

# TABLE OF CONTENTS

EXECUTIVE SUMMARY………………………………………3

ENTITY RELATIONSHIP DIAGRAM…………………………4 TABLES………………………………………………………….5

VIEWS………………………………………….......................16

REPORTS…………………………………….........................18

STORED PROCEDURES.……………………………………20

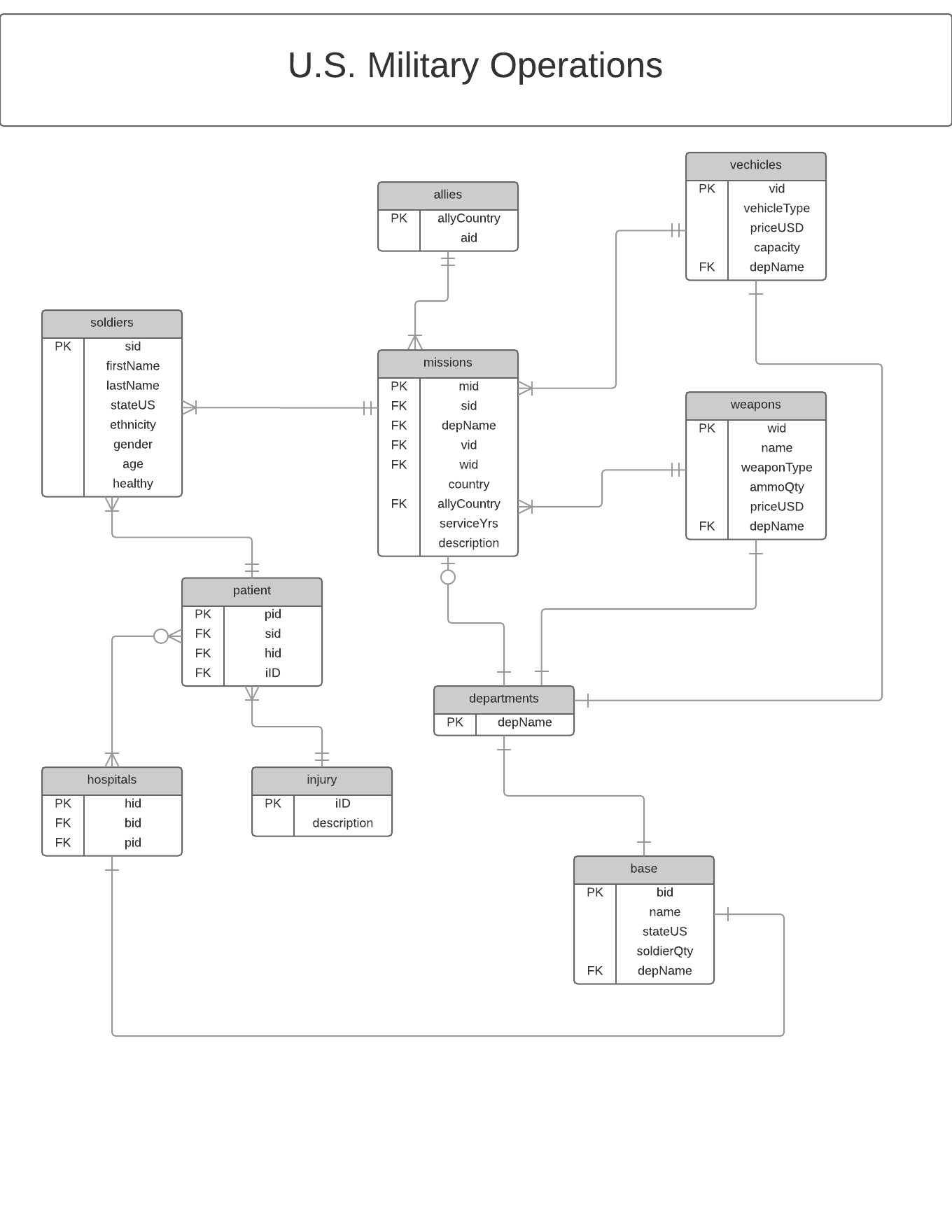
TRIGGERS…………………………………………................23

SECURITY…………………………………………………......24

NOTES—ISSUES—FUTURE………………………………..25

# EXECUTIVE SUMMARY

THIS document is a blueprint that displays the structure and entities contained in the design and input of a database for the U.S. Military Operations. The purpose of this database is to formulate a system that allows an organized method of documentation. At the dispense of this database, management will be able to obtain beneficial data from the implementation of queries that outputs stats and connections based off the requested specifics. U.S. Military Operations will be able track and produce accurate information by the implementation of this database. The end goals will be to provide a completely operating, normalized database that will fully support the U.S. Military Operations.



**Soldiers list all soldiers basic attributes functional dependencies**

CREATE TABLE soldiers ( sid char(3) not null unique,

|  |  |
| --- | --- |
| firstName | text not null, |
| lastName | text not null, |
| stateUS | text not null, |
| ethnicity | text not null, |
| gender | text not null, |
| age | integer not null, |
| healthy | text not null, |

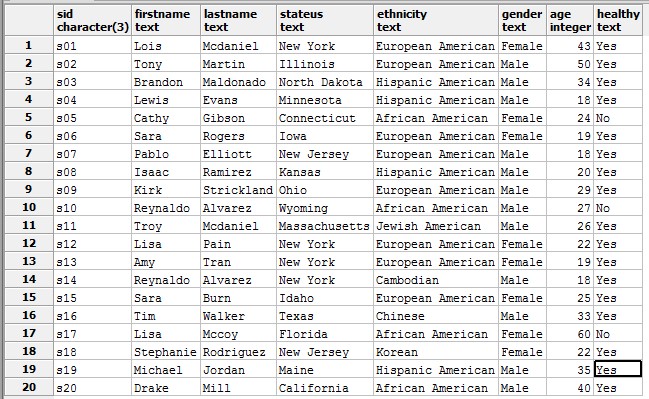
primary key(sid)

);

**functional dependencies** mid  firstName, lastName

stateUS, ethnicity, gender, age, healthy

## SAMPLE DATA ON FOLLOWING PAGE



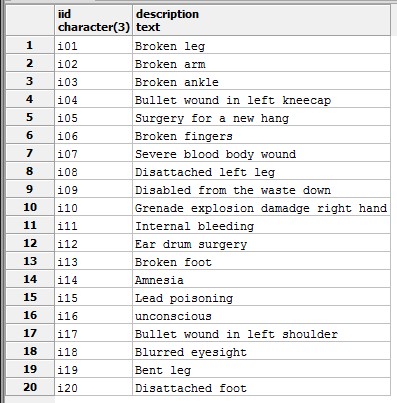
**Injury list all injury and basic attributes**

CREATE TABLE injury ( iID char(3) not null unique, description text not null, primary key(iID)

);

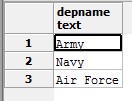
**functional dependencies**

iID  description



**Departments** **list all departments and basic attributes**

CREATE TABLE departments ( depName text not null unique, primary key(depName)

);

**functional dependencies**

depName  N / a

.

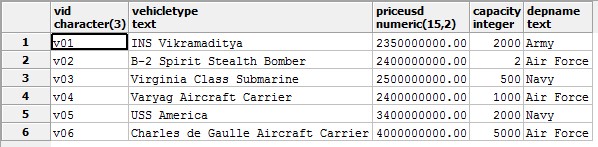
**Vehicles** **list all vehicles and basic attributes** CREATE TABLE vehicles (

|  |  |
| --- | --- |
| vid | char(3) not null unique, |
| vehicleType | text not null, |
| priceUSD | numeric(15, 2), |
| capacity | integer, |
| depName | text not null references departments(depName), |

primary key(vid)

);

**functional dependencies** vid  vehicleType, priceUSD, capacity, depName



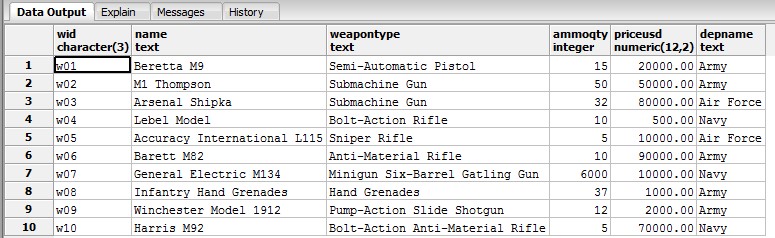
**Weapons** **list all weapons and basic attributes** CREATE TABLE weapons (

|  |  |
| --- | --- |
| wid | char(3) not null unique, |
| name | text not null, |
| weaponType | text, |
| ammoQty | integer, |
| priceUSD | numeric(12, 2), |
| depName | text not null references departments(depName), |

primary key(wid)

);

**functional dependencies** wid  name, weaponType, ammoQty, priceUSD, depName



**Base** **list all bases and basic attributes**

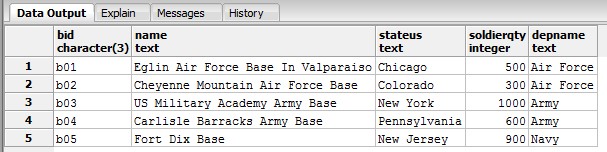
CREATE TABLE base ( bid char(3) not null unique, name text not null, stateUS text, soldierQty integer,

depName text not null references departments(depName), primary key(bid)

);

**functional dependencies**

bid  name, stateUS, soldierQty, depName



**Patient** **list all patients and basic attributes**

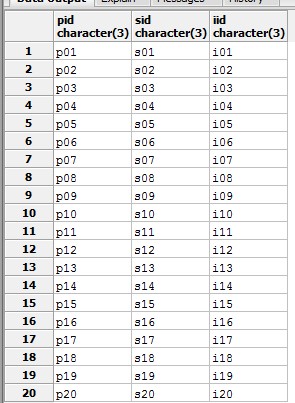
CREATE TABLE patient ( pid char(3) not null unique,

sid char(3) not null references soldiers(sid), iID char(3) not null references injury(iID), primary key(pid)

);

**functional dependencies** pid  sid, iID

## SAMPLE DATA ON FOLLOWING PAGE

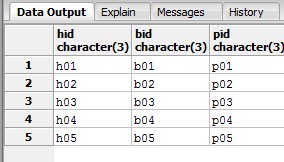


**Hospitals list all hospitals with patient and associated base**

CREATE TABLE hospitals (

hid char(3) not null unique,

bid char(3) not null references base(bid), pid char(3) not null references patient(pid), primary key(hid)

);

**functional dependencies**

hid  bid, pid

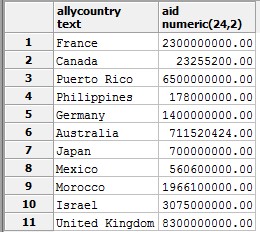
**Allies** **list all allies with amount of aid given to that ally**

CREATE TABLE allies ( allyCountry text not null unique, aid numeric(24, 2),

primary key(allyCountry)

);

**functional dependencies** allyCountry  aid



**Missions list all missions and basic attributes**

CREATE TABLE missions (

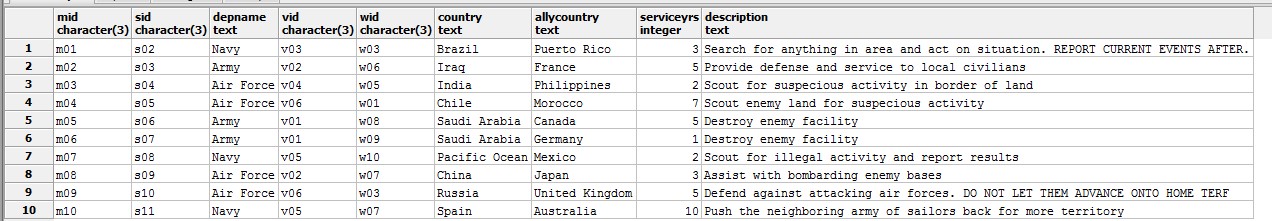
|  |  |
| --- | --- |
| mid | char(3) not null unique, |
| sid | char(3) not null references soldiers(sid), |
| depName | text not null references departments(depName), |
| vid | char(3) not null references vehicles(vid), |
| wid | char(3) not null references weapons(wid), |
| country | text not null, |
| allyCountry | text not null references allies(allyCountry), |
| serviceYrs | integer not null, |
| description | text not null, |

primary key(mid)

);

