# Some Specifics of ODBC with Python

5DV119 — Introduction to Database Management

Umeå University
Department of Computing Science

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

# ODBC Documentation for Python

- ODBC with Python is relatively new.
- Loading the module `pyodbc` and then issuing the Python command `help(pyodbc)` will give reference documentation for that modules.
- There is a reference site with some basic information, but it is not nearly as comprehensive as that for ODBC with C.
- The slides and the example programs will provide enough to do the ODBC laboratory exercise.
- Python is a very easy language to learn.
- Even students who know C but not Python may find it easy to learn Python and do the laboratory exercise using it.
- Although these examples are intended to be generic, they have been tested only with the PostgreSQL (and in some cases MySQL) DBMS, using Debian Linux as the client-side operating system.
- For programming submissions in this course, the reference machine will be *salt*, which runs Debian 7 Linux.

# Dialects of Python and ODBC Drivers for PostgreSQL

- There are two dialects of Python, *Python2* and *Python3*.
- Although they are similar, neither is completely compatible with the other.
- The syntax of even simple, basic commands differs in a few cases.
- The example programs were written to be as dialect independent as possible, and most run with both.
  - In the few cases in which there are unavoidable differences, comments in the programs indicate clearly what to do.

- The following is a crucial difference for use with PostgreSQL under the Debian 7 Linux systems of the department (`SQL_ANSI` server encoding):
  - Python2 requires the ANSI ODBC driver `psqlodbca.so` and will not work with the Unicode driver.
  - Python3 requires the Unicode ODBC driver `psqlodbcw.so` and will not work with the ANSI driver.
- For installations which use UTF8 server encoding (*e.g.*, newer versions of Ubuntu), `psqlodbcw.so` may be the driver which works for Python3.
- For MySQL, the ODBC driver `libmyodbc.so` works for both dialects.

# Running a Python Program with ODBC Calls

- Python is interpreted, not compiled.
- It is necessary to import the `pyodbc` module:

```
import pyodbc
```

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

# Some Basics of ODBC Calls in Python

Imports: To use ODBC API calls, import the module `pyodbc`.

Identifiers:

- There is no special convention for identifier names.

API calls:

- To see documentation for the calls, import `pyodbc` and then run `help(pyodbc)` as a Python command.
- The returned value of most calls is an identifier which is not particularly useful.
- Trapping errors must be done by other means, as shown in the examples.

# Using ODBC with Python

- The library `pyodbc` must be imported using `import pyodbc`.
- Complex type association is not necessary with Python because the interpreter determines the correct types dynamically.
- The API is not as rich as it is with C, and it is not clear how useful it is for true production-level programming.
- The details are best illustrated via example programs.

# Declaration and Use of ODBC Handles in Python

- Handles need not be declared, since data objects are not declared in Python.
- There are no explicit environment handles.
- A connection handle is allocated via a call to `pyodbc.connect()`.
- The rôle of a statement handle is assumed (more or less) by a `cursor`.
  - Use a call of the form `<connection_handle>.cursor()`.
- These are all illustrated in the example programs.

# Commit Mode

- The default commit mode is *manual*.
- This means that updates are not committed automatically.
- The commit mode may be set to *autocommit* when the ODBC connection is opened.
- Two example programs illustrate the details.