EVILCORP

Lluc Galceran Jodar
28/05/2023

Índice

1- Introducción

Un Evil Corp es una empresa ficticia o real que se dedica a actividades ilegales,muy poco éticas las cuales solo van en busca de dinero y poder.
Estas empresas a menudo tienen objetivos nefastos y persiguen sus propios intereses sin tener en cuenta las consecuencias para la sociedad o el medio ambiente.

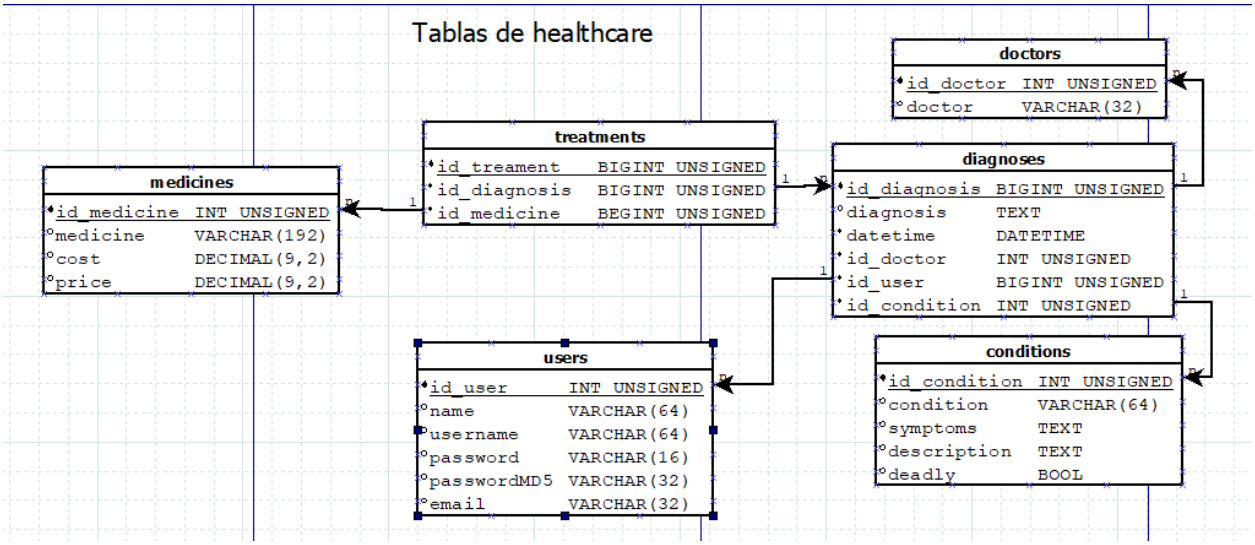
Los objetivos de una evil corp suelen estar centrados en obtener el control y la dominación en diferentes ámbitos. Pueden buscar el monopolio en un sector determinado, manipular mercados para maximizar sus ganancias. En nuestro caso nuestra Evil Corp está separada por 3 partes, la parte de las conspiraciones (**WORLD)**, en la parte de la salud (**HEALTH CARE)** y en la parte de las viviendas (**REAL STATE)**.

Estas corporaciones suelen utilizar tácticas despiadadas para lograr sus objetivos, como el soborno, la corrupción o el chantaje. También pueden emplear la tecnología y la innovación para desarrollar herramientas o productos peligrosos que pongan en riesgo la privacidad, la seguridad o la salud de las personas como en nuestro caso.

2- Health Care

En este apartado veremos la parte de la salud, veremos varias tablas con las medicinas los diagnósticos, los tratamientos entre otras, también veremos lo que de verdad le interesa a nuestra empresa que es el dinero que saca por cada medicina que vende.

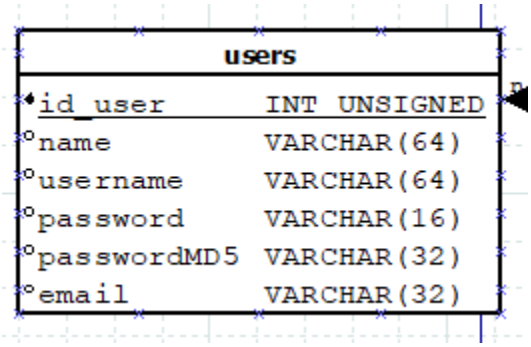
2.1- Diagrama Health Care



2.2- Explicación de las Tablas Health Care:

2.2.1- Users:

Esta tabla está en todas las partes de este proyecto, es la tabla principal, lo que pasa es que no es de este proyecto por eso no cuenta como tabla de la parte de Health Care, ya que diremos que es una tabla global.



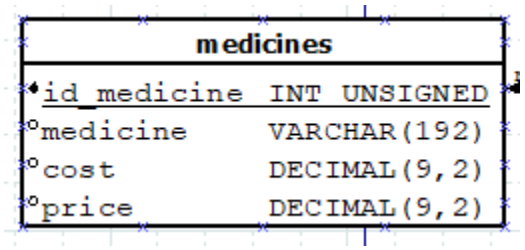
2.2.2- Medicines:

Esta tabla almacena las medicinas que Evil Corp tiene disponible, esta tiene los siguientes campos:

Medicine: Este campo es un **VARCHAR(192)** porque es el campo que almacena el nombre de la medicina y como ya sabemos hay medicinas con nombres muy largos de hecho la medicina con el nombre más largo tiene 192 caracteres.

Cost: Este campo es **Decimal (9,2)**, ya que representa el coste de los medicamentos para Evil Corp, él (9,2) significa que la columna "cost" puede almacenar números con hasta 9 dígitos en total, y de esos 9 dígitos, 2 se reservan para la parte decimal.

Price: Este campo es **Decimal (9,2)**, ya que representa el precio al cual Evil Corp vende estos productos al mercado, él (9,2) significa que la columna "cost" puede almacenar números con hasta 9 dígitos en total, y de esos 9 dígitos, 2 se reservan para la parte decimal.



