DBMS LAB

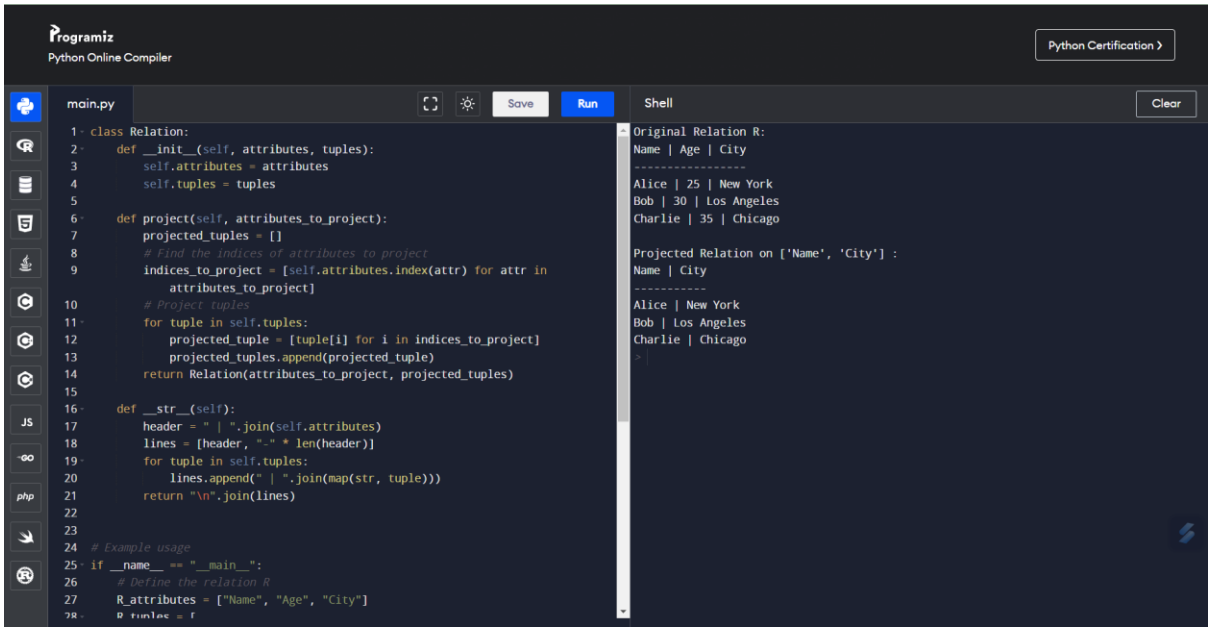
Name – RAVI SHEKHAR

Roll No. – 22CS3075

1.

```
class Relation:
  def __init__(self, attributes, tuples):
    self.attributes = attributes
    self.tuples = tuples

  def project(self, attributes_to_project):
    projected_tuples = []
    # Find the indices of attributes to project
    indices_to_project = [self.attributes.index(attr) for attr in attributes_to_project]
    # Project tuples
    for tuple in self.tuples:
      projected_tuple = [tuple[i] for i in indices_to_project]
      projected_tuples.append(projected_tuple)
    return Relation(attributes_to_project, projected_tuples)

  def __str__(self):
    header = " | ".join(self.attributes)
    lines = [header, "-" * len(header)]
    for tuple in self.tuples:
      lines.append(" | ".join(map(str, tuple)))
```

```
        return "\n".join(lines)



# Example usage
if __name__ == "__main__":
    # Define the relation R
    R_attributes = ["Name", "Age", "City"]
    R_tuples = [
        ["Alice", 25, "New York"],
        ["Bob", 30, "Los Angeles"],
        ["Charlie", 35, "Chicago"]
    ]
    R = Relation(R_attributes, R_tuples)

    # Attributes to project on
    attributes_to_project = ["Name", "City"]

    # Perform projection
    projected_relation = R.project(attributes_to_project)

    # Output the resulting relation after the projection operation
    print("Original Relation R:")
    print(R)
    print("\nProjected Relation on", attributes_to_project, ":")
    print(projected_relation)
```

