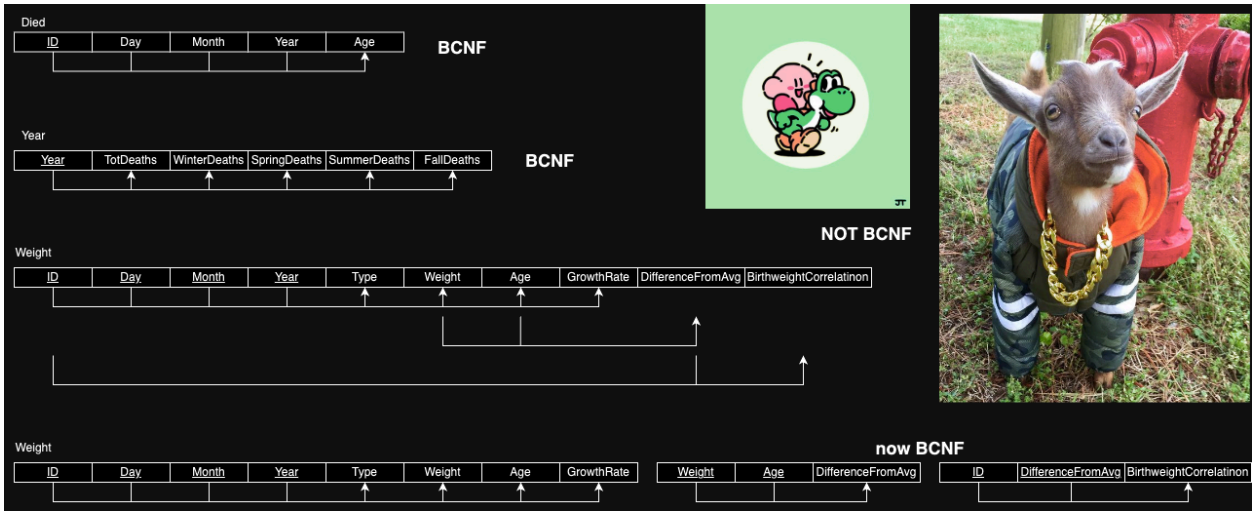
Phase IV –Database Design

1. Review the Database Model document with the stakeholder, and update the model as needed.
2. Demonstrate that all the relations in the relational schema are normalized to Boyce–Codd normal form (BCNF).
    - For each table, specify whether it is in BCNF or not, and explain why.
    - For each table that is not in BCNF, show the complete process that normalizes it to BCNF.
3. Define the different SQL views (virtual tables) required. For each view, list the data and transaction requirements.
4. Design a complete set of SQL queries to satisfy the transaction requirements identified in the previous phases, using the relational schema and views defined in tasks 2 and 3 above. For each query, list the data and transaction requirements.
5. Explain how your views and queries defined in tasks 3 and 4 above implement your use cases.

Submit to Canvas

• Task 2 normalization process (.pdf file)
• Task 3 SQL views (.sql file)
• Task 4 SQL queries (.sql file)
• Task 5 discussion (.pdf file)

**Task 2:**




We have three tables in our database: Died, Year, and Weight. Died and Year are already in BCNF form while Weight needs to be broken down into that form. Died and Year are in BCNF because it is in 1NF, 2NF, 3NF and for any dependency A → B, A is a superkey. Both of them are in 1NF because each column of the table has a single value, each value of each column is of the same domain, each column of the table has unique names, and the order in which the data is stored does not matter. Both of them are in 2NF because they are in 1NF and they do not have any partial dependency, and they are already in 3NF because they do not have transitive dependencies. Finally, they are in BCNF because for any dependency A → B, A is a super key.

The Weight table on the other hand has dependencies from attributes that aren't part of the super key, such as Weight and Age. To solve this problem, we split Weight into separate tables that result in Weight, Age, and DifferenceFromAvg becoming super keys, as the combination of those attributes uniquely identify each row.

**Task 3:**
What we need: ~~*(them to be consistent)*~~

GOAT TABLE

| Goat ID | Birthdate | Birthweight |
|---|---|---|
| Animal_id (Animal) | Dob (Animal) | Alpha_value \| traitcode = 357 (SessionAnimalTrait) |

WEIGHT TABLE

| Goat ID | Date_Weighed | Weight | Weight Type |
|---|---|---|---|
| Animal_id (SessionAnimalTrait) | | Alpha_value → (SessionAnimalTrait) | Identifier = "Live Weight", "%wt", "%WT" (idk if its case sensitive) (SessionAnimalTrait)<br><br>~~traitcode = 53, 369, 381, 393, 405, 436, 448, 963, 970~~ |

