Database Access via Programming Languages

5DV119 — Introduction to Database Management

Umeå University
Department of Computing Science

Stephen J. Hegner
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

The Limitations of Stand-Alone SQL

- SQL is primarily a language for data definition, retrieval, and update.
- It is not designed for complex computation.
- Enhancements such as OLAP are useful for certain specific tasks, but still leave many important tasks difficult or impossible to achieve.

Theoretical shortcoming: Unlike most programming languages, it is not *Turing complete*.

- There are computations which cannot be expressed in SQL at all.

Interoperability shortcoming: Stand-alone SQL clients are generally vendor specific.

- Concurrent access to databases of different vendors is not possible with a single client.
- Access to multiple databases via the same client is usually awkward, requiring vendor-specific directives.

The Limitations of Stand-Alone SQL: 2

Practical shortcomings: There is also a host of practical reasons why stand-alone SQL does not suffice:

Accessibility: Most users of databases are not computer scientists.

- They need a custom interface for non-experts.
- Even experts can often work more effectively via custom interfaces.

Simplicity: Real-world database schemata are often very large and complex.

- Users often need to work with custom views which present what they need to know and are allowed to know.

Security: The correct management of access rights is a very complex task.

- It is often easier to manage access by admitting access via specific interfaces.

Concurrency: The correct management of concurrent processes is also very complex.

- It is often easier to manage concurrency via properly designed interfaces.

Database Access via Programming Languages: Desiderata

Database access via standard SQL: Ça va sans dire !

Use with:
- traditional programming languages: C, C++, Java, Python.
- languages for Web-based access: PHP, via Apache Tomcat.

Interoperability: Access to several different databases, running the systems of many different vendors, perhaps on different platforms.

The Major Players in the DBMS arena:

The "big three" commercial systems:
- Oracle Database
- IBM DB2
- Microsoft SQL Server

The major open-source systems:
- PostgreSQL
- MySQL/InnoDB

Other significant commercial vendors: Mimer SQL, Sybase

Other products with widespread usage: Microsoft Access

Examples of Vendor-Specific Solutions

Oracle PL/SQL: A proprietary PL/1-like imperative programming language which supports the execution of SQL queries.

Advantages:

- Many Oracle-specific features of SQL and the Oracle Database systems are supported.
- Performance may be optimized in a manner not achievable with solutions which are not vendor specific.

Disadvantages:

- Vendor lock-in: applications are tied to a specific DBMS.
- Application development is dependent upon the existence of a development environment for the language (in this case, PL/SQL), which may not be available on all platforms.
- Big problems arise if the vendor goes out of business or chooses to stop supporting a given platform (*e.g.*, Linux).

- VBA + MS Access under Microsoft Windows is an even stronger vendor-specific example in the desktop environment.

Embedded SQL: a Vendor-Independent Solution

- In embedded SQL, calls to SQL statements are embedded in the host programming language.
- Typically, such statements are tagged by a special marker, usually `EXEC SQL`.
- A preprocessor is invoked to convert the source program into a “pure” program in the host language.
- The `EXEC SQL` statements are converted to statements in the host language 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 of Embedded SQL

Embedded SQL has a number of distinct disadvantages:

Preprocessed: Debugging preprocessed programs is not a pleasant experience.

