STUDIA INFORMATICA
Nr 1(12) Systems and information technology 2009

# Comparative analysis of database access technology

**Andrzej Barczak, Dariusz Zacharczuk, Stanisław Jastrzębowski**[1]
[1] Institute of Computer Science, Academy of Podlasie,
ul. 3 Maja 54, 08-110 Siedlce, Poland

**Abstract.** The paper's subject is comparative analysis of database access technologies. It presents the evolution of the discussed technologies, which are divided into the following categories: single-platform and multi-platform technologies. Sample programs showing the use of programming interfaces, used for initiating and establishing connections with databases and performing simple operations on them. A database diagram, which was used for measuring efficiency of selected technologies as well as testing plan and concept, system and hardware parameters, tested queries and the program skeleton that was used for measuring the efficiency of selected technologies, were presented. The results were grouped according to 4 query types and 3 operating systems. A description was attached to each graph. Conclusions were provided for query tests for each of the operating systems. Finally, the most efficient technologies were discussed, potential causes of increase or decrease in efficiency were presented, and the results were summarized.

**Keywords.** MySQL, Oracle, PostgreSQL

## 1 Introduction

A database is an important link in the process of strategic decision making and allows for efficient functioning of each organized unit or entity. Each company usually has a few different databases. In this case, a database management system must be able to establish connections and communicate with different data sources. It is a situation where database access technologies are particularly useful. Without them, a programmer writing an application that manages different database types would have to include the code for handling commands for each of the databases, used by the program. It would be much more time-consuming, and expensive. Another advantage of database access technologies is also the fact that the user needs to learn to use only one programming interface, and s/he can still use different databases in his or her applications.

There are a lot of technologies that make access to data available. Some of them are more popular, others are less popular, some are more efficient and others are less efficient. They may be commercial products or freeware, etc. The selection of an appropriate technology and customization of a database to suit a given project is not easy and depends on many factors such as the project scale, its use (web network, data warehouse, etc.), the need to provide simultaneous access to many users, database distribution, operation system or hardware configuration (workstation, server). It is a fact that cooperation between some database interfaces and particular databases is better, and it is worse in case of others. There are interfaces that allow access to many different data sources, and such interfaces that are much more efficient than others, although they allow access to only one database.

## 2 Access technologies

The history of access technologies dates back to the 1970s when such database systems as dBase, Paradox, Clipper and FoxPro dominated. Popularization of all kinds of database solutions made their authors create access mechanisms to various databases using different programming languages. In mid 70s Moshé M. Zloof from the IBM Research Centre created one of the first access technologies, which was called Query-By-Example (QBE). This technology is based on a high-level language which manages data, and allows one to build uniform queries, make convenient data updates or manage and control a relational database. One of the main QBE assumptions was facilitating inexperienced database users' learning process. The working principle of QBE is as follows: filling an empty record whose structure is identical to the structure of records in a database with sequences of characters to be found, e.g. putting "Nadarzyn" in the "Place" field or "Kowalski in the "Surname" field. The query returns a list of all the records containing a particular sequence of characters in a given field. QBE system converts the query asked by the user into a formal database query. Therefore, the user can execute complex database queries even though he has no knowledge of formal methods.

Query-By-Example gave the beginning to other data access technologies, such as DAO (Data Access Object), ODBC (Open DataBase Connectivity), OLE DB (Object Linking and Embedding DataBase), or ADO (ActiveX Data Object).

Data Access Object is a component that provides a uniform interface for communication between the application and the data source (e.g. a database or a file). It is often connected with design templates, universal solutions (tested in practice) to frequently appearing, repetitive design problems. Adding DAO to a given application, results in adding another interface layer and increasing the amount of code, which must be executed in order to perform the same action. For this reason, in applications where efficiency is critical, DAO is not added in order to ensure the fastest operation possible.

Another technology that became an alternative to Query-By-Example and Data Access Object is Open DataBase Connectivity (ODBC). Representatives of a few companies producing both hardware and software worked for a couple of

years in SQL-Access Group (SAG) trying to define a universal data access method in order to simplify client/server software. Microsoft company used the SAG group results to create Call-Level Interface (CLI)[1], an interface that was later used to create Open DataBase Connectivity Application Programming Interface. Structured Query Language (SQL)[2] was selected as a language for communication with databases.

One of the interface types that makes it easier for programmers to use database access technologies is Embedded SQL. It allows for using SQL queries directly in the programming language code. Then, preprocessor definitions are called - lower level objects, methods and functions - to which appropriate code is assigned that refers to the database interface. Before C language compiler carries out the final compilation of the program, all the pre-processor instructions will have to undergo pre-compilation – changing commands from SQL into a low-level code.

Another main type of API is the above mentioned Call-Level Interface (CLI). It is a method of SQL instruction execution in a conventional programming language program. First, a connection with a database is initiated using appropriate object that handles connections. Then, an appropriate method, an object, representing SQL instruction or an object handling connections with database is used to execute SQL instruction. This interface type became extremely popular in client/server-based systems. Call-Level Interface is an interface of a lower level than Embedded SQL. Instead of placing SQL instructions in the application code, a programmer places a sequence of standard subprogram calls, which send messages to a given DBMS[3].

ODBC defines a low-level set of functions that make data exchange and instruction transfer possible between client and server applications without the necessity to possess detailed information concerning the structure of both client and server. It applies to any operation performed within the joined area of client/server application, irrespective of whether the client and server run on the same or different computers, or whether they run on different programming as well as hardware platforms.

