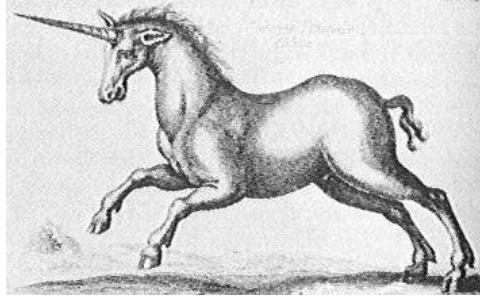


**An ODBC Interface  
for the Unicon language**

Federico Balbi and Clinton Jeffery

August 8, 1999



**Abstract**

The implementation of an ODBC interface for the Unicon language allows programmers to interface their applications to local and remote database management systems with a high level of interoperability and SQL language support.

Division of Computer Science  
The University of Texas at San Antonio  
San Antonio, TX 78255

## Introduction

The Unicon ODBC interface consists of a data type and a set of new functions to let Unicon access a DBMS. The standard language to retrieve and manipulate data in a DBMS is SQL, but in Unicon the ODBC function set let programmer interact with a database with minimum knowledge of the structured query language.

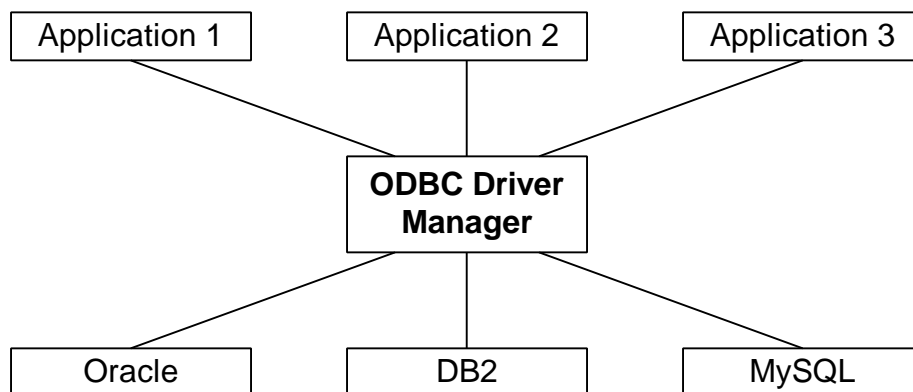
## Overview

ODBC is a programming interface (API) to access local and remote database management systems. ODBC is a de facto industry standard and works on many different operating systems and programming languages. It shields programmers from the complexity of different databases and communications the software used to access the data. ODBC defines an object called “data source” that is referenced by just a name and it maps the exact location of data (network software, server name, database name and user information if needed).

## ODBC Architecture

ODBC allows multiple applications to access to multiple data sources using an architecture that consists of five layers:

- A Driver Manager to add, configure and remove different DBMS drivers.
- A set of drivers that implement the ODBC API for a particular DBMS.
- Networking software to allow network access to the database.
- The DBMS.



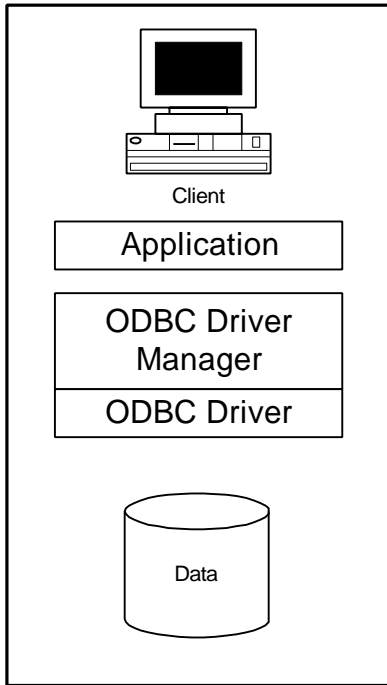
Applications call ODBC functions. The Driver Manager decides which driver is needed and loads it into memory. After this the Driver Manager routes ODBC calls to the driver. The minimum capability required from a driver is to be able to connect to a server, send SQL statements and retrieve the results. What is important is that the Driver Manager hides all the details related to the server so the application does not need to know if the data is on a local file, a network file server or a remote host.

