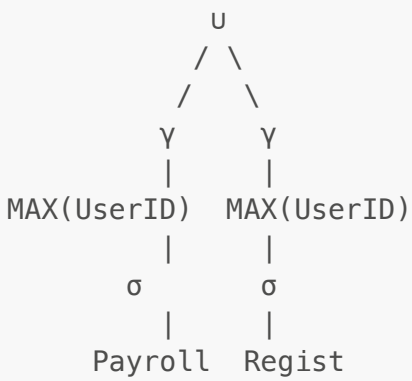
Takeaway Sheet 6 - Jonathan Jacobs

Question 1

Question 2

The relational algebra tree shows how to combine the highest `UserID` values from the `Payroll` and `Regist` tables. Here's a simple explanation of each part:

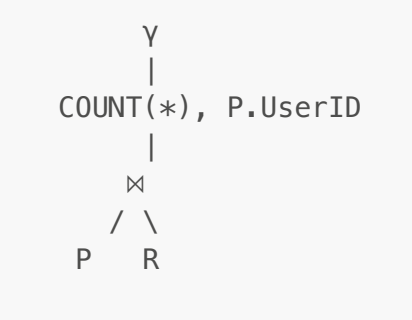
- **`Payroll` and `Regist`**: These are the two tables where the data is coming from.
- **`MAX(UserID)`**: This operation finds the highest `UserID` in each table.
- **`UNION`**: This combines the highest `UserID` from each table into a single list, removing any duplicates if both tables have the same highest `UserID`.



Question 3

The relational algebra tree illustrates how to join and aggregate data from the `Payroll` and `Regist` tables based on a common field, `UserID`. Here's a simple breakdown of each part:

- **`Payroll` and `Regist`**: These are the two tables involved. Data is pulled from both tables.
- **`Join Operation (⋈)`**: This node represents the process of combining records from `Payroll` and `Regist` where the `UserID` from `Payroll` matches the `UserID` from `Regist`.
- **`Group-By and Count (*)`**: After joining, this operation groups the results by `UserID` from `Payroll` and counts the number of records in each group, reflecting the total occurrences of each `UserID` across the joined tables.



```
    |   |
Payroll Regist
```

Question 4

An inner join in relational algebra is both **commutative** and **associative**. Here's why:

- **Commutative**: This means that the order in which you join two tables does not affect the result of the join. For example, if you have tables A and B, joining A to B (A ⋈ B) yields the same result as joining B to A (B ⋈ A). The key condition here is that the join criteria remain the same.
- **Associative**: This property means that when joining multiple tables, the way you group them for the join operation doesn't change the result. For example, if you have three tables A, B, and C, you can join them as ((A ⋈ B) ⋈ C) or (A ⋈ (B ⋈ C)), and the final result will be the same, provided the join conditions are appropriately managed.

Therefore, the correct choice for question 4 is:

- **Both Commutative and Associative**.

Question 5

Question 6

Question 7

Question 8

```
        Π<sub>(x.pid, y.pid, y.quantity)</sub>
                    |
                    |
    σ<sub>x.city and x.name='AAA Suppliers' and (y.pid = 5 or y.pid = 12)
</sub>
                    |
                    |
            ⋈<sub>x.sid = y.sid</sub>
                  /  \
                 /    \
Suppliers x                     Supplies y
T(Suppliers) = 1000             T(Supplies) = 10000
V(Suppliers, city) = 20         V(Supplies, pid) = 2500
V(Suppliers, state) = 10
V(Suppliers, name) = 950
```

To estimate the cardinality of the query described, consider the following:

1. **Join Condition**: The join on `x.sid = y.sid` links the `Suppliers` and `Supplies` tables. The actual cardinality of the join depends on the number of matching `sid` values.

2. **Selection Condition**: The selection restricts `Suppliers` based on `x.city` and `x.name = 'AAA Suppliers'`, and `Supplies` where `y.pid` must be 5 or 12. This limits the output to only those records that meet these specific conditions.

Estimation Details:

- **Base Table Sizes**:
  - `T(Suppliers) = 1000`
  - `T(Supplies) = 10000`
- **Relevant `Supplies` Entries**: Given `V(Supplies, pid) = 2500` but only considering `pid` 5 or 12, the likely subset is `(2 / 2500) * 10000 = 8` tuples.

Estimated Cardinality:

The estimated cardinality will be low due to the specific and restrictive conditions applied. The selectivity of the `name` and `pid` conditions dramatically reduces the number of tuples from the potential maximum. Assuming uniform distribution and perfect join conditions, the cardinality is primarily driven by the stringent selection criteria, likely resulting in a very small subset of the original datasets.