Rich Metadata Model for Business Application with Database Dictionary

Nikolay Pavlov
Faculty of Mathematics and Informatics
University of Plovdiv
236 Bulgaria Blvd., 4003 Plovdiv
Bulgaria

Abstract

In this paper we outline the limitations of the traditional relational database metadata model and compare them against the requirements of the framework for business applications. We describe the database dictionary component of a software framework for development of business applications. Its feasibility is proved in practice by development of several software applications and their successful implementations in production environments.

Keywords: software framework, business applications, metadata, database

1. Introduction

Object-oriented (OO) application frameworks are a promising technology for reifying proven software designs and implementations in order to reduce the cost and improve the quality of software. A framework is a reusable, semi-complete application that can be specialized to produce custom applications (Johnson and Foote, 1988). In contrast to earlier OO reuse techniques based on class libraries, frameworks are targeted for particular business units (such as data processing or cellular communications) and application domains (such as user interfaces or real-time avionics) (Fayad and Schmidt, 1997).

One of its key features is the variety of ready-to-use components which enable application developers implement various features without writing executable code (Johnson, 1997). These components rely on meta-data to bind themselves with the database and define their specific behavior at run-time. The object-oriented framework for distributed business applications (FDBA) is a powerful platform for developing state-of-art enterprise resource planning (ERP) and customer relation management (CRM) software products. FDBA builds on the approach for dynamic generation of content and runtime behavior, based on extensive metadata (Pavlov, 2011).

Metadata can be defined as data about data (McCray et al., 1999) or information about the information (Steinacker et al., 2001).IBM defines database metadata, or database dictionary, as a centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format (IBM, 1993). Hence, the term *database dictionary* has been adopted by many developers for their software frameworks. Metadata architectures must easily accommodate the notion of a base schema with additional elements that tailor a given application to local needs or domain-specific needs without unduly compromising the interoperability provided by the base schema (Duval et al., 2002).

In this paper we describe the *Database Dictionary* (DBD) of FDBA and how it extends the metadata of the database with specific business-oriented definitions. DBD allows FDBA to:

1. Provide tools to search for data and display data. Users can define custom criteria to find the data they need in addition to the default search, applied by the business logic. FDBA allows users to create a view with any columns from the current data table, but also with any columns from any referenced table. Further, users can define search criteria for data, or filters, on any column from the current data table, any column from any referenced table, and also from any column from any referencing table.
2. Generate dynamic forms to enter and modify data. The framework provides dedicated input controls for various data types. Ultimately, the generated forms can exercise client-side data validation and then respond to the result of the server-side data validation.
3. Perform full data validation checks at server-side, before updating the database.

These tasks are common for all business information systems, and all software frameworks work to make them easy for application developers. Most of them, however, require programmatic configuration of their components and have developers write programming code to achieve the desired effect. Programming code always increases the risk of errors. Therefore, FDBA tries to minimize programming code by providing feature-rich reusable components, which define their run-time behavior following the existing meta-data.

2. Limitations of Relational Database Metadata

Relational application databases contain various metadata: table structure, relations between tables, field types, data size, data importance (required fields), default values, and some data validation checks. This scope is not sufficient to enable the desired level of automation of the system, because:

1. It may not contain proper information how data is presented to users. For example, the display format for a numeric value is not part of the metadata.
2. Some data values may have special logical meaning on application level. For example, a bit value of 0 and 1 normally has no meaning to the end user. For him, a more appropriate presentation would be, for example, "closed" or "open".
3. Multiple data attributes can be presented as a single attribute to the user, or in a single complex control.
4. Application data validation rules may not fully overlap with the database validation rules. For example, on application level a data attribute can be required for some users, but optional for others; such behavior cannot be defined on database level.
5. On application level, there might be logical relations between entities, which are not feasible on database level.
6. Names of entities and attributes are usually not user friendly due to syntactic limitations of the database naming system. In addition, these names cannot be used in multi-language environment, where users of different language group require different translations of these names.

These limitations of the database metadata model are well-known to the application developers. Various frameworks work to overcome those by providing means to annotate data. For example, Microsoft ASP.NET uses the Data Annotations namespace with attribute classes that are used to define metadata for ASP.NET data controls (2013).