**functional dependencies** mid  sid, depName, vid, wid,

country, allyCountry, serviceYrs, description



**Missions list all missions with allies and department in charge**

CREATE TABLE missions (

|  |  |
| --- | --- |
| mid | char(3) not null unique, |
| sid | char(3) not null references soldiers(sid), |
| depName | text not null references departments(depName), |
| vid | char(3) not null references vehicles(vid), |
| wid | char(3) not null references weapons(wid), |
| country | text not null, |
| allyCountry | text not null references allies(allyCountry), |
| serviceYrs | integer not null, |
| description | text not null, |

primary key(mid)

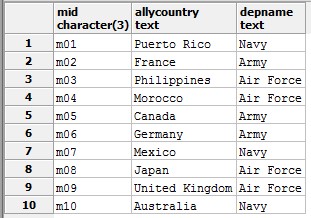
);

**functional dependencies** mid  sid, depName, vid, wid,

country, allyCountry, serviceYrs, description

## SAMPLE DATA ON FOLLOWING PAGE





**VIEW list all names and departments of all soldiers**

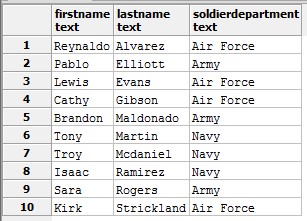
CREATE VIEW SoldierMissionDepartments AS SELECT firstName, lastName, missions.depName as soldierDepartment

FROM soldiers

INNER JOIN missions

ON soldiers.sid = missions.sid

ORDER BY lastName;



**VIEW list all the hospital locations with the base name**

CREATE VIEW HospitalLocation AS

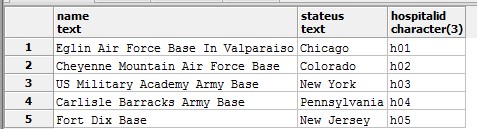
SELECT name, stateUS, hospitals.hid as hospitalID

FROM base

INNER JOIN hospitals

ON base.bid = hospitals.bid

ORDER BY hospitals.hid;



**VIEW list all the healthy soldiers and the given patient ID**

CREATE VIEW NonSickSoldiers AS

SELECT firstName, lastName, patient.pid as SoldierPatientID, healthy

FROM soldiers

INNER JOIN patient

ON soldiers.sid = patient.sid

Where healthy='Yes'

ORDER BY lastName;



**REPORTS**Interesting Queries **– these are queries that demonstrate the analytical potential of databases. These are mild examples, but nonetheless examples of the kinds of information that one can extrapolate from data.**

**1. Query to return the percentage of the soldiers that**

**went on a mission who’s age is under 23**

SELECT TRUNC (

CAST(

( SELECT COUNT(age) AS soldiersTotal

FROM soldiers

INNER JOIN missions

ON soldiers.sid = missions.sid

WHERE age < 23 ) as decimal(5,2) )

/

( SELECT COUNT(sid) AS allSoldiersOnMission FROM missions ) \* 100 ) as percent\_Under\_23



**2. Query to return the percentage of the soldiers that are aged is over 30**

SELECT TRUNC (

CAST (

( SELECT COUNT(age) AS soldiersTotal

FROM soldiers

WHERE age > 30 ) as decimal(5,2) )

/

( SELECT COUNT(sid) AS allSoldiersOnMission

FROM soldiers ) \* 100 ) as percent\_over\_30



**STORED PROCEDURES are functions conduct calculations automatically**

1. STORED PROCEDURE add\_soldier this automatically makes a newly created soldier if the soldier reports as

‘healthy’.

CREATE OR REPLACE FUNCTION add\_soldier() RETURNS trigger AS

$BODY$

BEGIN

IF NEW.healthy= "Yes" THEN

INSERT INTO soldiers(sid) VALUES (NEW.sid);

END IF;

RETURN NEW;

END;

$BODY$

LANGUAGE plpgsql;

**SAMPLE DATA FOR THIS PROCEDURE WILL BE PAIRED WITH THE**

**SAMPLE DATA FOR THE TRIGGER THAT ACTIVATES IT IN THE FOLLOWING SECTION**

2. **STORED PROCEDURE** weaponsUsed **this automatically returns a table of the soldier names and guns that the soldier has had on missions**

CREATE OR REPLACE FUNCTION weaponsUsed (IN sid varchar(3))