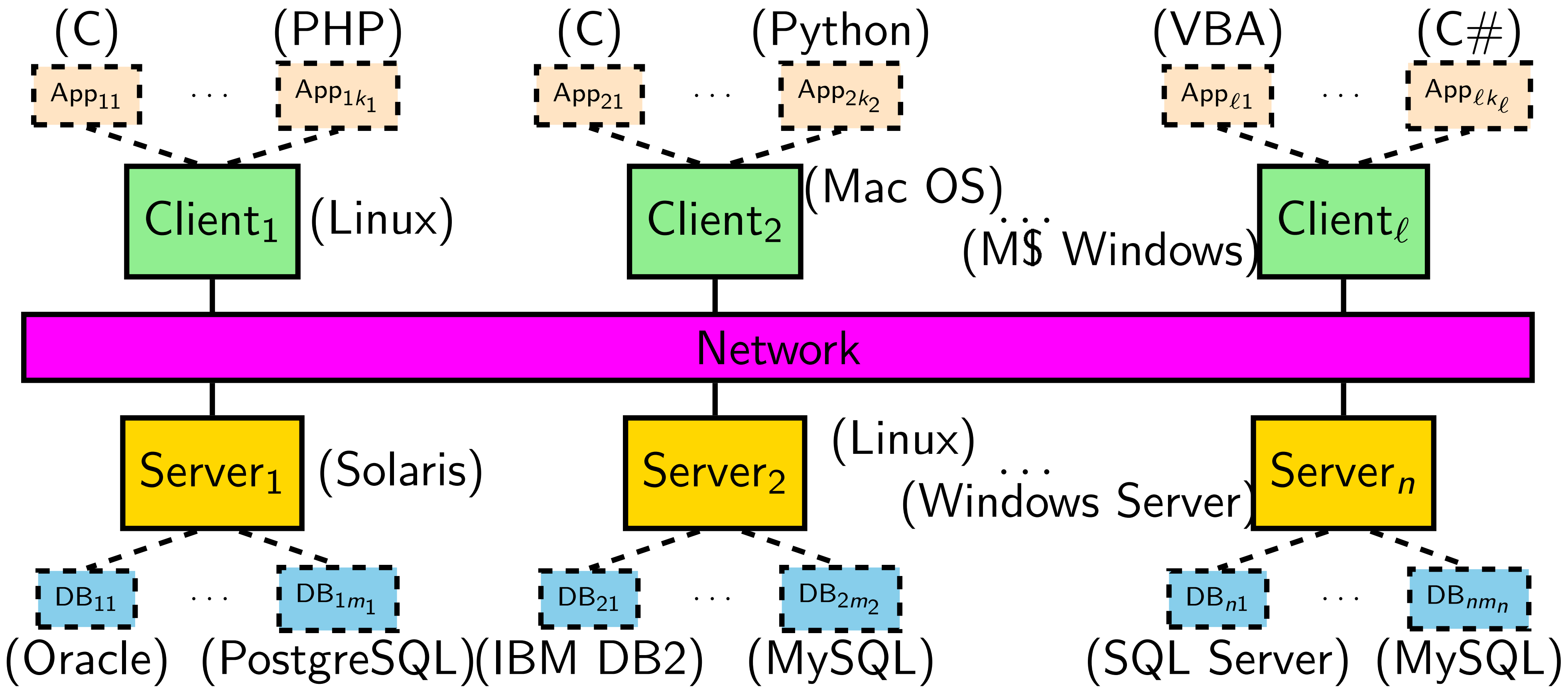
Program development environment: Because of the nature of preprocessed programs, it is not easy to provide support for the preprocessor directives within a programming environment.

Specificity: The preprocessor must be vendor specific, and at least in part, platform specific as well.

- Embedded SQL has been superseded in large part by CLI/ODBC.

A Closer Look at Interoperability

- A “real-world” situation might involve several DBMS, OSs, and PLs.
- The scenario might look something like this:



- In the ideal case, any application should be able to access any database using SQL ... subject only to limitations imposed by access rights.

The CLI/ODBC Solution to Interoperability

- There are two closely related specifications.

CLI (Call-Level Interface): An ISO/IEC standard developed in the early 1990s.

  - Defined only for C and COBOL.

ODBC (Open Data Base Connectivity): A specification based upon CLI.

  - Defined for many programming languages, including C, Python, Ruby, and PHP.

- in addition to ...

JDBC: An ODBC-like specification for Java.

- All of these solutions exhibit a large degree of interoperability.

☞ ODBC is not platform, OS, or DBMS specific.

OS: Unix, Linux, MacOS, MS Windows, IBM

DBMS: You name it.

- Interestingly, the major player which promoted ODBC was ... Microsoft!

Other Tools for Database Access via Programming Languages

- Approaches to interoperable DB access via PLs, other than ODBC, include:

Programming-language specific (so less interoperability): Some PLs have built-in features for accessing relational databases.

Example: *PHP* is a comprehensive language for server-side Web programming.

  - It has build-in command for DB access using SQL.
  - It also supports access via ODBC.

Programming-paradigm specific: Some approaches are focused upon a particular programming paradigm.

Example: *Active Record Pattern* is an approach for accessing relational databases which is particularly suited to the object-oriented paradigm.

  - *ColdFusion* is an open-source implementation of this idea.
  - Many programming languages, including *PHP*, *Ruby*, and *Java* have implementations available.

- These approaches are not considered in this course because they require knowledge of programming languages other than *C* and *Python*.

Use of ODBC

Myth: Nobody uses ODBC any more.

Reality: ODBC is still widely used.

- The other approaches complement it; they do not replace it.

Examples of ODBC usage:

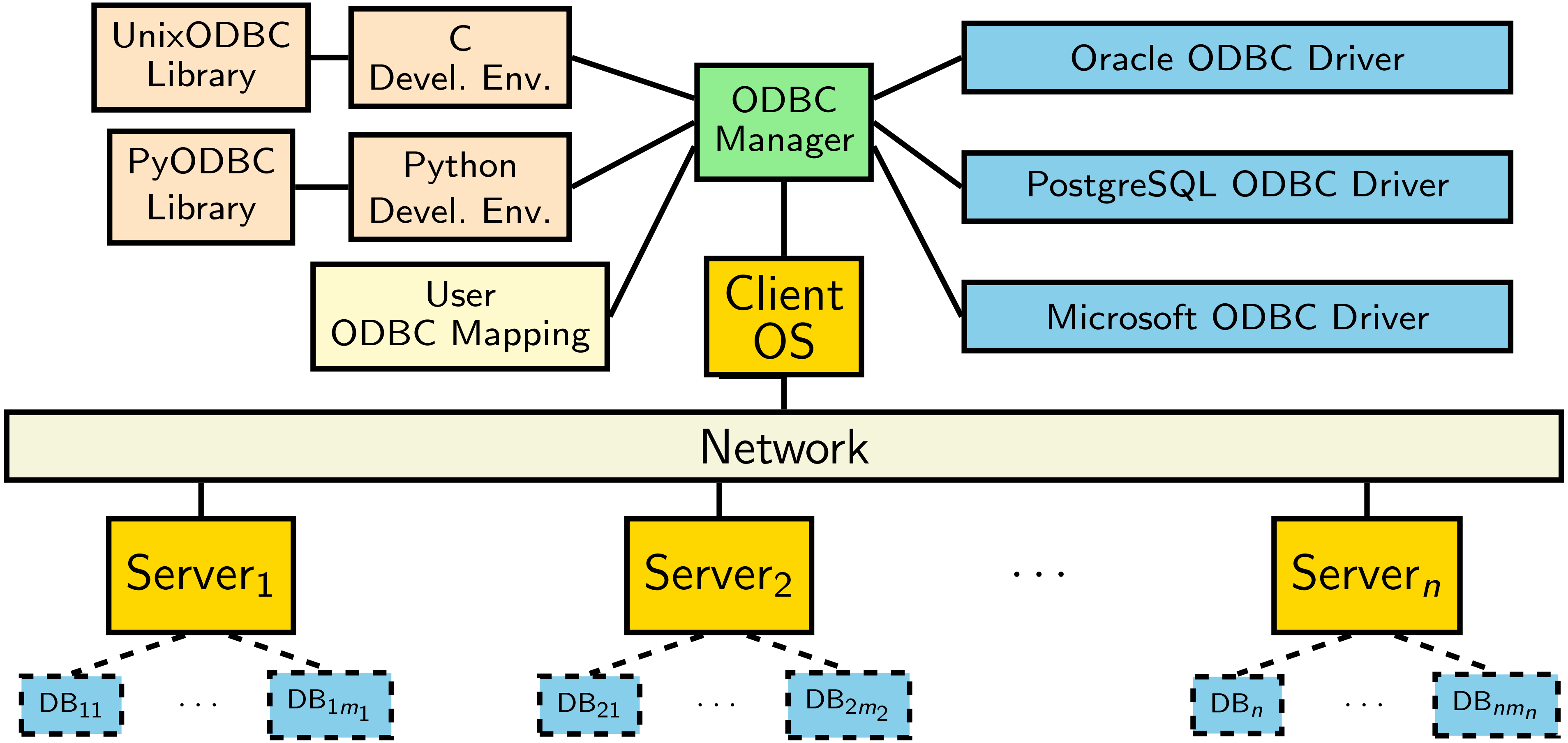
Virtuoso: The *Virtuoso Universal Server* provides access to a variety of types of databases, including but not limited to relational.

  - ODBC support is an integral part of Virtuoso.
  - Virtuoso with ODBC is used to support services in Linux.

Database access for research: It is widely used to for research applications which involve access to very large relational databases, particularly statistical databases.

- Even if you will use something else in your future employment, it is useful to learn the principles of interoperable DB access using ODBC.
- So, now to look at ODBC in some detail...

The Architecture of ODBC for a Single Client



Color code:

Operating system | OS-specific utility | DBMS-specific module

Development-environment-specific module | User configuration file

Using ODBC in the Linux Environment

- The main ideas are presented via a set of annotated programs in two languages:

  C: Using the standard *gcc* compiler.

  Python: Using the standard *python* interpreter.

- These slides provide only supporting information.

- The basic ODBC configuration file is the same for both *C* and *Python*.

- However, the details of usage are *very* different, since C and Python are very different languages.

  C: is a compiled language with explicitly declared data types.

  - This requires that there be declared type matching between C data types and SQL data types.

  Python: is an interpreted language with data typing upon assignment.

  - This implies that ODBC must also do run-time typing.

Data-Source Configuration: Linux + PostgreSQL

- 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 using ANSI encoding on the `postgres` server using Linux.
- Some of these parameters may be specified in the calling program as well.

```
# The ODBC data source names are are not used by PostgreSQL.
[ODBC Data Sources]
mydb1 = database1
# The name in square brackets is the ODBC DB name.
# It may be chosen arbitrarily.
[database1]
Description = PostgreSQL test database for Joe S. User
Driver      = /usr/lib/x86_64-linux-gnu/odbc/psqlodbca.so
# The name on the next line is the PostgreSQL DB name.
Database    = c5dv119_v14_jsu
# Username is the PostgreSQL user name and may be omitted with ident authentication.
Username    = c5dv119_v14_jsu
Servername  = postgres
Password    = "badidea"
```

Data-Source Configuration: ANSI and Unicode

- Actually, there are two drivers for PostgreSQL:

  ANSI: The ANSI driver psqlodbca.so.

  Unicode: The Unicode driver psqlodbcw.so.
- Information on which driver to use is contained in the slides which are specific to C and Python.
- An example configuration which uses the Unicode driver is shown below.

```
# The ODBC data source names are are not used by PostgreSQL.
[ODBC Data Sources]
mydb1 = database1U
# The name in square brackets is the ODBC DB name.
# It may be chosen arbitrarily.
[database1U]
Description = PostgreSQL test database for Joe S. User
Driver      = /usr/lib/x86_64-linux-gnu/odbc/psqlodbcw.so
# The name on the next line is the PostgreSQL DB name.
Database    = c5dv119_v14_jsu
# Username is the PostgreSQL user name and may be omitted with ident authentication.
Username    = c5dv119_v14_jsu
Servername  = postgres
Password    = "badidea"
```

Data-Source Configuration: Linux + MySQL

- To use MySQL instead of PostgreSQL, only the driver location and server name need be changed.
- There is one driver `libmyodbc.so` for both ANSI and Unicode.
- The keyword `Server`, not `Servername`, is used to identify the server.
- The keyword `User`, not `Username`, is used to identify the user.
- The `Password` may be given here as well, but for security reasons it is better to obtain it via prompting in the calling program.

```
# The ODBC data source names are are not used by MySQL.
[ODBC Data Sources]
mydb1 = database1M
# The name in square brackets is the ODBC DB name.
# It may be chosen arbitrarily.
[database1M]
Description = MySQL test database for Joe S. User
Driver      = /usr/lib/x86_64-linux-gnu/odbc/libmyodbc.so
# The name on the next line is the MySQL DB name.
Database    = v119v14jsu
# The name on the next line is the MySQL user name.
User        = v119v14jsu
Password    = "badidea"
Server      = mysql
```

Multiple Data-Source Configuration in One File

- Several data sources may be specified in the .odbc.ini file.

```
[ODBC Data Sources]
mydb1A = database1PA
mydb1M = database1M
mydb1U = database1PU
[database1PA]
Description = PostgreSQL test database with ANSI driver
Driver      = /usr/lib/x86_64-linux-gnu/odbc/psqlodbca.so
Database    = c5dv119_vt14_jsu
Username    = c5dv119_vt14_jsu
Servername  = postgres
[database1PU]
Description = PostgreSQL test database with Unicode driver
Driver      = /usr/lib/x86_64-linux-gnu/odbc/psqlodbcw.so
Database    = c5dv119_vt14_jsu
Username    = c5dv119_vt14_jsu
Servername  = postgres
[database1M]
Description = MySQL test database
Driver      = /usr/lib/x86_64-linux-gnu/odbc/libmyodbc.so
Database    = v119v14jsu
User        = v119v14jsu
Server      = mysql
```

A More Complete Specification of a Data Source

```
[ODBC Data Sources]
mydb3 = database3
[database3]
Description = PostgreSQL test database 1
Driver      = /usr/lib/x86_64-linux-gnu/odbc/psqlodbca.so
Database    = hegner1
Servername  = postgres
Port        = 5432
ReadOnly    = 0
Username    = hegner1
Password    = "badidea"
Trace       = No
TraceFile   = /tmp/odbc.log
```

- The fields not shown on the previous slide are optional.