ODBC interface ensures maximum versatility of the application, and makes it independent of a particular source and kind of processed data. It makes it possible to build an application that does not have to be dedicated to a particular database. It is the user that indicates an appropriate module called a database driver, and connects the application to a particular database containing data to be processed[4].
ODBC interface defines the following:

- Library of ODBC functions that make it possible to connect with database management system (DBMS), execute SQL queries and transfer their results.
- Uniform method of connecting to and logging in the DBMS,

[1] It defines the metod, in which programs should send queries to DBMS and methods used by applications to handle results obtained from a database.
[2] A paper on Open Database Connectivity in Wikipedia. Source: http://en.wikipedia.org/wiki/Open_Database_Connectivity
[3] P. Beynon-Davies, Systemy baz danych, Wydawnictwa Naukowo-Techniczne, Warszawa 2000, p. 123.
[4] W. J. Gilmore, PHP 4.0. Poradnik dla programistów, Mikom, Warszawa 2002, p. 262.

- Uniform data types set,

ODBC interface universality ensures:

- Defining SQL queries during compilation or during application's work,
- Using the same set of functions to connect to different DBMS,
- The possibility to connect with a numerous DBMS instances,
- Sending and receiving data in a uniform format; database driver converts data between types defined by ODBC and particular DBMS types.

ODBC standard defines two types of drivers that are compatible, which allows one system to use two types of drivers:

- Single-tier - the driver processes both ODBC functions as well as SQL queries,
- Multiple-tier - this driver processes ODBC functions and sends SQL queries to the target data source.

In a Single-tier implementation the database functionality is realized directly by the driver. The driver executes SQL queries and acquires information from the database. Local database drivers such as dBase, Paradox and FoxPro may be examples of such drivers.

In Multiple-tier configuration, the driver sends an SQL query to the server, which processes and executes the query. An application, a driver and a driver manager are within the system that is usually called a client. Both data source and software that controls access to data are usually located on another computer called a server.

## 3 Single-platform technologies

The fact that Microsoft created COM (Component Object Model) technology contributed to OLE DB development. COM technology enables efficient communication between applications by defining program components independent of programming language. It allows to include in the developed applications elements that belong to other programs and data exchange between individual objects using so called interfaces. Component Object Model is used in many applications, such as. Microsoft Office, where it allows to combine objects dynamically e.g. from MS Excel spreadsheet to MS Word word processor documents. It is also the basis of such access technologies to databases as OLE DB or ADO[5].

OLE DB technology Object Linking and Embedding Database) like ODBC, COM or ADO was developed by the Microsoft Company. OLE DB is a COM object, which works in a way that is similar to ODBC, but this applies to any data source, and not only to SQL databases. Applications may use OLE DB in order to reach data directly or they may call ODBC via OLE DB in order to get access to ODBC databases. In comparison with open interfaces such as ODBC, OLE DB is a very complex solution and it is difficult to use. Moreover, it is not portable, and

[5] CAS company website devoted to OPC (OLE for Process Control) http://www.cas.com.pl/opc/index.php?strona=com

can only be used by software that is compatible with the Microsoft Windows environment.

ADO (ActiveX Data Objects) technology, which derives from COM model and OLE DB technology is much easier and much more convenient to implement. ADO is an object-based technology, and it is built based on a set of classes and methods that implement defined COM[6] interfaces. ADO may be used in Visual C++, Visual Basic environment or another programming environment, which allows for making references to the types library, and as a result the programmer has access to objects that create these libraries. Fig. 1 presents the ODBC, OLE DB and ADO[7] interactions.

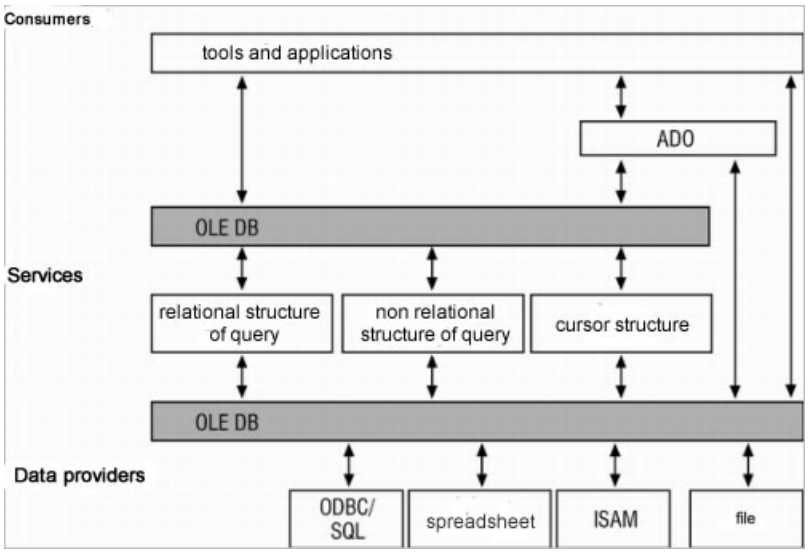


**Figure 3.1.** Interactions between access technologies to Microsoft's data

ODBC and OLE DB popularity resulted in the development of new technologies. It was also connected with the fact that such programming environments as Borland Delphi and Borland C++ Builder developed by the Borland company appeared on the market. New environments required new data access methods. Therefore, such technologies as IBX (InterBase Express), ), IDO (InterBase Objects), FIB (Free InterBase), ODAC (Oracle Data Access Components), SDAC (SQL Server Data Access Components), MyDAC (MySQL Data Access Components), Gemini, EasySoft, BDP.NET (Borland Data Provider for .NET), BDE (Borland Data Engine) were created for these environments. Several companies developed their own technologies for their programming environments e.g. Microsoft (ADO), Sun Microsystems (JDBC, Java DataBase Connectivity) or

