***This is the document for you to refer to write your relational algebra in python.***

**To write relational algebra expressions using python, the following components are available:**

* Relation(table\_name): Represents a table in the database.
* Projection(relation, columns): Projects specific columns from a relation.
* Selection(relation, condition): Selects tuples from a relation based on a condition.
* NaturalJoin(relation1, relation2): Joins two relations based on common attributes.
* **ThetaJoin(relation1, relation2, condition): Joins two relations based on a specific condition.**
* **Rename(relation, name): Rename a relation, this returns a relation.**
* **Rename(relation, mapping={original\_name: new\_name}): Rename an attribute, this returns a relation.**
* Set operators like / for set division, - for set difference, | for set union, and & for set intersection. Whenever you do a set operation, you should rename it for further operations.

The conditions (condition) can be constructed using classes like Equals, GreaterEquals, GreaterThan, LessEquals, LessThan, **And**, Or, Not.

### **Examples of Writing Relational Algebra Expressions**

Here’s a breakdown of how to write relational algebra expressions using this syntax:

1. **Projection**:

To select specific columns from a table, use:  
  
Projection(Relation("person"), ["name", "gender"])

This projects the name and gender columns from the person table.

1. **Selection**:

To filter rows based on a condition, use:  
  
Selection(Relation("equipment"), Equals("is\_available", 1))

This selects rows from the equipment relation where is\_available equals 1.

1. **Natural Join**:

To join two relations based on common columns, use:  
  
NaturalJoin(Relation("person"), Relation("university\_affiliate"))

This performs a natural join between person and university\_affiliate.

1. **Combining Operators**:

You can combine multiple operations to form complex queries:  
  
Projection(

Selection(

NaturalJoin(

Relation("person"),

Relation("university\_affiliate")

),

Equals("university\_affiliate.member\_type", "Faculty")

),

["person.name", "university\_affiliate.department"]

)

This expression selects rows where the member\_type is "Faculty" from the natural join of person and university\_affiliate, and then projects person.name and university\_affiliate.department.

1. **Theta Join**:

To join two relations using a specific condition, use:  
  
ThetaJoin(Relation("location\_reading"), Relation("person"), Equals("location\_reading.person\_id", "person.card\_id"))

This performs a theta join between location\_reading and person using the condition location\_reading.person\_id = person.card\_id.

### 

### 

### **Writing Your Own Relational Algebra Expressions**

To write relational algebra expressions using this syntax, follow these steps:

1. Start with the Relation class to refer to a specific table.
2. Use relational operators like Projection, Selection, NaturalJoin, and ThetaJoin to build the desired query.
3. For filtering conditions, use classes like Equals, GreaterEquals, LessThan, And, and Or to specify logical conditions.

Here is another example:

Suppose you want to find all events that occur in a "yoga studio" and return the names of attendees:  
  
expression = Projection(

NaturalJoin(

NaturalJoin(

Selection(

NaturalJoin(

Relation("attends"),

Relation("events")

),

Equals("space\_description", "yoga studio")

),

Relation("member")

),

Relation("person")

),

["name"]

)

This example involves multiple natural joins and selections, followed by a projection to get the final result.