***main.py***

# Classes for One-to-Many relationship

class OneToManyOperator:

    def \_\_init\_\_(self, operator\_id, symbol, description, salary, language\_id):

        self.operator\_id = operator\_id

        self.symbol = symbol

        self.description = description

        self.salary = salary

        self.language\_id = language\_id

class OneToManyProgrammingLanguage:

    def \_\_init\_\_(self, language\_id, language\_name, version):

        self.language\_id = language\_id

        self.language\_name = language\_name

        self.version = version

# Classes for Many-to-Many relationship

class Operator:

    def \_\_init\_\_(self, operator\_id, symbol, description, salary):

        self.operator\_id = operator\_id

        self.symbol = symbol

        self.description = description

        self.salary = salary

class ProgrammingLanguage:

    def \_\_init\_\_(self, language\_id, language\_name, version):

        self.language\_id = language\_id

        self.language\_name = language\_name

        self.version = version

class OperatorsLanguages:

    def \_\_init\_\_(self, operator\_id, language\_id):

        self.operator\_id = operator\_id

        self.language\_id = language\_id

# Function to create examples

def create\_examples():

    languages = [

        OneToManyProgrammingLanguage(1, "Python", "3.10"),

        OneToManyProgrammingLanguage(2, "Java", "17"),

        OneToManyProgrammingLanguage(3, "C++", "20"),

        OneToManyProgrammingLanguage(4, "JavaScript", "ES2021"),

    ]

    operators = [

        OneToManyOperator(1, "+", "Addition", 50000, 1),

        OneToManyOperator(2, "-", "Subtraction", 60000, 1),

        OneToManyOperator(3, "\*", "Multiplication", 70000, 1),

        OneToManyOperator(4, "/", "Division", 55000, 1),

        OneToManyOperator(5, "++", "Increment (Unary)", 52000, 2),

        OneToManyOperator(6, "--", "Decrement (Unary)", 52000, 2),

        OneToManyOperator(7, "!", "Logical NOT (Unary)", 53000, 3),

        OneToManyOperator(8, "~", "Bitwise NOT (Unary)", 54000, 3),

        OneToManyOperator(9, "&&", "Logical AND", 60000, 4),

        OneToManyOperator(10, "||", "Logical OR", 60000, 4),

    ]

    many\_to\_many\_operators = [

        Operator(1, "+", "Addition", 50000),

        Operator(2, "-", "Subtraction", 60000),

        Operator(3, "\*", "Multiplication", 70000),

        Operator(4, "/", "Division", 55000),

        Operator(5, "++", "Increment (Unary)", 52000),

        Operator(6, "--", "Decrement (Unary)", 52000),

        Operator(7, "!", "Logical NOT (Unary)", 53000),

        Operator(8, "~", "Bitwise NOT (Unary)", 54000),

    ]

    many\_to\_many\_languages = [

        ProgrammingLanguage(1, "Python", "3.10"),

        ProgrammingLanguage(2, "Java", "17"),

        ProgrammingLanguage(3, "C++", "20"),

        ProgrammingLanguage(4, "JavaScript", "ES2021"),

    ]

    operators\_languages = [

        OperatorsLanguages(1, 1),  # + in Python

        OperatorsLanguages(2, 1),  # - in Python

        OperatorsLanguages(3, 1),  # \* in Python

        OperatorsLanguages(4, 1),  # / in Python

        OperatorsLanguages(5, 2),  # ++ in Java

        OperatorsLanguages(6, 2),  # -- in Java

        OperatorsLanguages(1, 3),  # + in C++

        OperatorsLanguages(7, 3),  # ! in C++

        OperatorsLanguages(9, 4),  # && in JavaScript

        OperatorsLanguages(10, 4),  # || in JavaScript

        OperatorsLanguages(8, 3),  # ~ in C++

    ]

    return languages, operators, many\_to\_many\_operators, many\_to\_many\_languages, operators\_languages

