TECHNICAL SCIENCES
Abbrev.: Techn. Sc., 2011, 14(1)

THE IMPLEMENTATION OF THE SPATIALLY ENABLED DATABASE WITH OPEN SOURCE SOFTWARE AND OPEN SPECIFICATIONS USAGE

Michał Bednarczyk

Chair of Surveying
University of Warmia and Mazury in Olszytn

K e y w o r d s: open source software, OpenGIS specifications, relational database, MySQL, spatial database.

A b s t r a c t

This article describes the design approach in the elaboration and implementation of spatial database. This solution consists of computer programs, algorithms and appropriately designed database schema. The primary objective was to use open source software, and assumptions contained in the OpenGIS open specifications.

IMPLEMENTACJA PRZESTRZENNIE ZORIENTOWANEJ BAZY DANYCH Z WYKORZYSTANIEM WYBRANEGO OPROGRAMOWANIA OPEN SOURCE I OTWARTYCH SPECYFIKACJI

Michał Bednarczyk

Katedra Geodezji Szczegółowej
Uniwersytet Warmińsko-Mazurski w Olsztynie

S ł o w a k l u c z o w e: oprogramowanie open source, specyfikacje OpenGIS, relacyjna baza danych, MySQL, baza danych przestrzennych

A b s t r a k t

W artykule opisano autorskie podejście podczas opracowywania i implementacji bazy danych przestrzennych. Rozwiązanie to składa się z programów komputerowych, algorytmów i odpowiednio zaprojektowanego schematu bazy danych. Podstawowym założeniem było wykorzystanie oprogramowania open source oraz założeń zawartych w otwartych specyfikacjach OpenGIS.

Introduction

This article describes a way to develop a spatially-enabled database for GIS systems. The primary objective of the proposed solution was based on the assumptions contained in the OpenGIS specifications, developed by the Open Geospatial Consortium (OGC) and MySQL – an open source relational database management system. A database schema has been designed as a part of this objective, as well as algorithms and computer programs for converting data into a form suitable for the inclusion in the database schema, and an application which enables remote access to the data on the server. For the most part, the article describes the database itself, which, as the unitary data source, is the most important part of any GIS system. Although the applications needed for data conversion and loading are just as important, the paper will only mention them briefly, because a full description would take too much space.

Open source – means software developed with source code. It is possible to download it, use, make changes in source code and compile without any payment. It is mostly distributed under GNU/GPL license. The reason for addressing this subject was the increasing interest in open specifications and open source software, as well as its still growing quality and functionality. This creates a need for a closer look at the possibilities of its application in practice, because it often involves sizable financial savings.

OpenGIS specifications are rather extensive and have a wide range of applications. This paper discusses only the scope of the OpenGIS specification as implemented in the MySQL system and directly used in the approach described below. The reader will find here some general information about the OpenGIS specifications and a description of the scope of their implementation in the MySQL system. As the next step the author describes the spatially-enabled database he created. The proposed solution uses the capabilities of MySQL, resulting both from the work of its authors and the implementation of the OpenGIS specifications. The final section of the paper describes the practical use of the database created as a data source for Java application, developed by the author.

Open GIS Specifications

OpenGIS Specifications are technical documents, describing in detail interfaces or encoding. They are widely available, free of charge. They can be used by all the software developers wishing to use the proposed solutions in their products. Creation of these specifications is the basis of the Open Geospatial

Consortium to facilitate the interoperability of different computer programs. The ideal situation would be if the programs created by two different developers implementing the same specification worked without a problem.

These documents are created for GIS products, especially for web applications. OGC open standards are being largely taken over by International Organization for Standardization (ISO) to become ISO standards. The cooperation between OGC and International Organization for Standardization results in ISO standards and OGC specifications being identical. The difference is only in copyrights (GAŹDZICKI 2007). OGC specifications standardize many GIS issues such as: Catalogue Services, Coordinate Transformation Service, GML, Web Map Service, Web Feature Service, OpenLS Service, simple features implementation and many others. More information can be found on: http://www.opengeospatial.org.

For the purposes of this article the assumptions of OGC specification (*Implementation Specification for Geographic Information – Simple Feature Access*) were used. It describes interfaces for accessing spatial data, defines methods of publication, storage, retrieval and perform operations on vector graphical objects (simple features) comprising the digital map (points, lines, areas etc.). Assumptions of this OGC specification have been implemented in MySQL relational database management system.

MySQL Relational Database Management System

MySQL was created in the second half of 90's of the last century. MySQL's creator is Michael "Monty" Widenius from the Swedish company TcX DataKonsult AB (BEDNARCZYK 2005). The first open source version of MySQL was published in may 1995. The process of development and improvement has accelerated since that time. This happens because of the members of the Internet community who started to create MySQL together, who see in it an excellent database management system for web applications. Mainly because of its high performance.

Today MySQL is one of the most popular database management systems among the open source products (STONES, MATTHEW 2003). It is developed as an open source with an appropriate business model. It means that MySQL can be distributed under two different kinds of licenses: non-commercial free GNU/GPL or full paid commercial license. Commercial version is fully supported with additional services, such as 24 hours technical support (help in configuration, advices, increasing of functionality, use of advanced functions etc.), additional tools for better configuration, automatic updates. Commercial license is also recommended when MySQL source code is to be used as a part of another commercial system.

As the MySQL evolved, more abilities have become available. At first there were only basic functions, which are needed for data storage and distribution – table creation, data and table manipulation using SQL, safety mechanisms etc. MySQL ver.5 used by the author of this article has the following features (AXMARK, WIDENIUS 2007):
– transactions,
– replication
– cluster,
– spatial extensions (implemented according to Open GIS specifications),
– stored procedures and functions,
– triggers,
– views,
– information schema (system metadata).

In addition, extensive documentation and many tools for data administration and management are included. All these things make MySQL very similar to commercial systems such as Oracle for example. Due to spatial extensions included in MySQL and increase of functionality (compared to earlier versions used primarily for Websites), the possibility of using this system in the field of GIS is worth considering.

Spatial Extensions and Geometry Types in MySQL

As mentioned, spatial extensions in MySQL are implemented according to Open GIS specifications and based on the OpenGIS Geometry Model. They extend SQL language, available in MySQL, with new functions and geometry types for storing and managing spatial data. In MySQL 5 not all rthe ecommended by the OpenGIS functions are available yet – this applies mainly to spatial analyses. Spatial extensions appeared for the first time in MySQL version 4.1.

