# The ODBC interface

key concepts

- the concept of ODBC
- the structure of ODBC
- the ODBC drivers
- the tasks of drivers
- the main steps of the request processing
- ODBC handlers
- connecting to the data source
- SQL statement execution
- reading data from the data source
- isolation levels of the ODBC
- conformity levels
- error handling
- sample programs

# The concept of ODBC

ODBC: Open Database Connectivity

It offers a common API for different relational-like data sources

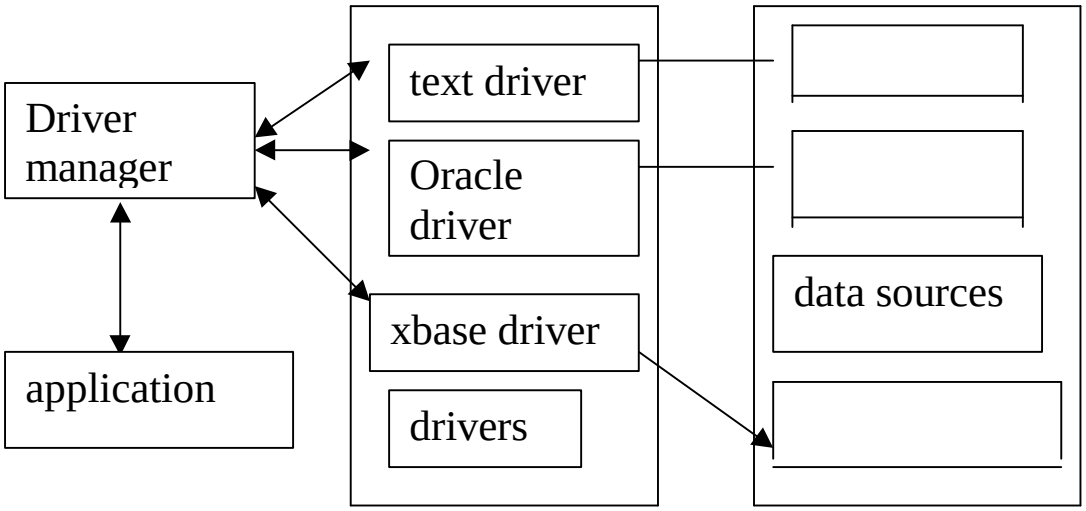
It provides:

- data source independence
- conversion
- function extension
- network independence
- parallel data access
- unified data access functions
- simple data access functions
- SQL based data management commands

It is supported by several data sources like

- Oracle
- DB2
- Informix
- Sybase
- SQLServer
- Xbase
- Excel
- text files
- …

# The structure of ODBC




Driver manager:

it accepts the ODBC CLI calls of the applications and performs the following steps:

- syntax checking of the call statement
- it loads the required drivers into the memory
- it controls the connections
- it passes the SQL command
- it performs some kind of conversion
- it makes the API easier

ODBC drivers

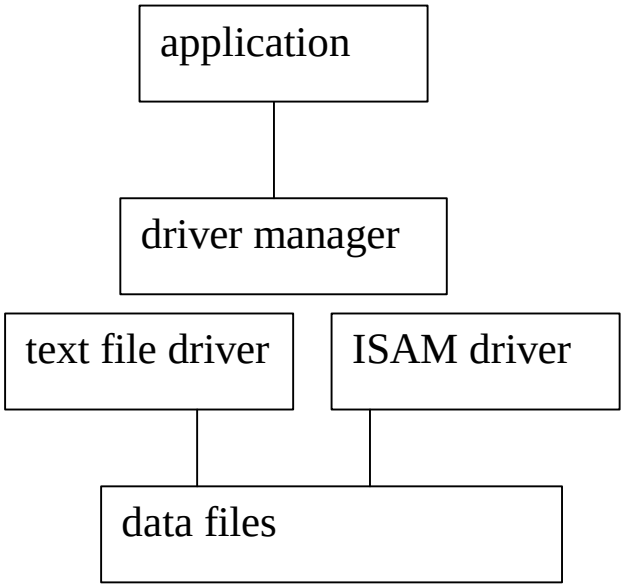
The driver converts the common API calls to data source specific DBMS commands. The amount of conversion work depends on the type of data source

Types of drivers:

- One layer drivers:
- Two layers drivers

*One layer drivers*

One layer drivers access the data source directly.




