfully implemented the empty shape handle on passing to GDAL. For example, for the Intersection operation the output is “*a new geometry representing the intersection or NULL if there is no intersection or an error occurs*” [4]. Curiously, the same GDAL geometry class has a handle to assign an empty geometry, or, to test for an empty geometry [1]. On another GDAL ticket register it is stated “*GML/WFS: by default does not advertise NOT NULL fields since we treat empty strings as being null for historical reasons. By setting the open option EMPTY_AS_NULL=NO, empty strings will be reported as such and NOT NULL fields might be advertised*” [21], it is clear that GDAL is not originally intended to distinguish between null and empty.

The retrospective addition of handles for empty geometry values in GDAL was mimicked by QGIS. Consequently, within QGIS there are likely to many set operators that convert empty values to null values as part of the set operation.

The definition of Null and Empty Values depends on context

The definition of Empty geometry values introduced in section 2.2 it was articulated that an Empty geometry is just one valid value in the set of valid values for the geometry data type. In contrast, null indicates that the geometry value is unknown and that the value can be any value from the set of valid values. Hence, an empty geometry values is one of many possible values for a null geometry (or, empty is a subclass of null). For example, for a point, the vertices of an empty point are $\{ \}$ as it has no coordinates, and, the vertices of a null point are $\{x, y\}$ where x and y are both variables designating unknown coordinates. These definitions

² In both Microsoft SQL and PostGis the geometry type that is empty is recorded [18][20]). Sometimes the geometry type gets changed to the generic ‘GEOMETRYCOLLECTION’ by set operations.

are consistent with SQL relational databases. Unfortunately, the definition of null used by SQL relational databases is different to that used by both mathematics for set theory, and, but most computer programming languages [6].

Let's consider the definition of null for set theory and computer programming using a common example. If one has a box of apples and a box of bananas, then the intersection of the two boxes of fruit is an empty box. For set theory, *null* – nothing is what you have when you take away the box.

Similar to set theory, for most object-oriented computer programming languages, an object that has not been instantiated is *null*. Hence, a pointer to something that doesn't exist is null. An empty object is an object that has been instantiated but not populated, for example, a list with no members, or, a box with no fruit.

So, from the perspective of an object-oriented computer programming language, null means no value. Hence, a null object is an object that does not exist, or, occupies no space in a computer's memory. Indeed, SQL relational databases implement null as an object that doesn't exist, so, the context difference between unknown and nothing does not have to create conflict.

The underlying conflict is based on logical deduction. It has been proved that an empty set is a subset of any set. It is argued that an object that doesn't exist can't contain anything, so, it must be empty³. Hence, *null* is a subclass of empty. However, an empty set is still a set, a container that is empty. In contrast, null states that no container exists. So, by arguing that a *null* object is empty you have just created an object. A *null* object neither contains anything nor has a container.

Unfortunately, QGIS employs the logic that a null geometry is also an empty geometry [11]. The QGIS test for null simply asks where the object points to anything [11]:

```
144 bool QgsGeometry::isNull() const
145 {
146     return !d->geometry;
147 }
```

Now, the QGIS test for empty performs the same test as for null, and, returns true if the test for null returns true [11]:

```
329 bool QgsGeometry::isEmpty() const
330 {
331     if ( !d->geometry )
332     {
333         return true;
334     }
335     return d->geometry->isEmpty();
337 }
```

3.5 Future Development

Some ideas for improvement:

1. Add a summary table

³ Unable to find a high quality reference for this argument. Several private conversations with C++ programmers reveal that this logic is common and is frequently used to test for empty in C++.

2. Add a column for number of parts
3. Distinguish curved shapes
4. Distinguish shapes with elevation information

3.6 Bibliography

The following modules are included within this plugin. The plugin's repository is on [GitHub](#).

4.1 modelVectorLayers module

Creates vector layers as QGIS memory layers. These layers are intended to demonstrate how the Geometry Attribute Table plugin renders datasets that have: multi-part, single part, empty, and, null geometries. A description of the created layer is included with each method.