2.2.3- Diagnoses:

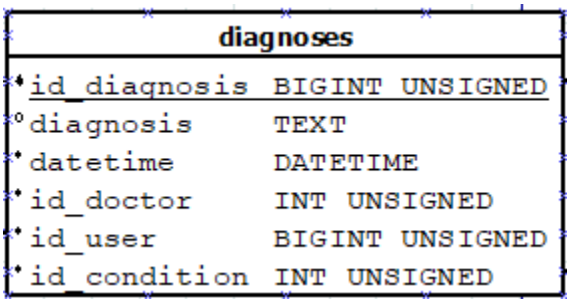
Esta tabla es donde se guardan los datos de una diagnóstico completo el cual tiene estos campos:

Diagnosis: Esta es tipo **TEXT, ya que** es una pequeña explicación del Diagnóstico completo i requiere de demasiados caracteres para ser un **CHAR o VARCHAR**

Id_user: Este campo hace referencia al identificador de la tabla **USERS** mediante una **FOREIGN KEY**, sirve para poder vincular un usuario a una diagnóstico en concreto.

Id_doctor: Este campo hace referencia al identificador de la tabla **DOCTORS** mediante una **FOREIGN KEY,** nos sirve para poder vincular un doctor a una diagnóstico en concreto.

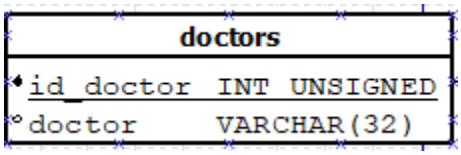
Id_condition: Este campo hace referencia al identificador de la tabla **CONDITIONS** mediante una **FOREIGN KEY**, nos sirve para poder vincular una condición a un diagnostico concreto.



2.2.4- Doctors:

En esta tabla se almacenan los nombres de todos los doctores que están dentro de nuestra Evil corp, esta tabla tiene solo este campo:

Doctor: Este campo es **VARCHAR(32)** este indica él nombre del doctor el cual tenemos.

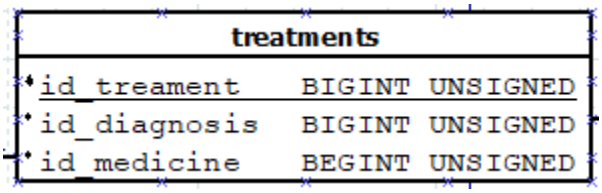


2.2.5- Treatments:

En esta tabla se almacenan las vinculaciones entre los **medicamentos y los diagnósticos**, podemos decir que si juntamos el medicamento más un diagnóstico tenemos un **tratamiento**.

Id_diagnosis: Este campo hace referencia al identificador de la tabla **DIAGNOSES** mediante una **FOREIGN KEY,** nos sirve para poder vincular un diagnóstico a un tratamiento especifico.

Id_medicine: Este campo hace referencia al identificador de la tabla **MEDICINES** mediante una **FOREIGN KEY,** esto nos permite poder vincular una medicina a un tratamiento especifico.



2.2.6- Conditions

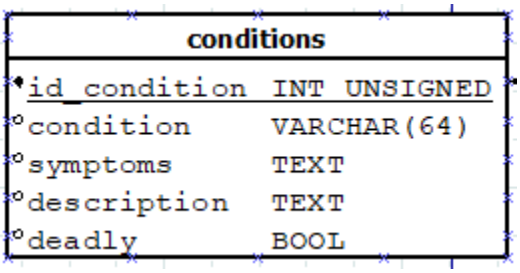
Esta tabla es donde se almacenan todas las distintas enfermedades que puede haber o las más importantes para que Evil Corp saque su dinérico.

Condición: Este campo hace referencia al nombre de la enfermedad, por eso es un **VARCHAR(64)**, ya que puede haber enfermedades con nombres bastante largos.

Symptoms: Este campo almacena los síntomas de la enfermedad correspondiente, es de tipo **TEXT,** ya que los síntomas son como una pequeña descripción y con los otros tipos no habría suficiente espacio.

Description: Este campo almacena las descripciones de las enfermedades por esto es tipo **TEXT.**

Deadly: Este campo hace referencia a si una enfermedad es mortal, por esto es un **BOOLEAN**, ya que solo puede ser o **TRUE** o **FALSE.**



2.3- Script Health Care:

2.3.1- Drops:

```
DROP TABLE IF EXISTS treatments;
DROP TABLE IF EXISTS diagnoses;
DROP TABLE IF EXISTS doctors;
DROP TABLE IF EXISTS conditions;
DROP TABLE IF EXISTS medicines;
```

2.3.2- Creates:

```
CREATE TABLE medicines(
    id_medicine INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    medicine VARCHAR(192) NOT NULL,
    cost DECIMAL(9,2),
    price DECIMAL(9,2)
);
CREATE TABLE conditions(
    id_condition INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    symptom TEXT NOT NULL,
    description TEXT NOT NULL,
    deadly BOOL NOT NULL
);

CREATE TABLE doctors(
    id_doctor INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    doctor VARCHAR(32)
);

CREATE TABLE diagnoses(
    id_diagnosis INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    diagnoses TEXT NOT NULL,
    id_user INT UNSIGNED NOT NULL,
    id_condition INT UNSIGNED NOT NULL,
    id_doctor INT UNSIGNED NOT NULL,
    FOREIGN KEY (id_user) REFERENCES users(id_user),
    FOREIGN KEY (id_condition) REFERENCES conditions(id_condition),
    FOREIGN KEY (id_doctor) REFERENCES doctors(id_doctor)
);
```

```
CREATE TABLE treatments(
    id_treatments INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    id_diagnosis INT UNSIGNED NOT NULL,
    id_medicine INT UNSIGNED NOT NULL,
    FOREIGN KEY (id_treatments) REFERENCES diagnoses(id_diagnosis),
    FOREIGN KEY (id_medicine) REFERENCES medicines(id_medicine)
);
```