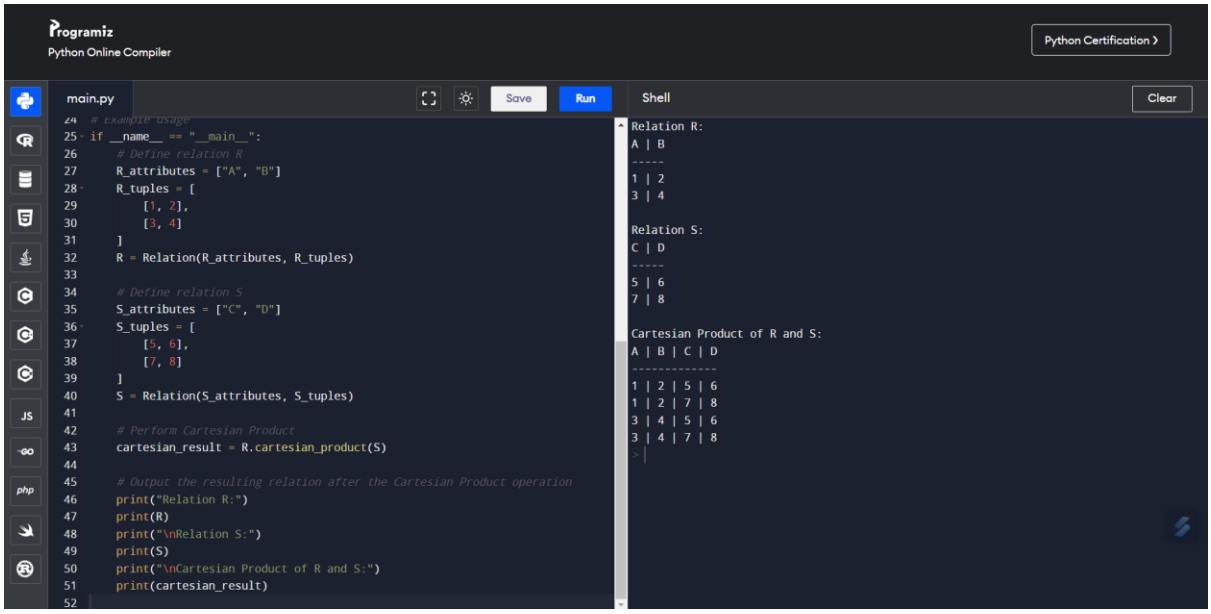


2.

```
class Relation:
    def __init__(self, attributes, tuples):
        self.attributes = attributes
        self.tuples = tuples

    def cartesian_product(self, other_relation):
        result_attributes = self.attributes + other_relation.attributes
        result_tuples = []

        for tuple1 in self.tuples:
            for tuple2 in other_relation.tuples:
                result_tuples.append(tuple1 + tuple2)

        return Relation(result_attributes, result_tuples)

    def __str__(self):
        header = " | ".join(self.attributes)
        lines = [header, "-" * len(header)]
```

```
        for tuple in self.tuples:
            lines.append(" | ".join(map(str, tuple)))
        return "\n".join(lines)


# Example usage
if __name__ == "__main__":
    # Define relation R
    R_attributes = ["A", "B"]
    R_tuples = [
        [1, 2],
        [3, 4]
    ]
    R = Relation(R_attributes, R_tuples)

    # Define relation S
    S_attributes = ["C", "D"]
    S_tuples = [
        [5, 6],
        [7, 8]
    ]
    S = Relation(S_attributes, S_tuples)

    # Perform Cartesian Product
    cartesian_result = R.cartesian_product(S)

    # Output the resulting relation after the Cartesian Product operation
    print("Relation R:")
    print(R)
    print("\nRelation S:")
    print(S)
```

```
    print("\nCartesian Product of R and S:")
    print(cartesian_result)
```

