# Is Interoperability Achievable With ODBC?

Christoph Steindl
Institute of Computer Science (System Software)
Johannes Kepler University Linz, Austria
steindl@ssw.uni-linz.ac.at

June 12, 1996

**Abstract**

Open Database Connectivity (ODBC) is a call level interface to databases. In contrast to Embedded SQL and PL/SQL, programs using ODBC are meant to be database independent. This is achieved by an additional layer between the application and the database. In this paper we regard interoperability not as the question if it is at all possible to work with different DBMS's [1] but how easy and convenient it is. We will show that there are some deficiencies in the design of ODBC which reduce the degree of interoperability but also that these can be remedied with moderate effort.

## 1 Accessing Databases

Many applications need to access large amounts of data. Therefore many of them use databases. For a query language SQL is a widely accepted and used standard. But depending on the degree of integration of SQL into the application there are at least three different ways to bind SQL statements to the application code [Ven95]:

- Embedded SQL, embedding SQL statements directly into the application code (raising the need for an additional pre-compilation step),
- Module Language, enhancing SQL with control flow statements and building executable units, and
- Direct Invocation, accessing database functionality via calls to a library.

The advantage of the latter approach is that only the functionality of the library and the way how an application (client) calls the library functions have to be defined. The calls are then forwarded to the database (server) which can even run on a different machine connected with the client via a network. Furthermore it is not necessary that the application knows the database in advance (as it is with Embedded SQL). The result is a gain in database independence and interoperability.

[1] DBMS - database management system

## 2 ODBC

ODBC [ODBC94] is Microsoft's implementation of the SQL Connectivity specification [SQL/CLI], which they jointly developed with Lotus, Sybase, and DEC. The ODBC library contains function calls that allow an application to connect to a DBMS, execute SQL statements, and retrieve results. Furthermore it defines standard error codes and standard data types. ODBC is an alternative to Embedded SQL [Emb89]. While Embedded SQL is efficient and portable across different hardware and operating systems, the source code must be recompiled for each new environment. Additionally the program is tightly coupled with the database it was compiled for. There is no need for implementation specific transformations on the source code of ODBC programs (such as pre-processing or pre-compiling).

ODBC is more flexible than Embedded SQL in the sense that the same programs (object code) can be used to access different DBMS's and can ignore underlying data communications protocols between the program and the DBMS.

In the following figure we show the overall structure of an ODBC program.

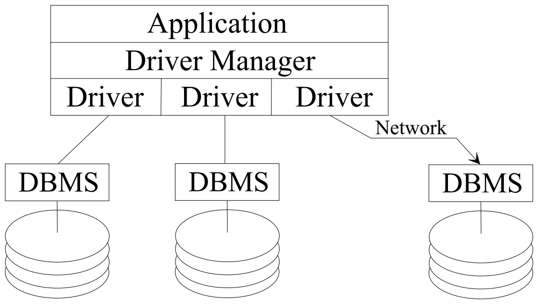


Figure 1: Architecture of an ODBC Program

A program calls functions from the driver manager which passes them on to the respective drivers of the DBMS's. Thus a program can be connected to multiple DBMS's at the same time which can even reside on different machines in a network.

## 3 Execution of Statements

ODBC provides functions to connect to a DBMS (e.g. SQLConnect). After the connection has been established, SQL statements can be executed over that connection. This is done as follows: The statement (supplied as an SQL string) is prepared with SQLPrepare, parameters are bound to the statement with SQLBindParameter, and finally the statement is executed with SQLExecute. After buffers for the values to be returned have been bound with SQLBindCol, the results of an SQL statement can be retrieved with SQLFetch. A statement can be executed repeatedly with different parameter values.