2.3.3- Inserts:

```
--------- MEDICINES ---------
INSERT INTO medicines (medicine, cost, price)
VALUES ('Aspirin', 5.99, 9.99);

INSERT INTO medicines (medicine, cost, price)
VALUES ('Paracetamol', 3.49, 7.99);

INSERT INTO medicines (medicine, cost, price)
VALUES ('Ibuprofen', 4.99, 8.99);

INSERT INTO medicines (medicine, cost, price)
VALUES ('Amoxicillin', 7.99, 12.99);

INSERT INTO medicines (medicine, cost, price)
VALUES ('Omeprazole', 6.49, 10.99);

--------- CONDITIONS ---------
INSERT INTO conditions (name, symptom, description, deadly)
VALUES ('Flu', 'Fever, cough, sore throat', 'Common viral infection',
FALSE);

INSERT INTO conditions (name, symptom, description, deadly)
VALUES ('Diabetes', 'Increased thirst, frequent urination', 'Chronic
metabolic disease', TRUE);

INSERT INTO conditions (name, symptom, description, deadly)
VALUES ('Hypertension', 'High blood pressure', 'Cardiovascular disorder',
FALSE);

INSERT INTO conditions (name, symptom, description, deadly)
```

```
VALUES ('Asthma', 'Shortness of breath, wheezing', 'Chronic respiratory
disease', FALSE);

INSERT INTO conditions (name, symptom, description, deadly)
VALUES ('Cancer', 'Unexplained weight loss, fatigue', 'Malignant tumor
growth', TRUE);

--------- DOCTORS ---------
INSERT INTO doctors (doctor)
VALUES ('Dr. Smith');

INSERT INTO doctors (doctor)
VALUES ('Dr. Johnson');

INSERT INTO doctors (doctor)
VALUES ('Dr. Anderson');

INSERT INTO doctors (doctor)
VALUES ('Dr. Davis');

INSERT INTO doctors (doctor)
VALUES ('Dr. Wilson');

--------- DIAGNOSES ---------
INSERT INTO diagnoses (diagnoses, id_user, id_condition, id_doctor)
VALUES ('You have the flu. Get plenty of rest and drink fluids.', 1, 1,
1);
INSERT INTO diagnoses (diagnoses, id_user, id_condition, id_doctor)
VALUES ('You have diabetes. Follow a healthy diet and take medication as
prescribed.', 2, 2, 2);
INSERT INTO diagnoses (diagnoses, id_user, id_condition, id_doctor)
VALUES ('You have hypertension. Monitor your blood pressure regularly.',
3, 3, 3);
INSERT INTO diagnoses (diagnoses, id_user, id_condition, id_doctor)
VALUES ('You have asthma. Use an inhaler as needed.', 4, 4, 4);
INSERT INTO diagnoses (diagnoses, id_user, id_condition, id_doctor)
VALUES ('You have cancer. Further tests and treatment options will be
discussed.', 5, 5, 5);

--------- TREATMENTS ---------
```

```
INSERT INTO treatments (id_diagnosis, id_medicine)
VALUES (1, 1);

INSERT INTO treatments (id_diagnosis, id_medicine)
VALUES (2, 3);

INSERT INTO treatments (id_diagnosis, id_medicine)
VALUES (3, 2);

INSERT INTO treatments (id_diagnosis, id_medicine)
VALUES (4, 5);

INSERT INTO treatments (id_diagnosis, id_medicine)
VALUES (5, 4);
```

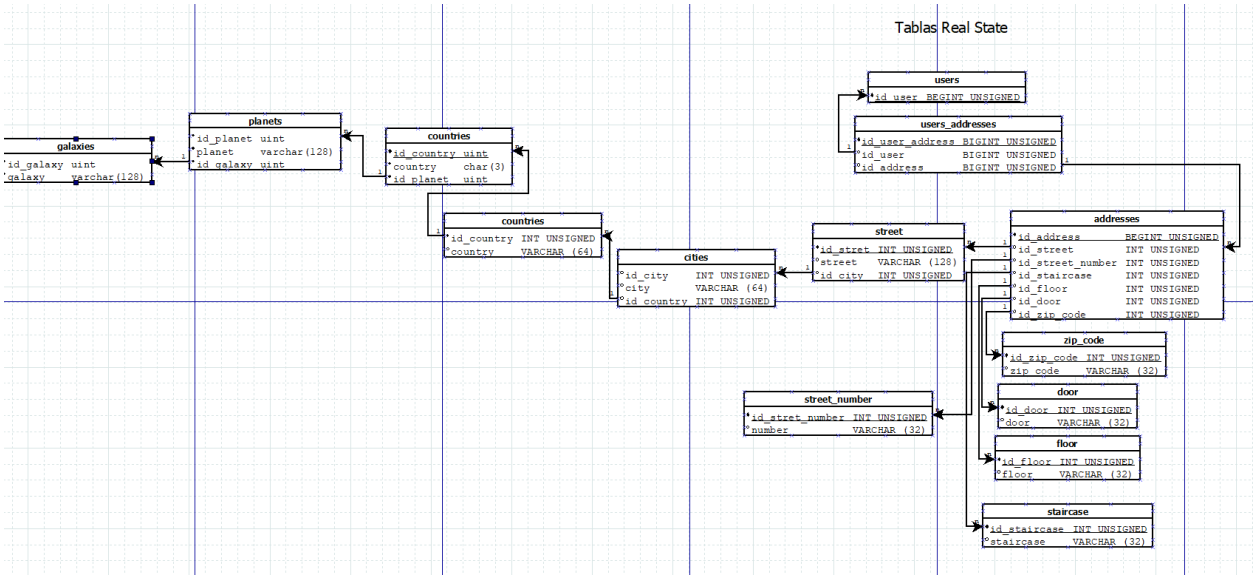
2.3.4- Views:

```
--------- VIEW PROFIT MEDICINES ---------
create view costes as select treatments.id_medicine,
COUNT(medicine) medicines, cost_production,
SUM(cost_production) total_cost_production, cost_sell,
SUM(cost_sell) total_cost_sell from medicines LEFT JOIN
treatments on medicines.id_medicine=treatments.id_medicine group
by cost_production;
```

3- Real State

En este apartado veremos la parte de la vivienda i de toda la información que tenemos sobre cada usuario, en esta parte tenemos estas tablas: streets, streets numbers, addresses, planets entre otras, este apartado es el más extenso y el más complejo, también veremos aquí las diferentes funciones y procedimientos creados para varios usos.

3.1- Diagrama



3.2- Explicación de las tablas Real State:

3.2.1-Addresses:

Esta tabla es la encargada de almacenar el conjunto de calle, número de calle, piso, puerta y código postal.

Id_street: Este campo hace referencia al identificador de la tabla **STREETS** mediante una **FOREIGN KEY**, esto nos sirve para vincular una calle a una dirección.