DEATH TABLE

| Goat ID | Death date |
|---|---|
| Animal_id \| _something_ \| traitcode = 546 (SessionAnimalTrait) | Status_date \| status = dead (Animal) |

```
DROP TABLE Goat;
CREATE TABLE Goat (
        goat_id integer PRIMARY KEY,
        birthdate timestamp,
        birthweight varchar(20),
        last_weight varchar(20),
        last_weight_date timestamp,
        goat_status varchar(20));

DROP TABLE Weight;
CREATE TABLE Weight (
```

```
        session integer,
        goat_id integer,
        trait integer,
        w_date timestamp,
        alpha_value varchar(20),
        primary key(session, goat_id, trait, w_date));

DROP TABLE Death;
CREATE TABLE Death (
        goat_id integer PRIMARY KEY,
        d_date timestamp);

INSERT INTO Goat (goat_id, birthdate, birthweight, last_weight, last_weight_date, goat_status)
SELECT Animal.animal_id, Animal.dob, MIN(SessionAnimalTrait.alpha_value) AS birthweight,
Animal.last_weight, Animal.last_weight_date, Animal.status
FROM Animal JOIN SessionAnimalTrait ON Animal.animal_id = SessionAnimalTrait.animal_id
WHERE SessionAnimalTrait.trait_code = 357 AND SessionAnimalTrait.alpha_value != '0' AND
SessionAnimalTrait.alpha_value != ''
GROUP BY
        Animal.animal_id,
        Animal.dob,
        Animal.last_weight,
        Animal.last_weight_date,
        Animal.status;

INSERT INTO Weight (session, goat_id, trait, w_date, alpha_value)
SELECT session_id, animal_id, trait_code, when_measured, alpha_value
FROM SessionAnimalTrait
WHERE trait_code IN (53, 369, 381, 393, 405, 436, 448, 963, 970)
AND alpha_value != '0';

INSERT INTO Death (goat_id, d_date)
SELECT animal_id, status_date
FROM Animal
WHERE status = 'Dead';
```

