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()
```