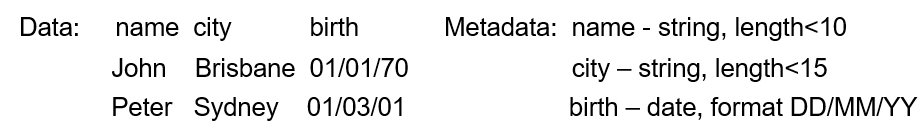
**Week 6 Databases and Excel**

**Databases**

**What is a database?**

A collection of logically related data, including metadata (data about data).

**Database management systems**

Piece of software designed to store and manage a database. Providing a way to interact with the database: It enables users to create, maintain, and query the database. E.g. Oracle, MySQL, sqlite

**SQL – structured query language**

Sql has 3 major components:

* A data definition language for defining database structure
* A data manipulation language for retrieving and updating data
* A data control language for data control, granting of rights etc.

**Data types:**

* Int (size) – whole number
* Decimal(p,s) - exact numerical, fixed precision
* Float (p) - approximate numerical
* Char (n) – fixed length string
* Varchar(n) – variable length string
* Boolean – TRUE or FALSE
* Date
* Time

The most common use of SQL is as a (DML) Data Manipulation Language.

* Databases are created once, structurally modified rarely, but are searched and updated regularly.

DML features include (but are not limited to)

* Inserting data (INSERT INTO <table-name>)
* Displaying data (SELECT FROM <table-name>)
* Updating a record (UPDATE <table-name>)

**SQL Queries**

SQL commands are case insensitive:

* Same: SELECT, Select, select
* Same: Product, product

Values are not:

* Different: ‘Brisbane’, ‘brisbane’

Use single quotes for constants:

* ‘bob’ - yes
* “bob” - no

**Conditions**

Condition is an expression that can be evaluated to TRUE or FALSE. Only rows satisfying the condition will be chosen. Condition can be a simple comparison or a compound expression.

* Basic comparison operators:

=, >, <, <=, >=, <> (!=) can be applied to numbers, strings and dates

* Basic comparisons can be compounded by AND, OR and NOT
  + Eg: prodNo=100 and ordDate='01-jan-2003‘

**Aggregate functions**

Aggregate functions are used to compute against returned columns of data from your SELECT statement. They basically summarize the results of a query.

* COUNT - returns the number of selected values
* SUM - returns the sum of selected (numeric) values
* AVG - returns the average of selected (numeric) values
* MIN - returns the minimum of selected values
* MAX - returns the maximum of selected values

|  |  |
| --- | --- |
| **Table creation** |  |
| Create table <Table-name> (  Column-1 datatype [not null], Column-n datatype [not null]  );  Create table <Table-name> (  Column-1 datatype [not null], Column-n datatype [not null],  CONSTRAINT constraint-name CONSTRAINT\_TYPE (column-name1, ….., column-namen)  ); | create table Person (  ID BigInt auto-increment,  name string [not null],  age int [not null]  );  Create table Person (  ID BigInt auto-increment,  name string [not null],  age int [not null],  companyID int [not null],  PRIMARY KEY (ID), FOREIGN KEY (companyID) REFERENCES Company(ID)  ); |
| **Inserting data** |  |
| INSERT INTO <table-name> [(attributes)] VALUES (value\_list); | INSERT INTO Person  VALUES  (“Ben Freeman”, 34, 001),  (“Meg Smith”, 27, 013); |
| **Displaying data** |  |
| SELECT <attributes> FROM <table-name>;  SELECT <attributes> FROM <table-name/s> WHERE <conditions>;  SELECT <attributes> FROM <table-name> WHERE <attribute> LIKE <value>;  SELECT <attributes> FROM <table-name> WHERE <attribute> between <value> and <value>;  SELECT <attributes> FROM <table-name> WHERE <attribute> in (list); | SELECT \* FROM Person;  SELECT name, companyID FROM Person WHERE companyID=002;  SELECT \* FROM Person WHERE name LIKE ‘%Ben%’;  SELECT \* FROM Person WHERE age between 25 and 35;  SELECT name FROM Person WHERE companyID IN (001, 012, 030); |
| **Distinct data values** |  |
| SELECT DISTINCT <attribute> FROM <table-name>; | SELECT DISTINCT companyID FROM Person; |
| **Order by** |  |
| SELECT <attribute> FROM <table-name> ORDER BY <attribute> <order-type> ; | SELECT \* FROM Person ORDER BY CompanyID asc; |
| **Update table** |  |
| UPDATE <table-name> SET <column1 = value1, column2 = value2, ….> WHERE <condition>; | UPDATE Person SET age=35, companyID=013 WHERE ID=1; |

