```
class Chapter:
    def __init__(self, id, title, pages, book_id):
            self.id = id
            self.title = title
            self.pages = pages
            self.book_id = book_id

class Book:
      def __init__(self, id, name):
            self.id = id
            self.name = name

class ChapterBook:
      def __init__(self, book_id, chapter_id):
            self.book_id = book_id
            self.chapter_id = chapter_id

# данные
books = [
    Book(1, "Boeing 737 flight(self-education)"),
    Book(2, "МЦТ США. История"),
    Book(3, "Апчихба и другие мемы"),

    Book(11, "Boeing 737 flight(self-education). Редакция 2"),
    Book(22, "МЦТ США. История (переписанная)"),
    Book(33, "Апчихба и другие мемы. Редакция 2"),
]

chapters = [
    Chapter(1,"Глава 1", 10, 1),
    Chapter(2,"Глава 2", 91, 1),
    Chapter(3,"Глава 33", 50, 2),
    Chapter(4,"Глава 5", 20, 3),
```

```
    Chapter(5,"Глава 69", 30, 3),


    Chapter(1,"Глава 1", 10, 11),
    Chapter(2,"Глава 2", 91, 11),
    Chapter(3,"Глава 33", 50, 22),
    Chapter(4,"Глава 5", 20, 33),
    Chapter(5,"Глава 69", 30, 33),
 ]


ch_b =[
    ChapterBook(1,1),
    ChapterBook(2,2),
    ChapterBook(3,3),
    ChapterBook(3,4),
    ChapterBook(3,5),



    ChapterBook(11,1),
    ChapterBook(22,2),
    ChapterBook(33,3),
    ChapterBook(33,4),
    ChapterBook(33,5),
]


def unit_test(**kwargs):
    def inner_decorator(func):
        def wrapped(*args):
            assert 'input' in kwargs and 'output' in kwargs
            for i in range(len(kwargs['input'])):
                response = func(*kwargs['input'][i])
                assert response == kwargs['output'][i], f'Incorrect
return of function {func.__name__}({kwargs["input"][i]}) - {response}.
Must be {kwargs["output"][i]}'
```

```
            print(f"Test function {func.__name__} complete. Not found
errors")
            return func(*args)
        return wrapped
    return inner_decorator


def main():
     # Запрос 1: Список всех книг, у которых название начинается с буквы
"A", и список их глав
    @unit_test(input=[], output=[{
        "Апчихба и другие мемы": ["Глава 5", "Глава 69"],
        "Апчихба и другие мемы. Редакция 2": ["Глава 5", "Глава 69"],
    }])
    def query_books_starting_with_a():
        result = {}
        for book in books:
            if book.name.startswith('A'):
                chapters_in_book = [ch.title for ch in chapters if
ch.book_id == book.id]
                result[book.name] = chapters_in_book
        return result


    # Запрос 2: Список книг с максимальной длиной глав
    @unit_test(input=[], output=[{
        "Boeing 737 flight(self-education)": ("Глава 2", 91),
        "Boeing 737 flight(self-education). Редакция 2": ("Глава 2",
91),
        "МЦТ США. История": ("Глава 33", 50),
        "МЦТ США. История (переписанная)": ("Глава 33", 50),
        "Апчихба и другие мемы": ("Глава 69", 30),
        "Апчихба и другие мемы. Редакция 2": ("Глава 69", 30),
    }])
    def query_books_with_max_chapter_length():
        result = {}
        for book in books:
```

```
            book_chapters = [ch for ch in chapters if ch.book_id ==
book.id]
            if book_chapters:
                max_chapter = max(book_chapters, key=lambda ch:
ch.pages)
                result[book.name] = (max_chapter.title,
max_chapter.pages)
        return dict(sorted(result.items(), key=lambda x: x[1][1],
reverse=True))


     # Соединение данных многие-ко-многим
    many_to_many_temp = [(b.name, cb.book_id, cb.chapter_id)
        for b in books
        for cb in ch_b
        if b.id==cb.book_id]


    many_to_many = [(c.title, c.pages, book_name)
        for book_name, book_id, chapter_id in many_to_many_temp
        for c in chapters if c.id==chapter_id]



    # Запрос 3: Список всех связанных глав и книг, отсортированных по
книгам
    @unit_test(input =[], output = [{
        "Boeing 737 flight(self-education)": [("Глава 1", 10, "Boeing
737 flight(self-education)"),
                                              ("Глава 2", 91, "Boeing 737
flight(self-education)")],
        "МЦТ США. История": [("Глава 33", 50, "МЦТ США. История")],
        "Апчихба и другие мемы": [("Глава 5", 20, "Апчихба и другие
мемы"),
                                  ("Глава 69", 30, "Апчихба и другие
мемы")],
        "Boeing 737 flight(self-education). Редакция 2": [("Глава 1",
10, "Boeing 737 flight(self-education). Редакция 2"),
                                                          ("Глава 2", 91,
"Boeing 737 flight(self-education). Редакция 2")],
```

```
        "МЦТ США. История (переписанная)": [("Глава 33", 50, "МЦТ США.
История (переписанная)")],
        "Апчихба и другие мемы. Редакция 2": [("Глава 5", 20, "Апчихба и
другие мемы. Редакция 2"),
                                              ("Глава 69", 30, "Апчихба и
другие мемы. Редакция 2")],
    }])
    def query_all_related_chapters_and_books():
        result = {}
        for book in books:
            related_chapters = list(filter(lambda i: i[2]==book.name,
many_to_many))
            result[book.name] = related_chapters
        return result


    # Выполнение запросов
    print("\n3апрос 1:\n", query_books_starting_with_a())
    print("\n3апрос 2:\n", query_books_with_max_chapter_length())
    print("\n3апрос 3:\n", query_all_related_chapters_and_books())


if __name__ == '__main__':
    main()
```

