Pruebas unitarias

Test unitarios realizados sobre las operaciones básicas de insertado y borrado, además también sobre la función generadora de informes, comprobando si ha generado dicho informe posterior a su ejecución

```
def test_insert(self):
    app = QApplication.instance()
    if app == None:
        app = QApplication([])
    apli = MainWindow()
    window = QWidget()

    before=apli.modelo.rowCount()
    apli.nueva()
    after=apli.modelo.rowCount()
    self.assertEqual(before+1,after)
    app.quit()
    window.close()
```

```
def test_borrar(self):
    app = QApplication.instance()
    if app == None:
        app = QApplication([])
    apli = MainWindow()
    window = QWidget()

    before=apli.modelo.rowCount()
    apli.borrar()
    after=apli.modelo.rowCount()
    self.assertEqual(before,after)
    app.quit()
    window.close()
```

```
def test_informe(self):
    app = QApplication.instance()
    if app == None:
        app = QApplication([])
    apli = MainWindow()
    window = QWidget()

    self.nombre = "juan"
    self.Apellidos = "Yuan"
    self.cursoId = 5
    self.media1 = 5.25
    self.media2 = 9.99
    self.media3 = 2.56
    apli.generate(self.nombre,self.Apellidos,self.cursoId)

    try:
        file = open('result.pdf')
        print(file) # File handler
        self.assertEqual(os.path.exists('result.pdf'),True)
        file.close()
        app.quit()
        window.close()
    except FileNotFoundError:
        print('Sorry the file we\'re looking for doesn\'t exist')
        exit()
```

pruebas de integración de los distintos elementos

Luego de añadir El Boton Animado a nuestra aplicación, sus test unitarios seguirían funcionando igual que antes

pruebas de regresión

Las pruebas unitarias realizadas antes seguirían teniendo el mismo resultado luego de añadir el botón

pruebas de volumen y estrés

Insertaríamos 10.000 nuevas tuplas para comprobar su correcto funcionamiento en situaciones de estrés

```
class Test():

    def Estres():
        app = QApplication.instance()
        if app == None:
            app = QApplication([])
        apli = MainWindow()
        window = QWidget()
        for i in range(10000):
            apli.nueva()

        app.quit()
        window.close()

if __name__ == '__main__':
    Test.Estres()
```

pruebas de seguridad

A la hora de hacer la prueba de Seguridad hemos observado que nuestra base de datos no esta cifrada, ni tiene protección de ningún tipo, y los datos que guardamos son un tanto sensibles ya que contiene información personal de los alumnos.

Para solucionar esto he propuesto utilizar la herramienta de Python **pysqlitecipher** que nos permite crear un objeto sobre el que realizaremos las operaciones actuando como modelo, y a su vez cifraría la base de datos.

Como implementar esta herramienta:

Instalamos su libreria

```
-pip install pysqlitecipher==0.11
```

La importamos en nuestro proyecto

```
-from pysqlitecipher import sqlitewrapper
```

Creamos el objeto/modelo indicándole la ruta de la DB y su password

```
-obj = sqlitewrapper.SqliteCipher(dataBasePath="pysqlitecipher.db"
, checkSameThread=False , password=None)
```

Sobre el objeto:

Para crear nuevas tablas

```
-obj.createTable(tableName , colList , makeSecure=False
,commit=True)
```

En la que colList estaría compuesta por

```
colList = [
[colname , datatype] ,
[colname2 , datatype] ,
]
```

Insertar Datos

```
-obj.insertIntoTable(tableName , insertList , commit = True)
```

Insertlist deberá tener el mismo formato que colList pero con sus respectivos datos ya rellenos tal que:

```
colList = [
[colname , colname2]
]
```

Para obtener datos de una tabla:

```
- obj.getDataFromTable(tableName , raiseConversionError = True ,
omitID = False)
```

Para borrar datos de una tabla:

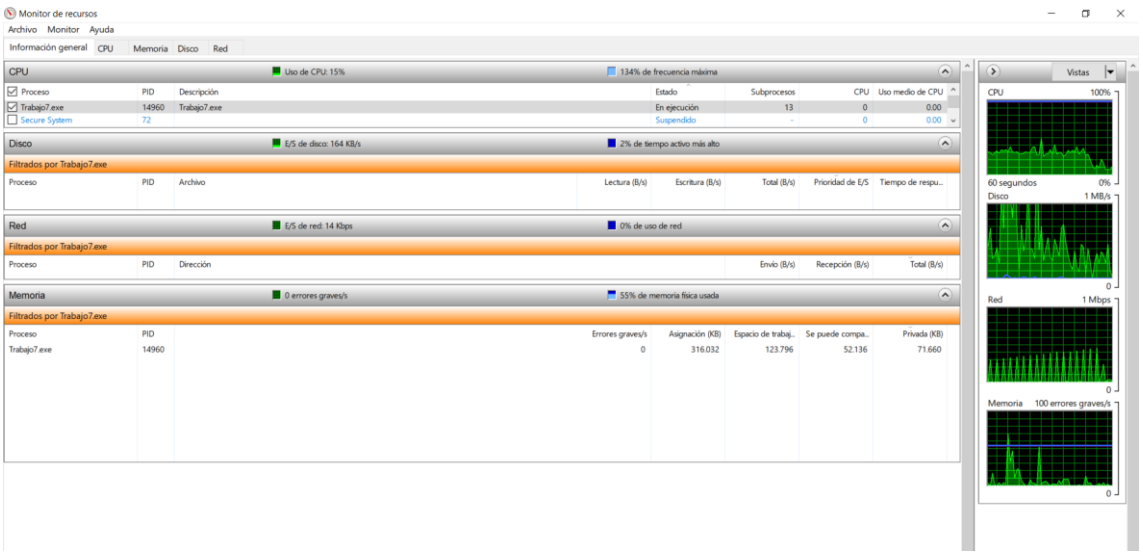
```
obj.deleteDataInTable(tableName , iDValue , commit = True ,
raiseError = True , updateId = True)
```

Para actualizarla:

```
obj.updateInTable(tableName , iDValue , colName , colValue ,
commit = True , raiseError = True)
```

pruebas de uso de recursos por parte de la aplicación

Los recursos utilizados por la aplicación cuando se esta ejecutando son mínimos y tendrían poco impacto en el sistema



Nombre	Estado	15% CPU	55% Memoria	0% Disco	0% Red	5% GPU	Motor de GPU
Oracle RDBMS Kernel Executable		0%	458,1 MB	0,1 MB/s	0 Mbps	0%	
Google Chrome (7)		0,2%	306,0 MB	0 MB/s	0 Mbps	0%	GPU 1 - 3D
Visual Studio Code (13)		0,1%	254,7 MB	0 MB/s	0 Mbps	0,1%	GPU 0 - 3D
Antimalware Service Executable		0,3%	179,5 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0%	155,5 MB	0 MB/s	0 Mbps	0%	
Spotify (6)		0,3%	153,8 MB	0,1 MB/s	0 Mbps	0%	
Discord (32 bits) (4)		0,4%	129,2 MB	0 MB/s	0 Mbps	0%	GPU 0 - 3D
Microsoft Word (2)		0,4%	85,1 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0%	71,1 MB	0 MB/s	0 Mbps	0%	
Trabajo7.exe		0%	70,0 MB	0 MB/s	0 Mbps	0%	
MainWindow							