Database Access via Programming Languages

- SQL is a direct query language; as such, it has limitations.

- Some reasons why access to databases via programming languages is needed :

  - Complex computational processing of the data.
  - Specialized user interfaces.
  - Access to more than one database at a time.

Desirable features of such systems:

- Ease of use.
- Simplicity of implementation.
- Conformance to standards for existing programming languages and database query languages.
- *Interoperability*: the ability to use a common interface to diverse database systems.

A Closer Look at Interoperability

- Consider SQL: it is a standard direct query language which may be used with virtually any relational database system.

- Question: To what extent can a similar standard be attained which focuses upon embedding SQL-style query support within a powerful programming language?

Some desirable properties:

- Make use of existing standards for database queries (*e.g.*, SQL).
- Make use of existing standards for programming languages (*e.g.*, C, C++, Ada, etc.)
- Make use of existing development environments for programming languages (*e.g.,* Borland C++, Microsoft Visual C++, Anjuta DevStudio, CodeForge, etc.).
- Admit (possibly *simultaneous*) access to a diverse collection of database systems (*e.g.,* Oracle, Sybase, Microsoft, Interbase products) using a common syntax and semantics.
- Solutions should not depend upon characteristics of particular operating systems. Rather, they should be available on a variety of systems, such as Microsoft Windows, Mac OS, Linux, and UNIX, *without having to depend upon the OS vendor for DBMS-specific support.*

Vendor-specific solutions:

- Oracle PL/SQL: A proprietary PL/1-like language which supports the execution of SQL queries:
  - Advantages:
    - Many Oracle-specific features, not common to other systems, are supported.
    - Performance may be optimized to Oracle-based systems.
  - Disadvantages:
    - Ties the applications to a specific DBMS.
    - The application programmer must depend upon the vendor for the application development environment. It may not be available for all platforms.
- Microsoft VBA (Visual Basic for Applications) support for MS Access:
  - Advantages:
    - Uses the common Microsoft application language VBA.
  - Disadvantages:
    - Totally specific to Microsoft systems.

Although they have some obvious advantages, these solutions do not embody interoperability.

Vendor-independent solutions based upon SQL:

There are three basic strategies which may be considered:

- Embedded SQL
- SQL modules
- SQL call level interfaces.

Embedded SQL:

- In this strategy, calls to SQL statements are embedded in a host programming language, such as C. Such calls are tagged by a special marker, such as “EXEC SQL.”

- A preprocessor is invoked to convert the source program into a “pure” host-language program. The SQL calls are converted to host-language statements via a preprocessor.

- In *static* embedded SQL, table and attribute names must be declared in the source program.

- In *dynamic* embedded SQL, they may be provided at run time.

- There is an ISO standard for embedded SQL.

Disadvantages:

- It is a real pain to debug preprocessed programs.

- The use of a program-development environment is compromised substantially.

- The preprocessor must be vendor and platform specific.

A very simple example of a static embeded SQL program is shown on the next slide.

```
/* Skeleton example of embedded SQL in C */
#include <stdio.h>
#include <stdlib.h>

/* Include communications area */
EXEC SQL INCLUDE sqlca

main ()
{
EXEC SQL BEGIN DECLARE SECTION
char Lname[16];
char SSN[10];
EXEC SQL END DECLARE SECTION


EXEC SQL CONNECT 'company';
if (sqlca.sqlcode < 0) exit(-1);

printf("Enter SSN: ");
scanf("%s", SSN);

EXEC SQL SELECT Lname FROM Employee
                      WHERE SSN = :SSN;

switch (sqlca.sqlcode)
 {
  case 0: printf("Last name is %s.\n", LName);
    break;
  case 100: printf("No last name for %s.\n, SSN);
    break;
  default: printf("SQL error %d.\n",
                   sqlca.sqlcode);
 };
return(0);
EXEC SQL DISCONNECT;
};
```

SQL Modules:

- In the module approach, invocations to SQL are made via libraries of procedures, rather than via preprocessing.

Advantages over embedded SQL:

- Clean separation of SQL from the host programming language.
- Debugging is much more straightforward, since no preprocessor is involved.

Disadvantages:

- The module libraries are specific to both the DBMS and programming language and environment. Thus, portability is compromised greatly.

Is there a standard for such systems???

Vaguely, PL/SQL is an example of such a system, although it does not really use PL/1 as the host language. Rather, it is an integrated system.

JDBC is perhaps another example, using Java as the host language. Again, it is proprietary.