The driver should perform the DBMS functions, it replaces the missing DBMS

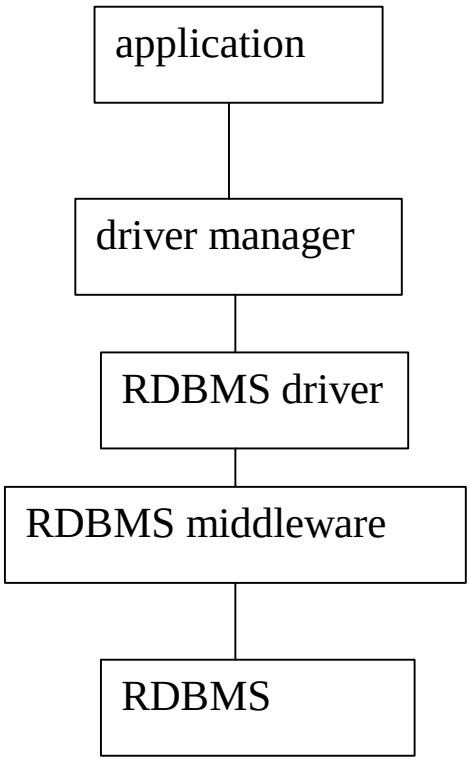
The one layer driver must contain a simple relational DBMS motor.

It provides usually only a simple DMBS functionality (no transaction management, no indexing)

The data source is usually located on the same node as the ODBC driver.

*The two layer drivers*

The two layer drivers access the data source via a running DBMS component.

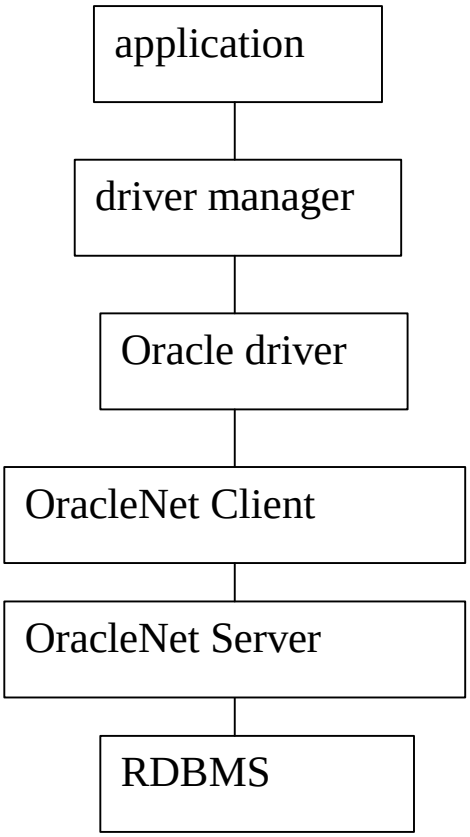


The driver should only transform the SQL commands to the existing RDBMS

It is based on the functionality of the existing DMBS (with transaction management and indexing)

The data source may be located on different node as the ODBC driver.

The driver usually access the network component of the corresponding RDBMS



## Tasks of the drivers

The main tasks of the drivers are

- management of connections to the specific data sources
  connection parameters:
  - connection identifier
  - data source identifier
  - authentication
  - transaction mode
- error handling
  - it should provide standard error codes
  - it should provide data source specific error codes too
- conversion of the SQL commands
  - it should perform conversion between the different SQL dialects
- providing information about the data sources
  - unified catalog identification
  - it should provide access to the different system meta-data tables

The main steps of the request processing

1. allocation of a session handler (connection to the ODBC system)
2. allocation of a connection handler
3. connect to the data source
4. the driver manager loads the required driver
5. allocation the driver part of the connection handler
6. connection to the data source
7. allocation of the statement handler both in the application and in the driver
8. sending the SQL statement to the ODBC
9. passing the SQL statement to the driver
10. transformation of the SQL statement to native SQL or to low level code module
11. execution of the SQL statement at the driver or at the data source
12. generating the status code
13. converting the error codes to standard form
14. passing status code back to the application
15. releasing the statement handler
16. releasing the connection handler
17. releasing the session handler

