```
from operator import itemgetter
import unittest


class Hrd:
    """Жесткий диск"""

    def __init__(self, id, memory, type, name, cmp_id):
        self.id = id
        self.memory = memory
        self.type = type
        self.name = name
        self.cmp_id = cmp_id


class Cmp:
    """Компьютер"""

    def __init__(self, id, name, type, os):
        self.id = id
        self.name = name
        self.type = type
        self.os = os


class CmpHrd:
    """Жесткие диски компьютера для реализации связи многие-ко-многим"""

    def __init__(self, cmp_id, hrd_id):
        self.cmp_id = cmp_id
        self.hrd_id = hrd_id


def get_one_to_many(hrds, cmps):
    return [
        (h.name, h.memory, c.name)
        for h in hrds
        for c in cmps
        if h.cmp_id == c.id
    ]


def get_many_to_many(cmps, hrds, cmps_hrds):
    many_to_many_temp = [
        (c.name, ch.cmp_id, ch.hrd_id)
        for c in cmps
        for ch in cmps_hrds
        if c.id == ch.cmp_id
    ]
```

```
    return [
        (h.name, h.memory, cmp_name)
        for cmp_name, cmp_id, hrd_id in many_to_many_temp
        for h in hrds if h.id == hrd_id
    ]


def compute_total_memory(cmps, one_to_many):
    res_12_unsorted = []
    for c in cmps:
        c_hrds = list(filter(lambda i: i[2] == c.name, one_to_many))
        if c_hrds:  # если есть жесткие диски
            c_memorys = [memory for _, memory, _ in c_hrds]
            c_memorys_sum = sum(c_memorys)
            res_12_unsorted.append((c.name, c_memorys_sum))

    return sorted(res_12_unsorted, key=itemgetter(1), reverse=True)


def get_disks_per_computer(cmps, many_to_many):
    res_13 = {}
    for c in cmps:
        c_hrds = list(filter(lambda i: i[2] == c.name, many_to_many))
        c_hrds_names = [x for x, _, _ in c_hrds]
        res_13[c.name] = c_hrds_names
    return res_13

class TestDiskComputerRelations(unittest.TestCase):

    def setUp(self):
        self.cmps = [
            Cmp(1, "Рабочая станция 1", "настольный", "Windows 10"),
            Cmp(2, "Ноутбук 2", "ноутбук", "Ubuntu 20.04"),
            Cmp(3, "Сервер 3", "сервер", "Windows Server 2019"),
            Cmp(4, "Игровой ПК", "настольный", "Windows 11"),
            Cmp(5, "Лаптоп 4", "ноутбук", "macOS Monterey"),
        ]

        self.hrds = [
            Hrd(1, 256, "SSD", "Samsung", 1),
            Hrd(2, 512, "HDD", "Lacie", 2),
            Hrd(3, 1024, "SSD", "Toshi", 3),
            Hrd(4, 2000, "HDD", "Fujitsu", 3),
            Hrd(5, 4000, "SSD", "BMSTU Special", 3),
        ]

        self.cmps_hrds = [
            CmpHrd(1, 1),
            CmpHrd(1, 2),
            CmpHrd(2, 1),
```

```
            CmpHrd(2, 2),
            CmpHrd(3, 1),
            CmpHrd(3, 3),
            CmpHrd(4, 4),
            CmpHrd(5, 5),
        ]

    def test_get_one_to_many(self):
        result = get_one_to_many(self.hrds, self.cmps)
        expected_result = [
            ('Samsung', 256, 'Рабочая станция 1'),
            ('Lacie', 512, 'Ноутбук 2'),
            ('Toshi', 1024, 'Сервер 3'),
            ('Fujitsu', 2000, 'Сервер 3'),
            ('BMSTU Special', 4000, 'Сервер 3')
        ]
        self.assertCountEqual(result, expected_result)

    def test_compute_total_memory(self):
        one_to_many = get_one_to_many(self.hrds, self.cmps)
        result = compute_total_memory(self.cmps, one_to_many)
        expected_result = [
            ('Сервер 3', 7024),
            ('Ноутбук 2', 512),
            ('Рабочая станция 1', 256),
        ]
        self.assertEqual(result, expected_result)

    def test_get_disks_per_computer(self):
        one_to_many = get_one_to_many(self.hrds, self.cmps)
        many_to_many = get_many_to_many(self.cmps, self.hrds, self.cmps_hrds)
        result = get_disks_per_computer(self.cmps, many_to_many)
        expected_result = {
            'Рабочая станция 1': ['Samsung', 'Lacie'],
            'Ноутбук 2': ['Samsung', 'Lacie'],
            'Сервер 3': ['Samsung', 'Toshi'],
            'Игровой ПК': ['Fujitsu'],
            'Лаптоп 4': ['BMSTU Special'],
        }
        self.assertEqual(result, expected_result)


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

Результаты тестов представлены ниже:

```
PS D:\studyin\metromap> python -m unittest rk2.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```