Supervision 1 Questions

Question 1

1a

```
select distinct p.name as name, m.year - p.deathYear as gap
    from people as p
    inner join has_position as h on h.person_id = p.person_id
    inner join movies as m on m.movie_id = h.movie_id
    where h.position='writer'
    and m.year>p.deathYear
order by gap desc, p.name
limit 10;
```

Join the `has_position` and `movies` tables to the `people` table where the person has a position in the movie. Select rows in which that position is that of being a writer and where the movie's release year is after the year of the person's death.

1b

```
select p.name as name, pr.role as role, count(*) as movie_count
    from people as p
    inner join plays_role as pr on pr.person_id = p.person_id
    inner join movies as m on pr.movie_id = m.movie_id
    where m.type = 'movie'
group by name, role
order by movie_count desc, name, role
limit 10;
```

Join the `plays_role` and `movies` tables to the `people` table where the person played a role in the movie. Return the rows in which the movie type is "movie", and group the rows by `name` and `role` such that the groups contain rows in which the same person plays the same role.

1c

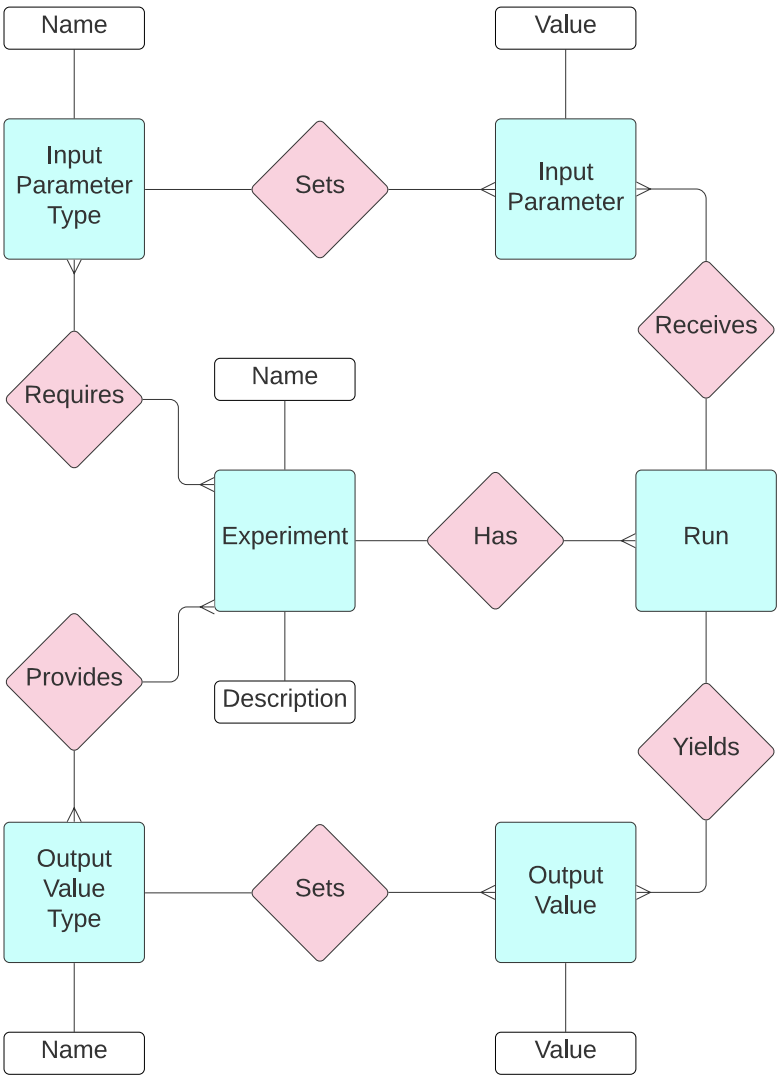
```
select r1.role as role, p.name as name, m1.title as movie_title, m2.title as tv_movie_title
    from people as p
    inner join plays_role as r1 on r1.person_id = p.person_id
    inner join plays_role as r2 on r2.person_id = p.person_id
    inner join movies as m1 on r1.movie_id = m1.movie_id
    inner join movies as m2 on r2.movie_id = m2.movie_id
    where m1.type='movie'
    and m2.type='tvMovie'
    and r1.role=r2.role
order by r1.role, p.name, m1.title, m2.title;
```