3. Existing Works

There are numerous frameworks and tools which feature a database dictionary component (Kumpati, 1988; Zhang et al., 2002; Base One Int., 2012; Lim, 2012; Radicore Soft., 2013; Entity Framework, 2013; Hibernate ORM, 2013;ASP.NETDataAnnotations, 2013). Most of them target one of the following goals, but others target two of them:

1. Facilitate creation and management of efficient database structures.
2. Hide the physical model of the database and provide a richer object-oriented model.
3. Automatic implementation of business logic and user interface.

These frameworks in total provide almost all functional features of the current work. However, as single frameworks they cover only part of the requirements and limitations, defined above. Combing these frameworks is not always possible due to interoperability issues. Therefore, FDBA features its own approach to building of database dictionary.

4. Key Features of DBD

DBD has the following key features:

1. Provides a topology map with the database entities, their attributes (fields) and the links between them.
2. Single-point of generating SQL. User searching activities require dynamic generation of SQL statements across the application. DBD provides all other components of FDBA with a programming interface to request data, and generates the SQL statements.
3. Extends the database data types to types of higher logical, i.e. application level. Data is not regarded in its raw form (integer numbers, floating-point numbers, decimal numbers, text, date and time, etc.). Instead, it is regarded as business types – the types that users operate with. For example, users distinguish between item quantity and monetary amount. The two logical types have different presentation and input requirements, while for the database both are persisted and decimal numeric data.

4. Describes how data should be presented to application users. This includes formats, which are not subject to the system regional settings. For example, the precision of quantities, or phone numbers.
5. Hides "internal" data attributes. Every database has certain data attributes, which are for internal use by application developers and database administrators, and are not for application users to access.
6. Enables automatic presentation of data by defining attributes, which identify the data in a user-meaningful way. It is well-known that natural primary keys are not always good options for primary keys, and then there should be clear indication which field or combination of fields defines the uniqueness of the record from users' perspective.
7. Defines validation rules. These rules are related to application logic, which cannot be enforced on database level. For example, a data attribute may require a value only if another data attribute has a certain value.
8. Defines virtual entities, or custom views of data tables. In effect, virtual entities are analogous to database views. The difference, however, is that these entities can be empowered with application logic. For example, the results of the entity can depend dynamically on a value of a data attribute.
9. Defines additional relations between data entities, where database foreign keys are inappropriate. Used with logical entities.
10. Provides user-friendly names of entities and attributes with multi-lingual support.
11. Builds on top of the database's metadata. The DBD can construct itself from the application database, and let application developers only add the necessary metadata, or override the one from the database. This significantly speeds the development process.

4.1 Data Topology Map

DBD creates a map of all database entities, their attributes and the links between them. The map is organized as an oriented graph. FDBA uses this map to generate dynamic user interface for users to select fields from the current, referenced and referencing entities. The selection of fields can be used in custom views of data, or for defining search criteria and locate records. The map is also essential for dynamic generation of SQL statements to retrieve data from the database.

The map is stored in-memory for fast access. The map enables the respective components of FDBA to quickly find all referenced and referencing entities for the current entity.

4.2 Single-point of Generating SQL

DBD provides a method which accepts:

- Name of root entity.
- List of field paths with columns to return in the result.
- List of conditions to search on. A condition is built from a field name, comparison operator, and searching value.

The method returns a result set with values from the database, which match the requested search.

4.2.1 Field Paths

The field paths describe how the actual field can be reached from the root entity by traversing the foreign keys. The format of the path is:

FieldName[\FieldName [\ …]]

The field path is parsed from left to right, and every name is analyzed:

1. DBD discovers the corresponding field for the current name.
2. If the field is not a lookup field, the actual field is returned. No further parsing is performed.
3. If the field is a lookup field, DBD identifies the referred table. Then, it continues with the next field on the path. If the path ends with a lookup field, the DBD automatically uses the default user identifier field for the referred table.

Examples:

PolicyNumber
Client\Name
Client\Country
Client\Country\Code

4.2.2 Construct SQL

The algorithm parses the field paths from both the columns to return and the search criteria.

For every item on the column path:

1. If column is not lookup – get column. Add the column to the list of selected columns.
2. If column is lookup – get lookup entity.
3. Construct the join link using the primary key of the lookup entity and the current column.
4. Check if the current join link is already available; if not:
    a. Add the lookup entity to the list of joined entities.
    b. Add the join link to that joined entity.

The algorithm guarantees unique joins. The risk of excessive joins is avoided, and there are no performance risks if proper indices are created in the database.