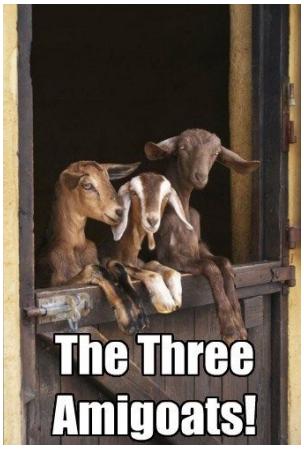
**Task 4:**

```
-- Average number of deaths per year and average age at death
SELECT year, ROUND(AVG(deaths_per_year), 0) AS num_deaths, AVG(age) AS avg_age
FROM (
        SELECT EXTRACT(YEAR FROM Death.d_date) AS year, COUNT(*) AS
deaths_per_year, AVG(AGE(Death.d_date, Goat.birthdate)) AS age
        FROM Goat JOIN Death ON Goat.goat_id = Death.goat_id
```

```
        GROUP BY year
)
GROUP BY year
ORDER BY year;

-- Total number of deaths each year and average age at death
SELECT COUNT(*) AS total_deaths, AVG(age) AS avg_age
FROM (
        SELECT AGE(Death.d_date, Goat.birthdate) AS age
        FROM Goat JOIN Death ON Goat.goat_id = Death.goat_id);

-- Avg deaths per season
WITH Spring AS (
        SELECT ROUND(AVG(spring_deaths), 2) AS avg_spring_deaths
        FROM (
        SELECT COUNT(*) AS spring_deaths
        FROM Death
        WHERE EXTRACT(MONTH FROM Death.d_date) IN (3, 4, 5)
        GROUP BY EXTRACT(YEAR FROM Death.d_date))
),
Summer AS (
        SELECT ROUND(AVG(summer_deaths), 2) AS avg_summer_deaths
        FROM (
        SELECT COUNT(*) AS summer_deaths
        FROM Death
        WHERE EXTRACT(MONTH FROM Death.d_date) IN (6, 7, 8)
        GROUP BY EXTRACT(YEAR FROM Death.d_date))),
Fall AS (
        SELECT ROUND(AVG(fall_deaths), 2) AS avg_fall_deaths
        FROM (
        SELECT COUNT(*) AS fall_deaths
        FROM Death
        WHERE EXTRACT(MONTH FROM Death.d_date) IN (9, 10, 11)
        GROUP BY EXTRACT(YEAR FROM Death.d_date))),
Winter AS (
        SELECT ROUND(AVG(winter_deaths), 2) AS avg_winter_deaths
        FROM (
        SELECT COUNT(*) AS winter_deaths
        FROM Death
        WHERE EXTRACT(MONTH FROM Death.d_date) IN (1, 2, 12)
        GROUP BY EXTRACT(YEAR FROM Death.d_date)))
SELECT Spring.avg_spring_deaths, Summer.avg_summer_deaths, Fall.avg_fall_deaths,
Winter.avg_winter_deaths
FROM Spring, Summer, Fall, Winter;
```

```
-- Deaths per season in a specific year
-- combines dec with jan/feb of the same year instead of the same winter
WITH Total AS (
   SELECT COUNT(*) AS total_deaths, EXTRACT(YEAR FROM Death.d_date) AS year
        FROM Death
        GROUP BY year),
Spring AS (
        SELECT COUNT(*) AS spring, EXTRACT(YEAR FROM Death.d_date) AS year
        FROM Death
        WHERE EXTRACT(MONTH FROM Death.d_date) IN (3, 4, 5)
        GROUP BY year),
Summer AS (
        SELECT COUNT(*) AS summer, EXTRACT(YEAR FROM Death.d_date) AS year
        FROM Death
        WHERE EXTRACT(MONTH FROM Death.d_date) IN (6, 7, 8)
        GROUP BY year),
Fall AS (
        SELECT COUNT(*) AS fall, EXTRACT(YEAR FROM Death.d_date) AS year
        FROM Death
        WHERE EXTRACT(MONTH FROM Death.d_date) IN (9, 10, 11)
        GROUP BY year),
Winter AS (
        SELECT COUNT(*) AS winter, EXTRACT(YEAR FROM Death.d_date) AS year
        FROM Death
        WHERE EXTRACT(MONTH FROM Death.d_date) IN (1, 2, 12)
        GROUP BY year)
SELECT Total.year, Total.total_deaths, Spring.spring, Summer.summer, Fall.fall, Winter.winter
FROM Total LEFT JOIN Spring ON Total.year = Spring.year LEFT JOIN Summer ON Total.year
= Summer.year LEFT JOIN Fall ON Total.year = Fall.year LEFT JOIN Winter ON Total.year =
Winter.year
ORDER BY Total.year;

-- Average weaning/winter/sale weight
--WITH Weaning AS (
        --????),
WITH Winter AS (
        SELECT AVG(CAST(alpha_value AS DECIMAL(4,1))) AS avg_winter_weight
        FROM Weight
        WHERE EXTRACT(MONTH FROM Weight.w_date) IN (1, 2, 12)),
Sale AS (
        SELECT AVG(CAST(last_weight AS DECIMAL(4,1))) AS avg_sale_weight
        FROM Goat
        WHERE goat_status = 'Sold')
```

```
--SELECT Weaning.avg_weaning_weight, Winter.avg_winter_weight, Sale.avg_sale_weight
SELECT Winter.avg_winter_weight, Sale.avg_sale_weight
--FROM Weaning, Winter, Sale;
FROM Winter, Sale;

-- Average weaning/winter/sale weight by birthweight
--WITH Weaning AS (
        --????),
WITH Winter AS (
        SELECT AVG(CAST(Weight.alpha_value AS DECIMAL(4,1))) AS avg_winter_weight
        FROM Weight JOIN Goat ON Weight.goat_id = Goat.goat_id
        WHERE EXTRACT(MONTH FROM Weight.w_date) IN (1, 2, 12)
        GROUP BY Goat.birthweight),
Sale AS (
        SELECT AVG(CAST(Goat.last_weight AS DECIMAL(4,1))) AS avg_sale_weight
        FROM Goat
        WHERE goat_status = 'Sold'
        GROUP BY Goat.birthweight)
--SELECT Weaning.avg_weaning_weight, Winter.avg_winter_weight, Sale.avg_sale_weight
SELECT Winter.avg_winter_weight, Sale.avg_sale_weight
--FROM Weaning, Winter, Sale;
FROM Winter, Sale;
```


**Task 5:** ***(Explain how your views and queries defined in tasks 3 and 4 above implement your use cases.)***

In task 3 and 4, we get userInput for the specific years they want to see the correlation between. We create a Death Table which is composed of goat ID, session, trait, and date. From that Death Table, we work to get the average deaths each year and the number of deaths each year. We then use this date to specify which season they want to search for and return the average, total, and difference in the death rates between the years. To calculate these, we

would have a script with equations utilizing the data extracted from our queries. To determine season-specific information, we use the results from the Death Table to find death per season from the Goat table, once again performing a series of equations to calculate certain trends or correlations. After getting this data, the Actor will be able to fine tune the specifications so they can further narrow down their search. This repeated task will be handled through our scripts. We are getting the info we need through the queries and our script will further use these results to find the correlation between the dates and find the trends in the deaths.

-- TO DO:
-- Difference between two (or more?) years specified by user
-- Average growth rate of goats
    -- total and grouped by birthweight
-- Difference between birthweight classes at different ages/stages (look for pattern/correlation)
-- Predict future weights of specified birthweight