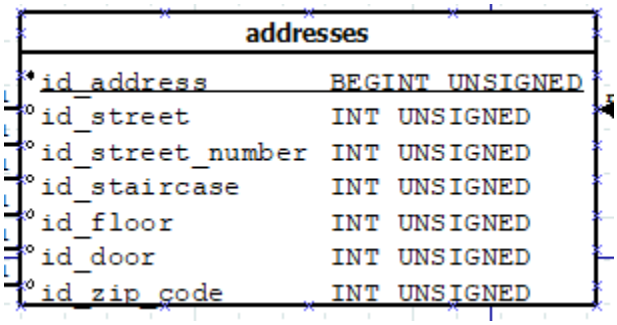
Id_street_num: Este campo hace referencia al identificador de la tabla **STREETS_NUM** mediante una **FOREIGN KEY**, esto es porque vinculamos un número calle a una dirección.

Id_staircase: Este campo hace referencia al identificador de la tabla **STAIRCASES** mediante una **FOREIGN KEY**, nos sirve para vincular un número de escalera a una dirección.

Id_floor: Este campo hace referencia al identificador de la tabla **FLOORS** mediante una **FOREIGN KEY**, nos sirve para vincular una planta a una dirección.

Id_door: Este campo hace referencia al identificador de la tabla **DOORS** mediante una **FOREIGN KEY,** esto es porque queremos saber a qué puerta pertenece esta dirección.

Id_zipcode: Este campo hace referencia al identificador de la tabla **ZIPCODES** mediante una **FOREIGN KEY**, esto es porque queremos saber qué código postal pertenece esta dirección.

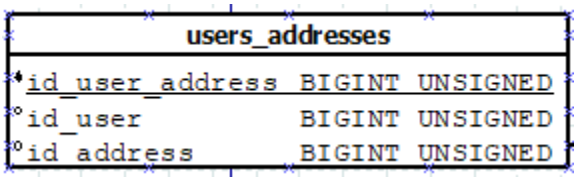


3.2.2- User_addresses:

Esta tabla es una tabla intermedia entre la tabla **users** y la tabla **addresses**, nos sirve para poder vincular a un usuario a su dirección correspondiente. Esta tabla está compuesta por:

Id_user: Este campo se refiere al identificador de la tabla **USERS** mediante una **FOREIGN KEY**, y no sirve para saber a qué usuario pertenece una dirección concreta.

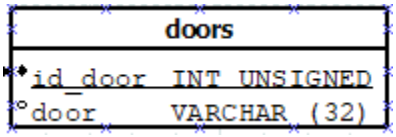
Id_address: Este campo se refiere al identificador de la tabla **ADDRESSES** mediante una **FOREIGN KEY,** nos sirve para saber que direcciones pertenecen a cada usuario.



3.2.3- Doors:

Esta tabla se encara de almacenar los números de puerta y tienes estos campos:

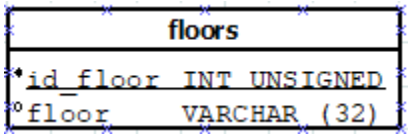
Door: Este campo es el que almacena el numero de la puerta, es tipo **VARCHAR(8)** porque solo necesitamos almacenar esta cantidad de caracteres.



3.2.4- Floors:

Esta tabla se encara de almacenar los pisos y tienes estos campos:

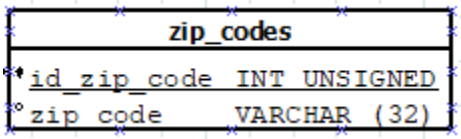
Floor: Este campo es el que almacena el piso, es tipo **VARCHAR(32)** porque solo necesitamos almacenar números y no superara esta cantidad de caracteres.



3.2.5- Zipcodes:

Esta tabla se encara de almacenar los codigos postal y tienes estos campos:

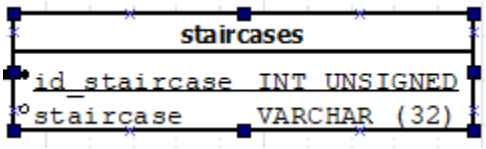
Zipcode: Este campo es el que almacena el numero de la puerta, es tipo **VARCHAR(32)** porque los codigos postales más largos solo tienen 32 caracteres.



3.2.6- Staircases:

Esta tabla se encara de almacenar los números de escalera y tienes estos campos:

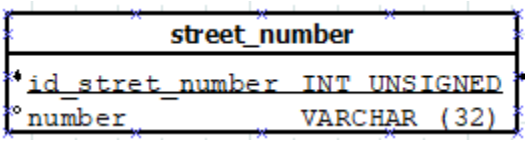
Staircase: Este campo es el que almacena el numero de la puerta, es tipo **VARCHAR(32)** porque con 32 caracteres tenemos suficientes para almacenar todas las escaleras.



3.2.7- Streets_Numbers:

Esta tabla se encara de almacenar los números de calle y tienes estos campos:

Street_number: Este campo es el que almacena el numero de la puerta, es tipo **VARCHAR(32)** porque con 32 caracteres tenemos suficientes para almacenar todos los números de calle posibles.

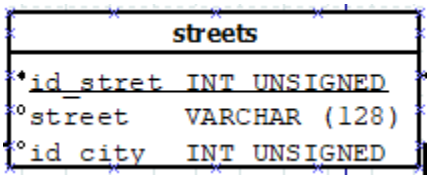


3.2.8- Streets:

Esta tabla se encara de almacenar los números de calle y tienes estos campos:

Street: Este campo es el que almacena nombre de la calle, es tipo **VARCHAR(128)** porque con 128 caracteres tenemos suficientes para almacenar el nombre entero de la calle.

Id_city : Este campo se refiere al identificador de la tabla **CITIES** mediante una **FOREIGN KEY,** nos sirve para saber a qué ciudad pertenecen esta calle.

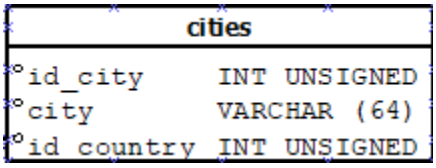


3.2.9- Cites:

Este campo almacena las distintas ciudades de cada país.

City: Este campo almacena el nombre de la ciudad, por esto es **VARCHAR(64)**, ya que puede haber ciudades que tengan nombres bastante largos.

Id_country: Este campo se refiere al identificador de la tabla **COUNTRIES** mediante una **FOREIGN KEY,** nos sirve para saber en qué país se encuentra esta ciudad.

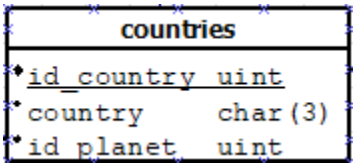


3.2.9- Countries:

Este campo almacena los distintos países de cada planeta.

Country: Este campo almacena el nombre del país, por esto es **VARCHAR(64)**, ya que puede haber países que tengan nombres bastante largos.

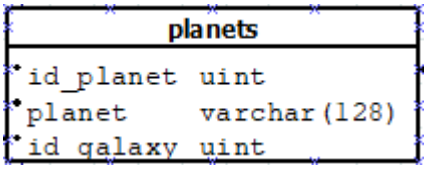
Id_planet: Este campo se refiere al identificador de la tabla **PLANETS** mediante una **FOREIGN KEY,** nos sirve para saber en qué planeta se encuentra este país.



3.2.9- Planets:

Este campo almacena los distintos países de cada planeta.

Planet: Este campo almacena el nombre del Planeta, por esto es **VARCHAR(128)**, ya que puede haber países que tengan nombres bastante largos.



3.2.10- Users:

Esta tabla está en todas las partes de este proyecto, es la tabla principal y su estructura no es la indicada en esta parte, ya que es un poco más amplia, lo que pasa es que no es de este proyecto por eso no la pongo.

3.3- Script:

3.3.1- Drops:

En los drops de esta parte debemos ir con cuidado ya que al haber muchas foreign keys se nos puede complicar.

```
-------- REAL STATE DROPS --------
DROP TABLE IF EXISTS adresses;
DROP TABLE IF EXISTS zip_codes;
DROP TABLE IF EXISTS doors;
DROP TABLE IF EXISTS floors;
DROP TABLE IF EXISTS staircases;
DROP TABLE IF EXISTS street_numbers;
DROP TABLE IF EXISTS streets;
DROP TABLE IF EXISTS cities;
DROP TABLE IF EXISTS countries;
DROP TABLE IF EXISTS planets;
```

3.3.2- Creates:

Aquí debemos crear por orden inverso a los drops para poder hacer bien la foreing keys.

```
-------- CREATE REAL STATE --------
CREATE TABLE planets(
    id_planet INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    planet VARCHAR(64)
);

CREATE TABLE countries(
    id_country INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    country VARCHAR(64),
    id_planet INT UNSIGNED NOT NULL,
    FOREIGN KEY (id_planet) REFERENCES planets(id_planet)
);

CREATE TABLE cities(
    id_city INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    city VARCHAR(64),
    id_country INT UNSIGNED NOT NULL,
    FOREIGN KEY (id_country) REFERENCES countries(id_country)
);
```

```sql
CREATE TABLE streets(
    id_street INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    street VARCHAR(128),
    id_city INT UNSIGNED NOT NULL,
    FOREIGN KEY (id_city) REFERENCES cities(id_city)
);

CREATE TABLE street_numbers(
    id_street_number INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    street_number VARCHAR(32)
);

CREATE TABLE staircases(
    id_staircase INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    staircase VARCHAR(32)
);

CREATE TABLE floors(
    id_floor INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    flooor VARCHAR(32)
);

CREATE TABLE doors(
    id_door INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    door VARCHAR(32)
);

CREATE TABLE zip_codes(
    id_zip_code INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    zip_code VARCHAR(32)
);

CREATE TABLE adresses(
    id_adress INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    id_street INT UNSIGNED NOT NULL,
    id_street_number INT UNSIGNED NOT NULL,
    id_staircase INT UNSIGNED NOT NULL,
    id_floor INT UNSIGNED NOT NULL,
    id_zip_code INT UNSIGNED NOT NULL,
```

```
    id_user INT UNSIGNED NOT NULL,
    FOREIGN KEY (id_street) REFERENCES streets(id_street),
    FOREIGN KEY (id_street_number) REFERENCES
street_numbers(id_street_number),
    FOREIGN KEY (id_staircase) REFERENCES staircases(id_staircase),
    FOREIGN KEY (id_floor) REFERENCES floors(id_floor),
    FOREIGN KEY (id_zip_code) REFERENCES zip_codes(id_zip_code),
    FOREIGN KEY (id_user) REFERENCES users(id_user)
);
```