As mentioned earlier, the implementation of spatial extensions in MySQL is based on the considerations of OpenGIS Specifications. For this system, they are an extension of spatial SQL environment (geometric) data types (geometry types), together with the features to support them. Not all functions recommended by the OpenGIS are available in MySQL 5 – this applies mainly to spatial analyses. For the first time they appeared in MySQL version 4.1.

Geometry types are used to store information about feature geometry (point, line, area etc.). They can be divided into:

– types which store information about simple-geometry features, such as: POINT, LINESTRING, POLYGON, GEOMETRY – each of the above geometry types.

– Types which store information about complex-geometry features, which can contain many simple geometry features, such as: MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION.

They can be used in the same way as the standard SQL data types (for example INTEGER, CHAR or REAL). Therefore, they can be used to create a column in table or define a variable of type, for example POLYGON. Class hierarchy of geometry types is presented in the diagram below (Fig. 1).

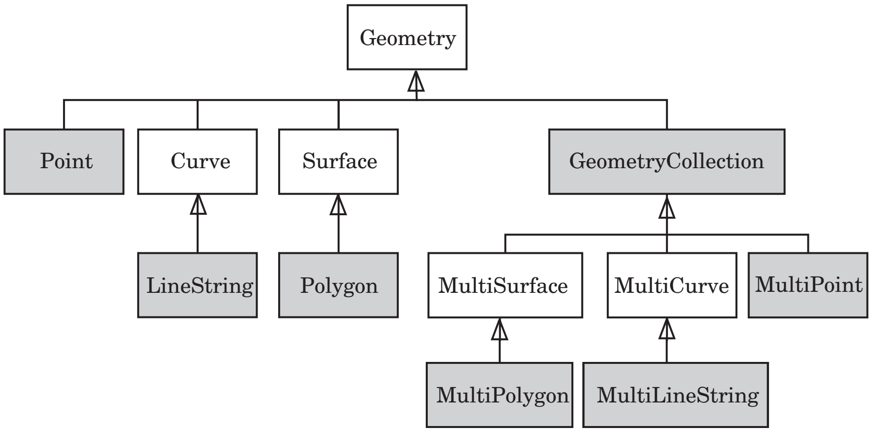


Fig. 1. Geometry types class hierarchy diagram, according to Open GIS specification
Source: OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 2: SQL option. http://www.opengeospatial.org.

GEOMETRY class is the parent class for all these which inherit from it. Therefore, all classes derive attributes and properties from GEOMETRY, depending on the type of feature they represent. A detailed description of all classes can be found in Open GIS specification: *OpenGIS Implementation Specification for Geographic information – Simple Feature Access.*

Data Formats for Spatial Information

MySQL implements – also according to Open GIS specification – two kinds of spatial data formats. All functions called by client applications, operating on spatial data, can use one of them. These are:
– WKT – Well Known Text Format,
– WKB – Well Known Binary Format.
Here are some examples of data in WKT format:
– a point with coordinates (10, 12): POINT(10,12),

– a linestring consisted of four vertices: LINESTRING(0 0, 10 10, 20 28, 50 65),
– a polygon with a linear ring inside: POLYGON((0 0, 10 0, 10 10, 0 10, 0 0),(5 5, 7 5, 7 7, 5 7, 5 5)),
– Line and point in one geometry type: GEOMETRYCOLLECTION(POINT (10 10), LINESTRING(11 12, 15 20)).

The data represented in WKB format is specially prepared BLOB[1] values. For example: Point with coordinates (1.0,1.0) will be represented as:
0101000000000000000000F03F000000000000F03F

Where the different parts of the data stream specified above in hexadecimal notation, means:

01 – byte order (determine whether the most significant bit is at the beginning or end of a stream, affects the order of reading the bytes),

01000000 – feature geometry type (point, line poligon etc. 1 to 7),

000000000000F03F – *X* coordinate (double),

000000000000F03F – *Y* coordinate (double).

Functions Operating on Spatial Data

Internally, the data of the geometry objects are not stored in any of the above formats. MySQL uses its own format, while WKT and WBK are used to exchange data. Spatial extensions are available through appropriate functions, whose interfaces are described in the OpenGIS specifications. These functions can be divided into:

1) Functions for creating spatially enabled database. They return geometry value of the specified type (eg. POLYGON, LINESTRING, etc.). Available functions are using WKT or WKB format or created for MySQL (not included in the OpenGIS specification) format.. They are primarily used for such tasks as:

– creating spatial data type columns,
– inserting spatial values into database tables,
– selecting spatial values from database,
– conversion between spatial data formats.

2) Functions for analysis of spatial information. They are used for:

– obtaining information about the graphical attributes of the object – e.g. length, the number of vertices, area, the number of objects in the collection, etc.

[1] BLOB – Binary Long Object – binary data type used in relational databases. It is used to store binary data such as media files, pictures etc.

– creating geometry from existing elements – e.g. join, difference, buffer zone, next vertex etc.,

– the analysis of the relationship between objects – e.g. intersection, disjoint, contain, overlap, etc.

These functions can be used in SQL queries. This makes it possible to manipulate spatial data and perform spatial analysis through a commonly known language of structured queries. For example, to insert into the field *geo_kol* in table *geo_tab* value that represents the point with the coordinates (1,1) the following query can be used:

INSERT INTO geo_tab(geo_kol) VALUES(GeomFromText('POINT(1,1)'))

Where GeomFromText() is one of the functions described above, which converts string POINT(1,1) in WKT format into a value which can be saved in the database.

Spatial analysis can be performed in the same way, except that a different function and SQL keywords must be used.

User-Defined Procedures and Functions

MySQL ver.5 makes it possible to create user-defined functions and procedures (stored routines). According to the information contained in the *MySQL Reference Manual*, their syntax is compatible with SQL: 2003 standard. Use of the routines increases functionality of the system. After the procedures are created and saved in the system, they can be used as a part of SQL queries, just like functions operating on spatial data, which were described before. Such functionality can be found in commercial systems e.g. Oracle. Routines allow to:

– automate internal processes such as: fields update, replication, performing several operations in one call and other operations on the data in the database,

– enhance the safety of performed operations – the client cannot access tables directly, only through the routines; in this way operations performed on the database are invisible for him.

– enhance performance – single procedure call instead of many consecutive commands to the server.

– improve overall functionality (beyond SQL) – the syntax of the routines is similar to well known programming languages and includes loops, conditions, declaration of variables etc.; SQL commands also can be used.