[6] W. Dudek, Bazy danych SQL. Teoria i praktyka, Helion, Gliwice 2006, s. 52-53.
[7] IIS 5.0 Resource Kit, rozdział 7: Dostęp do danych oraz transakcje, http://www.microsoft.com/poland/windows2000/win2000serv/IIS/roz_07.mspx

ActiveState (DBI, DataBase Interface). ADO technology was used by other companies. Therefore, ADO uses are not restricted to the Microsoft's developer environments and it may be used in applications developed using programming tools that belong to Borland or ActiveState companies.

In response to more and more imprecise ODBC technology specification, a BDE (Borland Database Engine) was developed. It was developed on the basis of IDAPI (Independent Database API) standard, created by the consortium made up of the biggest IT companies such as Borland, Microsoft, Hewlett-Packard, IBM and Oracle. IDAPI is a BDE programming interface, which, in order to get access to supported databases allows for calls execution at the dBASE and Paradox applications and C++ language level. Unfortunately, despite initial consensus of the consortium members' opinions, it was not widely accepted. As a result, only Borland company supplied drivers based on this technology. The specification itself has not been modified, despite many changes that have taken place in IT world since it was developed.

Despite its undeniable advantages such as speed and simplicity of use, BDE technology, which once used to be very popular, is not widely used, and Borland company ceased to develop it. BDE's disadvantage was its limited portability, and applications that used it could only work in environments where BDE was functioning. Fig. 3.2. shows how the BDE technology working principle.

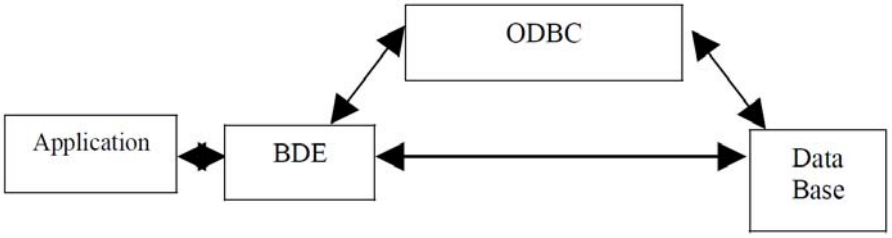


**Figure 3.2.** A diagram showing the BDE technology working principle

# 4 Multiplatform technologies

## 4.1 JDBC technology

Together with the development and popularization of Java, a language that was portable and independent of system platform and hardware architecture, a need arose to create a new database interface that would comply with the assumptions that were similar to those the language itself complied with. The following observations were also in favour of it:

- ODBC was a technology developed and used in environments that were developed on the basis of C and C++. Applications created using them are dependent on the operating system, which is in disagreement with one of the Java language foundations – software portability,

- Rewriting ODBC in Java language did not make any sense due to the differences in language construction between Java and C/C++,
- ODBC, in the opinion of Java authors, was too complex. The newly developed interface was to be much simpler in use, and, at the same time, more functional[8].

This technology, written exclusively in the Java language, was called Java DataBase Connectivity (JDBC). It quickly became popular and was widely used by more and more Java programmers. Nowadays, it is not only used as a uniform access technology to any database but it is also a significant element of higher level applications/interfaces, which allow to access a database at a higher level. Examples of such uses are presented below:

- SQLJ – SQL is embedded in Java code. This code is then pre-processed in order to extract proper SQL commands, which are executed by JDBC.
- Java Blend – allows to map (transform) tables in a relational database directly into Java objects.

Since not all database systems producers prepared JDBC standard implementation in the Java language, and some of them developed hybrid solutions, four basic JDBC driver types may be distinguished:

- JDBC-ODBC bridges – queries formulated in Java are translated into the ODBC driver language. ODBC driver handles communication with the database. Such a solution is connected with potential costs of ODBC programming[9].
- Java to API SZBD (Native-API partly-Java) – a program written in the Java language communicates with the DBMS, and not with the ODBC driver, as above.
- Direct JDBC (JDBC-Net pure Java) – JDBC driver communicates with the intermediate server using DBMS-independent protocol. The server translates commands into a given DBMS protocol and sends it the queries it received. A server may act as a go-between for the data exchange between many clients and many different DBMS. Thus, it is the most general and heterogeneous solution, burdened with security reasons.
- Indirect JDBC (Native-protocol pure Java) – JDBC driver communicates directly with a database using its network protocol. It is the most general solution, successfully used in intranets.

[8] Sieluszko, op. cit.
[9] R. Stones, N. Matthew, Bazy danych i MySQL. Od podstaw, Helion, Gliwice 2002, p. 458-471.