3.3.3- Inserts:

```
--------- PLANETS ---------
INSERT INTO planets (planet)
VALUES ('Mercury');

INSERT INTO planets (planet)
VALUES ('Venus');

INSERT INTO planets (planet)
VALUES ('Earth');

INSERT INTO planets (planet)
VALUES ('Mars');

INSERT INTO planets (planet)
VALUES ('Jupiter');

--------- COUNTRIES ---------
INSERT INTO countries (country, id_planet)
VALUES ('United States', 3);

INSERT INTO countries (country, id_planet)
VALUES ('Canada', 3);

INSERT INTO countries (country, id_planet)
VALUES ('United Kingdom', 3);

INSERT INTO countries (country, id_planet)
VALUES ('Germany', 3);
```

```
INSERT INTO countries (country, id_planet)
VALUES ('Australia', 3);

--------- CITIES ---------
INSERT INTO cities (city, id_country)
VALUES ('New York', 1);

INSERT INTO cities (city, id_country)
VALUES ('Toronto', 2);

INSERT INTO cities (city, id_country)
VALUES ('London', 3);

INSERT INTO cities (city, id_country)
VALUES ('Berlin', 4);

INSERT INTO cities (city, id_country)
VALUES ('Sydney', 5);

--------- STREETS ---------
INSERT INTO streets (street, id_city)
VALUES ('Main Street', 1);

INSERT INTO streets (street, id_city)
VALUES ('Queen Street', 2);

INSERT INTO streets (street, id_city)
VALUES ('Oxford Street', 3);

INSERT INTO streets (street, id_city)
VALUES ('Unter den Linden', 4);

INSERT INTO streets (street, id_city)
VALUES ('George Street', 5);

--------- STREETS NUMBERS ---------
INSERT INTO street_numbers (street_number)
VALUES ('123');

INSERT INTO street_numbers (street_number)
```

```
VALUES ('456');

INSERT INTO street_numbers (street_number)
VALUES ('789');

INSERT INTO street_numbers (street_number)
VALUES ('10A');

INSERT INTO street_numbers (street_number)
VALUES ('20B');

--------- STAIRCASES ---------
INSERT INTO staircases (staircase)
VALUES ('A');

INSERT INTO staircases (staircase)
VALUES ('B');

INSERT INTO staircases (staircase)
VALUES ('C');

INSERT INTO staircases (staircase)
VALUES ('D');

INSERT INTO staircases (staircase)
VALUES ('E');

--------- FLOORS ---------
INSERT INTO floors (flooor)
VALUES ('1st Floor');

INSERT INTO floors (flooor)
VALUES ('2nd Floor');

INSERT INTO floors (flooor)
VALUES ('3rd Floor');

INSERT INTO floors (flooor)
VALUES ('4th Floor');
```

```
INSERT INTO floors (flooor)
VALUES ('5th Floor');

--------- DOORS ---------
INSERT INTO doors (door)
VALUES ('Door 1');

INSERT INTO doors (door)
VALUES ('Door 2');

INSERT INTO doors (door)
VALUES ('Door 3');

INSERT INTO doors (door)
VALUES ('Door 4');

INSERT INTO doors (door)
VALUES ('Door 5');

--------- ZIP CODES ---------
INSERT INTO zip_codes (zip_code)
VALUES ('12345');

INSERT INTO zip_codes (zip_code)
VALUES ('67890');

INSERT INTO zip_codes (zip_code)
VALUES ('24680');

INSERT INTO zip_codes (zip_code)
VALUES ('13579');

INSERT INTO zip_codes (zip_code)
VALUES ('97531');

--------- ADDRESSES ---------
INSERT INTO adresses (id_street, id_street_number, id_staircase, id_floor,
id_zip_code, id_user)
VALUES (1, 1, 1, 1, 1, 1);
```

```sql
INSERT INTO adresses (id_street, id_street_number, id_staircase, id_floor,
id_zip_code, id_user)
VALUES (2, 2, 2, 2, 2, 2);

INSERT INTO adresses (id_street, id_street_number, id_staircase, id_floor,
id_zip_code, id_user)
VALUES (3, 3, 3, 3, 3, 3);

INSERT INTO adresses (id_street, id_street_number, id_staircase, id_floor,
id_zip_code, id_user)
VALUES (4, 4, 4, 4, 4, 4);

INSERT INTO adresses (id_street, id_street_number, id_staircase, id_floor,
id_zip_code, id_user)
VALUES (5, 5, 5, 5, 5, 5);
```

3.3.4- Views:

Esta view lo que hace es calcular el profit que saca cada planeta por las medicinas que vende:

```
CREATE VIEW profit_4_planet AS
SELECT planets.id_planet, planets.planet, SUM(medicines.price) AS profit
FROM planets

LEFT JOIN countries
    ON planets.id_planet=countries.id_planet
LEFT JOIN cities
    ON countries.id_country=cities.id_country
LEFT JOIN streets
    ON cities.id_city=streets.id_city
LEFT JOIN adresses
    ON streets.id_street=adresses.id_street
LEFT JOIN users
    ON adresses.id_user=users.id_user
LEFT JOIN diagnoses
    ON users.id_user=diagnoses.id_user
LEFT JOIN treatments
    ON diagnoses.id_diagnosis=treatments.id_diagnosis
LEFT JOIN medicines
    ON treatments.id_medicine=medicines.id_medicine


GROUP BY planets.id_planet, planets.planet;
```

Esta view lo que hace es contar cuantas personas hay en cada planeta:

```
CREATE VIEW persones_per_planeta AS
SELECT planets.id_planet, planets.planet, COUNT(users.id_user) AS personas
FROM planets

LEFT JOIN countries
    ON planets.id_planet=countries.id_planet
LEFT JOIN cities
    ON countries.id_country=cities.id_country
LEFT JOIN streets
    ON cities.id_city=streets.id_city
LEFT JOIN adresses
    ON streets.id_street=adresses.id_street
LEFT JOIN users
    ON adresses.id_user=users.id_user

GROUP BY planets.id_planet, planets.planet;
```

Esta view lo que hace es contar cuantas direcciones hay por cada planeta:

```
CREATE VIEW planet_addresses AS SELECT planets.planet,
COUNT(*) planet_addresses FROM planets, addresses, users,
users_planets, users_addresses WHERE
users.id_user=users_planets.id_user AND
users_planets.id_planet=planets.id_planet AND
users.id_user=users_addresses.id_user AND
addresses.id_address=users_addresses.id_address GROUP BY
planets.planet;
```

3.3.5- Functions:

Esta funcion lo que hace es añadir una ciudad con su pais y planeta correspòndiente en la tabla de cities:

```
DELIMITER $$
CREATE FUNCTION get_city(city_name VARCHAR(64),
country_name VARCHAR(64), planet_name VARCHAR(64))
RETURNS INT UNSIGNED
BEGIN
DECLARE planet_id INT UNSIGNED;
DECLARE country_id INT UNSIGNED;
DECLARE city_id INT UNSIGNED;
SELECT id_planet INTO planet_id FROM planets WHERE name =
planet_name;
IF planet_id IS NULL THEN
INSERT INTO planets (name) VALUES (planet_name);
SET planet_id = LAST_INSERT_ID();
END IF;
SELECT id_country INTO country_id FROM countries WHERE
country = country_name AND id_planet = planet_id;
IF country_id IS NULL THEN
INSERT INTO countries (country, id_planet) VALUES
(country_name, planet_id);
SET country_id = LAST_INSERT_ID();
END IF;
SELECT id_city INTO city_id FROM cities WHERE city =
city_name AND id_country = country_id;
IF city_id IS NULL THEN
INSERT INTO cities (city, id_country) VALUES (city_name,
country_id);
SET city_id = LAST_INSERT_ID();
ELSE
SET city_id = (SELECT id_city FROM cities WHERE city =
city_name AND id_country = country_id LIMIT 1);
END IF;
RETURN city_id;
END$$
DELIMITER ;
```

Esta función lo que hace es escoger un usuario al azar de un planeta que le pasemos:

```
DROP FUNCTION IF EXISTS randomuser;

DELIMITER $$

CREATE FUNCTION randomuser(planeta VARCHAR(64))
RETURNS VARCHAR(64)
BEGIN
    DECLARE random_user VARCHAR(64);

    SELECT username INTO random_user FROM users

    LEFT JOIN adresses ON users.id_user=adresses.id_user
    LEFT JOIN streets ON adresses.id_street=streets.id_street
    LEFT JOIN cities ON streets.id_city=cities.id_city
    LEFT JOIN countries ON cities.id_country=countries.id_country
    LEFT JOIN planets ON countries.id_planet=planets.id_planet
    WHERE planets.planet=planeta

    ORDER BY RAND()
    LIMIT 1;

    RETURN random_user;

END $$

DELIMITER ;
```