### **Client/Server Model**

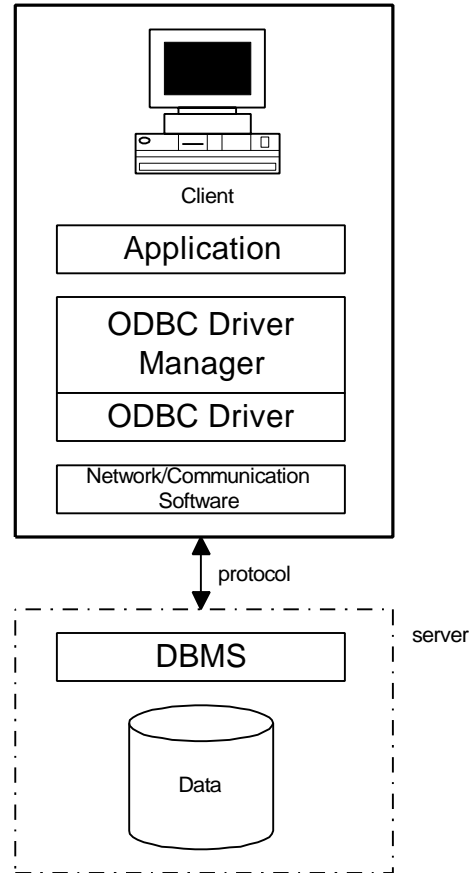
ODBC was designed to work with the client/server model in order to satisfy the following requirements:

- A standard application programming interface.
- Access to the particular features of any DBMS.
- Performance comparable to DBMS native API.

## File Oriented Model



## Client/Server Model



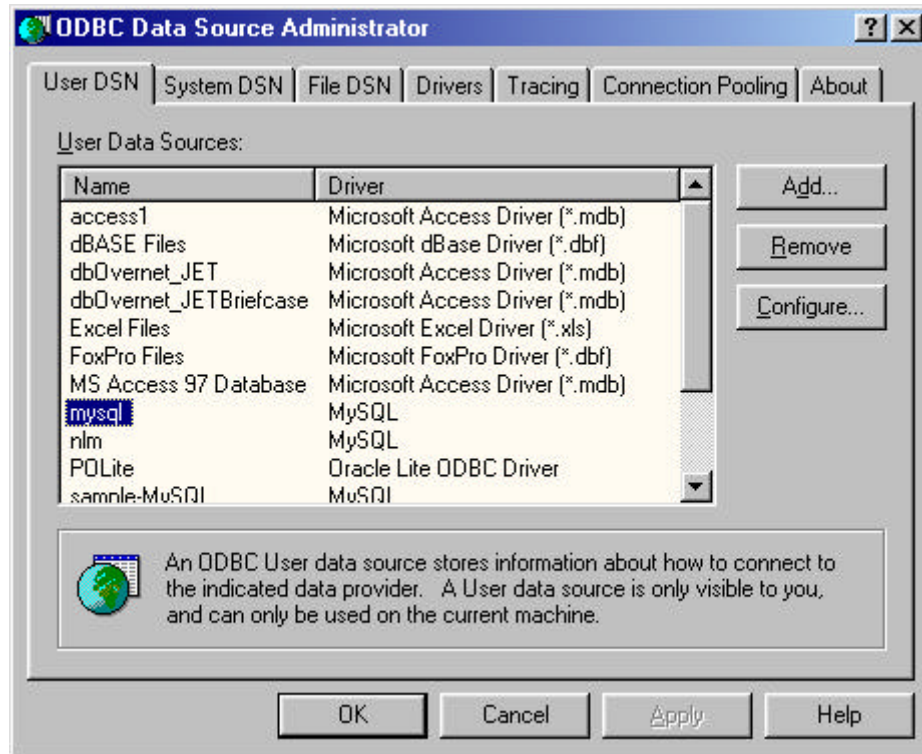
## ODBC Installation

After deciding which DBMS an application is going to use it is necessary to install its related ODBC driver to let the application establish a connection with the DBMS.

For our ODBC tests we decided to use MySQL. MySQL is a free SQL server downloadable at [www.mysql.org](http://www.mysql.org) and available for most popular platforms. We tested both Linux and Sun/Solaris version. MySQL comes with MyODBC, the ODBC driver for the server, and it can be downloaded at the same web site. There are driver versions both for Unix and Windows.

