Using ODBC

- The main ideas are presented via a sequence of four annotated C programs.

- These slides provide only supporting information.

Context:

- These notes deal with ODBC progtrams written using the gcc compiler under Linux/Unix.

- Although the examples are intended to be generic, they have been tested only with the PostgreSQL database system, using Debian Linux as the client-side operating system.

- For information regarding ODBC with Microsoft Visual C++ and Windows operating systems, consult the slides from 2001.

- Currently, ODBC support is not available for Kexi, an open-source DBMS with features similar to those of Microsoft Access.

References:

- On-line documentation for ODBC is available at the Microsoft web site. Follow the link on the course home page.

The following hardcopy references are provided only as information for those with insatiable appetites for knowledge.

These notes, together with the accompanying sample programs and lectures, should provide enough information to write reasonable ODBC-based applications.

- One may also purchase hardcopy of the Microsoft documentation and software from booksellers. (*Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference, Second Edition* -- Two books plus a CD, 1997)

- A decent book for the eager is *ODBC 3.5 Developers Guide,* by Roger E. Sanders, McGraw-Hill, 1999.

- A great reference????? If one were available at a reasonable price, it would become part of the course literature.

.

Compiling a C Program with ODBC Calls under the Linux Installation:

- To compile a C program with ODBC calls, one of the ODBC client-side libraries must be included.

- One of the following should work:

```
cc -lodbc program.c
cc -liodbc program.c
```

- Although they are functionally equivalent, the libraries `odbc` and `iodbc` cannot co-exist on the same system.

- Try one, if a list of error messages appear, try the other.

- Currently, use `-liodbc`.

- The actual structure of C programs which contain ODBC calls will is illustrated via accompanying example programs, with some basic principles discussed later in these slides.

Data-Source Configuration under the Linux Installation:

- Every data source which is to be reached via ODBC calls must be declared in the `.odbc.ini` file in the home directory of the user.

- A minimal example file is shown below for connection to PostgreSQL databases on the `postgres` server when using Linux.

```
[ODBC Data Sources]
mydb1 =  database1
mydb2 =  database2

[database1]
Description         = PostgreSQL test database 1
Driver  = /usr/lib/postgresql/lib/psqlodbc.so
Database            = hegner1
Servername          = postgres

[database2]
Description         = PostgreSQL test database 2
Driver  = /usr/lib/postgresql/lib/psqlodbc.so
Database            = hegner2
Servername          = postgres
```

- The `Database` field gives the name of the database which was issued by the system administrator.

- The header name (*e.g.* `[database1]`) is the ODBC name for the database, and may be chosen arbitrarily.

Variations:

- Shown below is a more complete `.odbc.ini` entry, which expands some default entries.

```
[ODBC Data Sources]
mydb3 =  database3

[database3]
Description         = PostgreSQL test database 1
Driver  = /usr/lib/postgresql/lib/psqlodbc.so

Database            = hegner1
Servername          = postgres
Port                = 5432
ReadOnly            = 0
Username            = hegner1
Password            = "badidea"
Trace               = No
TraceFile           = /tmp/odbc.log
```

- Attributes such as `Port` and `ReadOnly` need only be specified if they differ from the default values.
- `Trace` and `TraceFile` need only be specified if tracing is desired.

- To use options such as `UserName` and `Password`, it is necessary to use the `SQLDriverConnect` call, which is not discussed in these notes. See the example program `iodbc.c` which comes with the iodbc library for details.

- Needless to say, it is not a good idea to put the password in this file.

Variations for the Solaris Installation:

- Under the Solaris installation, use the `unixodbc` library, and specify `gcc` and the load path explicitly:

```
gcc -L/usr/local/lib -lodbc program.c
```

- Under the Solaris installation, the location of the PostgreSQL ODBC driver is specified in the `.odbc.ini` file as follows:

```
[ODBC Data Sources]
mydb1 =  database1
mydb2 =  database2

[database1]
Description         = PostgreSQL test database 1
Driver  = /usr/local/lib/psqlodbc.so
Database            = hegner1
Servername          = postgres

[database2]
Description         = PostgreSQL test database 2
Driver  = /usr/local/lib/psqlodbc.so
Database            = hegner2
Servername          = postgres
```

- To use a single `.odbc.ini` file for under both Linux and Solaris, a simple solution is to use different ODBC names:

```
[ODBC Data Sources]
mydb1Solaris =  database1S
mydb2Solaris =  database2S
mydb1Linux   =  database1L
mydb2Linux   =  database2L

[database1S]
Description         = PostgreSQL test database 1
Driver  = /usr/local/lib/psqlodbc.so
Database            = hegner1
Servername          = postgres

[database2S]
Description         = PostgreSQL test database 2
Driver  = /usr/local/lib/psqlodbc.so
Database            = hegner2
Servername          = postgres

[database1L]
Description         = PostgreSQL test database 1
Driver  = /usr/lib/postgresql/lib/psqlodbc.so
Database            = hegner1
Servername          = postgres

[database2L]
Description         = PostgreSQL test database 2
Driver  = /usr/lib/postgresql/lib/psqlodbc.so
Database            = hegner2
Servername          = postgres
```

- This requires changing the name of the database to be connected to when changing systems.

- A more elegant solution would be to create your own soft link to the driver, which is set in a system-dependent fashion in a login script.

Some Basics of ODBC calls in C:

Identifiers:

- Most ODBC identifiers begin with `SQL` (note the capitalization). Thus, it is a very good idea to avoid using this sequence as the beginning of user-defined identifiers.