The following example is over-simplified but shows the principle. A question mark within an SQL string ("?") is a parameter marker and stands for the value of the variable that is bound to the parameter.

```
VAR
  firstName, lastName: ARRAY 32 OF CHAR;
```

```
BEGIN
  ODBC.SQLPrepare(stat, "INSERT INTO Friends VALUES (?, ?)");
  ODBC.SQLBindParameter(stat, 1, VARCHAR, firstName);
  ODBC.SQLBindParameter(stat, 2, VARCHAR, lastName);
  firstName := "Christoph"; lastName := "Steindl";
  ODBC.SQLExecute(stat)
END
```

The more general case is depicted in figure 2.

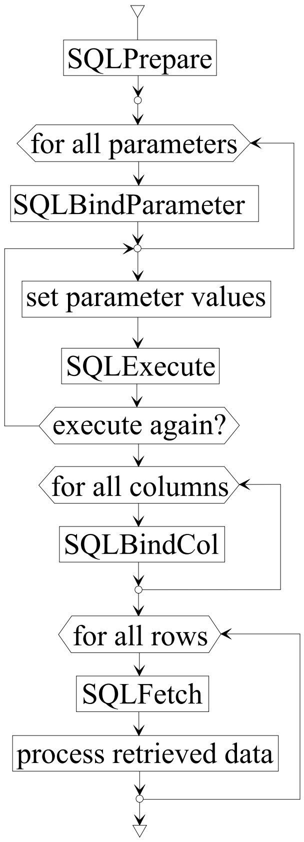


Figure 2: Execution of Statements

## 4 Data types

ODBC defines some data types (which we will call ODBC types) according to [SQL/CLI]. There are predefined data types for character data, for numeric data and for time and date (see table 1). These data types are used both for data definition (CREATE TABLE statements) and for data manipulation (INSERT, UPDATE, SELECT statements). Unfortunately, not all databases support all the ODBC types, and if they support them, they often use different names (we will call them DB types). Additionally the programming language in use typically only supports a different set of data types (which we will call application types).

Table 1 gives a flavor of the variety of data type names (blank fields indicate not-supported data types).

| Data Type | Oracle | MS Access | Programming Language |
|---|---|---|---|
| BIT | | BIT | BOOLEAN |
| TINYINT | | UNSIGNEDBYTE | SHORTINT |
| BIGINT | | | |
| LONG VAR BINARY | LONG RAW | LONGBINARY | binary file |
| VAR BINARY | RAW | | ARRAY x OF BYTE |
| BINARY | | | ARRAY x OF BYTE |
| LONG VAR CHAR | LONG | MEMO | text file |
| CHAR | CHAR | TEXT | ARRAY x OF CHAR |
| NUMERIC | NUMBER | CURRENCY | |
| DECIMAL | | | |
| INTEGER | | LONG | LONGINT |
| SMALLINT | | SHORT | INTEGER |
| FLOAT | FLOAT | | REAL |
| REAL | | | |
| DOUBLE PRECISION | | DOUBLEFLOAT | LONGREAL |
| DATE | | | record |
| TIME | | | record |
| TIMESTAMP | DATE | DATETIME | record |
| VARCHAR | VARCHAR2 | | ARRAY x OF CHAR |

Table 1: Supported Data Types

A further obstacle is that the ODBC driver manager does not do any mapping of ODBC types to DB types. Thus, the names of the data types given in a CREATE TABLE statement must be exactly in the form as they are supported by the database. Not only the names of the data types differ, but also their parameters when used for data definition. For example, ODBC defines a string of 32 characters as CHARACTER(32) while MS Access calls it just TEXT without parameters. This makes an application program effectively dependent on the underlying DBMS. Using a different DBMS requires adaptation of the SQL statements and a recompilation of the application.

## 5 Mapping of Data Types for Data Definition

An interoperable program should only use ODBC types. The mapping of these types to specific DB types (e.g. to MS Access types) should ideally be transparent and hidden in the ODBC driver manager. Since this is not the case we added the mapping as a small set of functions.

In the following we show how this mapping is done in an ODBC interface [Ste96] implemented for the Oberon System [WiGu89] under MS Windows.