RETURNS TABLE("First Name" text, "Last Name" text, “Gun” text) AS

$BODY$

BEGIN

RETURN QUERY SELECT DISTINCT soldiers.firstName as

first\_name, soldiers.lastName as last\_name, weapons.name as Gun

FROM soldiers

INNER JOIN missions

ON soldiers.sid = missions.sid

INNER JOIN weapons

ON missions.wid = weapons.wid

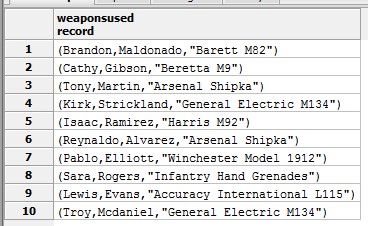
WHERE weapons.wid IN ( SELECT weapons.wid

FROM weapons);

END;

$BODY$

LANGUAGE PLPGSQL;



**3. STORED PROCEDURE** vehiclesUsed **this automatically returns a table of the soldier names and vehicles that the soldier has had on missions**

CREATE OR REPLACE FUNCTION vehicleUsed (IN sid varchar(3))

RETURNS TABLE("First Name" text, "Last Name" text, “Vehicle” text) AS

$BODY$

BEGIN

RETURN QUERY SELECT DISTINCT soldiers.firstName as first\_name, soldiers.lastName as last\_name, vehicles.vehicleType as Vehicle

FROM soldiers

INNER JOIN missions

ON soldiers.sid = missions.sid

INNER JOIN vehicles

ON missions.vid = vehicles.vid

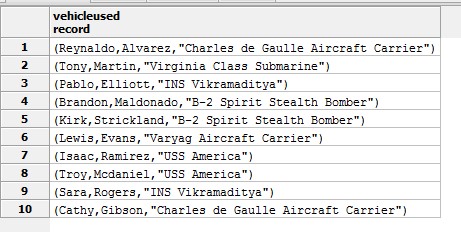
WHERE vehicles.vehicleType IN ( SELECT vehicles.vehicleType

FROM vehicles);

END; $BODY$

LANGUAGE PLPGSQL;





## **TRIGGERS** these call functions upon a specified activity on a certain table such as insert, update, or delete

1. **TRIGGER add\_soldier automatically makes a newly created soldier if healthy= “Yes” attribute of the inserted file is false.**

CREATE TRIGGER add\_soldier

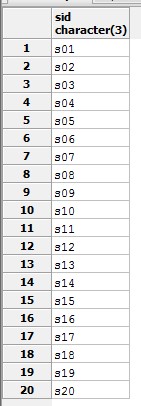
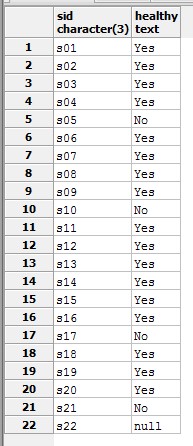
AFTER INSERT ON soldiers

FOR EACH ROW

EXECUTE PROCEDURE add\_soldier();



**BEFORE AFTER**



**SECURITY The purpose of this section is to identify and define the user roles associated with this system and then grant or revoke privileges to the various groups**

## **ADMIN**

CREATE ROLE admin;

GRANT ALL ON ALL TABLES IN SCHEMA PUBLIC TO admin;

## **CADET**

CREATE ROLE cadet;

GRANT SELECT ON soldiers, patient,

base, weapons, vehicles,

hospitals

TO cadet;

GRANT INSERT, UPDATE ON soldiers, patient,

base, weapons, vehicles, hospitals

TO cadet;

## **President**

CREATE ROLE President;

GRANT ALL ON ALL TABLES

IN SCHEMA PUBLIC

TO President;

**NOTES – ISSUES – FUTURE CONSIDERATIONS**

I now realize that I made little mistakes that had retrogressed my progress when creating the U.S. Military Operations database. The issue that I had encountered dealt with the logical relational concepts of forming an entity and perceiving the potential relationships incorrectly. One small thing that I could have done to fix this issue was to simply drop everything and restart from scratch. I was deeply engrossed with the work that I had begun causing my focus to never once think to simply take a step back and reevaluate my steps taken during my creation phase. I also would want consider not adding a Trigger to my next database.