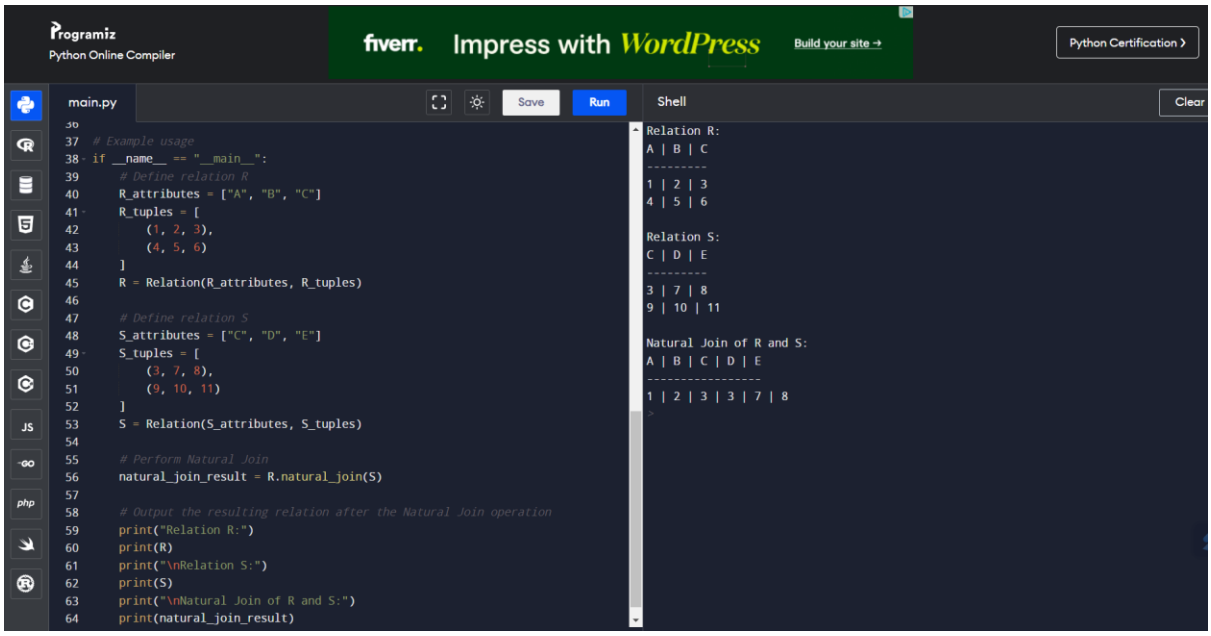


3.

```
class Relation:
    def __init__(self, attributes, tuples):
        self.attributes = attributes
        self.tuples = tuples

    def natural_join(self, other_relation):
        common_attributes = list(set(self.attributes).intersection(other_relation.attributes))

        # Find indices of common attributes in each relation
        self_indices = [self.attributes.index(attr) for attr in common_attributes]
        other_indices = [other_relation.attributes.index(attr) for attr in common_attributes]

        # Build the resulting attributes list
        result_attributes = self.attributes + [attr for attr in other_relation.attributes if attr not in common_attributes]
```

```
        # Build the resulting tuples
        result_tuples = []
        for tuple1 in self.tuples:
            for tuple2 in other_relation.tuples:
                # Check if tuples match on common attributes
                match = all(tuple1[i] == tuple2[j] for i, j in zip(self_indices, other_indices))
                if match:
                    # Combine tuples
                    new_tuple = tuple1 + tuple2
                    result_tuples.append(new_tuple)

        return Relation(result_attributes, result_tuples)

    def __str__(self):
        header = " | ".join(self.attributes)
        lines = [header, "-" * len(header)]
        for tuple in self.tuples:
            lines.append(" | ".join(map(str, tuple)))
        return "\n".join(lines)


# Example usage
if __name__ == "__main__":
    # Define relation R
    R_attributes = ["A", "B", "C"]
    R_tuples = [
        (1, 2, 3),
        (4, 5, 6)
    ]
    R = Relation(R_attributes, R_tuples)
```

```
# Define relation S
S_attributes = ["C", "D", "E"]
S_tuples = [
    (3, 7, 8),
    (9, 10, 11)
]
S = Relation(S_attributes, S_tuples)

# Perform Natural Join
natural_join_result = R.natural_join(S)

# Output the resulting relation after the Natural Join operation
print("Relation R:")
print(R)
print("\nRelation S:")
print(S)
print("\nNatural Join of R and S:")
print(natural_join_result)
```



4.

```
class Relation:
    def __init__(self, attributes, tuples):
        self.attributes = attributes
        self.tuples = tuples

    def division(self, other_relation):
        # Check if the attributes of S are a subset of the attributes of R
        if not set(other_relation.attributes).issubset(set(self.attributes)):
            print("Error: The attributes of S are not a subset of the attributes of R.")
            return None

        # Find the indices of attributes in R that are not in S
        indices_to_keep = [self.attributes.index(attr) for attr in self.attributes if attr not in other_relation.attributes]

        # Build the resulting attributes list
        result_attributes = [attr for attr in self.attributes if attr not in other_relation.attributes]

        # Build the resulting tuples
        result_tuples = []
        for tuple1 in self.tuples:
            match_found = False
            for tuple2 in other_relation.tuples:
                # Check if tuple2 is a subset of tuple1
                if all(tuple1[i] == tuple2[other_relation.attributes.index(attr)] for i, attr in enumerate(other_relation.attributes)):
                    match_found = True
                    break
            if not match_found:
                result_tuples.append([tuple1[i] for i in indices_to_keep])
```

```
        return Relation(result_attributes, result_tuples)

    def __str__(self):
        header = " | ".join(self.attributes)
        lines = [header, "-" * len(header)]
        for tuple in self.tuples:
            lines.append(" | ".join(map(str, tuple)))
        return "\n".join(lines)


# Example usage
if __name__ == "__main__":
    # Define relation R
    R_attributes = ["A", "B"]
    R_tuples = [
        (1, 2),
        (3, 4),
        (5, 6)
    ]
    R = Relation(R_attributes, R_tuples)

    # Define relation S
    S_attributes = ["A"]
    S_tuples = [
        (1,),
        (3,)
    ]
    S = Relation(S_attributes, S_tuples)

    # Perform Division (R ÷ S)
```

```
division_result = R.division(S)


# Output the resulting relation after the Division operation
print("Relation R:")
print(R)
print("\nRelation S:")
print(S)
print("\nDivision of R by S (R ÷ S):")
print(division_result)
```