The main part of the mapping is to determine the DB type that best corresponds to the

ODBC type. Additionally the so-called *creation parameters* (How long is a string of characters? What is the precision of an integer number?) have to be considered as they also vary.

Figure 3 shows, how ODBC types can be emulated if they are not supported by the database (an arrow means that the data type pointed to can be used instead of the other data type). Without loss of information it is possible to use Integer instead of SmallInt if SmallInt is not available (arithmetic promotion). It is also possible to convert a number into a string of characters if the data type were not available. Since SmallInt and Integer can be emulated with Numeric and vice versa (if the specified precision does not exceed the range of SmallInt or Integer), the resulting structure contains cycles.

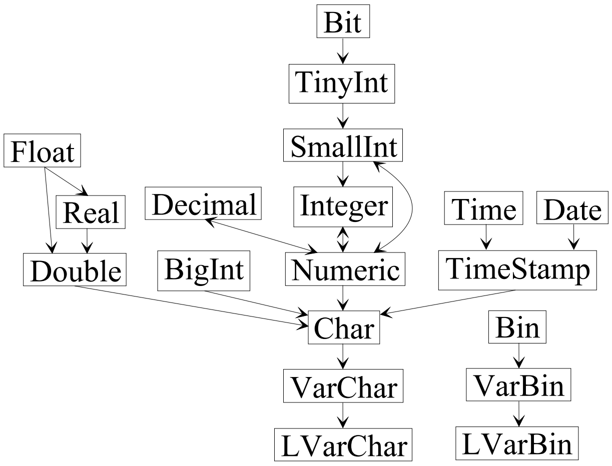


Figure 3: Mapping of Data Types

The procedure Map maps the ODBC type odbcType to the DB type dbType that is supported by the connection c. Map returns TRUE if the mapping can be done, otherwise FALSE which means that the database cannot represent the ODBC type in any way. Care is taken to avoid endless loops (because of the cycles mentioned) during the mapping.

```
PROCEDURE Map (c: Connection; odbcType: Type; VAR mapType: Type): BOOLEAN;
```

Now we can assemble the CREATE TABLE statement where the columns have DB types only. Therefore we use the ODBC function SQLGetTypeInfo to get the exact name of the DB types.

## 6 Mapping of Data Types for Data Manipulation

There are two more cases where the programmer specifies data types: when parameters are bound to the statement and when the results of the statement are retrieved. In each case one must specify the application type and the DB type used in the database. If these data types differ, the ODBC driver converts the data "on the fly".

The following figure shows the problems when the application type, the ODBC type and the DB type are not the same. In the following example an application program inserts an integer value into a table. For example, ORACLE only provides NUMBER and FLOAT as numeric types, whereas the application types might be LONGINT and LONGREAL. The application type LONGINT is mapped to the ODBC type INTEGER and the DB type NUMBER. The DB type NUMBER in return is mapped to the ODBC type CHAR and application type ARRAY OF CHAR because it is generally not possible, to represent the DB type NUMBER (fixed numeric precision) within the numeric application types.

The main point is that the mapping of types application types to DB types and back cannot be done without loss of information. The programmer must therefore be aware of the fact that the data types of the inserted data do not necessarily equal the data types of the retrieved data.