Implementation of Spatially Enabled Database in Mysql

The database discussed in this paper can be supplied with data from an external data source which is a database from Intergraph MGE-PC GIS system, running in Microstation environment. A special application, called Sipos-export, was created by the author for this purpose. It can convert and upload data into the MySQL database. A short description of Sipos-export can be found in the chapter *The Role of User-Defined Routines in the Process of Data Processing*.

However, in the data conversion process, user defined routines, mentioned above, are used. They were programmed and applied specifically to automate activities related to data conversion and uploading into the server.

The data presented in this work was originally produced, as already mentioned, in the MGE-PC GIS system. It has been moved and transformed from this system by the author, with a specially developed application, as described here. The general scheme of the database after transformation consists of the following tables:

1) feature – a table with information about the feature classes in the system – this table contains following columns:

– fclassname – taken over from MGE-PC feature class name (fname),

– fcode – taken over from MGE-PC feature class code,

– ftype – taken over from MGE-PC and modified feature type – takes the following values: 1 – not present in the MGE-PC – this code is used for graphical elements which are not GIS features (not having identifiers binding them with attributes), 0 – undefined, 1 – point, 2 – line, 3 – area, 4 – centroid, 5 – label,

– category – taken over from MGE-PC category identifier,

– mgetablename – MGE-PC table name in which feature attributes were originally stored (MGE-PC feature table name),

– mgelevel – layer number, to which features of this class were assigned in MGE-PC,

– csid – coordinate system identifier,

– f-geometry-column – column name in the feature table, where geometry attributes are stored;

2) primitive – stores vertex coordinates of individual features (point primitives). Contains columns:

– pid – the identifier of the point primitive,

– *X* – the *X* coordinate,

– *Y* – the *Y* coordinate,

3) prim-feature – stores connection schemas of vertexes for every single feature. Columns are:

– pid – point primitive identifier from *primitive* table,

– fclassname – feature class name,

– fid – feature identifier,

– elnum – the number of a graphical element as a part of the feature; the numbering proceeds according to the order of writing elements in DGN project file,

– eltype – integer specifying the type of graphical element as a part of the feature (eg. line – 3, polyline – 4, shape – 6, etc.),

– num – the number of a node in the graphical element as a part of the feature; always starts from 1, within the component;

4) <feature table> – a table, with any name, which contains descriptive and graphical data for all features of a given class. A single line of such a table corresponds to a single feature, while a column corresponds to a single attribute. The number of tables in the table depends on the number of classes defined by the user in the system. It may contain the following columns:

– <descriptive attribute> – a column, with any name, which contains a descriptive attribute, eg. name, size, etc.). The table may contain any number of these columns,

– <graphical attributes> – a geometry column (in accordance with Open-GIS) which contains geometric attributes of a feature. There can only be one column of this type. The default name during export is: GEOMETRY_ATT.

The schemas of all tables except feature tables are predetermined, so their creation in the system is not problematic. The construction of a feature table is automatically set during export, based on the information in MGE-PC database.

According to OpenGiIS specifications, feature tables contain a column with geometric attributes (feature geometry). The remaining attributes may be of any type and are derived from the original versions of these tables (MGE-PC). An example of database schema is shown on figure 2.

In geodesy points with coordinates obtained by calculations or in a measurement play a pivotal role. It is a basic piece of data needed for the preparation of cartographic document. In this situation, the most logical path in the creation of a GIS system is to (simplified):

1) Obtain data (by a measurement, calculation, etc.)

2) Feed the system with points,

3) Construct features based on these points.

Following this line of reasoning, the proposed way of converting and storing data makes use of two tables: *primitive* and *prim-feature.* The former table stores the points' coordinates – point primitives. They can be obtained from different sources (e.g. calculations, direct measurements, other systems). Features are then contructed on the basis of these coordinates. Each feature

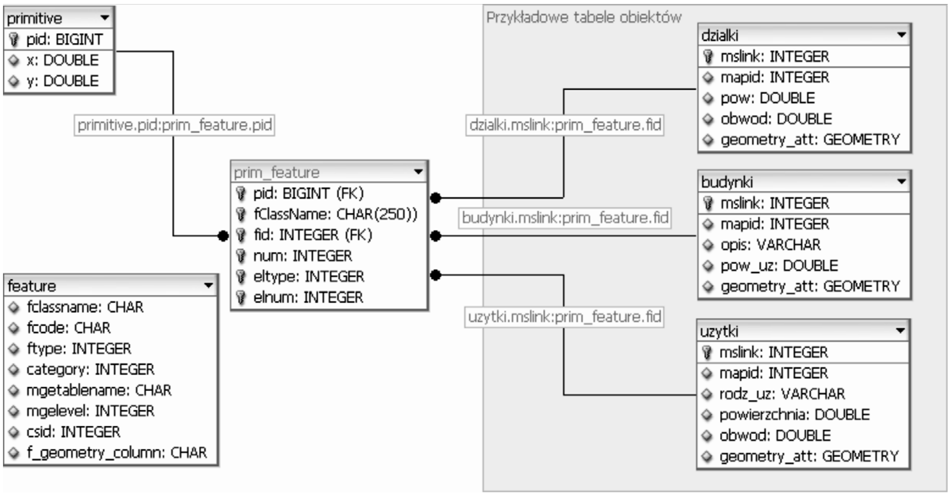


Fig. 2. An example of a MySQL server-side database schema
Source: own work.

has a map of node connections – in this case stored in the the table *prim-feature*. Each node is assigned a point with specific coordinates – point primitive – stored in the table *primitive*. A primitive is assigned to the node by an identifier (PID). Such a format makes it easier to conduct operations on separate points, to use them for calculations, measurements or setting up another system. The conversion of all the coordinates simultaneously is also easier. In this format the connection maps of different features may share the same point primitives. This eliminates the problem of duplicating the same coordinate values for different features. A change in the coordinates of a primitive will change the boundaries of all the features containing the primitive.

The data including point primitives is saved according to the following rules:

1) The point coordinates are stored as a separate set – point primitives (table *primitive*).

2) Each feature consists of elements. A feature is defined by a pair of identifiers: *nazwa klasy* (class name) and *identyfikator obiektu* (feature identifier). An element of the feature may be a point, line, multiline or area.

3) Each element has nodes ordered by their occurrence in the original project file – i.e. in the order they were digitized.

4) The coordinates of each node are assigned by the identifier (PID) from the table of primitives (*primitive*).