Пример работы программы:

```
Test function query_books_starting_with_a complete. Not found errors


Запрос 1:
 {'Апчихба и другие мемы': ['Глава 5', 'Глава 69'], 'Апчихба и другие
мемы. Редакция 2': ['Глава 5', 'Глава 69']}
Test function query_books_with_max_chapter_length complete. Not found
errors


Запрос 2:
 {'Boeing 737 flight(self-education)': ('Глава 2', 91), 'Boeing 737
flight(self-education). Редакция 2': ('Глава 2', 91), 'МЦТ США.
```

История': ('Глава 33', 50), 'МЦТ США. История (переписанная)': ('Глава 33', 50), 'Апчихба и другие мемы': ('Глава 69', 30), 'Апчихба и другие мемы. Редакция 2': ('Глава 69', 30)}

Test function query_all_related_chapters_and_books complete. Not found errors

Запрос 3:

{'Boeing 737 flight(self-education)': [('Глава 1', 10, 'Boeing 737 flight(self-education)'), ('Глава 1', 10, 'Boeing 737 flight(self-education)')], 'МЦТ США. История': [('Глава 2', 91, 'МЦТ США. История'), ('Глава 2', 91, 'МЦТ США. История')], 'Апчихба и другие мемы': [('Глава 33', 50, 'Апчихба и другие мемы'), ('Глава 33', 50, 'Апчихба и другие мемы'), ('Глава 5', 20, 'Апчихба и другие мемы'), ('Глава 5', 20, 'Апчихба и другие мемы'), ('Глава 69', 30, 'Апчихба и другие мемы'), ('Глава 69', 30, 'Апчихба и другие мемы')], 'Boeing 737 flight(self-education). Редакция 2': [('Глава 1', 10, 'Boeing 737 flight(self-education). Редакция 2'), ('Глава 1', 10, 'Boeing 737 flight(self-education). Редакция 2')], 'МЦТ США. История (переписанная)': [('Глава 2', 91, 'МЦТ США. История (переписанная)'), ('Глава 2', 91, 'МЦТ США. История (переписанная)')], 'Апчихба и другие мемы. Редакция 2': [('Глава 33', 50, 'Апчихба и другие мемы. Редакция 2'), ('Глава 33', 50, 'Апчихба и другие мемы. Редакция 2'), ('Глава 5', 20, 'Апчихба и другие мемы. Редакция 2'), ('Глава 5', 20, 'Апчихба и другие мемы. Редакция 2'), ('Глава 69', 30, 'Апчихба и другие мемы. Редакция 2'), ('Глава 69', 30, 'Апчихба и другие мемы. Редакция 2')]}