# Functions for One-to-Many queries

def get\_languages\_starting\_with\_J(languages, operators):

    result = []

    for lang in languages:

        if lang.language\_name.startswith("J"):

            language\_info = {"language\_name": lang.language\_name, "operators": []}

            for op in operators:

                if op.language\_id == lang.language\_id:

                    language\_info["operators"].append({"symbol": op.symbol, "description": op.description})

            result.append(language\_info)

    return result

def get\_languages\_with\_max\_operators(languages, operators):

    operator\_count = {lang.language\_id: 0 for lang in languages}

    for op in operators:

        operator\_count[op.language\_id] += 1

    sorted\_languages = sorted(languages, key=lambda lang: operator\_count[lang.language\_id], reverse=True)

    result = [{"language\_name": lang.language\_name, "operators\_count": operator\_count[lang.language\_id]} for lang in sorted\_languages]

    return result

# Functions for Many-to-Many queries

def get\_operators\_by\_language(languages, operators, operators\_languages):

    result = []

    sorted\_languages = sorted(languages, key=lambda lang: lang.language\_name)

    for language in sorted\_languages:

        language\_info = {"language\_name": language.language\_name, "operators": []}

        for ol in operators\_languages:

            if ol.language\_id == language.language\_id:

                operator = next(op for op in operators if op.operator\_id == ol.operator\_id)

                language\_info["operators"].append({"symbol": operator.symbol, "description": operator.description})

        result.append(language\_info)

    return result

***tests.py***

import unittest

class TestProgramFunctions(unittest.TestCase):

    def setUp(self):

        self.languages, self.operators, self.many\_to\_many\_operators, self.many\_to\_many\_languages, self.operators\_languages = create\_examples()

    def test\_get\_languages\_starting\_with\_J(self):

        expected\_result = [

            {

                "language\_name": "Java",

                "operators": [

                    {"symbol": "++", "description": "Increment (Unary)"},

                    {"symbol": "--", "description": "Decrement (Unary)"}

                ]

            },

            {

                "language\_name": "JavaScript",

                "operators": [

                    {"symbol": "&&", "description": "Logical AND"},

                    {"symbol": "||", "description": "Logical OR"}

                ]

            }

        ]

        result = get\_languages\_starting\_with\_J(self.languages, self.operators)

        self.assertEqual(result, expected\_result)

    def test\_get\_languages\_with\_max\_operators(self):

        expected\_result = [

            {"language\_name": "Python", "operators\_count": 4},

            {"language\_name": "C++", "operators\_count": 3},

            {"language\_name": "Java", "operators\_count": 2},

            {"language\_name": "JavaScript", "operators\_count": 2}

        ]

        result = get\_languages\_with\_max\_operators(self.languages, self.operators)

        self.assertEqual(result, expected\_result)

    def test\_get\_operators\_by\_language(self):

        expected\_result = [

            {

                "language\_name": "C++",

                "operators": [

                    {"symbol": "+", "description": "Addition"},

                    {"symbol": "!", "description": "Logical NOT (Unary)"},

                    {"symbol": "~", "description": "Bitwise NOT (Unary)"}

                ]

            },

            {

                "language\_name": "Java",

                "operators": [

                    {"symbol": "++", "description": "Increment (Unary)"},

                    {"symbol": "--", "description": "Decrement (Unary)"}

                ]

            },

            {

                "language\_name": "JavaScript",

                "operators": [

                    {"symbol": "&&", "description": "Logical AND"},

                    {"symbol": "||", "description": "Logical OR"}

                ]

            },

            {

                "language\_name": "Python",

                "operators": [

                    {"symbol": "+", "description": "Addition"},

                    {"symbol": "-", "description": "Subtraction"},

                    {"symbol": "\*", "description": "Multiplication"},

                    {"symbol": "/", "description": "Division"}

                ]

            }

        ]

        result = get\_operators\_by\_language(self.many\_to\_many\_languages, self.many\_to\_many\_operators, self.operators\_languages)

        self.assertEqual(result, expected\_result)

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()