No matter what the format of graphical data is in the system, it usually has to be converted if it is to be displayed on the computer screen or a printout. As a result, the user can be presented with certain geometrical shapes (lines, polygons, points, etc.). They are built based on the data collected by the system and drawn using appropriate software functions responsible for the visual display. These functions must access the information about the type of the shape to be drawn, the coordinates of the nodes and, additionally, colour and line style. They must, independently of how the feature is described, "construct" geometrical shapes to be displayed, on the basis of the data obtained from, for example, a database. This model of data presentation is analogous to a simple vector model, where each feature is defined separately, including geometry. Thus, even if a program uses a completely different format of storing graphical data, for visualization purposes they will by converted to a format resembling a simple vector model. A simple vector model, despite certain drawbacks (resulting, for example, from node repetition for features sharing a boundary) has undeniable advantages, such as simplicity and display speed. It focuses on those properties of graphical elements, which are important for visual representation, and does not require additional interpretation.

Applications which use a relational database, especially over the Internet, should generate the lowest possible data transfer. A large amount of information sent between clients and the server has a significant impact on efficiency. One way to optimize the cooperation between the program and the database is to reduce the number of inquires. If the same amount of data can be retrieved or sent to the server using one or many queries, the time necessary for the server to send and process the data will be much shorter in the former case.

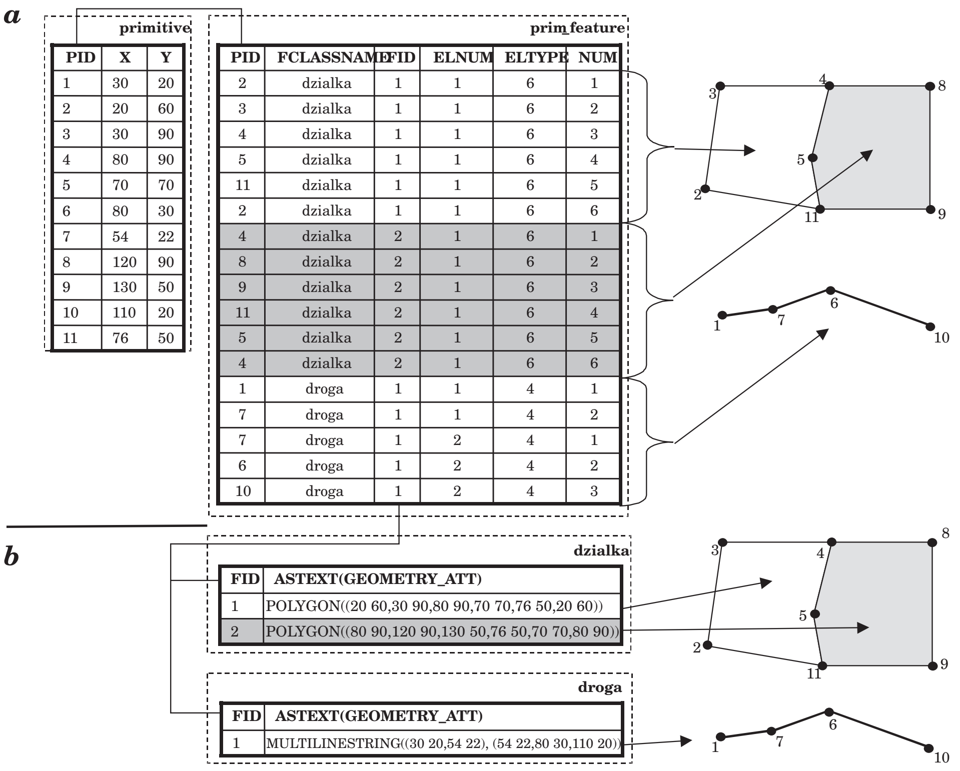
Taking this into consideration, I decided to take advantage of the possibilities offered by MySQL and implement in it, in accordance with OpenGIS specification, spatial data types. They are discussed in more detail in the chapter on MySQL server. OpenGIS specification suggests a feature table schema, which, apart from descriptive attributes, contains a column with spatial data type, containing geometric data. Henceforth I will call this table geometric attribute of the feature. In taking this step I focused on simplicity, speed and efficiency, which are particularly important when the information is accessed via the Internet. This is especially true when using the popular SQL, as processing speed might become a problem.

A similar treatment of geometric data (node coordinates and the identifier in one record of the same table) has been proposed elsewhere and refered to as object-oriented topological model (ECKES 2006) or a physical model of a topological database (LONGLEY et al. 2006). The difference is that in the physical model a set of topological rules for different classes of features (for example different buildings cannot be located in the same area, plots cannot overlap, etc.) is recorded, together with the information on topological errors and areas which have been edited, but not yet corrected with respect to topology. Using this information the system reproduces topological relations among features on the fly, when requested by the user. This kind of formatting is used, for example, in ESRI ArcGIS. It need not involve storing the topology in, for example, the structure of nodes and edges. Consequently, it is not necessary to reconstruct their geometry every time they are being displayed and thus to slow down this operation.

The solution described here uses the functions of MySQL server created by the author, which produce a geometric attribute of the feature based on the information from the database (point primitives and a map of node connections). This attribute is naturally recorded and stored in a spatial data type column, in a specific feature table. Once the feature table is created in this way, a simple SQL query may retrieve both descriptive attributes and feature geometry. The output of the query is a single record containing all the necessary information. This considerably speeds up the loading of data from the server, which is an important issue on the Internet. The application which retrieves the data does not need to “assemble” the feature from point primitives and analyse the connection map, because this operation is only done once after each feature update. This means that the update of the feature geometry causes the geometric attribute needed for visualization to be updated as well. This is the method used in my applet – *Sipos-klient*, The geometric attribute retrieved by the client is not only used for the visual presentation of the feature, but, above all, serves as the interface for the identification of the feature in the database. By clicking on the picture on the computer screen the user retrieves the feature identifier. This gives him access to any data on the selected feature, and the possibility to modify it.

The feature tables contain – in compliance with OpenGIS specification – geometric attribute. The usage of such tables has an additional advantage of interoperability with other systems reading this format.