For every unique join, a temporary unique alias name is generated. The alias is used to reference the fields and the tables in the SQL statement.

After that the resulting lists are enumerated and the actual SQL statement is constructed using the physical table and field names, and the generated alias names. If the foreign key fields are required, the joins are created as INNER, otherwise – as OUTER. CROSS JOINS are not supported.
Search criteria are always applied in the WHERE clause of the SQL statement.

4.3 Logical Field Types

Databases generally operate with “simple” data types (Nordstorm, 1978; SQL Server Data Types, 2013), such as text, numeric, date time, binary, etc. These simple types contain little specifics about how data is interpreted by the above-standing application. On the other hand, applications and users work with both simple data types and with high level concepts such as names, phone numbers, zip codes, etc. For example, a phone number and a zip code can both be persisted in the database into text fields, but the two bear very different presentation and validation rules.

FDBA introduces several high level data types:

- Flag / option.
- Multi-language foreign key field.
- Monetary amount with currency.
- Time field.
- Period field.
- Password field.
- Memo field.
- Rich content field.
- Colors.

4.4 Internal Fields

Internal fields are fields in the database which are user internally by the system, and are never presented to the users in the application. FDBA should be aware of these, and hide them from the user interface. DBD supports that fields are marked as “internal”.

4.5 Entity Identifiers

For various reasons, FDBA employs the model of surrogate primary keys. As a result, the natural key of tables is not known on database level. DBD solves this problem by letting application developers specify the natural key, or *user identifier*, for every entity. DBD goes one step further by providing support for extended user identifiers, i.e. allow the user interface to present the values of multiple fields to help users identify the record.
Extended user identifies are especially useful for m→n-relationship entities, where the user-meaningful identifier is usually some extra field, out of the indexed field.

4.6 Validators

Data validators are key instruments to guarantee data integrity, security, and enforce the business logic of the application on the database.

DBD inherits some validation rules from the database, such as: required fields, and unique values. Other database validation mechanisms, such as range checks and validation behavior of triggers, are not recognized by the DBD. They take effect on actual attempt to update data. Failure to meet these requirements results in an error message, which is logged by FDBA and passed to the user interface as user notification.

DBD supports two primary types of validation, based on their execution environment: client-side and server-side. Client-side validation is performed by the client application of FDBA before data is sent over the network to the application server. Client-side validators do simple checks on the current data record only:

1) Required values.
2) Value within range.
3) Custom validation via a developer-supplied predicate method in the application.

Server-side validators are performed by the server application when the client application requests a data update. Server-side validations include all client-side validations plus additional validation mechanisms, which require access to the database, such as:

1) Uniqueness of values.
2) Check condition using a custom SQL statement.
3) Check condition using a database stored procedure.

The application server also executes all client-side validations to guarantee data integrity as a security measure against malfunctioning client, or a rogue attempt to change data during network communications.

Based on their impact, validators are also classified as errors and warnings. Error validators do not allow data modifications if the validation condition is not met. Warnings allow the modification, but ask the user for explicit confirmation for the action to be taken.

4.7 Virtual Entities

DBD defines virtual entities as custom views of data from one physical table, or from a database view. Virtual entities are very similar to the database views, but have one significant advantage: virtual entities can be defined with wildcards, and will get updated automatically when structural changes are made in the database. In some relational database systems, database views do not handle table rebuilds and structural changes.
Virtual entities in DBD allow application developers to assign different roles to database fields depending on the context of business logic.

