Takeaway Sheet 2 (Lectures 4-6)

L4.1

What does this Java pseudocode do?

```
foreach row in Payroll:
 if row.Job == 'TA':
   sum = sum + row.Salary
output(sum)
```

The provided Java pseudocode iterates over a data structure named `Payroll`, summing up the `Salary` values for entries where the `Job` is equal to 'TA'. It then outputs the total sum of salaries for all 'TA' positions.

L4.2

What is the result of these 3 COUNT() queries?

1.
   ```
    COUNT(*) -- This counts all rows in the table, regardless of any null
   values in any columns. It gives the total number of rows in the table.
   ```

2.
   ```
   COUNT(job) -- This counts all rows where the job column is not null.
   If every row in the table has a non-null value for job, it will be the
   same as COUNT(*). However, if any row has a null value for job, those
   rows won't be counted.
   ```

3.
   ```
    COUNT(salary) --  Similar to COUNT(job), this counts all rows where
   the salary column is not null. Rows with a null value in the salary
   column are not counted.
   ```

L4.3

For Leslie:

- COUNT(*): 1
- COUNT(job): 1
- COUNT(salary): 1

For Frances:

- COUNT(*): 1

- COUNT(job): 1
- COUNT(salary): 1

L4.4

```
SELECT COUNT(*) FROM Payroll; -- 4
SELECT COUNT(job) FROM Payroll; --  4
SELECT COUNT(salary) FROM Payroll; -- 4
```

L4.5

```
SELECT SUM(salary) FROM Payroll; -- 330k.

SELECT MIN(salary) FROM Payroll; -- 50k.

SELECT MAX(salary) FROM Payroll; --  'Prof' is 120k
```

L5.1

```
# Find users who own both Civic and Ferrari
civic_owners = set(entry["UserID"] for entry in regist if entry["Car"] ==
"Civic")
ferrari_owners = set(entry["UserID"] for entry in regist if entry["Car"]
== "Ferrari")

# Intersection to find who owns both
both_owners = civic_owners.intersection(ferrari_owners)

# Display names of those who own both
for person in payroll:
    if person["UserID"] in both_owners:
        print(person["Name"], "owns both a Civic and a Ferrari.")
```

L5.2

Which orange section best describes the following join operations?

- LEFT OUTER JOIN: C
- RIGHT OUTER JOIN: D
- FULL OUTER JOIN: B
- INNER JOIN: B

L5.3

Write a SQL query to find everyone who drives a Civic AND a Ferrari

```sql
SELECT p.Name
FROM Payroll p
JOIN (
    SELECT r1.UserID
    FROM Regist r1
    INNER JOIN Regist r2 ON r1.UserID = r2.UserID
    WHERE r1.Car = 'Civic' AND r2.Car = 'Ferrari'
) AS owners ON p.UserID = owners.UserID;
```

L5.4

At which stage in FJWGHOS did ... Frances' tuple disappear? Answer: JOIN

Quinn's tuple disappear? Answer: JOIN

Both Frances and Quinn are excluded during the JOIN operation because the condition for joining (owning both a Civic and a Ferrari) is not met for them.

L5.5

Invent a phrase to help you remember FJWGHOS Answer: Friendly Jaguars Will Generally Help Other Species

L6.1

Write a SQL query to find the names of people who own more than one car

```sql
SELECT p.Name
FROM Payroll p
JOIN (
    SELECT UserID
    FROM Regist
    GROUP BY UserID
    HAVING COUNT(*) > 1
) AS multiple_cars ON p.UserID = multiple_cars.UserID;
```

L6.2

Write Java pseudocode to output the person and their salary who has the highest salary per job

```
# Process each payroll record
for record in payroll:
    # Convert the salary to an integer for comparison
    salary = int(record["salary"].rstrip('k')) * 1000
    job = record["job"]
    # Check if this is the highest salary for the job category
    if salary > max_salary_per_job[job]:
        max_salary_per_job[job] = salary
        person_with_max_salary_per_job[job] = record["name"]
```

L6.3

Modify this query to calculate all-pairs by job

```
SELECT P1.Name AS Name1, P2.Name AS Name2, P1.Job
FROM Payroll AS P1
JOIN Payroll AS P2 ON P1.Job = P2.Job AND P1.UserID < P2.UserID
```