As can be seen in Figure 3, the table *prim-feature* stores the maps of node connections for all features in the system. This concept of the table *prim-feature* was partly determined by technical problems in MySQL routine implementation. Among others, it is impossible to use a table name in the routine as an input parameter. As a result, the routine can only operate on tables whose names are contained in the routine code as constants. The solution of this problem would offer greater elasticity in constructing database schemas. For example, it would enable the creation of a separate table with point primitives for each feature class. I hope it will be possible in forthcoming editions of MySQL server.



primitive

PID	X	Y
1	30	20
2	20	60
3	30	90
4	80	90
5	70	70
6	80	30
7	54	22
8	120	90
9	130	50
10	110	20
11	76	50

prim_feature

PID	FCLASSNAME	FID	ELNUM	ELTYPE	NUM
2	dzialka	1	1	6	1
3	dzialka	1	1	6	2
4	dzialka	1	1	6	3
5	dzialka	1	1	6	4
11	dzialka	1	1	6	5
2	dzialka	1	1	6	6
4	dzialka	2	1	6	1
8	dzialka	2	1	6	2
9	dzialka	2	1	6	3
11	dzialka	2	1	6	4
5	dzialka	2	1	6	5
4	dzialka	2	1	6	6
1	droga	1	1	4	1
7	droga	1	1	4	2
7	droga	1	2	4	1
6	droga	1	2	4	2
10	droga	1	2	4	3

dzialka

FID	ASTEXT(GEOMETRY_ATT)
1	POLYGON((20 60,30 90,80 90,70 70,76 50,20 60))
2	POLYGON((80 90,120 90,130 50,76 50,70 70,80 90))

droga

FID	ASTEXT(GEOMETRY_ATT)
1	MULTILINESTRING((30 20,54 22), (54 22,80 30,110 20))

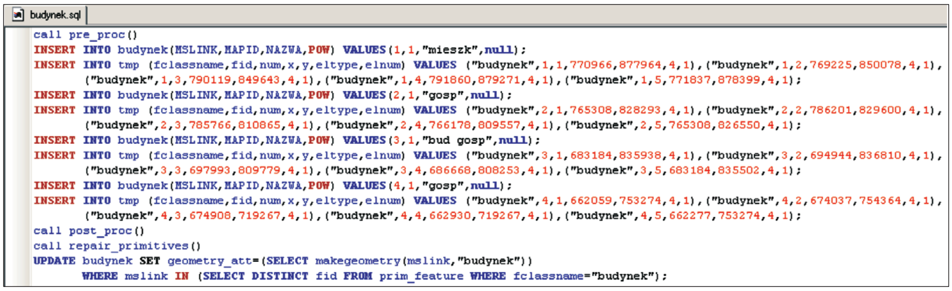
Fig. 3. The format of features' geometric data in database tables: *a*) – the format of features' geometric data in a relational database table based on point primitives; *b*) – the format of features' geometric data using spatial data types. The geometric attributes have been recorded in the feature tables in the column GEOMETRY_ATT (only this column and FID identifier are visible in figure) *Source*: own work.

The Role of User-Defined Routines in the Process of Data Processing

The author of this paper has created several MySQL server routines, which play a rather important part in the process of uploading the data to the server: pre_proc(), post_proc(), Repair_primitives(), as well as function: MakeGeometry().

They were designed to automate and simplify the process of exporting and data processing when working with the system. Once implemented they are an integral part of the database and can be called in the same way as any standard command, function or routine.. Besides, because they are stored on the server, there is an additional possibility to modify what routines do, by editing their code. Thus, the process may be modified and improved without the necessity to tamper with the structure of the application which exports the data. Naturally, such modifications may only be made by a user with the appropriate authorization in the database management system.

The data is uploaded to the server with a program created just for this purpose – Sipos-eksport. It is a desktop program, written in Object Pascal in Delphi environment. Its role is to read data from the original MGE-PC database, convert it into a text file with SQL commands and export it to the spatially-enabled database on MySQL server, discussed above. The process uses the routines discussed here, which make it possible to create tables, load data, convert it to the appropriate format and finalize the process. An sample content of the data file created to export the data to the server is shown below:



```sql
call pre_proc()
INSERT INTO budynek(MSLINK,MAPID,NAZWA,POW) VALUES(1,1,"mieszk",null);
INSERT INTO tmp (fclassname,fid,num,x,y,eltype,elnum) VALUES ("budynek",1,1,770966,877964,4,1),("budynek",1,2,769225,850078,4,1),
     ("budynek",1,3,790119,849643,4,1),("budynek",1,4,791860,879271,4,1),("budynek",1,5,771837,878399,4,1);
INSERT INTO budynek(MSLINK,MAPID,NAZWA,POW) VALUES(2,1,"gosp",null);
INSERT INTO tmp (fclassname,fid,num,x,y,eltype,elnum) VALUES ("budynek",2,1,765308,828293,4,1),("budynek",2,2,786201,829600,4,1),
     ("budynek",2,3,785766,810865,4,1),("budynek",2,4,766178,809557,4,1),("budynek",2,5,765308,826550,4,1);
INSERT INTO budynek(MSLINK,MAPID,NAZWA,POW) VALUES(3,1,"bud gosp",null);
INSERT INTO tmp (fclassname,fid,num,x,y,eltype,elnum) VALUES ("budynek",3,1,683184,835938,4,1),("budynek",3,2,694944,836810,4,1),
     ("budynek",3,3,697993,809779,4,1),("budynek",3,4,686668,808253,4,1),("budynek",3,5,683184,835502,4,1);
INSERT INTO budynek(MSLINK,MAPID,NAZWA,POW) VALUES(4,1,"gosp",null);
INSERT INTO tmp (fclassname,fid,num,x,y,eltype,elnum) VALUES ("budynek",4,1,662059,753274,4,1),("budynek",4,2,674037,754364,4,1),
     ("budynek",4,3,674908,719267,4,1),("budynek",4,4,662930,719267,4,1),("budynek",4,5,662277,753274,4,1);
call post_proc()
call repair_primitives()
UPDATE budynek SET geometry_att=(SELECT makegeometry(mslink,"budynek"))
     WHERE mslink IN (SELECT DISTINCT fid FROM prim_feature WHERE fclassname="budynek");
```

Fig. 4. A sample content of a file with SQL queries which exports descriptive and geometric feature attributes
Source: own work.

As shown in Figure 4, procedures are called in the appropriate places before and after the data is loaded by SQL commands. The execution of all calls from the file results in the data being exported to the server in the appropriate format, designed by the author.

The procedures mentioned above, *pre-proc* and *post-proc* perform the following actions:

– pre_proc – among others, initializes the export process, creates temporary tables, assigns identifiers.

– Post_proc – finalizes the data export process, deletes temporary tables.

The role of the routine *Repair_primitives()* and function *MakeGeometry()* is the conversion of the data after it has been exported to the server.