The example defines two virtual entities for one physical table, which present different subsets of the data, and are used as referenced fields in two other entities. On physical level, there are only two tables, with a single relationship between them. The example clearly demonstrates how DBD can extend the metadata from the database.

```
<EntityPhysicalName="CLUBS_UWRITERS"Name="InsurersPolicies"MainTable="CLUBS_UWRITERS"Cust
omSql="select * from CLUBS_UWRITERS where CU_KIND = 1">
<IntegerFieldPhysicalName="CU_KIND"Internal="true"DefaultValue="1"/>
<LookupFieldPhysicalName="RELATION_ID"Name="RelationId"ToEntity="Relations"Required="true"/>
<FixedListFieldPhysicalName="CU_CLUB_IT_SCENARIO"Required="true"DefaultValue="4">
… omitted for readability
</FixedListField>
</Entity>
<EntityPhysicalName="CLUBS_UWRITERS"Name="InsurersCorrespondents"MainTable="CLUBS_UWRITE
RS"CustomSql="select * from CLUBS_UWRITERS where CU_KIND = 2">
<IntegerFieldPhysicalName="CU_KIND"Internal="true"DefaultValue="2"/>
<LookupFieldPhysicalName="RELATION_ID"Name="RelationId"ToEntity="Relations"Required="true"/>
<FixedListFieldPhysicalName="CU_CLUB_IT_SCENARIO"Required="true"DefaultValue="4">
… omitted for readability
</FixedListField>
</Entity>
<EntityPhysicalName="VIEW$CLAIMS_A"Name="ClaimsA"UserIdentifier="FileNumber"MainTable="CLAI
MS">
… omitted for readability
```

```
<LookupFieldPhysicalName="CLAIM_INSURER_CLUB_ID"ToEntity="InsurersPolicies"ReadOnly="true"/>
… omitted for readability
</Entity>
<EntityPhysicalName="VIEW$CLAIMS_C"Name="ClaimsC"UserIdentifier="FileNumber"MainTable="CLAI
MS">
… omitted for readability
<LookupFieldPhysicalName="CLAIM_INSURER_CLUB_ID"ToEntity=" InsurersCorrespondents
"ReadOnly="true"/>
… omitted for readability
</Entity>
```

4.8 Relationship between entities

By default, DBD uses database foreign keys to create relations between entities. In addition, it allows application developers to define relations, which do not or cannot exist in the database.

4.9 User-Friendly Names

User-Friendly names are the names of entities and attributes used by the user interface. Database management systems always have syntactic restrictions for naming, which can conflict with the names of the real-life facts they present.

DBD is designed with multi-language support in mind. Every name is mapped to a key. Keys can be assigned a different actual text value for every language, supported by the system. The text values are stored in Unicode, thus providing support for virtually any language.

During user logon the system retrieves the actual text values for the language of the user and caches them in the user session.

4.10 Initialization

Business applications contain a large number of small tables with simple structure. This is a common effect of normalization levels, higher or equal to 3.To save development costs DBD tries to retrieve as much information from the application database as possible. Application developers are required only to provide the missing additional details, and override default information.

DBD analyzes the structure of the application database and populates its internal structures. It uses the system tables from this task. DBD identifies:

- All tables.
- All relations between tables.
- All fields – their names, types and attributes.

The next step is to load the definitions from the persistence storage. The definition can:

1) Extend the metadata with additional information.
2) Add new elements to DBD: entities, fields, relations.
3) Completely override the definitions from the database.

Examples:

```
<EntityPhysicalName="FUNCTIONS"/>
```

Defines a table, where all fields are defaulted. The name of the table in the database is also considered friendly and is defaulted.

```
<EntityPhysicalName="PROPERTIES_CON"Name="PropertiesOfContacts"/>
```

Defines a table, where all fields are defaulted. The name of the table is overrode, because the database name is considered obscure.

```
<EntityPhysicalName="CONTACTS"AdditionalIdentifier="Fullname">
<FixedListFieldPhysicalName="CNTCT_SEX">
<OptionValue="0"TextKey="sexmale" />
<OptionValue="1"TextKey="sexfemale" />
</FixedListField>
<MemoFieldPhysicalName="CNTCT_MEMO"/>
<DateTimeFieldPhysicalName="CNTCT_BIRTHDATE"DateWithTime="false" />
</Entity>
```

Defines a table, where most fields are defaulted. Field “CNTCT_SEX” is defined with a list of values for the user to choose from; these are also all valid values for the field. Field “CNTCT_MEMO” is defined as a memo field, meaning the UI should define a larger text editing area for it. Field CNTCT_BIRTHDATE is defined as a DATETIME field and that only the DATE part of the data is relevant for the system; TIME should be ignored.