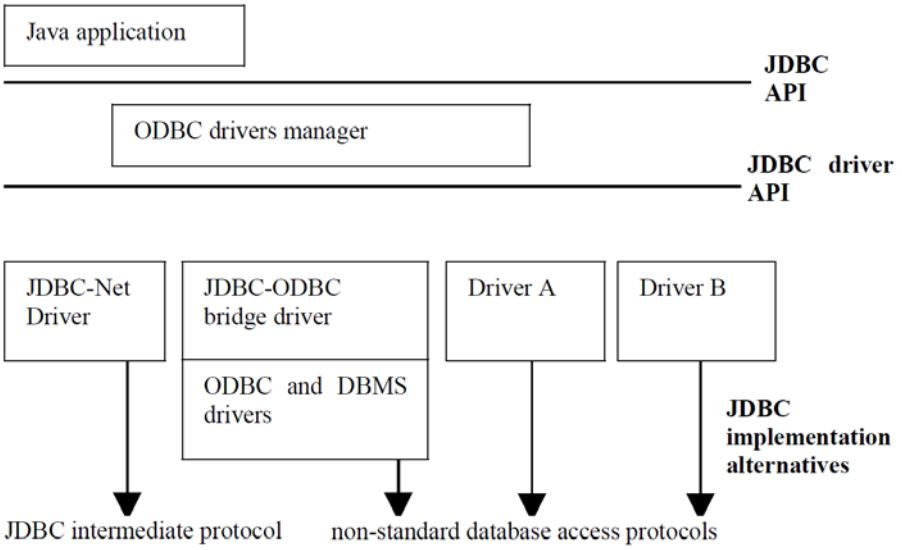


**Figure 4.1.** JD drivers types and their uses

Using a JDBC-Net driver or a JDBC-ODBC bridge driver demands providing software dedicated to a particular platform. Thus, none of these solutions is independent of hardware/system platform.

## 4.2 ADO.NET technology

ODBC and BDE technologies success, as well as programmers' aversion to OLE DB caused by too complex process of configuration and implementation of this technology made the Microsoft company develop a new data access mechanism. This resulted in developing ADO (ActiveX Data Objects) technology, mentioned above, which, similarly to OLE DB, is based on COM objects.

ADO developers aimed at creating a technology that provides access to a database, without the necessity to know its internal structure. BDE technology had a similar property. However, BDE consisted of ordinary components using additional libraries. ADO, in turn, is both an intermediate layer and an access technology. Another assumption ADO developers made, was providing the same support for all database systems. Various DBMS have different functionalities. Therefore, in order to handle all common situations it was necessary to provide support to all functions used in all databases. It was also connected with the need to omit a few functions specific to a given DBMS.

Apart from access to data, ADO (like OLE DB) allows one to access Excel files, flat files, Lotus files, HTML files and many more data sources. Objects that create ADO were shown in fig, 4.2. They use the SQL language, and,

as a consequence, they may be treated as language components for data manipulation.

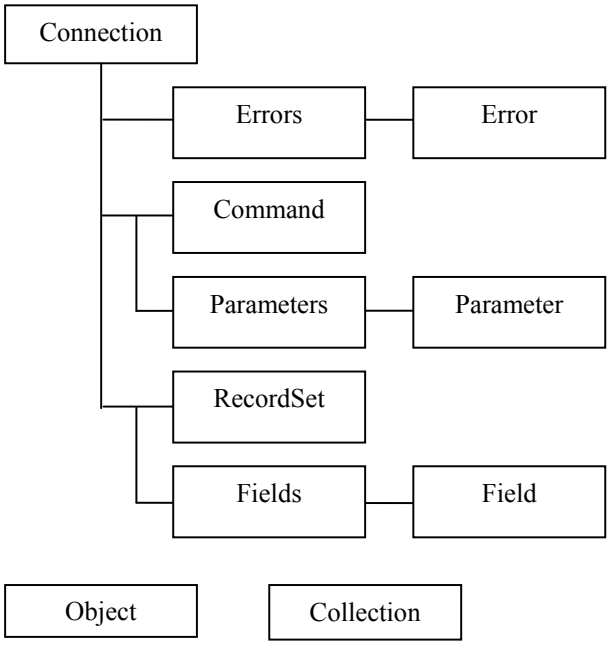


**Figure 4.2.** Hierarchy of objects creating ADO technology

An object, located at the highest level in the ADO hierarchy, is the Connection object, which, by setting appropriate properties in combination with the Open method or just by calling it with specified parameters, allows to connect to the data source. RecordSet object is used to handle all access methods to data. When it is used, one works with unidirectional result streams of queries from a database, views data on the server or views buffered query results. Changes made to the data may be transferred to the database immediately or batch updates using search operation and data update may be performed. The desired functionality of the RecordSet object is defined while its instance is being created, and the obtained RecordSet object may function in very different ways depending on the selected properties[10].

ADO technology provides programmers, who create COM objects, with efficient and extended interface for work with data. Since ADO objects may be called using any programming language e.g. Microsoft Visual Basic, Microsoft Visual C++, as well as many scripting interfaces, this technology was widely used as an interface that provided access to many different data warehouses until its successor, ADO.NET, was developed.

[10] Doug Rothaus, Mike Pizzo, "ADO.NET dla programistów ADO". The Microsoft company website: http://www.microsoft.com

Together with .NET Framework platform, Microsoft introduced ADO.NET – an extension of data access technology, provided by ActiveX Data Objects (ADO). ADO.NET is a new version of library, extending possibilities ADO provides. It provides better cooperation with other platforms and scalable access to data. Due to the fact that ADO.NET technology is a set of .NET environment classes, it is not directly based on OLE DB. However, it is compatible and can communicate with it.