Repair_primitives() deletes the duplicates of primitives. Duplicates of primitives are deifined as those points, for which the coordinates $X_n = X_m$ and $Y_n = Y_m$, while PID identifiers $_n \neq$ PID$_m$. Procedure *Repair_primitives()*, first of all, deletes the duplicated records in the table *primitive* so that only one instance remains. Secondly, it updates the appropriate fields in the table *prim_feature* where PIDs of deleted primitives were located and replaces them with the identifier of the primitive which was not deleted. Duplicates of this kind occur when linear elements meet (the end of one element meets the beginning of another) and when features share boundaries. Procedure *Repair_primitives()* is called directly with: call Repair_primitives().

Funcion *MakeGeometry()* (Fig. 5) creates and updates geometric attributes of features in the feature table. The function has the following input parameters: feature identifier (*feature_id*) and class name (*fclassname*). The output is a geometric (spatial) type value. The input data on which the function operates is taken from the tables *primitive* and *prim_feature*. Based on the information from these tables the function creates and returns geometric values appropriate for a given feature, using MySQL functions which operate on spatial data types.

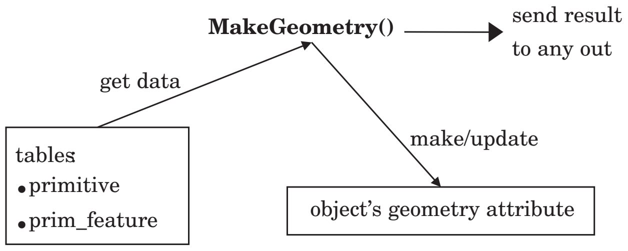


Fig. 5. The role of the function MakeGeometry()
Source: own work.

Function *MakeGeometry()* is used as a part of an SQL query, so it can serve a number of purposes, for example to update geometric values in an feature table. An example of a query which performs this task is shown below.

– A query updating geometric attributes of all features of the class autostrada in the table autostrada:

```
UPDATE autostrada
SET geometry-att=SELECT MakeGeometry(mslink,"autostrada")
          WHERE mslink IN
          (SELECT DISTINCT fid FROM prim_feature
                    WHERE fclassname="autostrada");
```

Result:

All values of geometric attributes will be generated and inserted into the field *geometry_att* of the table *autostrada.*

When selecting data from a table with spatial data type, geometric attributes are treated in the same way in SQL queries as descriptive attributes. For this reason they can be retrieved together for a feature or a set of features, for example:

```
SELECT mslink,szerokosc,pasy_ruchu,astext(geometry_att) as geometria
         FROM autostrada WHERE mslink=3892;
```

Result: shown in table 1.

Table 1

Result of SQL query

Mslink	szerokosc	Pasy_ruchu	Geometria
3892	10	2	MULTILINESTRING((81058880 43978312,80650010 43893707,80366165 43881624,80172909 43911833,79949457 43990377,79798476 44068921,79587102 44141423,79492733 44158282),(79492733 44158282,79418003 44171632,79280809 44152701),(79280809 44152701,79073766 44123297,78892588 44093088,78669136 44002461,78439644 43905791,78343016 43875582))

Descriptive data is readable immediately after download, independently of the method used. An application able to read and draw an feature based on geometric value is needed to see a visualisation of the geometry. This format can be read, among others, by GRASS – a free GIS system. For the purpose of this paper the author wrote his own application in Java.

Internet Client

I have written the program *Sipos-klient* as an example of a tool for data retrieval from the database discussed in the article. The program can be used to test the basic capabilities of the database. Its functions make it possible to:

– connect to MySQL database,

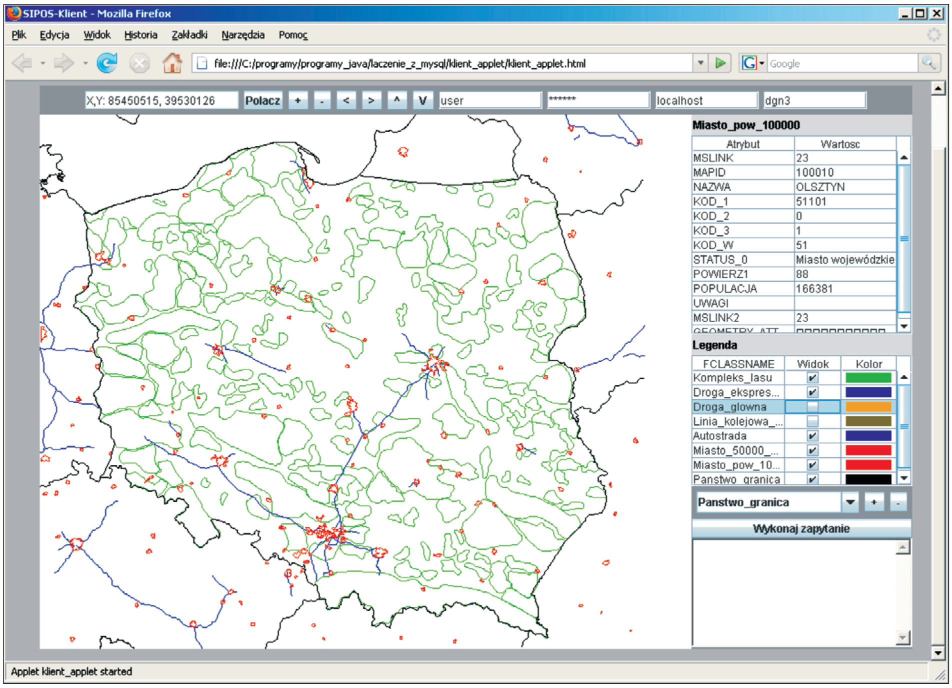
– retrieve the classes of features available in the database,

– send a query which selects features – attribute analysis,

– display and browse through features form the selected classes, retrieve the values of features' attributes, in a simple way adjust the view of the displayed feature classes and the results of queries (on/off, colour change, scaling, movement).

The addition of other functions to the program *Sipos-klient*, like analytic or editorial functions, would make it useful for specific purposes expected of this kind of software.