3.3.6- Procedures:

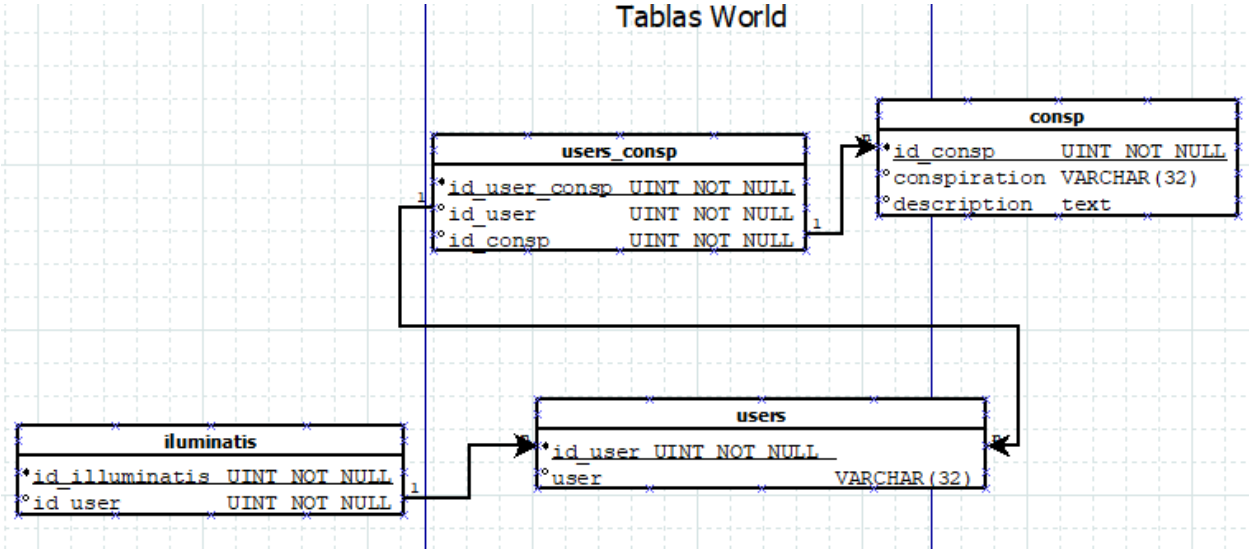
Este procedimiento lo que hace es matar al usuario que le passamos:

```
DROP PROCEDURE IF EXISTS kill_user;

DELIMITER $$

CREATE PROCEDURE kill_user(IN username_proc VARCHAR(64))
BEGIN
    DECLARE id_user_proc INT UNSIGNED;
    DECLARE user_proc VARCHAR(64);
    DECLARE contador INT;
    DECLARE IsDead INT;

    SELECT COUNT(*) INTO contador FROM users WHERE username =
username_proc;
    ALTER TABLE users ADD COLUMN dead BOOLEAN DEFAULT FALSE;
    IF contador > 0 THEN

        SELECT id_user, name, dead INTO id_user_proc, user_proc, IsDead
FROM users WHERE username = username_proc;

        IF IsDead = FALSE THEN
            UPDATE users SET dead = 1 WHERE username = username_proc;

            SELECT CONCAT('El señor', username_proc,' ha sido assesinado
por Kennen Papu') AS output;
        ELSE
            SELECT CONCAT('Ya esta muerto no puedes volver a matara a: ',
username_proc) AS output;
        END IF;
    ELSE
        SELECT CONCAT('Usuario no encontrado, no es possible matarlo',
username_proc) AS output;
    END IF;
END $$
DELIMITER ;
```

4- World Conspirations

En esta parte vamos a ver la parte de las conspiraciónes como los illuminatis las chemtrails entre otras, esta parte es un poco más pequeña que las otras.

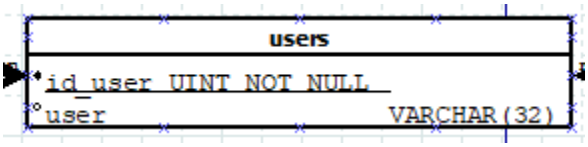
4.1- Diagrama:



4.2- Explicación de las tablas de World Conspirations

4.2.1-Users:

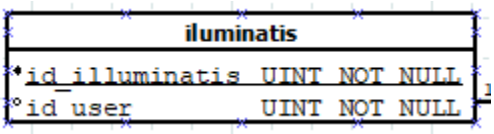
Esta tabla está en todas las partes de este proyecto, es la tabla principal y su estructura no es la indicada en esta parte, ya que es un poco más amplia, lo que pasa es que no es de este proyecto por eso no la pongo. Podemos decir que es una tabla global.



4.2.2- Iluminatis:

En esta tabla es donde se almacenaran los Usuarios que si son illuminati, esta tiene este campo:

Id_user: Este campo se refiere al identificador de la tabla **USERS** mediante una **FOREIGN KEY,** nos sirve para saber en qué usuario es illuminati.

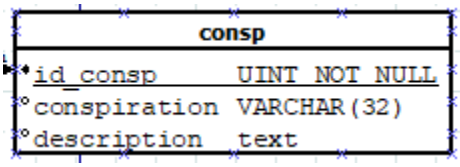


4.2.3- Conspirations:

En esta tabla es donde se almacenaran las distintas Conspiraciones y que usuarios si creen en ellas, esta tabla tiene estos campos:

Conspiration: Este campo se refiere al nombre de la conspiración por eso es **VARCHAR(32).**

Description: Este campo tiene la función de almacenar una pequeña descripción de en que consta la Conspiración, por esto es tipo **TEXT.**

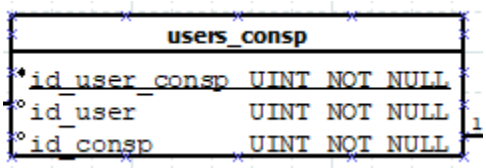


4.2.4- User_conspirations:

En esta tabla es donde se almacenaran las distintas Conspiraciones y que usuarios si creen en ellas, esta tabla tiene estos campos:

Id_user: Este campo se refiere al identificador de la tabla **USERS** mediante una **FOREIGN KEY,** nos sirve para saber qué usuario cree en una conspiración concreta.