5. Database Dictionary Format and Persistence

DBD is defined as an XML document. The hierarchical structure and machine-parsing are the two key features of XML which made it the best candidate for defining DBD.

```
<?xmlversion="1.0"encoding="utf-8" ?>
<DataDictionaryxmlns="http://www.framestory.nl">
<EntityPhysicalName="FUNCTIONS"/>
<EntityPhysicalName="COUNTRY"Name="Countries"/>
<EntityPhysicalName="PROPERTIES_CON"Name="PropertiesOfContacts"/>
<EntityPhysicalName="CONTACTS"AdditionalIdentifier="Fullname">
<DefaultView>
<ColumnName="Search" />
<ColumnName="Fullname" />
<ColumnName="Phone" />
<ColumnName="RelId/Search" />
</DefaultView>
<FixedListFieldPhysicalName="CNTCT_SEX">
<OptionValue="0"TextKey="sexmale" />
<OptionValue="1"TextKey="sexfemale" />
</FixedListField>
<MemoFieldPhysicalName="CNTCT_MEMO"/>
<DateTimeFieldPhysicalName="CNTCT_BIRTHDATE"DateWithTime="false" />
</Entity>
<EntityPhysicalName="CNTCT_COMMS"Name="ContactCommunications"UserIdentifier="Description">
<LookupFieldPhysicalName="CNTCTCOM_TYPE"Name="TypeId"ToEntity="CommunicationTypes" />
<TextFieldPhysicalName="CNTCTCOM_DESCRIPT"Name="Description" />
<LookupFieldPhysicalName="CONTACT_ID"Name="ContactId"ToEntity="Contacts" />
</Entity>
<EntityPhysicalName="COMTYPE"Name="CommunicationTypes">
<TextFieldPhysicalName="COMTPE_DESCRIPT"Name="Description" />
</Entity>
<EntityPhysicalName="CONTACT_PROPERTIES">
<LookupFieldPhysicalName="CNTCT_ID"Name="ContactId"ToEntity="Contacts" />
<LookupFieldPhysicalName="PROP_ID"Name="PropertyId"ToEntity="PropertiesOfContacts" />
</Entity>
</ DataDictionary>
```

An XML schema is also created, which enables validation of the DBD. The schema enables development of graphical tools for authoring the database dictionary in future.

DBD is stored in as a resource in the installation folder of the server application, along with the binaries on FDBA. On application startup it is loaded in memory. When a user logs in the system, the server application creates a copy of DBD for the current user, stores it in the user session, and sends it to the client application.

6. Data Structures and Helper Functions

6.1 Data Structures

The Database Dictionary is stored in memory as a collection of entities, organized in a bi-directional graph. FDBA traverses the graph to find fast how entities are connected not only directly but also via other entities.
Each entity is a collection of field objects, a default view, and validator objects.

Fields objects have the following hierarchy:

```
FieldBase
    SystemField
        IdField
        FlagDeletedField
    NumericField
        IntegerField
            FixedListField
        FloatField
        MoneyField
            CurrencyField
        TimeUnitField
    StringField
        MemoField
        RichTextFormatField
        LabelField
        PasswordField
    LookupField
    MultilingualLookupField
    DateTimeField
        DateField
        TimeField
    ColorField
    BinaryField
```

Validators have the following hierarchy:

```
ConstraintBase
    RequiredConstraint
    UniqueConstraint
    ClientConstraint
        MethodConstraint
    DatabaseConstraint
        SqlConstraint
        StoredProcedureConstraint
```

6.2 Helper Functions

DBD exposes a programming interface, which enable external components:

1) Retrieve full information about an item: field, or entity.
2) Quickly find an entity, or field by name, or field path.
3) Find the shortest relation between two entities. This function uses Dijkstra's algorithm.
4) Find all the relations between two entities.

7. Conclusion

With the help DBD, FDBA makes it possible to develop the backbone of a business application without writing programming code. DBD increases the reusability level of software components in FDBA and enables features like fully customizable search screens and powerful search filters. FDBA is successfully used for the development of several software products. Among the prominent implementations are enterprise management systems for shipping insurance brokerage and underwriting for companies in the Netherlands. The latest version of FDBA is used to develop a CRM system for insurance management company. The application was developed fully only with metadata definitions of DBD and the Application Dictionary (Pavlov, 2011) of FDBA; no programming code was required.

Another area of application is distributed services and systems for e-learning. Several systems have been developed, as described in (Rahnev et al., 2005; Rahnevand Rahenva, 2007; Pavlov and Rahnev, 2006;Golevet al., 2009; Rahneva, 2003). The Distributed eLearning Platform – DisPeL (Rahnev et al., 2014) is implemented in several high educational institutions in Bulgaria.

8. Acknowledgements

Parts of this work are supported by NPD of Plovdiv University "PaisiiHilendarski", Bulgaria, grant NI13 FMI-002.