All classes and interfaces connected with ADO.NET belong to the name space System.Data. Mechanisms responsible for data gathering are contained in the following classes DataSet, DataTable, DataRow, DataColumn. Data is accessed using a set of interfaces called Data Provider. Data Provider acts as a go-between in the exchange of data between the DataSet object and databases. Fig 4.3 shows a hierarchy of name space and ADO.NET objects.

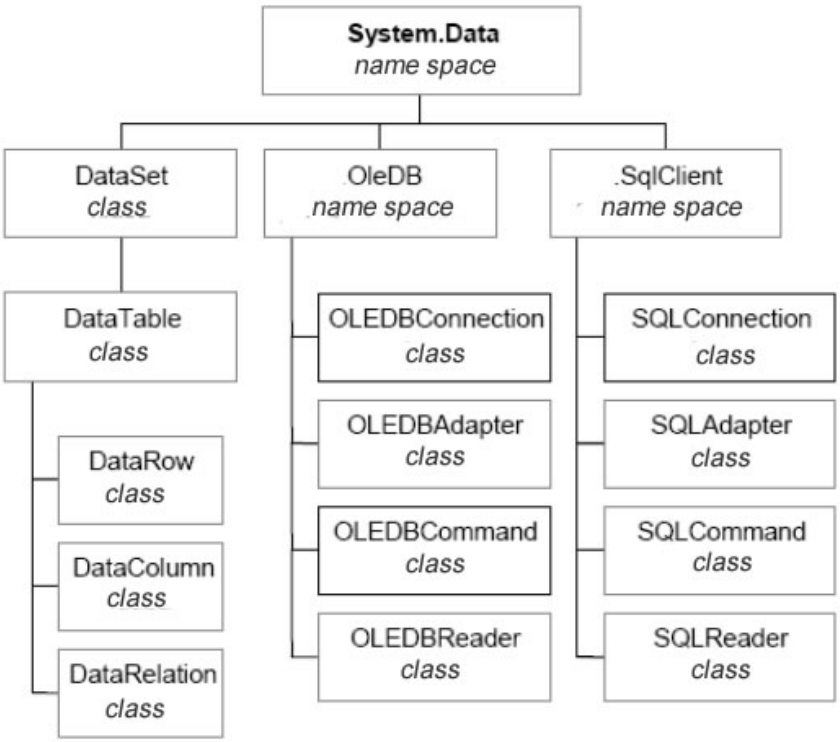


**Figure 4.3.** A hierarchy of name space and ADO.NET objects

Data providers for OLEDB or ODBC are universal providers – they may provide access to any database, which can be communicated with using OLEDB or ODBC. However, dedicated data providers such as providers for SQL Server or Oracle are more efficient. The Borland Company created additional drivers, Borland Data Provider (BDP), which allow to establish connection with other data sources i.a. with InterBase and DB2 databases.

Each data provider consists of the following objects:

- Connection object (SqlConnection, OleDbConnection, OdbcConnection, OracleConnection) – represents a connection with a particular database. It contains all the information that are necessary for connecting with a database, authorization as a defined user and carrying out further communication.
- Command object (SqlCommand, OleDbCommand, OdbcCommand, OracleCommand) – is used to perform commands concerning data gathered in the database. This command may be defined as an SQL query to be executed or as a procedure name stored in a database. The Parameters property of the Command object contains a collection of parameters transferred to the command. Communication with a database takes place using Connection object,
- DataReader object (SqlDataReader, OleDbDataReader, OdbcDataReader, OracleDataReader) – is used to read the data returned as a result of the execution of the query saved in the Command object. Data is sent in portions, one line at a time,
- DataAdapter object (SqlDataAdapter, OleDbDataAdapter, OdbcDataAdapter, OracleDataAdapter) – acts as a go-between in the communication between the DataSet object and a database.

### 4.3 DBI technology

Another technology that belongs to multi-platform and portable data access mechanisms family is DBI (DataBase Interface). It is a part of a package of a very popular scripting language called Pearl. Pearl has a wide range of access technologies, which is combined with portability of programs written in this language, and DBI layer allows for easy and fast data access and modification. A strength of this technology is simultaneous and transparent access to many different data sources such as MySQL, SQL Server, Oracle, Informix, Sybase, without the necessity of knowing each of them.

The process of development resulted in creating technologies, which need a virtual machine or such environments as .NET Framework or its independent, multi-platform counterpart – Mono, in order to work. The development also brought technologies dedicated to scripting languages such as Pearl, which is available on each hardware and system platform. Thanks to such an approach, implementation of these technologies on various systems is much easier, and their portability increases greatly. Figure 4.4. shows the development of database access technologies, divided into specializations and types.

## 5 Tests of selected database access technologies

For the purpose of performance and efficiency tests the following database was created with the following structure:

- Departments (id, department_name, description) - Dzialy (id, nazwa_dzialu, opis)