Id_conspiration: Este campo se refiere al identificador de la tabla **CONSPIRATIONS** mediante una **FOREIGN KEY,** nos sirve para saber que conspiraciónes hay para poder relacionarlas con una usuario concreto.



4.3- Script:

4.3.1- Drops:

```
-------- CONSPIRATIONS DROPS --------
DROP TABLE IF EXISTS users_conspirations;
DROP TABLE IF EXISTS illuminats;
DROP TABLE IF EXISTS conspirations;
```

4.3.2- Creates:

```
-------- CREATE CONSPIRATIONS --------
CREATE TABLE conspirations(
    id_conspiration INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    conspiration TEXT NOT NULL
);

CREATE TABLE users_conspirations(
    id_user_conspiration INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    id_user INT UNSIGNED NOT NULL,
    id_conspiration INT UNSIGNED NOT NULL,
    FOREIGN KEY (id_user) REFERENCES users(id_user),
    FOREIGN KEY (id_conspiration) REFERENCES
conspirations(id_conspiration)
```

```
);

CREATE TABLE illuminatis(
    id_illuminati INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    id_user INT UNSIGNED NOT NULL,
    FOREIGN KEY (id_user) REFERENCES users(id_user)
);
```

4.3.3 Inserts:

```
--------- CONSPIRATIONS ---------
INSERT INTO conspirations (conspiration)
VALUES ('Conspiracy 1');

INSERT INTO conspirations (conspiration)
VALUES ('Conspiracy 2');

INSERT INTO conspirations (conspiration)
VALUES ('Conspiracy 3');

INSERT INTO conspirations (conspiration)
VALUES ('Conspiracy 4');

INSERT INTO conspirations (conspiration)
VALUES ('Conspiracy 5');

--------- USER CONSPIRATIONS ---------
INSERT INTO users_conspirations (id_user, id_conspiration)
VALUES (1, 1);

INSERT INTO users_conspirations (id_user, id_conspiration)
VALUES (2, 2);

INSERT INTO users_conspirations (id_user, id_conspiration)
VALUES (3, 3);

INSERT INTO users_conspirations (id_user, id_conspiration)
VALUES (4, 4);
```

```
INSERT INTO users_conspirations (id_user, id_conspiration)
VALUES (5, 5);

--------- ILLUMINATIS ---------
INSERT INTO illuminatis (id_user)
VALUES (1);

INSERT INTO illuminatis (id_user)
VALUES (2);

INSERT INTO illuminatis (id_user)
VALUES (3);

INSERT INTO illuminatis (id_user)
VALUES (4);

INSERT INTO illuminatis (id_user)
VALUES (5);
```

4.3.4 Views Conspirations:

Esta view lo que hace es mostrar todos los illuminatis:

```
CREATE VIEW ilumi_show AS SELECT users.id_user, users.name
FROM users LEFT JOIN iluminatis ON users.id_user =
iluminatis.id_user;
```

Esta view lo que hace es contar cuantos illuminatis hay:

```
CREATE VIEW ilumi_count AS SELECT COUNT(users.id_user)
users FROM users,ilumi_show;
```

4.3.5: Procedures Conspirations:

Este procedure lo que hace es crear una nueva conspiracion:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS procedimiencito;
CREATE PROCEDURE procedimiencito (id_conspiration INT, id_user INT,
id_telacreiste INT)
BEGIN
```

```
    DECLARE handlercito INT DEFAULT 1;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION SET handlercito = 0;

    START TRANSACTION;

    INSERT INTO users_conspiraciones (id_user, id_conspiration) VALUES
(id_user, id_conspirartion)

    IF id_telacreiste = 1 THEN
        COMMIT;
    ELSE
        ROLLBACK;
    END IF;

END $$

DELIMITER ;
```

5- Usuario La Parca

Aquí lo que hacemos es crear un usuario que solo tenga los permisos de SELECT y UPDATE que se llame parca lo que hace este es matar a los usuarios que le pasamos por parámetros añadiendo una columna nueva a la tabla users nombrada **dead** si esta es igual a TRUE quera decir que el usuario está muerto, pero si es igual a FALSE querrá decir que sigue vivo.

5.1- Creamos la parca:

```
CREATE USER 'parca'@'localhost' IDENTIFIED BY 'parca';
GRANT SELECT, UPDATE ON evilcorp.users TO 'parca'@'localhost';
```

5.2- Creamos el Procedimiento para matar:

```
DROP PROCEDURE IF EXISTS kill_user;

DELIMITER $$

CREATE PROCEDURE kill_user(IN username_proc VARCHAR(64))
BEGIN
    DECLARE id_user_proc INT UNSIGNED;
    DECLARE user_proc VARCHAR(64);
    DECLARE contador INT;
    DECLARE IsDead INT;

    SELECT COUNT(*) INTO contador FROM users WHERE username =
username_proc;
    ALTER TABLE users ADD COLUMN dead BOOLEAN DEFAULT FALSE;
    IF contador > 0 THEN

        SELECT id_user, name, dead INTO id_user_proc, user_proc, IsDead
FROM users WHERE username = username_proc;

        IF IsDead = FALSE THEN
            UPDATE users SET dead = 1 WHERE username = username_proc;

            SELECT CONCAT('El señor', username_proc,' ha sido assesinado
por Kennen Papu') AS output;
        ELSE
```

```
            SELECT CONCAT('Ya esta muerto no puedes volver a matara a: ',
username_proc) AS output;
        END IF;
    ELSE
        SELECT CONCAT('Usuario no encontrado, no es possible matarlo',
username_proc) AS output;
    END IF;
END $$
DELIMITER ;
```

5.3- Matar a un usuario al azar:

Aquí juntamos el procedimiento de matar a un usuairo con el de escoger un usuario random a l'azar de un planeta:

```
DROP PROCEDURE IF EXISTS matar_usuario_planeta;

DELIMITER $$

CREATE PROCEDURE matar_usuario_planeta(IN planesito VARCHAR(64))
BEGIN
    DECLARE user_name VARCHAR(64);
    SET user_name = randomuser(planesito);

    CALL kill_user(user_name);



END$$

DELIMITER ;
```