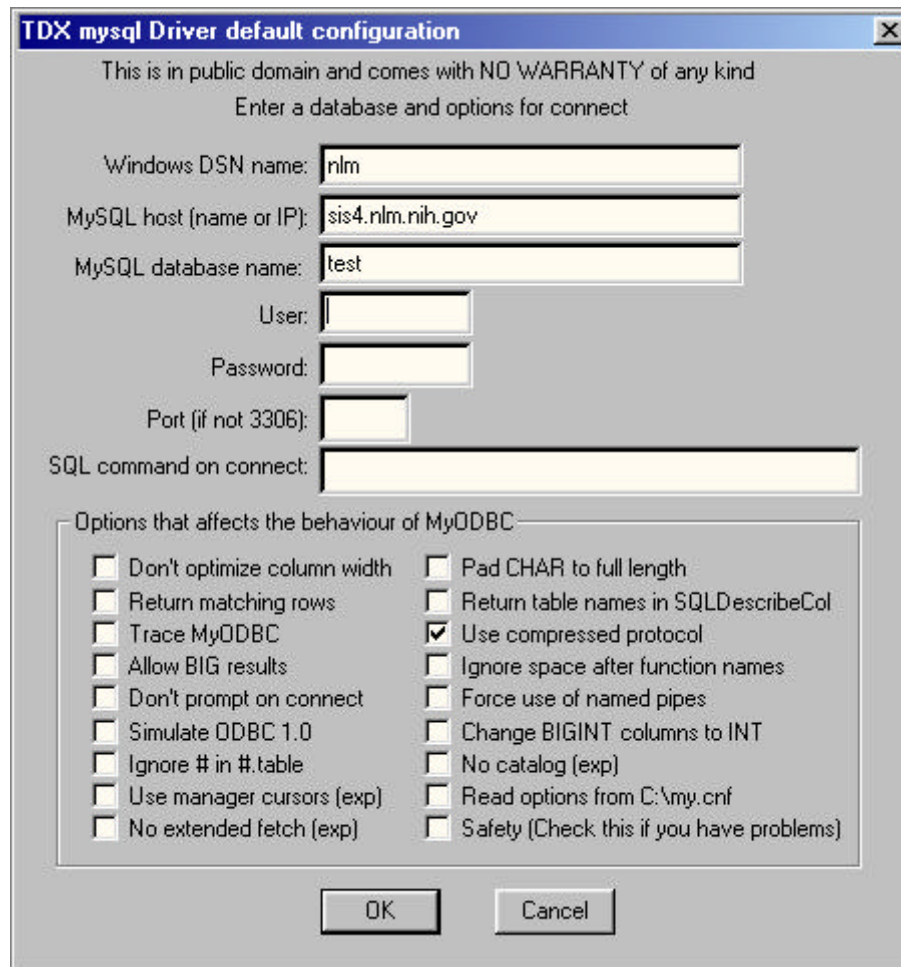


*The Control Panel windows*



*The Driver Manager*

The ODBC Driver Manager lists the drivers installed for the different DBMSes. We can add new drivers, remove or configure existing ones just with the simple click of a button. Anyway, when MyODBC is downloaded from MySQL web page it comes with a setup program that automatically add and setup the driver into the Driver Manager's driver list. After installing MyODBC we can open the Driver Manager and take a look at the different configuration parameters.



### *MyODBC configuration*

MyODBC configuration panel has the following fields:

- **Windows DSN name:** DSN stands for Data Source Name and it contains the alias that we will use in our applications to reference a particular database.
- **MySQL host:** is contains the DSN name or the IP address of the server we want to connect to.
- **MySQL database name:** name of the database the alias is referred to.
- **User:** database user name. For better security this field can be left empty and specified within an application.
- **Password:** database user password. For security this field can be also left empty and specified during application runtime.

- **Port:** this field is the port where MySQL server listens for connection. If left empty the default value is 3306.
- **SQL command on connect:** this field can contain an SQL command that is executed when the application first connects to the server.

All the other check boxes are driver options. Of particular note are the following options:

- **Trace MyODBC:** traces the driver activity on a log file C:\MYODBC.TXT. This is useful to find error within an application.
- **Use compressed protocol:** this option enables data compression in order to speed up data transfer on slow connections. This is recommended on fast computers.
- **Read options from C:\my.cnf:** this option lets the driver read a particular configuration from a file. This helps administrators to easily configure several client machines by including c:\my.cnf in the application distribution.

### Unicon ODBC Interface

Unicon ODBC interface (*IODBC*) is a set of built-in functions that allow an easy way to write database applications without knowledge of SQL or the particular DBMS used.