# ODBC handlers

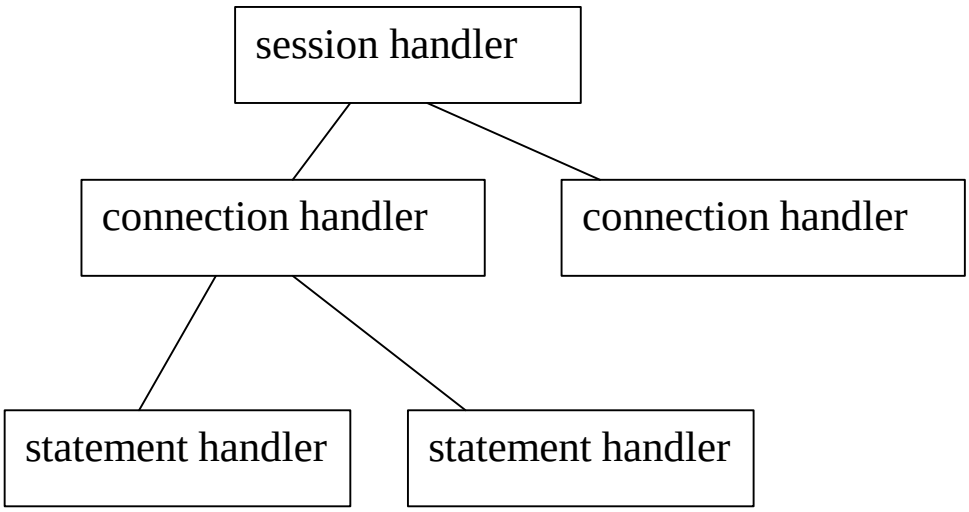
The ODBC objects are identified and accessed by handlers

The handler is a data structure containing the object attributes and access parameters. It hides the information details.

Types of handlers:

- session
- connection
- statement

The relation among the handlers can be described by a handler tree (hierarchy)



Benefits of the handler method

- parallel connections
- parallel statement execution
- information hiding
- OO concepts
- security aspects

Session handler:

- it contains the children handlers
- it stores the global information related to the user
- it describes the ODBC level transaction parameters

Connection handler:

- it describes the data access parameters
- it performs transaction handling
- and the error handling

Statement handler:

- it receives and executes the SQL statement
- it passes the result data back to the user
- error handling

SQLAllocEnv()
SQLAllocConnect()
SQLAllocStmt()

## Connecting to the data source

The data source is configured by the ODBC control module.

DSN : data source name, parameters:

- id name
- driver type
- location
- access parameters, authentication
- special parameters

Connection:

- direct connection
  SQLConnect()
  all required parameters are passed in the arguments of the SQLConnect function
- driver connection
  SQLDriverConnect()
  not every required parameters are passed, the ODBC asks for the missing values using a standard GUI dialog window
- browse connection
  SQLBrowseConnect()
  not every required parameters are passed, the ODBC asks for the missing values using the return string value

SQL statement execution

Execution modes:

- direct, one phase
- two phases

One phase statement execution:

After generating the statement string, the statement is executed by only one function call:

SQLExecDirect (..,statement,..)

- it provides simple execution model
- flexible, ad-hoc operations

Two phases statement execution:

After generating the statement string, the statement is first prepared for execution by the function call:

SQLPrepare (..,statement,..)

and then it can be executed several times by the call:

SQLExec()

- it provides more efficiency in the case of repeated executions
- flexible, ad-hoc operations

Execution of stored procedures:

SQLExecDirect (‘call procedure-name’)

## Reading data from the data source

The result of a query can be:

- one field
- one record
- more records

reading one field into a host variable:

SQLGetData()

reading one record from the data source

- it is based on the cursor model
- binding the output of the query to host variables:
  SQLBindCol()
- transfer the next record from the cursor into the host variables
  SQLFetch()

reading more records from the data source:

- it is based on the extended cursor model:
  SQLExtendFetch()

Isolation levels of the ODBC

It is related to the transaction management capabilities of the data source

Not every isolation level can be realized at every data source

Transaction isolation levels:

- READ UNCOMMITTED
  it allows to read data before committing the transaction
- READ COMMITTED
  data can be read only after committing the transaction
- REPEATABLE READ
  the same data value is read during the whole transaction
- SERIALIZABLE
  the transactions have no effect to each others

The transaction isolation parameter can be set by the

SQLSETStmtOption()

function call.