SQL Call-Level Interfaces:

- A call-level interface provides a libarary of functions for access to DBMS's.

- The DBMS drivers are stored separately; thus the library used by the programming language is DBMS independent. The programming language functions provided only an interface to the DBMS drivers.

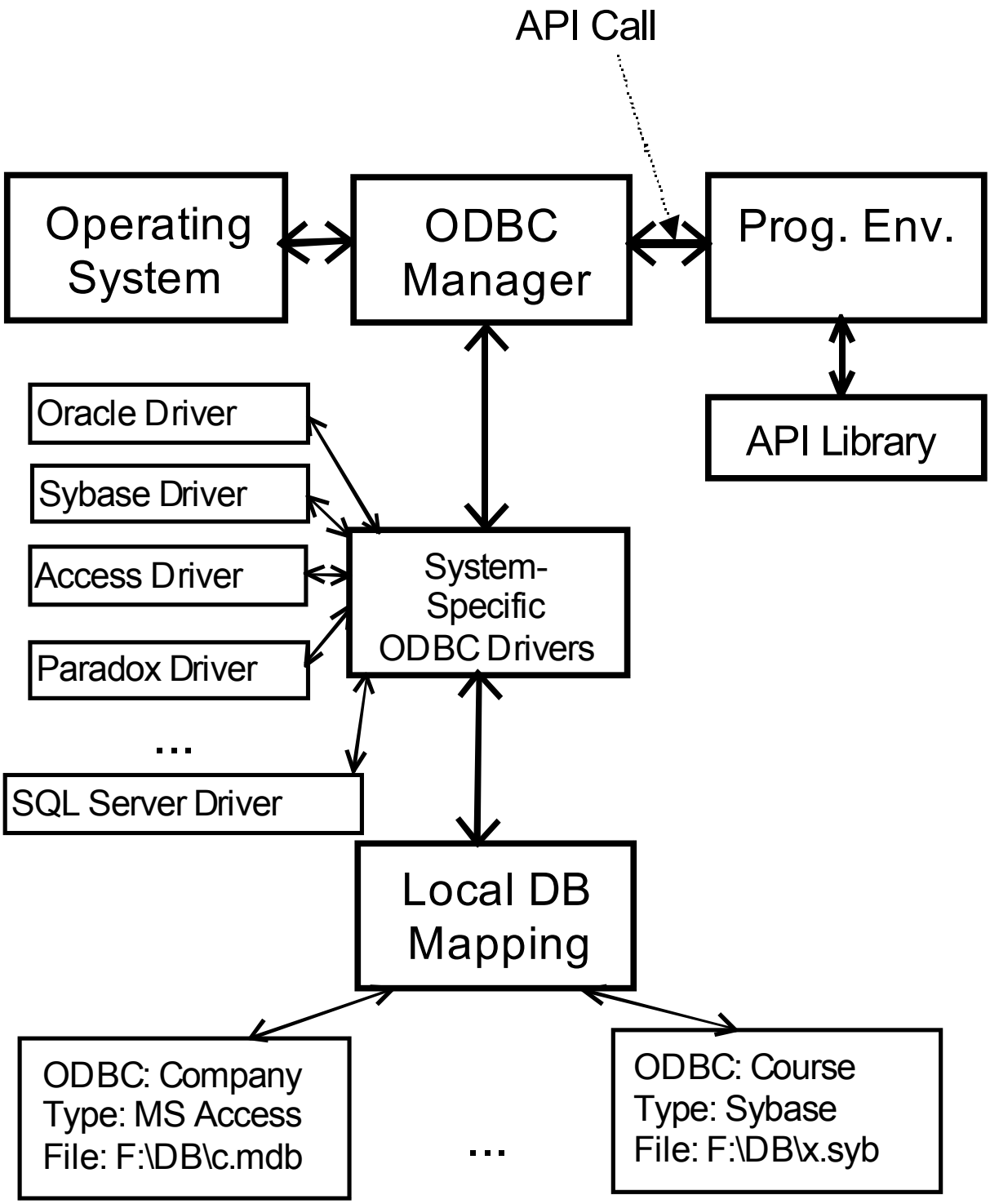
Advantages:

- The development environment is not tied to a particular DBMS, operating sytem, or even a particular development environment.

Disadvantages:

- Some low-level optimization may be more difficult or impossible to achieve.

- Key example:
  - The SQL CLI (X/Open CLI)
  - Microsoft ODBC (Open Database Connectivity)

  - The two are closely aligned.

An oversimplified picture of ODBC: