Databases Supervision 1 (srns2)

Question 1

Exercise 1a

```sql
SELECT DISTINCT p.name AS name, m.year - p.deathYear AS gap
FROM people AS p
JOIN has_position AS hp ON hp.person_id = p.person_id
JOIN movies AS m ON m.movie_id = hp.movie_id
WHERE hp.position = 'writer' AND m.year > p.deathYear
ORDER BY gap DESC, p.name
LIMIT 10;
```

The `people` table is joined with the `has_position` table using the key `person_id` on both tables. The `movies` table is then joined to this using the key `movie_id`. This results in a table with people and the movies they're in connected by the positions they have. This is used to find if a writer died before their film was released by comparing the death year of the writer with the release year of their corresponding movies.

Exercise 1b

```sql
SELECT p.name AS name, pr.role AS role, COUNT(*) AS movie_count
FROM people AS p
JOIN plays_role AS pr ON p.person_id = pr.person_id
JOIN movies AS m ON m.movie_id = pr.movie_id
WHERE m.type <> 'tvMovie'
GROUP BY name, role
ORDER BY movie_count DESC, name, role
LIMIT 10;
```

Similarly to in exercise 1a, the each person in the `people` table is joined with the movies they acted in by joining it with the `movies` tale using the `plays_role` table. Then TV movies are excluded from the results. These results are then returned, with `COUNT(*)` being used to find the number of films.

Exercise 1c

```sql
SELECT r1.role AS role, p.name AS name, m1.title AS movie_title, m2.title AS tv_movie_title
FROM people AS p
JOIN plays_role AS r1 ON r1.person_id = p.person_id
JOIN plays_role AS r2 ON r2.person_id = p.person_id
JOIN movies AS m1 ON m1.movie_id = r1.movie_id
JOIN movies AS m2 ON m2.movie_id = r2.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;
```

The `people` table is joined to the `movies` table twice using the `plays_role` table to find each role of every actor. Then, the two movie types are compared and if one is a movie and the other is a TV movie, and the actor plays the same role in both, the movies are returned.

Question 2

Redundancy of data refers to where the same information is represented several times. This has some advantages and disadvantages.

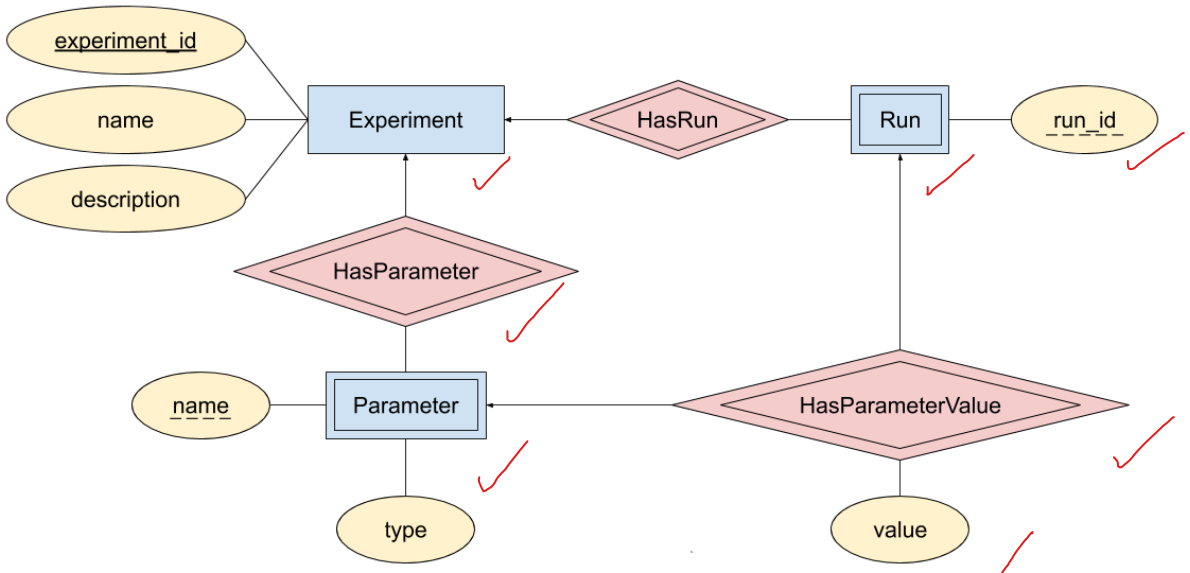
An advantage is that if the data is accidentally deleted in one place, it can be recovered by looking for its other occurrences. It also generally speeds up database reads as it typically requires fewer queries by storing all relevant information in fewer tables.

It also has some disadvantages. It is obviously memory-inefficient as there are several occurrences of the data when there only needs to be one. Updates are also inefficient as the data needs to be updated several times instead of just once, and a lot of data needs to be locked. Furthermore, if the last occurrence of the data is deleted then it is irrecoverable if needed again.

Redundancy is therefore generally minimised in a relational implementation of an ER model by separating a table into multiple tables if the same data occurs several times.

Redundancy is different to duplication of data. Redundancy refers to part of each record having the same data, while duplication refers to different records being entirely identical.

Question 3



Question 4

A possible relational implementation of this model is as follows:

- A table called `experiments` with the columns `experiment_id`, `name`, and `description`, where `experiment_id` is the primary key
- A table called `runs` with the columns `run_id` and `experiment_id`, where `experiment_id` is a foreign key to the `experiments` table, and `experiment_id` and `run_id` form a composite primary key.
- A table called `parameters` with the columns `experiment_id`, `name`, and `type`, where `experiment_id` is a foreign key to the `experiments` table, and `name` and `experiment_id` form a composite primary key.
- A table called `parameter_values` with the columns `experiment_id`, `name`, `run_id`, and `value`, where `name` and `experiment_id` form a composite foreign key to the `parameters` table, and `run_id` and `experiment_id` form a composite foreign key to the `runs` table.

These tables can be created using the following code:

```
CREATE TABLE experiments
(
  experiment_id INTEGER NOT NULL AUTO_INCREMENT,
  name VARCHAR(32) NOT NULL,
  description VARCHAR(255),
  PRIMARY KEY (experiment_id)
);

CREATE TABLE runs
(
  run_id INTEGER NOT NULL,
  experiment_id INTEGER NOT NULL,
  FOREIGN KEY (experiment_id) REFERENCES experiments(experiment_id),
  PRIMARY KEY (run_id, experiment_id)
);

CREATE TABLE parameters
(
  name VARCHAR(32) NOT NULL,
  experiment_id INTEGER NOT NULL,
  type VARCHAR(6) NOT NULL,
  FOREIGN KEY (experiment_id) REFERENCES experiments(experiment_id),
  PRIMARY KEY (name, experiment_id)
);

CREATE TABLE parameter_values
(
  experiment_id INTEGER NOT NULL,
  name VARCHAR(32) NOT NULL,
  run_id INTEGER NOT NULL,
  value INTEGER,
  FOREIGN KEY (name, experiment_id) REFERENCES parameters(name, experiment_id),
  FOREIGN KEY (run_id, experiment_id) REFERENCES runs(run_id, experiment_id)
);
```

Question 5

```
SELECT i1.value AS grid_width, i2.value AS grid_height, AVG(o1.value) AS message_count, AVG(o2.value) AS run_
FROM experiments AS e
JOIN runs AS r ON r.experiment_id = e.experiment_id
JOIN parameters AS p1 ON p1.experiment_id = e.experiment_id
JOIN parameters AS p2 ON p2.experiment_id = e.experiment_id
JOIN parameters AS p3 ON p3.experiment_id = e.experiment_id
JOIN parameters AS p4 ON p4.experiment_id = e.experiment_id
JOIN parameter_values AS i1 ON i1.run_id = r.run_id
  AND i1.experiment_id = e.experiment_id
  AND i1.name = p1.name
JOIN parameter_values AS i2 ON i2.run_id = r.run_id
  AND i2.experiment_id = e.experiment_id
  AND i2.name = p2.name
JOIN parameter_values AS o1 ON o1.run_id = r.run_id
  AND o1.experiment_id = e.experiment_id
  AND o1.name = p3.name
JOIN parameter_values AS o2 ON o2.run_id = r.run_id
  AND o2.experiment_id = e.experiment_id
  AND o2.name = p4.name
WHERE e.name = 'grid'
  AND i1.name = 'grid_width'
  AND i2.name = 'grid_height'
  AND o1.name = 'message_count'
  AND o2.name = 'run_time'
GROUP BY i1.value, i2.value
ORDER BY i1.value, i2.value;
```