**Python for SQL**

**Sqlite & DB-API**

To use sqlite in python we import the module at the top of the python file ‘import sqlite3’. To connect to a database & create or use an existing database file we use ‘connection = sqlite3.connect(“databasename.db”) cursor = connection.cursor()’. Within python all access to the database is via the ‘cursor’ object. From here we can use SQL queries and commands to access the database.

Simple database access through python

* import sqlite3  
    
  connection = sqlite3.connect(“company.db”)  
  cursor = connection.cursor()  
    
  sql = “”” CREATE TABLE Person (

ID BigInt auto-increment,

name string [not null],

age int [not null],

companyID int [not null],

PRIMARY KEY (ID),  
FOREIGN KEY (companyID) REFERENCES Company(ID)

);”””  
   
 cursor.execute(sql)  
  
 sql2 = “”” INSERT INTO Person (  
 name, age, companyID)

VALUES (“Ben Freeman”, 35, 001

);”””

cursor.execute(sql2)

connection.commit()

connection.close()

**Python and Excel**

The openpyxl module allows your Python programs to read and modify Excel spreadsheet files.

The following code opens an Excel workbook and accesses worksheet by name.

import openpyxl

wb = openpyxl.load\_workbook('Stocks.xlsx')

sheet = wb.get\_sheet\_by\_name('Microsoft')

anotherSheet = wb.active

querying the above python program:

print(wb.get\_sheet\_names())

# will print ['Google', 'IBM', 'Microsoft']

sheet

# will echo <Worksheet "Microsoft">

type(sheet)

# will echo <class 'openpyxl.worksheet.worksheet.Worksheet'>

sheet.title

# will echo ‘Microsoft'

anotherSheet

# will echo <Worksheet “Microsoft">

The active member variable of a Workbook object gives the workbook’s active sheet.

**Accessing worksheet cells**   
when a worksheet is created in memory, it contains no ‘cells’, these are created when the worksheet is first ‘accessed’. Scrolling through cells instead of accessing them directly will create then all in memory, even if they’re not assigned a value.

* Something like:

>>> **for** i **in** range(1,101):

... **for** j **in** range(1,101):

... ws**.**cell(row**=**i, column**=**j)

will create 100x100 cells in memory, for nothing.

* Given a Worksheet object, the Cell object can be accessed:
  + >>> import openpyxl

>>> workbook = openpyxl.load\_workbook(‘Stocks.xlsx')

>>> sheet = workbook.get\_sheet\_by\_name(‘IBM')

>>> sheet['A1']

<Cell IBM.A1>

>>> sheet['A1'].value

datetime.datetime(2010, 1, 4, 0, 0)

>>> c = sheet['C1']

>>> c.value

131.179993

>>> sheet.cell(row=1, column=3, value=11)

<Cell IBM.C1>

>>> print('Row ' + str(c.row) + ', Column ' + c.column + ' is', c.value)

Row 1, Column C is 131.179993

>>> print('Cell ' + c.coordinate + ' is', c.value)

Cell C1 is 131.179993

>>> sheet['B10'].value

'IBM'

* we can also get a cell using the sheet’s cell() method and passing integers for its row and column keyword arguments. The first row or column integer is 1, not 0.
  + Cell B1: sheet.cell(row=1, column=2)
  + Value of cell C1: sheet.cell(row=1, column=3).value
  + Range of values in a column:

for i in range(1, 8, 2):

print(i, sheet.cell(row=i, column=3).value)

* we can determine the size of a sheet using the max\_row and max\_column variables.
  + How many rows: sheet.max\_row
  + How many columns: sheet.max\_column

**Accessing rows, columns & cells**

We can ‘slice’ worksheet objects to get all the cell objects in a row, colums or rectangular area of a sheet.