`modelVectorLayers.createMultiLines (layerName='Multi-Part Lines')`

A geometry collection of multi-part lines and lines (single part constrained).

Args: `layerName (str)`: The name of the layer that is loaded into QGIS.

Returns: A memory vector layer containing features with the following geometries: 2 part multi-line; 3 part multi-line, 1 part multi-line, line, Empty multi-line, and, Null.

`modelVectorLayers.createMultiPoints (layerName='Multi-Part Points')`

A geometry collection of multi-part points and points (single part constrained).

For an empty point, QGIS erroneously creates a point with the coordinates (0 0).

Args: `layerName (str)`: The name of the layer that is loaded into QGIS.

Returns: A memory vector layer containing features with the following geometries: 2 point multi-point; 4 point multi-point, 1 point multi-point, line, Empty multi-line, and, Null.

4.2 parseQGISGeometry module

The QGIS Expression and methods in this module parse a vector's geometry for a QGIS vector layer.

`parseQGISGeometry.geometryField (feature, parent)`

Creates a QGIS expression called `geometryField`. This expression returns a string that represents the geometry in the following order of decreasing precedence:

- Null
- Empty
- Well known binary type string

Args: `QgsFeature`

Returns: A string that represents the QgsFeature's geometry.

`parseQGISGeometry.layerAddVirtualGeometryField(vectorLayer)`

Uses the 'geometryField' expression to provide string values that represent the feature's geometry. These string values are appended to the input *vectorLayer* as a virtual field.

Args: *vectorLayer* (QgsVectorLayer): A QGIS vector layer.

`parseQGISGeometry.layerRemoveVirtualGeometryField(vectorLayer)`

Removes the virtual field called 'Geometry' from a vector layer if it exists.

Args: *vectorLayer* (QgsVectorLayer): A QGIS vector layer.

`parseQGISGeometry.layer_review(layer)`

IN PROGRESS - NOT CURRENTLY USED BY PLUGIN

Reviews any vector layer for the presence of null or empty geometries. Provides a summary of a layer by geometry type.

Args: *vectorLayer* (QgsVectorLayer): A QGIS vector layer.

The following modules are included in the plugin's repository and are useful to construct the plugin from the source scripts.

4.3 zipGeomAttribute module

Contains methods for zipping the minimum compiled source scripts needed for this plugin to function within QGIS. These methods do not do any of the compilation (e.g. update docs, or, compile the QT Designer interface).

`zipGeomAttribute.installGeomAttributeFromZip(pluginPath)`

Unzips the zipped plugin folder. An automation used for testing.

Args: *pluginPath* (str): The path of the plugin folder in the user's installation of QGIS.

`zipGeomAttribute.zipGeomAttribute()`

Zips the minimum number of files needed for the plugin to be installed into QGIS

CHANGELOG

All notable changes to this project will be documented in this file.

This project adheres to [Semantic Versioning](#).

5.1 Version 0.1.0

- First release
- Shows attribute table for vector layers in QGIS v3.x with icons that represent the geometry.
- Has the option to create demonstration vector datasets.

5.2 Version 0.1.1

- Removes the virtual field called 'Geometry' when the geomAttribute Attribute table is closed.

GLOSSARY

Geometry The shape that is associated with a feature. In many GIS systems the a feature can only have a single geometry type, where the type is constrained to being the same for the entire dataset.

Dataset A table containing features.

Empty It is known that a geometry does not exist within the feature's domaine. Normally only needed for attribute tables with multiple geometry columns.