The function set can be divided in five main groups:

- **Connection:** handle Unicon connection with a DBMS.
- **Catalog or information:** used to retrieve information about drivers, databases and tables (for example driver capabilities, database organization, tables description).
- **Data retrieval:** to load data from a DBMS to Unicon data structures.
- **Data manipulation:** to modify information stored in a DBMS, like insert , delete and update table rows.
- **General/SQL:** to interact directly with the DBMS using SQL commands. This can be used to send arbitrary SQL commands including extensions particular to a specific DBMS implementation.

## Unicon ODBC Functions

### Connection

- *dbopen*: opens a connection to the database server and the specified table.
- *dbclose*: closes a table and the related database connection.

### Catalog

- *dbcolumns*: gets information about a specific table columns.
- *dbdriver*: gets information about a specific ODBC driver in use.
- *dbkeys*: gets information about a particular table primary keys.
- *dbproduct*: gets information about the DBMS product the in use.
- *dbtables*: get information about a tables that belong to a specified database.

### Data Retrieval

- *dbfetch*: fetches a row from a rowset.
- *dbselect*: selects a rowset from a database.

### Data Manipulation

- *dbdelete*: deletes one or more rows of a database.
- *dbinsert*: insert a row in a database.
- *dbupdate*: update a database row.

### General/SQL

- *dbsql*: sends an SQL command string to the server for execution.

## An Example Phonebook Application

A typical database application performs the following tasks:

- Connects to a database
- Processes database information
- Disconnects from the database

This section presents a simple Unicon phonebook application that takes advantage of the ODBC interface and work with MySQL server.

The example will show how to use the main IODBC functions in order to connect, read and write data on a DBMS.

Our application will have the following menu:

- Insert a phone number
- Delete a phone number
- Modify a phone number
- List phone number in database

Let's create a **phones** table on our server with the following columns:

Column Name	Type
Name (KEY)	VARCHAR(40)
Phone	VARCHAR(12)
Address	VARCHAR(60)

From our server machine we invoke mysql client program to talk to the SQL server:

```
[fbalbi@icon bin]$ ./mysql -ufbalbi -p mysql
Enter password:
Reading table information for completion of table and column
names
You can turn off this feature to get a quicker startup with
-A
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 96 to server version: 3.22.15-
gamma
```

```
Type 'help' for help.
```

Now Let's create the example table with the column **Name** as primary key:

```
mysql> create table phones (name varchar(40) primary key,
phone varchar(12), address varchar(60));
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> describe phones;
```

```
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name  | varchar(40)   |      | PRI |          |       |
| phone | varchar(12)   | YES  |     | NULL    |       |
| address | varchar(60)  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

```
3 rows in set (0.01 sec)
```

Now the table is properly created and empty, in fact the following select commands returns an empty set:

```
mysql> select * from phones;
```

```
Empty set (0.00 sec)
```

Here the full list of our phonebook application:

```
# global variables

global db
global user, password

record person(name, phone, address) # database row

procedure main() # main program
  write("*** IODBC phonebook ***\n\n")

  login() # get user name and password

  # connect to "mysql" data source and open table "phones"

  db:=dbopen("mysql", "phones", user, password)

  if &errornumber~=0 then { # error during login
    write(&errortext)
  }
  else {
    getdbinfo() # print database information

    repeat {
      menu() # print menu options
      option:=read()

      case option of {
        "i": insertphone()
        "d": deletephone()
        "u": updatephone()
        "l": listphones()
        "q": break
        default: write("*** wrong selection ***")
      }
    }
    dbclose(db) # close table and database connection
  }
  write("bye")
end
```

```

#
# user information
#
procedure login()
  writes("user: ")
  user:=read()
  writes("password: ")
  password:=read()
end

#
# get database name and version
#
procedure getdbinfo()
  info:=dbproduct(db)
  write("\nDBMS: ", info["name"])
  write("version: ", info["ver"])
end

#
# display menu options
#
procedure menu()
  write("\nI)nsert")
  write("D)elete")
  write("U)pdate")
  write("L)ist")
  write("Q)uit\n")
end

#
# insert a new record
#
procedure insertphone()
  writes("name: ")
  name:=read()
  writes("phone: ")
  ph:=read()
  writes("address: ")
  addr:=read()
  row:=person(name, ph, addr)
  dbinsert(db, row) # insert row into database
  if &errornumber~= 0 then
    write("*** couldn't insert person ***")
  end
end

```