Sipos-klient connects to MySQL server using a JDBC driver[2] for MySQL. It can be used on a desktop computer or over the Internet. Java technology makes it possible to run this program under any operating system with a browser and Java Runtime Environment installed.



file:///C:/programy/programy_java/laczenie_z_mysql/klient_applet/klient_applet.html

X,Y: 85450515, 39530126 | user | ****** | localhost | dgn3

Miasto_pow_100000

Atrybut	Wartosc
MSLINK	23
MAPID	100010
NAZWA	OLSZTYN
KOD_1	51101
KOD_2	0
KOD_3	1
KOD_W	51
STATUS_0	Miasto wojewódzkie
POWIERZ1	88
POPULACJA	166381
UWAGI	
MSLINK2	23

FCLASSNAME	Widok	Kolor
Kompleks_lasu	✓	🟩
Droga_ekspres...	✓	🟦
Droga_glowna	☐	🟧
Linia_kolejowa_...	☐	🟫
Autostrada	✓	🟦
Miasto_50000_...	✓	🟥
Miasto_pow_10...	✓	🟥
Panstwo_granica	✓	⬛

Panstwo_granica

Applet klient_applet started

Fig. 6. Sipos-klient. Main window

Source: own work.

Figure 6 shows the interface of *Sipos-klient* run in Firefox browser. The upper part contains navigation buttons for scaling and scrolling the map, as well as editing fields for entering database connection parameters. The left-hand bottom part of the window, which occupies the most space, shows a map consisting of the feature classes retrieved from the database. The right-hand side shows the attributes of a selected feature, the key – a list of the displayed feature classes, a context menu which allows to select a feature class from the database, buttons which add and remove classes from the list and a window for

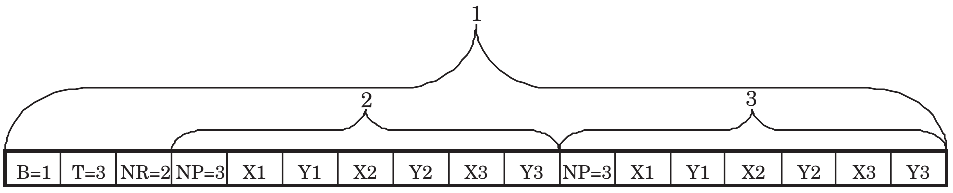
[2] JDBC – *Java Database Connectivity* – an interface for Java programs, analogous to ODBC. It allows to connect to the database using SQL. In order to work with a Java application a database must have a dedicated driver which uses JDBC interface. For MySQL, *Conector/J* is one of such drivers. It translates JDBC calls into MySQL network protocol.

entering and executing queries for the table with the attributes of the selected feature class.

Sipos-klient uses a database created earlier using *Sipos-eksport*. Feature tables with geometric attributes filled in (field *geometry-att*) and table *feature* are also necessary.

Sipos-klient retrieves geometric attributes of a feature from the database as binary streams in WKB format. OpenGis (ECKES 2006) specification (*OPEN-GIS Implementation Specification* 2005) contains a description of how to implement structures representing geometry in this format. According to this description, each geometry is built out of basic elements, such as points (*Point*) or linear rings (*Linear ring*). Each structure consisting of simpler structures (for example MultiLinestring) is built using already defined features. The following is an example of a definition of a structure for geometry of *WKBPolygon* type in C programming language:

```
WKBPolygon {
byte byteOrder;
uint32 wkbType; // 3
uint32 numRings;
LinearRing rings[numRings];
};
```



explanations:
1 – WKB Polygon
2 – ring 1
3 – ring 2

Fig. 7. Feature geometry in WKB representation. Byte order – little-endian[3] (B=1), type POLYGON (T=3), contains two elements linear ring (NR=2), each consisting of three points (NP=3).
Source: *OpenGIS Implementation Specification for Geographic information – Simple feature access –* Part 1: *Common architecture*. http://www.opengeospatial.org/.

The definitions discussed above have been used to determine how to represent, buildand manage a map composed of geometrical features. Using this information, a model of the classes implemented in *Sipos-klient* has been designed.

[3] *Little-endian* byte order means that the least significant byte is first. It is an important piece of information when reading consecutive values in the stream, which determines the order in which bytes are read.