References

ASP.NET Data Annotations, Microsoft (2013).http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations(v=vs.110).aspx

Base One International (2012).Base One Database Dictionary,http://www.boic.com/b1ddic.htm

Duval E., Hodgins W., Sutton S., and Weibel S. (2002).Metadata Principles and Practicalities. D-Lib Magazine, Volume 8, Number 4, April 2002, ISSN 1082-9873.

Entity Framework, Microsoft (2013).http://msdn.microsoft.com/en-us/data/ef.aspx

Fayad M., Schmidt D. (1997).Object-oriented frameworks. Communications of the ACM, Special Issue on Object-Oriented Application Frameworks, Vol. 40, No. 10, October 1997.

Golev A., O. Rahneva, A. Rahnev (2009). Algorithms to Minimize the Number of Unique Tests in Real Group Testing Examination.*Scientific Works, Plovdiv University*, vol. 36, book 3, 2009 – Mathematics, ISSN 0204–5249, pp 39-49.

Hibernate ORM (2013).[Online] Available: http://hibernate.org/

IBM (1993). IBM Dictionary of Computing, 10th edition, ACM, 1993, ISBN:0070314888.

Johnson R., Foote B. (1988).Designing Reusable Classes, Journal of Object-Oriented Programming. SIGS, 1, 5 (June/July. 1988), 22-35.

Johnson R. (1997). Frameworks = (components + patterns), Communications of the ACM, Special Issue on Object-Oriented Application Frameworks, Vol. 40, No. 10, October 1997.

Kumpati, Murari (Thornton, CO), Patent (1988). Database management system with active data dictionary, United States, American Telephone and Telegraph Company, AT&T Information Systems (Holmdel, NJ), 4774661, http://www.freepatentsonline.com/4774661.html

Lim J.(2012).ADOdbData Dictionary Library for PHP.http://phplens.com/lens/adodb/docs-datadict.htm

McCray A., Gallagher M., Flannick M.(1999).Extending the role of metadata in a digital library system.*IEEE Forum on Research and Technology Advances in Digital Libraries*, 190 –199.

Nordstorm B. (1978).Assignments and high level data types.*ACM '78 Proceedings of the 1978 annual conference* - Volume 2 ISBN:0-89791-000-1, pp. 630-638.

Pavlov,N., Rahnev,A. (2006). Architecture and Design of Customer Support System using Microsoft .NET technologies.*.NET Technologies 4th International Conference*, May 29 – June 1 2006, Plzen, Czech Republic, Short Papers Proceedings ISBN 80-86943-11-9, pp 21-26.

Pavlov, N. (2011).*Object-Oriented Framework for Development of Distributed Business Applications,* Ph.D. Thesis, Plovdiv University, Plovdiv, Bulgaria (in Bulgarian).

Radicore Software Limited (2013).Rapid Application Development toolkit for building Administrative Web Applications.http://www.radicore.org

Rahnev A., Pavlov N., Rahneva O. (2005).Architecture & Design of Distributed Electronic Testing Server (DeTC) based on Microsoft .NET Framework– *IMAPS CS International Conference 2005*, September 15-16, 2005, Brno, Czech Republic, pp 417-422.

Rahnev A., O. Rahneva (2007).Algorithms for Generation of Circuitries and Drafts in Distributed e-Testing Cluster (DeTC).*Scientific Works, Plovdiv University*, vol. 35, Book 3, 2007 – Mathematics, ISSN 0204–5249, pp 85-100.

Rahnev A., Pavlov N., Kyurkchiev V. (2014).Distributed Platform for e-Learning – DisPeL, European International Journal of Science and Technology, Vol 3, No 1, ISSN: 2304-9693.

Rahneva O. (2003).Testing and Assessment in Distributed e-Testing Cluster – DeTC, *12th International Conference ELECTRONICS'03*, Sozopol, Bulgarian, Conference proceedings, v. 4, pp. 214-219.

SQL Server Data Types, Microsoft(2013).http://technet.microsoft.com/en-us/library/ms187752.aspx

Steinacker A., Ghavam A., Steinmetz R. (2001).Metadata standards for web-based resources.*IEEE Multimedia*, 8(1), 70 –76.

Zhang J., Gruenwald L., Candler Ch, McNutt G, and Chung (2002).Database and Metadata Support of a Web-Based Multimedia Digital Library for Medical Education.*ICWL 2002*, LNCS 2436, pp. 339-350.