```

#
# remove a record
#
procedure deletephone()
  writes("name to remove: ")
  name:=read()

  # delete row with specified name column
  dbdelete(db, "name=' " || name | | "'")
end

#
# update a record
#
procedure updatephone()
  writes("name to update: ")
  name:=read()

  # select all columns of rows with specified name column
  dbselect(db, "*", "name=' " || name | | "'")

  if (row:=dbfetch(db)) then { # data found
    writes("phone (",row["phone"],"): ")
    row["phone"]:=read()
    writes("address (",row["address"],"): ")
    row["address"]:=read()

    dbupdate(db, row) # update row on server
  }
  else write("\n\n*** person not found ***")
end

#
# list all people in the database
#
procedure listphones()

  dbselect(db, "*", "") # select all columns and all rows

  while(row:=dbfetch(db)) do { # while data found

```

```
# write row fields
every i:=(1 to *row) do writes("[",row[i],"]")

write()
}
end
```

## Unicon Runtime System

For the implementation of the ODBC interface it has been necessary to modify several files of the Unicon runtime system and add new files for the new functions implementation.

### New files

- **FDB.R:** This is the main file of IODBC. It contains the Unicon ODBC function set implementation. It is written in standard C with RTT extensions.
- **RDB.R:** contains the C implementation of odbcerr function that is widely called in FDB.R.

### Modified files

- RPROTO.H: contains odbcerr function definition.
- OMISC.R: "\*" operator implementation for ODBC file type.
- FDEFS.H: ODBC function definitions.
- DATA.R: runerr error code for ODBC file mismatch.
- RSTRUCTS.H: ISQLFile definition (ODBC connection type).
- REXTERNS.H: ISQLEnv extern definition.
- RMACROS.H: Fs\_ODBC file status flag and ODBC error codes.
- SYS.H: VisualC++ ODBC header files inclusion (windows.h and sqlext.h).
- INIT.R: ODBC Environment structure release.
- DEFINE.H: ISQL symbol definition for conditional compilation.
- GRTTIN.H: new ODBC types definitions.
- MAKEFILE.RUN: Runtime system makefile (FDB.R and RDB.R definitions added)
- ICONX.LNK: Link file (XFDB.OBJ and XRDB.OBJ definitions added)

# **ODBC Function Reference**

## **dbclose()**

**Description:** closes an ODBC file

**Syntax:** dbclose(f)

### **Parameters**

- **f:** ODBC file previously opened with dbopen()

### **Code Example**

```
procedure main()  
  # open table "mytable" in mydb datasource name defined in  
  # ODBC Data Sources (see Windows 9x Control Panel folder)  
  # using username "federico" and password "mypassword"  
  
  db:=dbopen( "mydb" , "mytable" , "federico" , "mypassword" )  
  
  #  
  # program body  
  #  
  
  dbclose(db) # close table and disconnect  
end
```

## dbcolumns()

**Description:** returns the list of column information related to f

**Syntax:** dbcolumns(f)

### Parameters

- **f:** ODBC file previously opened with dbopen()

**Return Type:** list of records with the following string fields:

- **catalog:** catalog name
- **schema:** schema name
- **tablename:** table name
- **colname:** column name
- **datatype:** SQL data type
- **typename:** data source-dependent data type name
- **colsize:** if **datatype** is SQL\_CHAR or SQL\_VARCHAR this column contains the maximum length in characters of the column
- **buflen:** length in bytes of data transferred on a fetch operation
- **decdigits:** the total number of significant digits to the right of the decimal point
- **numprecradix:** for numeric data types either 10 or 2. If it is 10, the values in **columnsize** and **decimaldigits** give the number of decimal digits allowed for the column. If it is 2, the values in **columnsize** and decimal digits give the number of bits allowed in the column
- **nullable:** 0 if the column could not include NULL values; 1 if the columns accept NULL values; 2 if it is not known whether the column accepts NULL values.
- **remarks:** a description of the column

### Code Example

```
procedure main()
  f:=dbopen("mysql","test","federico","") # open table
  colinfo:=dbcolumns(f) # get columns information
  write("column info\n")

  every i:=(1 to *colinfo) do { # for each column
    writes("col #",i," : ")
    every j:=(1 to *colinfo[i]) do # write column's info
      writes("[",colinfo[i][j],"]")
    write()
  }
  write()
  dbclose(f) # close table and connection to the database
end
```

## **dbdelete()**

**Description:** removes one or more rows from a database table

**Syntax:** dbdelete(f [, [criteria]])

**Parameters:**

- **f:** ODBC file previously opened with dbopen()
- **criteria:** search condition string

**Note:** if *criteria* parameter is omitted dbdelete() is going to empty the table.