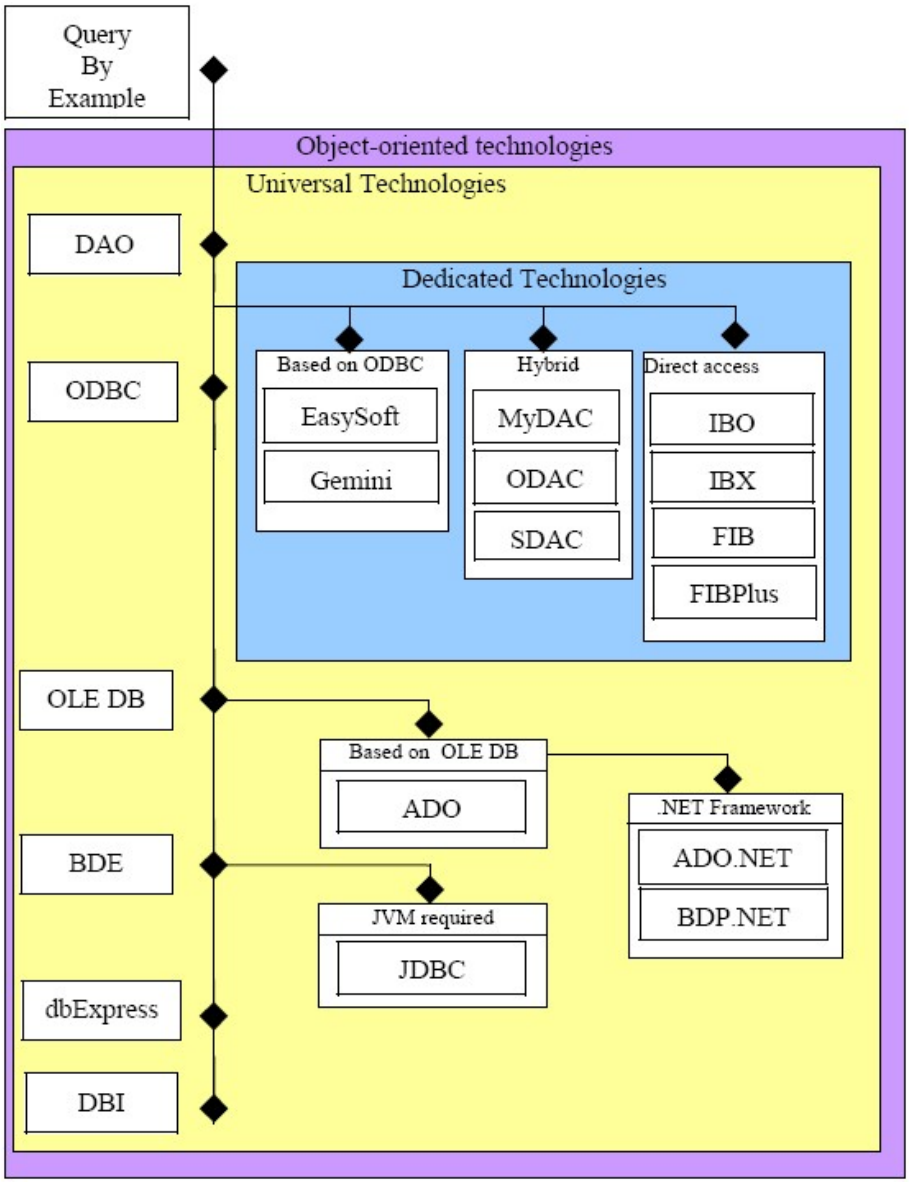


**Figure 5.1.** Developing database access technology

- Products (id, department_id, weight, price, pieces_available) - Towary (id, id_dzial, nazwa, masa, cena, ilosc_dostepnych)

- Clients (id, name, surname, street, code, city) – Klienci (id, imię, nazwisko, ulica, kod, miasto)
- Orders (id, client_id, product_id, order_date, completion_date, comment) - Zamowienia (id, id_klient, id_towar, data_zamowienia, data_Realizacji, komentarz).

The main table in the database is Orders table that stores information about client who orders a given product, about a product, order submission date, order completion date and about optional comments concerning this order. Performance test of each access technology was conducted using this table.

Technologies performance tests were conducted using the following operating systems: Microsoft Windows XP Professional SP2, Microsoft Windows 2003 Server Enterprise Edition SP1 and Linux Fedora Core 7, installed as virtual machines in the VMware Server, working under control of Windows XP Professional SP2 system. Each system, which was used to test the technologies was run on the same machine with identical configuration. However, neither the parent system nor the virtual systems are real-time systems. Therefore, it is necessary to remember that the obtained results are influenced by such factors as processor load and clock speed, operating memory available, the number of running processes, etc. For the testing purposes the number of processes was minimized, leaving only those that were essential. Moreover, the tests results might be different if the computer's hardware configuration was changed.

Table 5.1. shows the hardware and system platform used to conduct tests. The following database servers Oracle XE, MySql, PostgreSQL and Microsoft SQL Server were used to conduct the tests (the last one was only used with Microsoft Windows systems).

**Table 5.1.** Hardware and software parameters used for the tests

| **Windows XP Professional SP2** | **Windows 2003 Server Enterprise Edition SP1** | **Linux Fedora Core 7** |
|---|---|---|
| Procesor: Intel Pentium Mobile 1.73 GHz<br>Pamięć RAM: 512 MB<br>Dysk twardy: Seagate 80GB, 7200 RPM, 4MB cache | Procesor: Intel Pentium Mobile 1.73 GHz<br>Pamięć RAM: 512 MB<br>Dysk twardy: Seagate 80GB, 7200 RPM, 4MB cache | Procesor: Intel Pentium Mobile 1.73 GHz<br>Pamięć RAM: 512 MB<br>Dysk twardy: Seagate 80GB, 7200 RPM, 4MB cache |