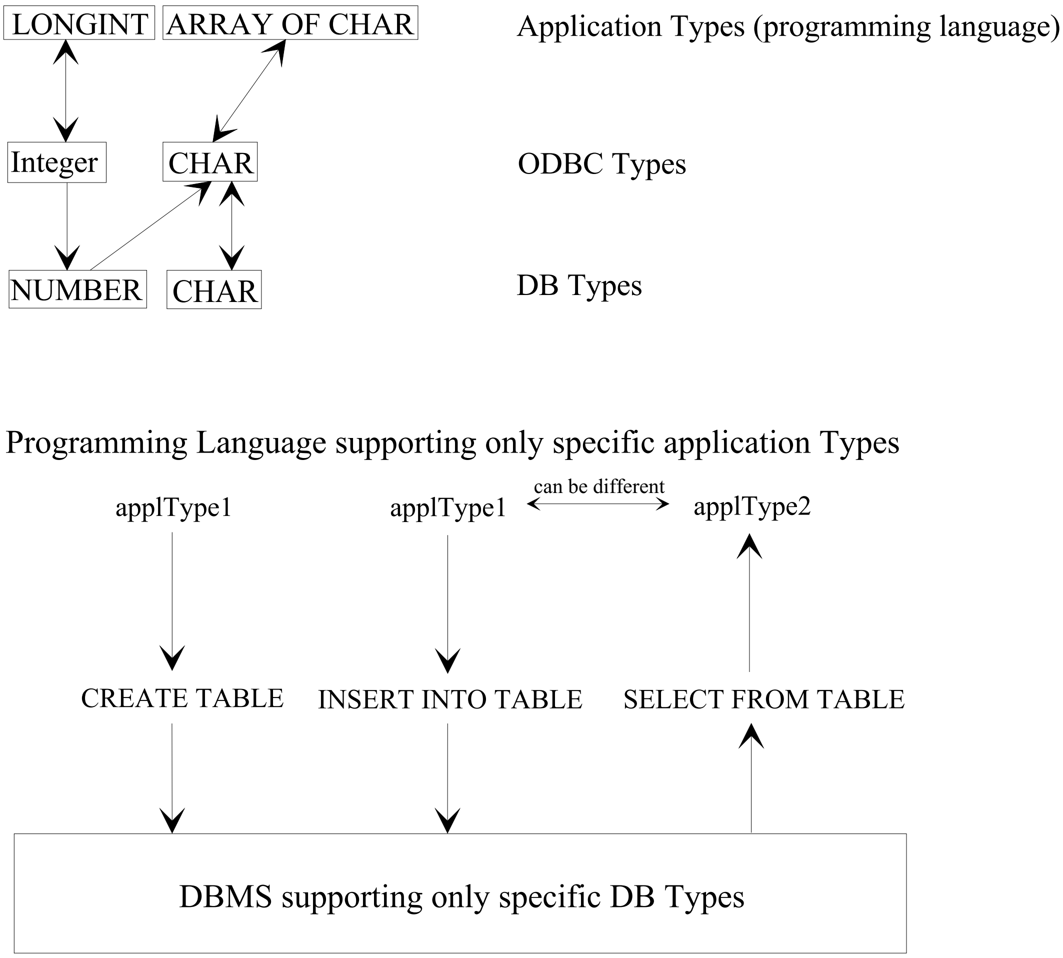


Figure 4: Mismatch of Data Types

## 7 Summary

We showed that ODBC does not automatically yield interoperable programs, rather must the programmer be aware of some obstacles in order to be able to circumvent them. This can be done with library support by a few auxiliary functions. We conclude that for the programmer's convenience this should already be done within the driver manager of ODBC.

## References


[Emb89] *Database Language - Embedded SQL* (X3.168-1989). American National Standards Institute, Technical Committee X3H2

[SQL/CLI] IS 9075-3 *International Standard for Database Language SQL - Part 3: Call Level Interface*, document ISO/IEC 9075-3:1995

[ODBC94] *Microsoft Open Database Connectivity Software Development Kit Version 2.0.* Microsoft Press, 1994, ISBN 1-55615-658-8

[Ste96] C. Steindl. *Entwurf und Implementierung einer Stücklistenverwaltung mittels einer Client/Server-Datenbank.* Diploma thesis, Johannes Kepler University Linz, 1996

[Ven95] M. Venkatrao, M. Pizzo. *SQL/CLI - A New Binding Style For SQL.* SIGMOD Record, Vol. 24, No. 4, December 1995

[WiGu89] N. Wirth, J. Gutknecht. *The Oberon System*, Software - Practice and Experiences, 1989/9