```
dbdelete(f) # to empty table f (dangerous!)  
dbdelete(f, ) # equivalent to dbdelete(f)
```

### **Code Example**

```
procedure main()  
  f:=dbopen("mysql", "test", "fbalbi", "mypass") # open table  
  test  
  
  dbdelete(f, "id=25") # delete row(s) with column id=25  
  
  dbclose(f) # close table  
end
```

## **dbdriver()**

**Description:** returns information about the driver being used

**Syntax:** dbdriver(f)

**Parameters:**

- **f:** ODBC file previously opened with dbopen()

**Return Type:** Record with the following string fields:

- **"name":** filename of the driver used to access the data source.
- **"ver":** version of the driver and, optionally a description of the driver.
- **"odbcver":** version of ODBC that the driver supports.
- **"dsn":** A character string with the data source name used during connection.
- **connections:** maximum number of active connections that the driver can support (zero for no specified limit or if the limit is unknown).
- **statements:** maximum number of statements that the driver can support for a connection (zero for no specified limit or if the limit is unknown).

### **Code Example**

```
procedure main()  
  f:=dbopen("mydb","mytable","fbalbi","") # open mytable  
  
  dinfo:=dbdriver(f) # get driver information record  
  
  write("driver name      : ", dinfo["name"])  
  write("driver version   : ", dinfo["ver"])  
  write("driver ODBC ver  : ", dinfo["odbcver"])  
  write("connections     : ", dinfo["connections"])  
  write("statements       : ", dinfo["statements"])  
  write("data source name: ", dinfo["dsn"])  
  
  dbclose(f) # close table  
end
```

## **dbfetch()**

**Description:** fetches and returns a row from a rowset

**Syntax:** dbfetch(f)

**Parameters:**

- **f:** ODBC file previously opened with dbopen()

**Return Type:** record with fields names equal to the selected table columns (see dbselect for column selection)

### **Code Example**

```
procedure main()
  f:=dbopen("mydb","mytable","fbalbi","mypass")

  # select 3 existing columns from table mytable and
  # "", "*": all rows are selected
  # (see dbselect link for more information)

  dbselect(f, "id, name, amt","")

  # *f = number of selected rows
  # may not work with some DBMS

  write(*f, " row(s) selected")

  write("\nrow values")

  # dbfetch returns a record
  # fields name are the columns names selected with dbselect
  # in case of "" or "*" all columns are selected
  # in this example we can reference fields using row["id"],
  # row["name"] and row["amt"]

  while (row:=dbfetch(f)) do { # while rows to retrieve
    every col:=(1 to *row) do # for each col of row
      writes("[",row[col],"]") # write row field
    write()
  }

  dbclose(f) # close table
end
```

## **dbinsert()**

**Description:** insert a row into a database table

**Syntax:** dbinsert(f, rec)

**Parameters:**

- **f:** ODBC file previously opened with dbopen()
- **rec:** row to insert

### **Code Example**

```
record article(id,title,abstract) # define database row

procedure main()
  db:=dbopen("nlm","articles","fede","passwd") # open table

  # set new row value
  row:=article(1,"Unicon ODBC","ODBC in Unicon is FUN!")

  dbinsert(db, row) # insert row into database

  dbclose(db) # close table
end
```

## **dbkeys()**

**Description:** returns information about the primary key columns

**Syntax:** dbkeys(f)

**Parameters:**

- **f:** ODBC file previously opened with dbopen()

**Return Type:** list of records with the following string fields:

- **col:** column name.
- **seq:** sequence number.

### **Code Example**

```
procedure main()  
  f:=dbopen("mydb","user","fbalbi","passwd") # open table  
  
  write(*f, " row(s) selected")  
  
  write("\ntable keys")  
  
  krec:=dbkeys(f) # retrieve primary key information  
  
  every i:=(1 to *krec) do {  
    r:=krec[i]  
    write("[", r["col"], "]") # print key name  
  }  
  