API calls:

- ODBC contains a large number of functions (around 80). They have names like `SQLAllocHandle`, and `SQLCloseCursor`. Only a few will be used in this course.
- All (most?) return a value of type `SQLRETURN`. This value is zero if the execution was normal, and nonzero if it was special.

Includes:

- To run ODBC API calls, the following two includes must be issued:

```
#include <sql.h>
#include <sqlext.h>
```

Variable types:

There are three classes of variables associated with ODBC.

1. Types to be used as declarations to C. These begin with `SQL`, and continue with a sequence of capital letters, without underscores. They are `#define`d within the header files to be certain C types. Here are some of the principal ones:

ODBC Type	C Type
SQLCHAR	char
SQLSCHAR	signed char
SQLINTEGER	long int
SQLUINTEGER	unsigned long int
SQLSMALLINT	short int
SQLUSMALLINT	unsigned short int
SQLREAL	float
SQLDOUBLE,SQLFLOAT	double
SQLDATE	a large struct..

There are also a number of special ones for date, time etc., which correspond to `struct`s in C.

The definitions are found in the library file `sqltypes.h`. Consult this file or the ODBC documentation for complete information.

For types involved in API calls, these types, rather than the C types, should be used.

2. C data type encodings. These are not true data types, but rather numerical encodings of the types listed in the previous group. These numerical encodings are used as arguments to API function calls. The following table lists some of the principal types.

Integer Encoding	ODBC Type
SQL_C_CHAR	SQLCHAR
SQL_S_STINYINT	SQLSCHAR
SQL_C_SLONG	SQLINTEGER
SQL_C_ULONG	SQLUINTEGER
SQL_C_SSHORT	SQLSMALLINT
SQL_C_USHORT	SQLUSMALLINT
SQL_C_FLOAT	SQLREAL
SQL_C_DOUBLE	SQLDOUBLE,SQLFLOAT
SQL_C_TYPE_DATE	SQLDATE

The definitions for these types are found in the file `sqlext.h`. Consult that file or the ODBC documentation for further information.

It is important to remember that these are **not** C-language data types. They cannot be used in type declarations!!!

3. SQL data types encodings. These provide an association between the types allowed in SQL declarations, and those of the programming language. They are used in arguments to API calls, but never in variable declarations in the program itself.

These are not true data types, but rather numerical encodings which correspond to the numerical encodings of the types in the previous list. They cannot be used in type declarations!!!

The following table gives some principal examples.

Integer Encoding	SQL Type
`SQL_CHAR`	`Char(n)`
`SQL_VARCHAR`	`Varchar(n)`
`SQL_SMALLINT`	`Smallint`
`SQL_INTEGER`	`Integer`
`SQL_REAL`	`Real`
`SQL_DECIMAL`	`Decimal(p,s)`
`SQL_TYPE_DATE`	`Date`

The exact mapping between these types and those of the previous table is implementation dependent.

The definitions for these types are found in the file `sqlext.h`. Consult that file or the ODBC documentation for further information.

Handles:

- Handles are numerical values which are associated with certain items.
- Example: File handles are familiar in operating system programming.

In ODBC, there are four types of handles:

- **Environment handles**: In order to access a database via ODBC, an ODBC environment must be established. There is normally only one such environment per program.

- **Connection handles**: Just as one must have a file handle for every open file in an operating system, so too must one have a connection handle for every ODBC database which is opened.

- **Statement handles**: A statement handle is associated with an SQL statement which is to be issued to an ODBC database for execution.

- **Descriptor handles**: Descriptors are metadata which describe formats associated with SQL statements. They will not be studied in this course.

In ODBC 3.0 and higher:

- Handles are declared using the type `SQLHANDLE`.
- Handles are allocated using the function `SQLAllocHandle`.
- Handles are freed using the function `SQLFreeHandle`.

The slides show examples of these activities.

Remark: There are older, ODBC 2 data types and calls which deal with each of the first three types of handles (all except descriptor handles) separately.

- The types are `HENV`, `HDBC`, and, `HSTMT`.
- The allocation functions are `SQLAllocEnv`, `SQLAllocDbc`, and `SQLAllocStmt`.
- The freeing functions are `SQLFreeEnv`, `SQLFreeDbc`, and `SQLFreeStmt`.

Although most ODBC implementations are backwards compatible with these calls, their use is to be discouraged in new software. (Translation: Do not use them in your project!)

Other important general operations:

- Inform the system of the ODBC version in use: `SQLSetEnvAttr`.

- Connect to a database identified by an allocated connection handle: `SQLConnect`.

- Disconnect from the database allocated to a connection handle: `SQLDisconnect`.
  - The handle remains available for connection to another database.

Special query operations:

- Prepare ("compile") an SQL statement for execution: `SQLPrepare`.

- Execute a compiled SQL statement: `SQLExecute`.

Note: The function `SQLExecDirect` combines the above two functions, and is appropriate in situations in which the SQL statement is executed only once.

- Bind an input parameter index in an SQL statement with a variable in the program: `SQLBindParameter`.

- Bind a column of a query result (output parameter) to a variable in the program. `SQLBindCol`.

- Fetch the next tuple from the result of a query: `SQLFetchTuple`.

- Close the cursor on a given query, so that the statement handle may be used to collect the results of a new query: `SQLCloseCursor`.

Other classes of API calls:

- Catalog queries: Find out which relations are in a given database, what the types of the columns are, what the constraints are, etc.

- Optimization directives: Handle large queries with efficient batch operations.

- Error management: If something goes wrong, find out what the problem is.

All in all, there are over 80 API calls in ODBC.