Project overview:

Sciospec is currently developing a measurement system that determines the water content of medical drug tablet using a method based on electric measurements. A voltage pulse signal is applied to the tablets and the resulting current is measured. Voltage and current are sampled and used to calculate the water content of the tablets.

Task overview:

The main goal of this task is to implement a class that handles the storage of the measurements into a SQLite Database while satisfying requirements related to the data integrity. The measurements are fetched from the device as raw voltage and current samples and then processed. Therefore, there are two kinds of datasets: raw datasets and processed datasets. Raw datasets should be identifiable by a UUID. A processed dataset is always derived from a raw dataset and should always refer to it. Further, it is possible to derive different versions of processed datasets from the same raw dataset. The version of the processed dataset should be tracked. Both kinds of datasets are handled throughout the code in a data structure called a Sequence Frame, which is a collection of Sequences (Indexed array of values) together with metadata, timestamps and other attributes (Please find an example of a Sequence Frame and its usage at the end of the document).

Task:

You are asked to create a class called *MeasurementDatabase* that handles storing, querying, updating and versioning the Sequence Frames in a SQLLite Database.
This class should allow :

- Creating the database and database table
- Merging existing database tables with database tables that were created other computers
- Inserting Sequence Frames that contain raw data, each Sequence Frame that contains raw data should be identifiable by a UUID
- Inserting Sequence Frames that contain processed data, each Sequence Frame that contains processed data should refer to the raw dataset it was derived from and should have a version number to distinguish it from other processed Sequence Frames that were derived from the same raw dataset
- Fetching Sequence frames by name, group, timestamp, metadata, UUID and version
- Fetching available raw Sequence Frames
- Fetching available versions of processed Sequence Frames by the UUID of the raw Sequence Frame they were derived from

In general, the requirements that we must respect are that:

- Data should have a unique ID
- Data version should be trackable
- Data should have relevant metadata
- Data should have a timestamp
- Data should be saved local
- Data should be saved in an SQL database
- Data should be saved as E-Record
- Data should not be changeable, a change is a new version of the data

In order to store this data, some of the classes that Sciospec already implemented must be used : *SQLiteDatabase* and *SequenceFrame*.

SQLiteDatabase:

This class was implemented to abstract the database queries, so it would be easier if we migrate from an SQLite database to another type. You should always keep in mind that the databases will be migrated to Microsoft SQL in the future.

It contains mainly class methods that abstract the basic queries of SQL like:

```
def create_table(self, table_name, *args): ...


def insert(self, table_name, value_list, *args): ...


def update(self, table_name, new_value_dict, where=None, where_and=None,
where_or=None): ...


def remove(self, table_name, where=None, where_and=None, where_or=None):
...


def get_certain_rows_by_where(self, table_name, row_list=None, where=None,
where_or=None, where_and=None): ...
```

Please also keep in mind that it’s totally prohibited to use queries like `cursor.execute(‟INSERT INTO … ‟)` throughout the code, everything should be abstracted in the SQLiteDatabase class.

SequenceFrame:

This class is used to store raw and processed datasets. There is a Sequence and a Sequence Frame class. Each column of a Sequence Frame is a Sequence.

A Sequence is based on the Pandas Series class. It contains metadata, timestamps and other attributes (name, group, abscissa_type, ordinate_type, timestamp, metadata).
Please find below a part of the signature of the Sequence class

```
class Sequence:
      def __init__(self, name, group, abscissa_type, ordinate_type):…
      def add_value_pair(self, abscissa_value, ordinate_value): …
      def add_metadata(self, key, metadata): …
      …
```

These Sequences are simply added to a Sequence Frame, that concatenates the values, the metadata and other attribute into a single structure. A Sequence Frame is based on the Pandas DataFrame class.

This is a part of the signature of the SequenceFrame class :

```
class SequenceFrame:
def __init__(self, name, abscissa_type):
      self.name = name
      self.abscissa_type = abscissa_type
      self.data_frame = pd.DataFrame()
      self.sequence_index_name_dict = {}
      self.sequence_index_group_dict = {}
      self.sequence_index_timestamp_dict = {}
      self.sequence_index_ordinate_type_dict = {}
      self.sequence_index_meta_data_dict = {}
def add_sequences(self, sequence_list): ...
def set_value(self, value, row, names, groups, timestamps,
ordinate_types): ...
def get_data_frame(self, column_names): ...
def get_column_groups(self): ...
def get_unique_column_groups(self): …
def get_metadata(self, key): …
def get_rows(self, names, groups, timestamps, ordinate_types,
index): …
def filter(self, names, groups, timestamps, ordinate_types,
row_indices): …
def apply(self, function, names, groups, timestamps,
ordinate_types): …
def to_csv(self, path): …
```

Please find below an example of creating a SequenceFrame :

```
attribute_key_list = [["2022-03-04 12:22:58", "Method 1"], ["2022-
03-03 21:16:37", "Method 2"],
["2022-03-04 12:30:58", "Method 1"], ["2022-03-04 09:51:45", "Method
2"],
["2022-03-03 23:01:33", "Method 2"]]
sample_id_list = ["0000-0000-0000-0000", "0000-0000-0000-0001",
"0000-0000-0000-0002", "0000-0000-0000-0003",
"0000-0000-0000-0004"]
batch_list = ["0000-0000-0000-0000", "0000-0000-0000-0001", "0000-
0000-0000-0002", "0000-0000-0000-0003",
"0000-0000-0000-0004"]
sequence_list = []
for measurement_index in range(5):
attribute_key = attribute_key_list[measurement_index]
```

```
sample_id = sample_id_list[measurement_index]

batch = batch_list[measurement_index]

sample_count = 10

comment = "This is a comment"

sequence_water_content = Sequence(name="Water Content",
group=attribute_key, abscissa_type="Number")

sequence_water_content.add_metadata("sample_id", sample_id)

sequence_water_content.add_metadata("sample_count", sample_count)

sequence_water_content.add_metadata("batch", batch)

sequence_water_content.add_metadata("comment", comment)

sequence_water_content.add_metadata("UUID", str(uuid.uuid4()))


sequence_uncertainty = Sequence(name="Uncertainty",
group=attribute_key, abscissa_type="Number")

sequence_uncertainty.add_metadata("sample_id", sample_id)

sequence_uncertainty.add_metadata("sample_count", sample_count)

sequence_uncertainty.add_metadata("batch", batch)

sequence_uncertainty.add_metadata("comment", comment)


sequence_uncertainty.add_metadata("UUID", str(uuid.uuid4()))


sequence_use_average_content = Sequence(name="Use Average",
group=attribute_key, abscissa_type="Number")

sequence_use_average_content.add_metadata("sample_id", sample_id)

sequence_use_average_content.add_metadata("sample_count",
sample_count)

sequence_use_average_content.add_metadata("batch", batch)

sequence_use_average_content.add_metadata("comment", comment)

sequence_use_average_content.add_metadata("UUID", str(uuid.uuid4()))


for data_index in range(1, 11):

sequence_water_content.add_value_pair(data_index,
np.abs(np.random.randn()))

sequence_uncertainty.add_value_pair(data_index,
np.abs(np.random.randn()))
```

```
sequence_use_average_content.add_value_pair(data_index, False)

sequence_list.append(sequence_water_content)

sequence_list.append(sequence_uncertainty)

sequence_list.append(sequence_use_average_content)

sequence_frame = SequenceFrame(name="seq0", abscissa_type="Number")


sequence_frame.add_sequences(sequence_list)
```