  dbclose(f) # close table  
end
```

## dblimits()

**Description:** returns information about the limits applied for identifiers and clauses in SQL statements.

**Syntax:** dblimits(f)

**Return Type:** record with the following string fields:

- **"maxbinlitlen"**: maximum length of a binary literal in an SQL statement. if there is no maximum length or the length is unknown, this value is set to zero.
- **"maxcharlitlen"**: maximum length of a character literal in an SQL statement. if there is no maximum length or the length is unknown, this value is set to zero.
- **"maxcolnamelen"**: maximum length of a column name in the data source. if there is no maximum length or the length is unknown, this value is set to zero.
- **"maxgroupbycols"**: maximum number of columns allowed in a GROUP BY clause. If there is no specified limit or the limit is unknown, this value is set to zero.
- **"maxorderbycols"**: maximum number of columns allowed in a ORDER BY clause. If there is no specified limit or the limit is unknown, this value is set to zero.
- **"maxindexcols"**: maximum number of columns allowed in an index. If there is no specified limit or the limit is unknown, this value is set to zero.
- **"maxselectcols"**: maximum number of columns allowed in a SELECT list. If there is no specified limit or the limit is unknown, this value is set to zero.
- **"maxtblcols"**: maximum number of columns allowed in a table. If there is no specified limit or the limit is unknown, this value is set to zero.
- **"maxcursnamelen"**: maximum name length of a cursor name in the data source. If there is no specified limit or the limit is unknown, this value is set to zero.
- **"maxindexsize"**: maximum number of bytes allowed in the combined fields of an index. If there is no specified limit or the limit is unknown, this value is set to zero.
- **"maxownnamelen"**: maximum length of an owner name in the data source. If there is no specified limit or the limit is unknown, this value is set to zero.
- **"maxprocnamelen"**: maximum length of a procedure name in the data source. If there is no specified limit or the limit is unknown, this value is set to zero.
- **"maxqualnamelen"**: maximum length of a qualifier name in the data source. If there is no specified limit or the limit is unknown, this value is set to zero.
- **"maxrowsize"**: maximum length of a single row in a table. If there is no specified limit or the limit is unknown, this value is set to zero.
- **"maxrowsizealong"**: a character string: "Y" if the maximum row size returned for the "maxrowize" information type includes the length of all SQL\_LONGVARCHAR and SQL\_LONGVARBINARY columns in the row; "N" otherwise.
- **"maxstmtlen"**: maximum length (number of characters, including white space) of an SQL statement. If there is no maximum length or the length is unknown, this value is set to zero.
- **"maxtblnamelen"**: maximum length of a table name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.

- **"maxselecttbls"**: maximum number of tables allowed in the FROM clause of a SELECT statement. If there is no maximum length or the length is unknown, this value is set to zero.
- **"maxusername"**: maximum length of a user name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.

### Code Example

```
procedure main()  
  f:=dbopen("mydb","mytable","fbalbi","") # open mytable  
  
  dbl:=dblimits(f) # get DBMS limits information  
  
  # print out all DBMS limits  
  every i:=(1 to *dbl) do write(dbl[i])  
  
  dbclose(f) # close table  
end
```

## **dbopen()**

**Description:** connects to a database, opens a table and returns the associated ODBC file.

**Syntax:** dbopen(db, table, user, password)

### **Parameters:**

- **db:** Data Source Name string (defined in ODBC Manager)
- **table:** database table string
- **user:** database user string
- **password:** user password string

### **Code Example**

```
procedure main()  
  # open "mytable" in mydb data source name defined in  
  # ODBC Data Sources (see Windows 9x Control Panel folder)  
  # using username "federico" and password "mypassword"  
  
  db:=dbopen("mydb", "mytable", "federico", "password")  
  
  #  
  # program body  
  #  
  
  dbclose(db) # close table and disconnect  
end
```

## **dbproduct()**

**Description:** returns information about the DBMS accessed by the driver

**Syntax:** dbproduct(f)

**Parameters:**

- **f:** ODBC file previously opened with dbopen()

**Return Type:** record with the following string fields:

- **name:** DBMS product name
- **ver:** DBMS version

### **Code Example**

```
procedure main()  
  f:=dbopen("mydb","test","fbalbi","mypasswd") # open table  
  
  p:=dbproduct(f) # get DBMS product information  
  
  write("product name: ", p["name"]) # print product name  
  write("product ver : ", p["ver"]) # print product version  
  
  dbclose(f) # close table  
end
```

## dbselect()

**Description:** selects a rowset from a table

**Syntax:** dbselect(f [, [columns] [, [criteria] [, [order]]])

### **Parameters:**

- **f:** ODBC file previously opened with dbopen()
- **columns:** list of columns to retrieve. If columns is "\*" or a null string or not specified all columns in the table are selected.
- **criteria:** search condition string. If criteria is a null string or not specified all rows are selected.
- **order:** column(s) by which the result set will be sorted.

### **Examples**

Suppose we want to perform the following SQL queries on table f:

```
SELECT * FROM f
```

Using dbselect() we can do this in several ways:

```
dbselect(f, "*")  
dbselect(f, "")  
dbselect(f, , , )  
dbselect(f)
```

```
SELECT name FROM f
```

can be written using dbselect() as:

```
dbselect(f, "name")  
dbselect(f, "name", , )
```

```
SELECT name FROM f ORDER BY id
```

is equivalent to:

```
dbselect(f, "name", , "id")  
dbselect(f, "name", "", "id")
```

```
SELECT id, name FROM f WHERE id > 100 ORDER BY name
```

would be written as:

```
dbselect(f, "id,name", "id > 100", "name")
```

## Code Example

```
procedure main()
  f:=dbopen("mysql","test","fbalbi","passwd") # open table

  # select columns 'name', 'id', 'amt' from test
  # rowset is made of rows with name='Fred'
  # and order the rowset by due date

  r:=dbselect(f,"id,name,amt,paid,due","name='Fred'", "due")

  write(*f, " row(s) selected") # may not work on some DBMS

  rec:=dbfetch(f) # retrieve first record in the rowset

  every i:=(1 to *rec) do # write record fields values
    write("[", rec[i], "]")

  dbclose(f) # close table
end
```

## dbsql()

**Description:** submits an SQL query using the connection opened by f

**Syntax:** dbsql(f, query)

**Parameters:**

- **f:** ODBC file previously opened with dbopen()
- **query:** SQL statement string

### **Code Example**

```
procedure main()  
  # connect to DBMS and open table  
  
  db:=dbopen("personnel_db", "table", "manager", "passwd")  
  
  # prepare SQL query string to create an employees table  
  # of 5 columns  
  
  query := "CREATE TABLE employees (id INTEGER PRIMARY KEY,  
        name VARCHAR(40), phone VARCHAR(12), DOB DATE,  
        pay FLOAT)"  
  
  # dbsql needs an opened connection in order to work;  
  # that's why we have to open an existing table associated  
  # to db. This requirement will be avoided on next IODBC  
  # versions  
  
  dbsql(db, query) # execute query  
  
  dbclose(db) # close  
end
```

## **dbtables()**

**Description:** returns a list of records with information about tables stored the database related to f

**Syntax:** dbtables(f)

**Parameters:**

- **f:** ODBC file previously opened with dbopen()

**Return Type:** list of records with the following string fields:

- **qualifier:** table qualifier
- **owner:** table owner
- **name:** table name
- **type:** table type
- **remarks:** table remarks

### **Code Example**

```
procedure main()  
  
    # get current database tables information  
  
    f:=dbopen("mysql","test","fbalbi","xxxxxxx")  
  
    tablelist:=dbtables(f)  
  
    # write number of tables  
    write("size list = ", *tablelist)  
  
    every i:=(1 to *tablelist) do { # for each table  
        r:=tablelist[i]  
  
        # print table information fields  
        every j:=(1 to *r) do writes("[",r[j],"]")  
        write()  
    }  
  
    dbclose(f) # close table  
end
```

## **dbupdate()**

**Description:** update a database table row

**Syntax:** dbupdate(f, rec)

**Parameters:**

- **f:** ODBC file previously opened with dbopen()
- **rec:** table row to update

### **Code Example**

```
procedure main()  
  f:=dbopen("mydb","mytable","fbalbi","xxxx") # open mytable  
  
  # select columns id, amt  
  # select row where id<6  
  
  r:=dbselect(f, "id, amt", "id<6")  
  
  write(*f, " row(s) selected")  
  
  write("\nrow values")  
  
  while (row:=dbfetch(f)) do { # for each row in the rowset  
    # update amt field value  
  
    row["amt"]+=1000.0  
  
    # update database row  
  
    dbupdate(f, row)  
  }  
  
  dbclose(f) # close table  
end
```

## References

- [1] Kyle Giger. Inside ODBC. Microsoft Press.
- [2] Roger E. Sanders. ODBC 3.5 Developer's Guide.
- [3] Microsoft ODBC 2.0 Programmer's Reference Guide.
- [4] Microsoft ODBC 3.0 Programmer's Reference.
- [5] Ralph E. Griswold, Madge T. Griswold. The Icon Programming Language.
- [6] Ralph E. Griswold. The Icon Language Implementation.
- [7] Clinton Jeffery, Programming with Unicon.
- [8] MySQL Reference Manual for version 3.23.2-alpha.