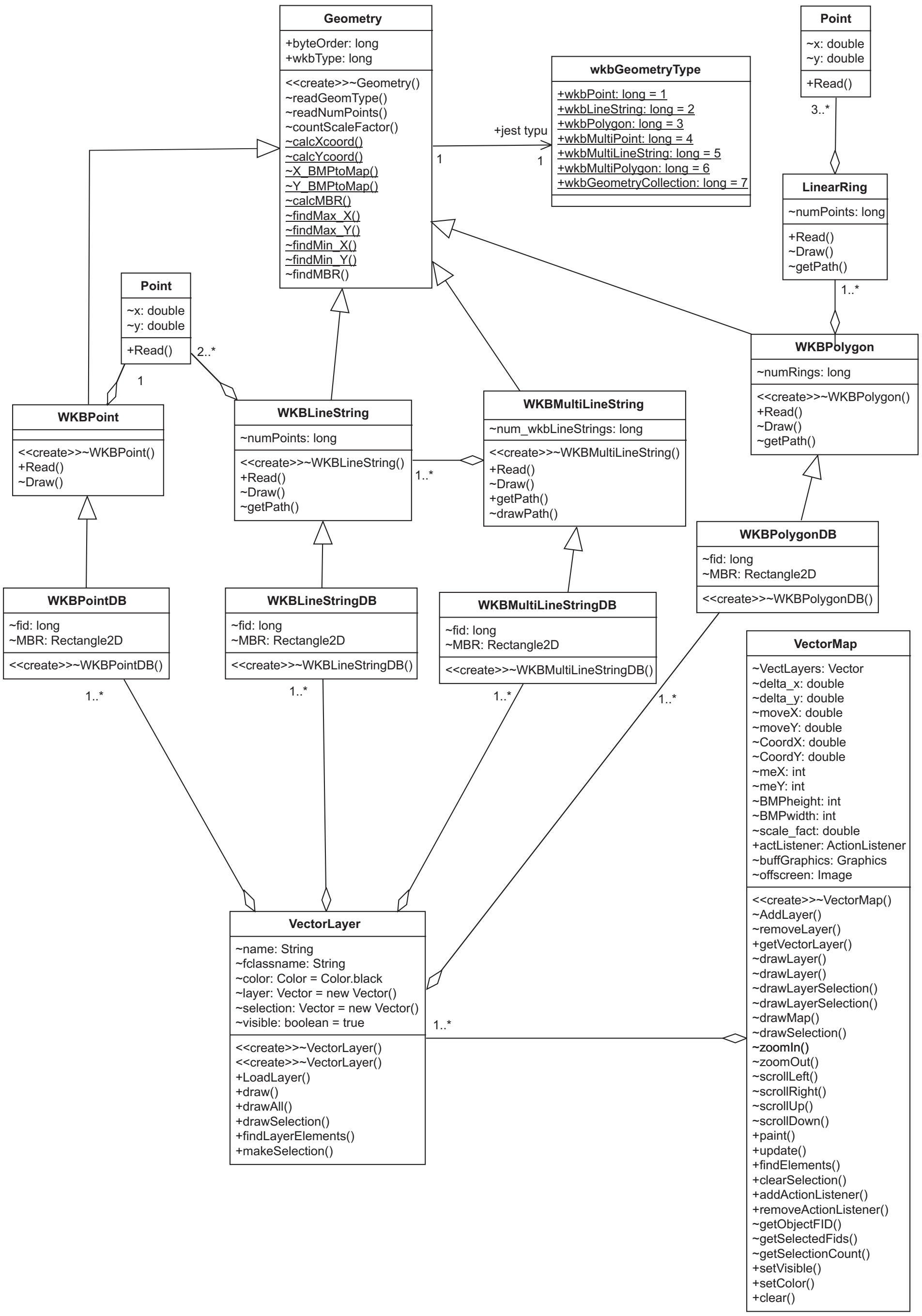


Fig. 8. UML diagram showing classes in Sipos-klient

Source: own work.

The flow chart below (Fig. 8) shows the classes which are the most important for the structure of the inner layer of the program, which operates on geometric features. The chart does not show the relationships with classes used in the user interface. The whole structure may be categorised into two interrelated class areas:

1. Geometric model, with the class *Geometry* as the main element. Classes which represent feature types are derived form it.

2. Vector map, consisting of layers, which in turn are constructed as collections of specific GIS features. Features are assigned to a layer based on a specified criterion. The default criterion is the membership of a specific class. However, the model allows the use of other rules, for example the result of a query which contains features of different classes may be presented as a separate layer.

There is an intermediate layer between the geometric model and the vector map. It connects geometric and descriptive attributes. The layer consists of classes derived from the geometric model classes: *WKBPointDB, WKBLinestringDB, WKBMultiLinestringDB, WKBPolygonDB*. Features in these classes have an identifier *fid* linking them to a corresponding record in the attribute table. This information is used for the identification of the feature by the user, for example when the user points at the feature on the map and retrieves its descriptive attributes.

Conclusion

The concept of a spatially-enabled database discussed in this paper is based mostly on open source software and open standards. The relational MySQL database management system used here can be used for GIS purposes, as shown by the functionality of spatial extensions, data types, user-defined functions and routines described above. The mechanisms of spatial data processing implemented in MySQL comply with OpenGIS specifications. As a result, so does the created database. This enables interoperability with other databases created in the same standard. It should also be noted that MySQL is available free of charge, which considerably reduces costs.

A spatially-enabled database on a MySQL server makes it possible for its data to be accessible on the Internet. The data is accessible on the condition that MySQL is installed on a computer connected to the Internet, with a public IP address. The data can be accessed with any MySQL client, but it is easier and more efficient to use a special application, such as Sipos-klient. The best solution would be for the data to be made accessible on the Internet via a WWW website. The free Apache or Windows Server 2003 Web Edition may be used as the HTTP server. It is not the only method of sharing this kind of data however. More complex solutions can also be employed (for example, using so called distributed architecture), which additionally use script languages (for example PHP[4], Python, VBScript, JScript i in.), application servers (Apache Tomcat[5], IBM WebSphere Application Server, Oracle Application Server, Windows Server etc.) and web services (a large number of useful web services has been proposed in OGC open specifications). MapServer – an environment for building web applications – can also be used for sharing the data. Access to the data from a spatially-enabled database may also be provided using a desktop client application, e.g. written in Delphi environment, which directly accesses the server without the help of a website.

Free software and open standards are becoming more and more popular not only among individuals and companies, but even governments in Europe and worldwide. It is reasonable to bear this tendency in mind when choosing a commercial product. It is often worthwhile to consider its open source counterparts. It may turn out that despite a smaller number of functions a free alternative will meet the user's expectations. The database proposed here as an answer to a specific problem using open source software and own solutions shows that the goal can be achieved at a lower financial cost.

Translated by Łukasz Arendarski, Michał Bednarczyk

Accepted for print 22.03.2011

References

Axmark D., Widenius M. 2007. *MySQL 5.0 Reference Manual*. MySQL AB.

Banachowski L. 1998. *Bazy danych Tworzenie aplikacji*. Akademicka Oficyna Wydawnicza PLJ, Warszawa.

Bednarczyk M. 2005. *Oprogramowanie open source na potrzeby systemów informacji przestrzennej na przykładzie systemu zarządzania relacyjną bazą danych MySQL*. Biuletyn Naukowy UWM, 25: 215–222.

Bednarczyk M. 2008. *Zastosowanie systemu zarządzania relacyjną bazą danych MySQL do automatyzacji przetwarzania informacji geograficznej*. Biuletyn Naukowy UWM, 29: 5–15.

Eckes K. 2006. *Modelowanie rzeczywistości geograficznej w systemach informacji przestrzennej*. Roczniki Geomatyki, IV(2): 43–73.

Gaździcki J. 2007. *Standardy otwarte w geomatyce*. Roczniki Geomatyki, V(2): 7–11.

Hamlet D., Maybee J. 2003. *Podstawy techniczne inżynierii oprogramowania*. Wydawnictwo Naukowo-Techniczne, Warszawa.

Longley P.A., Goodchild M.F., Maguire D.J., Rhind D.W. 2006. *GIS Teoria i praktyka*. Wydawnictwo Naukowe PWN, Warszawa.

[4] PHP, an Apache HTTP server and a MySQL database server on a Linux system are a very common Open Source software bundle for a website (so called LAMP solution stack).

[5] These are J2EE (Java 2 Platform Enterprise Edition) application servers, with Java the programming language. Of these, only IIS as a Microsoft product supports ASP.NET technology (VBScript, JScript lannguages).

OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 1. *Common architecture.* 2005. OGC (Open Geospatial Consortium), http://www.opengeospatial.org/.

OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 2. *SQL option.* 2005. OGC (Open Geospatial Consortium), http://www.opengeospatial.org/.

STONES R., MATTHEW N. 2003. *Bazy danych i MySQL od podstaw.* HELION, Gliwice.

STARZYŃSKI P. 2004. *Zamówienia publiczne a Open Source – stan prawny*. In: *Budowa społeczeństwa informacyjnego w Polsce i Unii Europejskiej.* Ed. P. Bała. Uniwersytet Mikołaja Kopernika, Toruń, pp. 107–109.