In order to conduct the technology performance tests, programs were developed for establishing connection with a particular database, executing appropriate command (SELECT, INSERT, UPDATE, DELETE) and measuring the time, in which it was completed. After the measurement is finished, the program shows the time, which was necessary to execute the series of queries. The performance of four dedicated DBI drivers providing access to the most popular databases was measured during the tests. The performance of a universal ODBC driver was also tested:

- DBD-mysql – DBI driver for MySql
- DBD-Oracle –DBI driver for Oracle
- DBD-Pg –DBI driver for PostgreSQL

- Win32-SqlServer DBI driver for SQL Server (onlyWindows)
- DBD-ODBC –DBI driver for ODBC

JDBC technology efficiency was measured using JDBC-ODBC driver. It is a bridge where the queries formulated in Java are translated by JDBC into ODBC driver language. ADO.NET technology was tested on its parent platform .NET in the Windows systems as well as the Mono[11] environment using the Linux system. The performance of the following drivers was tested:

- MySql Connector/NET is fully managed provider and does not need a client library.
- Npgsql .NET Data Provider for PostgreSQL is a fully managed provider and does not need a client library.
- ADO.NET Provider for Microsoft SQL Server and Sybase uses TDS protocol, version 7.0; it is based on FreeTDS and jTDS designs, which, in turn, are based on JDBC.
- ADO.NET Data Provider for Oracle which is based on Oracle Call-level interface (OCI), library of functions written in C language used to perform operations on Oracle databases.

Due to the fact that there is no version of .NET platform dedicated to the Linux system, Mono environment was used. Mono was created by independent programmers and is an open source project, which means that there is access to the source code. The main sponsor of the initiative is the Novell company. At present, Mono contains the implementation of most methods that .NET platform, version 2.0, has. Methods that are part of .NET platform, version 3.0 and 3.5 are being implemented and compatibility with these versions is being provided.

Programs based on the skeleton presented below were developed for each of the four DML (Data Modification Language). As a result, four programs for testing the efficiency of access technologies were obtained for each database, installed on three operating systems. The exception is Microsoft SQL Server, which is not compatible with the Linux operating system.

During efficiency measurements of database access technologies each program worked on ten thousand records and was executed seven times. Two extreme values obtained after time limit were omitted, and the remaining five were used to calculate the average time of the execution of a given type of query. Table 5.2 shows queries, which were used during efficiency measurements of each driver.

**Table 5.2.** Tested queries

| Query content | Description |
|---|---|
| `SELECT nazwa, cena, imie, nazwisko, nazwa_dzialu, komentarz FROM zamowienia JOIN towary ON zamowienia.id_towar=towary.id JOIN klienci ON zamowienia.id_klient=klienci.id` | Data is fetched from table Orders and from related tables, for which the department name equals “IT”, and order as well as completion dates have appropriate values. |

[11] Multi-platform version of .NET environment.

| | |
|---|---|
| `JOIN dzialy ON dzialy.id=towary.id_dzial`<br>`WHERE dzialy.nazwa_dzialu = 'IT'`<br>`AND (zamowienia.data_zamowienia BETWEEN`<br>`'2007-10-01' AND '2008-12-31')`<br>`AND zamowienia.data_realizacji='2008-03-31'` | |
| **`INSERT`** `INTO Zamowienia (id_klient, id_towar,`<br>`data_zamowienia, data_realizacji, komentarz)`<br>`VALUES (1, 1, '2008-03-30', '2008-03-31',`<br>`'to jest komentarz')` | A new row, which contains sample data is inserted into table Orders. |
| **`UPDATE`** `Zamowienia`<br>`SET komentarz = 'to jest nowy komentarz'`<br>`WHERE data_zamowienia < '2008-03-31'` | 'Comment' field is set in Orders table for the records, which have appropriate order date. |
| **`DELETE`** `FROM Zamowienia`<br>`WHERE data_zamowienia < '2008-12-31'` | All the records which meet the date condition are deleted from table Orders. |

# 6 Test results of selected technologies

Database efficiency depends on multiple factors, from hardware configurations, database server settings, selection of an appropriate access technology to optimization of the database structure and queries. Therefore, a programmer who writes software using databases must use the skills administrators have and take care of appropriate indexing of significant columns as well as optimum state space of tables, code of stored procedures or other queries. Only then the programmer shall be able to appreciate the possibilities offered by efficient data access technologies and drivers. Table 6.1. shows efficiency results of the tested technologies.