Join the `plays_role` and `movies` tables to the `people` table twice, such that the row contains the information about the person, and the roles they've played in two movies. Select all such rows for which the first movie is of type "movie" and the second is of type "tvMovie", and the person played the same role in both.

Question 2

Redundancy is when more information is being stored than is strictly necessary to specify all of the data. If data is redundant, that means it could be reconstructed perfectly from the rest of the data. Not all duplicate data is redundant. For example, two people might have died in 1999, and so 1999 will be stored in the database twice, but this is not redundancy, because the death year of one alone cannot tell you the death year of the other.

Question 3



Each experiment is associated with many input parameter types. For example, one simulation experiment might be associated with input parameter types named "grid width" and "grid height" while another experiment might be associated with a single input parameter type named "tree depth".

Each run is associated with many input parameters which each have a value and are also associated with a single input parameter type. For example, for a certain run, one input paramter might be associated with the input parameter type named "grid width" and might have a value of 5.

Similar relationships exist for the outputs. Having the types separate from the values will allow users to ask questions such as "what inputs are required for a certain experiment?" without needing to have a specific run in mind.

Question 4

One possible relational implementation is represented by the following tables.

experiments
experiment_id
name
description

input_parameter_types
input_parameter_type_id
experiment_id
name

output_value_types
output_value_type_id
experiment_id
name

runs
run_id
experiment_id

input_parameters
input_parameter_id
run_id
input_parameter_type_id
value

output_values
output_value_id
run_id
output_value_type_id
value

Note This implementation does not capture the many-to-many relationships between experiments/input parameter types and experiments/output value types which are indicated in the ER diagram. This is because although there could indeed be many experiments which have an input parameter named "grid width", if the experimenter decided that for a certain experiment, the name "rectangle width" might be more appropriate, the above relational model makes it easier to update this for only that one experiment.

Note This implementation does not require that all of the input parameters and output values associated with a run are each associated with a type which is then in turn associated with the same experiment as the run itself.

For example, an input parameter type called "grid_width" might be associated with the experiment with id 1729, but there's no safeguard against a run from experiment 1337 having an input parameter with that type.

This sort of consistency would need to be implemented on the application level.

Question 5

```
SELECT
    ip1.value AS grid_width,
    ip2.value AS grid_height,
    avg(ov1.value) AS average_message_count,
    avg(ov2.value) AS average_run_time
FROM
    experiments AS e
    INNER JOIN runs AS r ON r.experiment_id = e.experiment_id
    INNER JOIN input_parameters AS ip1 ON ip1.run_id = r.run_id
    INNER JOIN input_parameters AS ip2 ON ip2.run_id = r.run_id
    INNER JOIN input_parameter_types AS ipt1 ON ipt1.input_parameter_type_id = ip1.input_parameter_type_id
    INNER JOIN input_parameter_types AS ipt2 ON ipt2.input_parameter_type_id = ip2.input_parameter_type_id
    INNER JOIN output_values AS ov1 ON ov1.run_id = r.run_id
    INNER JOIN output_values AS ov2 ON ov2.run_id = r.run_id
    INNER JOIN output_value_types AS ovt1 ON ovt1.output_value_type_id = ov1.output_value_type_id
    INNER JOIN output_value_types AS ovt2 ON ovt2.output_value_type_id = ov2.output_value_type_id
WHERE
    e.name = 'grid'
    AND ipt1.name = 'grid_width'
    AND ipt2.name = 'grid_height'
    AND ovt1.name = 'message_count'
    AND ovt2.name = 'run_time'
GROUP BY grid_width, grid_height
```