MySQL: Use `User` instead of `Username` and `Server` instead of `Servername`.

- `Port` and `ReadOnly` need be specified only if they differ from the defaults.
- `Trace` and `Tracefile` need only be specified if tracing is desired.
- Other DB systems may use different keywords for some of these entries, and support additional entries as well.

ODBC Handles

- A *handle* is a numerical value which is associated with a certain object.
- File handles are familiar in systems 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.

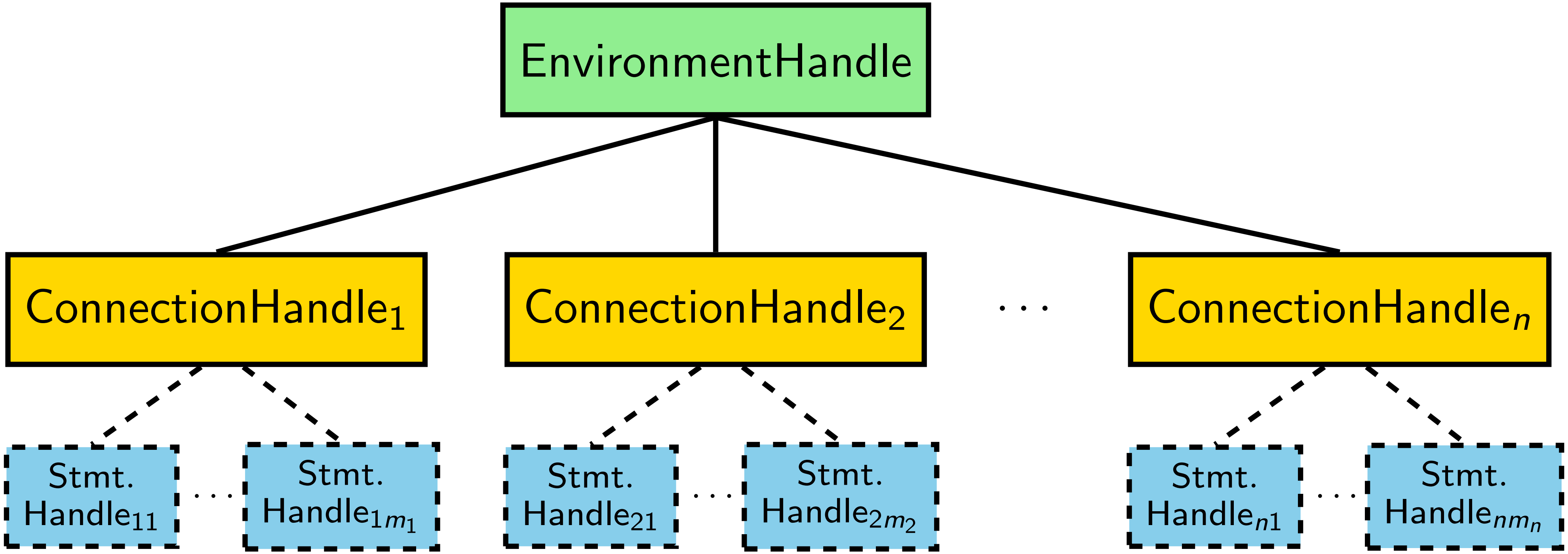
Connection handles: Just as there must be a file handle for every open file in an operating system, so too must there be a connection handle for each connection to an ODBC database.

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

Descriptor handles: Descriptors are metadata which describe formats associated with SQL statements.

- Descriptor handles will not be used in this course.

Visualization of the Hierarchy of ODBC Handles



- There is usually one environment handle per program.
- Distinct connection handles may refer to distinct databases.
- Each statement handle is for the database associated with its environment handle.
  - Multiple statement handles are useful for parallel execution of queries.