**Table 6.1.** Access technologies efficiency within a given database and operating system

| | | | Average time of query execution | | | |
|---|---|---|---|---|---|---|
| Operating system | Database | Language and access technology | SELECT | INSERT | UPDATE | DELETE |
| Windows XP Professional SP2 | SQL Server | Perl – Win32--SqlServer | 0,9343 | 14,7070 | **0,1591** | **0,1791** |
| | | Perl – ODBC | 0,1754 | 12,1718 | 0,1748 | 0,2192 |
| | | C# - ADO.NET Provider | 0,3212 | **9,6558** | 0,2026 | 0,2276 |
| | | Java - JDBC | **0,1566** | 11,8938 | 0,2284 | 0,2248 |
| | MySql | Perl – DBD--mysql | 0,1978 | **2,3406** | **0,3145** | **0,2734** |
| | | Perl – ODBC | **0,1783** | 3,0906 | 0,4164 | 0,3497 |
| | | C# – MySql Connector/NET | 0,3902 | 3,2170 | 0,5464 | 0,4934 |
| | | Java - JDBC | 0,1938 | 2,9938 | 0,3750 | 0,3188 |
| | PostgreSQL | Perl – DBD-Pg | **0,2219** | **16,1353** | 0,3482 | 0,1696 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | Perl – ODBC | 0,2613 | 17,7656 | **0,3157** | 0,1679 |
| | | C# – Npgsql Data Provider | 0,7438 | 17,2594 | 0,7840 | 0,5814 |
| | | Java - JDBC | 0,2968 | 17,0408 | 0,4000 | **0,1646** |
| | Oracle | Perl – DBD--Oracle | **0,1502** | 23,6250 | **0,5969** | **0,2811** |
| | | Perl – ODBC | 0,2213 | 26,7937 | 1,6449 | 2,2969 |
| | | C# - ADO.NET Provider Oracle | 1,7440 | **17,6312** | 1,5158 | 2,3816 |
| | | Java - JDBC | 0,2406 | 23,0280 | 0,6782 | 1,7423 |
| Windows 2003 Server SP1 | SQL Server | Perl – Win32-SqlServer | 0,2690 | 15,4281 | **0,2031** | **0,2164** |
| | | Perl – ODBC | **0,1671** | 11,5062 | 0,2127 | 0,2385 |
| | | C# - ADO.NET Provider | 0,4964 | **10,9808** | 0,3120 | 0,4902 |
| | | Java - JDBC | 0,2066 | 11,6562 | 0,2532 | 0,2500 |
| | MySql | Perl – DBD--mysql | 0,2409 | **2,1812** | 0,4260 | **0,2904** |
| | | Perl – ODBC | **0,1465** | 2,7093 | **0,3988** | 0,3031 |
| | | C# – MySql Connector/NET | 0,7844 | 3,5248 | 0,9438 | 0,8062 |
| | | Java - JDBC | 0,1720 | 2,7996 | 0,4218 | 0,3032 |
| | PostgreSQL | Perl – DBD-Pg | **0,2522** | **15,1816** | 0,3313 | **0,1320** |
| | | Perl – ODBC | 0,2894 | 15,2531 | 0,3511 | 0,1879 |
| | | C# – Npgsql Data Provider | 0,8688 | 15,6112 | 0,8376 | 0,7032 |
| | | Java - JDBC | 0,2624 | 15,8592 | **0,3156** | 0,2000 |
| | Oracle | Perl – DBD--Oracle | **0,1518** | 22,4062 | 1,4250 | 2,5468 |
| | | Perl – ODBC | 0,1990 | 26,5562 | 0,8853 | 2,0000 |
| | | C# - ADO.NET Provider Oracle | 1,5406 | **16,9052** | 1,0030 | 2,2092 |
| | | Java - JDBC | 0,1782 | 22,8906 | **0,8464** | **1,9562** |
| Linux Fedora Core 7 | MySql | Perl – DBD--mysql | **0,1357** | **2,1221** | **0,3704** | **0,2595** |
| | | Perl – ODBC | 0,1626 | 2,4762 | 0,3753 | 0,2661 |
| | | C# – MySql Connector/NE[12] | 0,6062 | 3,8421 | 0,9175 | 1,0124 |
| | | Java - JDBC | 0,1812 | 2,7510 | 0,4030 | 0,2958 |
| | PostgreSQL | Perl – DBD-Pg | **0,1922** | 13,2816 | **0,3313** | **0,1200** |
| | | Perl – ODBC | 0,3621 | 15,1100 | 0,4223 | 0,2553 |
| | | C# - Npgsql Data Provider | 1,0186 | **12,7354** | 1,1156 | 0,8390 |
| | | Java - JDBC | 0,5150 | 15,2330 | 0,6268 | 0,4134 |
| | Oracle | Perl – DBD--Oracle | **0,1073** | 26,0134 | 1,2386 | **2,3901** |
| | | Perl – ODBC | 0,1393 | 31,5081 | 1,1346 | 2,4911 |
| | | C# - ADO.NET Provider Oracle | 0,5092 | **24,4706** | 0,9214 | 2,6874 |
| | | Java - JDBC | 0,1430 | 25,0492 | **0,4262** | 2,4114 |

[12] MySql Connector/NET during tests performed Rusing Linux was unstable, which caused frequent freezing or hanging of the testing program.

The best results were obtained during tests of access technologies using MySQL database. This base is designed for use in smaller companies where small amount of data is stored. Moreover, it is dedicated for WWW websites, which, in a way, results in its minimalist character (e.g. in comparison with such databases as Oracle or Sybase) and optimum efficiency.

To sum up, the obtained results allow to state that the databases themselves have a significant influence on access technologies efficiency. Drivers written in different technologies may be fast and efficient. However, it will not improve the situation if the query execution time is prolonged because of the delay caused by the database server. Thus, on the basis of the above results the following interdependence may be observed: less complex database server – faster operation of data access drivers and shorter times of testing queries execution; more complex database server – slower operation of data access drivers and longer times of testing queries execution.

## References

1 Wiesław Dudek, *Bazy danych SQL. Teoria i praktyka*, Helion, Gliwice 2006.
2 Paul Beynon-Davies, *Systemy baz danych*, Wydawnictwa Naukowo-Techniczne, Warszawa 2000.
3 William. J. Gilmore, *PHP 4.0. Poradnik dla programistów*, Mikom, Warszawa 2002.
4 Richard Stones, Neil Matthew, *Bazy danych i MySQL. Od podstaw*, Helion, Gliwice 2003.