## Conformity levels

The functionality of the ODBC may be vary in the different versions and implementations

Two types of conformity:

- API level
- SQL level

API levels:

- core functionality
  simple connection management
  SQL execution
  transaction settings
  error handling
- first level
- second level

SQL levels:

- minimal
  CREATE DROP SELECT
  INSERT UPDATE DELETE
- core SQL
- extended SQL
  stored procedures

Error handling:

the return value of the function calls depends on the result

```
x = SQLExecDirect()
if (x ==…)
```

In the case of error the error code and error message can be get by recursive call of the

```
SQLError()
```

function.

Sample Programs

Host language and OS : C, Windows

Creating the ODBC driver:
- Control panel
  - add USR DSN
    Name: HELLO
    Driver: MS Text Driver
    Database Directory:
    C:\MSVC20\peldak\HELLO

C source code

main block

```
BOOL C the App: initInstance()
{
  void sqlproba();
  …
  m_pMainWnd=new CmainWindow;
  m_pMainWnd→ShowWindow(m_n(mdShow);
  m_pMainWnd→UpdateWindow();
  Sqlproba();
  Return TRUE;
}
```

sqlproba function

```
Void sqlproba()
{
  RETCODE rc;
  HEVN henv;
  HDBC hdbc
  HSTMT hstmt
  Char szData[MAX_DATA]
  SWORD cbData

  SQLAllocEnv(&henv);
  SQLAllocConnect(henv,&hdbc);
  SQLConnect(hdbc,(unsigned                    char*)
  ”HELLO”,SQL_NTS,NULL,0, NULL,0);

  SQLAllocStmt(hdbc,&hstmt);
  SQLExecDirect(nstmt,(unsigned char*)
  “SELECT * FROM Minta “, SQL_NTS);
  for (rc=SQLFetch(hstmt);rc==SQL_SUCCESS;
  rc=SQLFetch(hstmt)){
  SQLGetData(hstmt,1,SQL_C_CHAR,SzData,
        Siyeof(sydata),&cbData);
  MessageBox(NULL,syData,”ODBC”,MB_OK);
  }
  SQLFreeStmt(hstmt,SQL_DROP);
  SQLDisconnect(hdbe);
  SQLFreeConnet(hdbe);
  SQLFreeEno(henv)
```

}

Data Source:
C:\MSC20\peldak\HELLO – ban
Minta.txt

Access to RDBMS

Data source : Sybase RDBMS

C source code:

```
SQLAllocEnv(&henv);
SQLAllocConnect(henv,Rdbc);
Rc=SQLConnect(hdbe,(unsigned char*)”sybaseminta”,
  “dba”, SQL.NTS,”sql”,SQL_NTS);
  (nev)
if (rc!=SQL_SUCCESS) {
  char sqlst[10],
  char msg[100];
  SDWORD nNUM;
  SDWORD cbm;
  While
  (SQLError(SQL_NULL_HENV,hdbe,SQL_NULL_HS
  TMT,
  (unsigned char *) sqlst,&nNUM,(unsigned shar *)msg,
  siyeof(msg),
```

```
   &chm)==SQL_SUCCESS){
   Messagebox(NULL,msg,”ODBC hiba”,MB_OK);
   }
}
```

```
SQLAllocStmt(    )

Type SWPD clols;
SQLExecDirect(hstmt,(unsigned char *)” select * from auto”, SQL.NTS);
SQLNumResultCols(hstmt,&clols);
For (rc=SQLFetch(hstmt); rc==SQL_SUCCESS;
   Rc=SQLFetch(hstmt)){
   Inti;
   For(i=1;i<=clols;i++){
   SQLGetData(hstmt,i,SQL.C
   SCAR,Szdata,Sizeof(szData))
   MessageBox(NULL,szData,”ODBC”,MB.OK);
   }
}
```