Null The geometry is unknown. It may or may not be empty depending on the dataset.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

BIBLIOGRAPHY

- [1] Empty(self, *args). URL: <http://gdal.org/python/osgeo.ogr.Geometry-class.html#Empty>.
- [2] How To : Insert a NULL or empty st _ geometry attribute in Oracle. URL: <https://support.esri.com/en/technical-article/000010466>.
- [3] QGIS - The Leading Open Source Desktop GIS. URL: <https://www.qgis.org/en/site/about/index.html#>.
- [4] def Intersection(self, *args):. URL: <http://gdal.org/python/osgeo.ogr-pysrc.html#Geometry.Intersection>.
- [5] ISO 19125-2 Geographic information - Simple feature access. Part 2: SQL Option. First Edition. 2004.
- [6] What Is Null. 2010. URL: <http://wiki.c2.com/?WhatIsNull>.
- [7] QgsGeometry Class Reference. 2017. URL: <https://qgis.org/api/2.18/classQgsGeometry.html#details>.
- [8] Bug report #19190: QgsPoint class does not store empty point geometries - QGIS Application - QGIS Issue Tracking. 2018. URL: <https://issues.qgis.org/issues/19190>.
- [9] Changelog for QGIS 3.0. 2018. URL: <https://www.qgis.org/en/site/forusers/visualchangelog30/>.
- [10] INTERSECT | meaning in the Cambridge English Dictionary. 2018. URL: <https://dictionary.cambridge.org/dictionary/english/intersect>.
- [11] QGIS API Documentation: /tmp/builddd/qgis-3.3.0+git20180710+2688a9d+15buster/src/core/geometry/qgsgeometry.cpp Source File. 2018. URL: https://qgis.org/api/qgsgeometry_8cpp_source.html.
- [12] QGIS API Documentation: QgsAbstractGeometry Class Reference. 2018. URL: <https://qgis.org/api/classQgsAbstractGeometry.html>.
- [13] QGIS/src/providers/. 2018. URL: <https://github.com/qgis/QGIS/tree/master/src/providers>.
- [14] Tutorial: Perform web editing using data from a database—Documentation | ArcGIS Enterprise. 2018. URL: <http://enterprise.arcgis.com/en/server/latest/get-started/windows/tutorial-performing-web-editing-using-data-from-a-database.htm>.
- [15] Wikipeday Contributors. QGIS. URL: <https://en.wikipedia.org/wiki/QGIS>.
- [16] Wikipedia Contributors. Set (mathematics). 2018. URL: [https://en.wikipedia.org/w/index.php?title=Set_\(mathematics\)&oldid=866126059](https://en.wikipedia.org/w/index.php?title=Set_(mathematics)&oldid=866126059).
- [17] ESRI. ESRI Shapefile Technical Description. Technical Report, ESRI, 1998. URL: www.esri.com,.

- [18] Mateusz Loskot. SqlGeometry and POINT EMPTY in WKB. 2010. URL: <http://mateusz.loskot.net/post/2010/02/26/sqlgeometry-and-point-empty-in-wkb/>.
- [19] OGC. OpenGIS\textregistered Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture. *Open Geospatial Consortium, Inc*, pages 93, 2010. URL: http://portal.opengeospatial.org/files/?artifact_id=25355E+Implementation+Standard+for+Geographic+information+-+Simple+feature+access#1.
- [20] Paul Ramsey. Nothing, Nada, Zip, Bupkus. 2010. URL: <http://blog.cleverelephant.ca/2010/03/nothing-nada-zip-bupkus.html>.
- [21] Even Rouault. Empty elements in GML : empty or NULL. 2015. URL: <https://trac.osgeo.org/gdal/ticket/5968>.
- [22] Gary Sherman. History of QGIS Committers. 2011. URL: <http://spatialgalaxy.net/2011/09/23/history-of-qgis-committers/>.
- [23] Philip Whitten. Bug report #19397: Parssing of empty geometry values from MicroSoft SQL Server - QGIS Application - QGIS Issue Tracking. 2018. URL: <https://issues.qgis.org/issues/19397>.

PYTHON MODULE INDEX

m

`modelVectorLayers`, [31](#)

p

`parseQGISGeometry`, [31](#)

z

`zipGeomAttribute`, [32](#)