* >>> import openpyxl

>>> wb = openpyxl.load\_workbook('Stocks.xlsx')

>>> sheet = wb.get\_sheet\_by\_name('IBM')

>>> tuple(sheet['A1':'C3'])

((<Cell IBM.A1>, <Cell IBM.B1>, <Cell IBM.C1>),

(<Cell IBM.A2>, <Cell IBM.B2>, <Cell IBM.C2>),

(<Cell IBM.A3>, <Cell IBM.B3>, <Cell IBM.C3>))

To access values of cells in a particular row or column, use the rows and columns attribute.

* Column access:   
  columns = tuple(sheet.columns)  
  print(columns[1])  
    
  for cellObject in columns:

Print(cellObject[1].value)

* Row access:   
  rows = tuple(sheet.rows)  
  print(rows[1])  
    
  for cellObj in rows:

Print(cellObj[1].value)

* Iterate through the worksheet (print out row items in row 1-2 for column 1-3)

For row in sheet.iter\_rows(min\_row=1, max\_col=3, max\_row=2):

For cell in row:

Print(cell)

The generator object contains three tuples: one for each row, from the top of the desired area to the bottom.

* for rowOfCellObjects in sheet['A1':'C3']:

for cellObj in rowOfCellObjects:

print(cellObj.coordinate, cellObj.value)

print('--- END OF ROW ---')

A1 2010-01-04 00:00:00

B1 IBM

C1 131.179993

--- END OF ROW ---

A2 2010-01-05 00:00:00

B2 IBM

C2 131.679993

--- END OF ROW ---

A3 2010-01-06 00:00:00

B3 IBM

C3 130.679993

--- END OF ROW ---

**Creating workbooks**

Using the openpyxl.Workbook() function, we can create new, blank workbook objects.

* workbook = openpyxl.Workbook()  
  sheet = workbook.active  
  sheet.title = “sheet-title”  
  workbook.save(“workbook-name.xlsx”)

Whenever you edit a spreadsheet you’ve loaded from a file, you should always save the new, edited spreadsheet to a different filename than the original. That way, you’ll still have the original spreadsheet file to work with in case a bug in your code caused the new, saved file to have incorrect or corrupt data.

**Creating and removing worksheets**

Creating: create\_sheet()   
Removing: remove\_sheet()

Editing and copying worksheets  
worksheets are automatically given a name at creation based on it’s creation number (sheet1, sheet2 etc) this title can be changed at any-time using the .title property

* workbook.title = “title”

worksheet title tabs can be changed color using the .tabColor property

* workbook.sheet\_properties.tabColor=”RRGGBB”

worksheet copies can be made within a single workbook using the .copy\_worksheet() function, worksheets cannot be copied between workbooks.

* source = workbook.active  
  target = workbook.copy\_worksheet(source)

**Writing to cells**

cells can be written to much like value keys in a dictionary.

* Workbook = openpyxl.Workbook()  
  sheet = workbook.get-sheet\_by\_name(‘Sheet’)  
  sheet[‘A1’] = ‘value’

**Formulas**

Formulas, which begin with an equal sign, can configure cells to contain values calculated from other cells. The following code writes the value of 500 to cell A3.

* Workbook = openpyxl.Workbook()  
  sheet = workbook.active  
  sheet[‘A1’]=200  
  sheet[‘A2’]=300  
  sheet[‘A3’]=’=Sum(A1:A2)’  
  workbook.save(‘writeforumla.xlsx’)

Formulas in excel are very unreadable and complicated, this is much easier done through python coding.

* Create a workbook: workbook.py
* Using number formats: number\_format.py

**Chart generation in python**

Python supports chart types of: area, bar, column, bubble, line, scatter plots, pie, doughnut, radar, stock, surface.

**Working with tubular data**

This can be done through excel aswell as the use of the pandas module.