Aircraft Carrier Operations Management System

Relational Database

Jackson O’Brien

CMPT 308 - May 2024

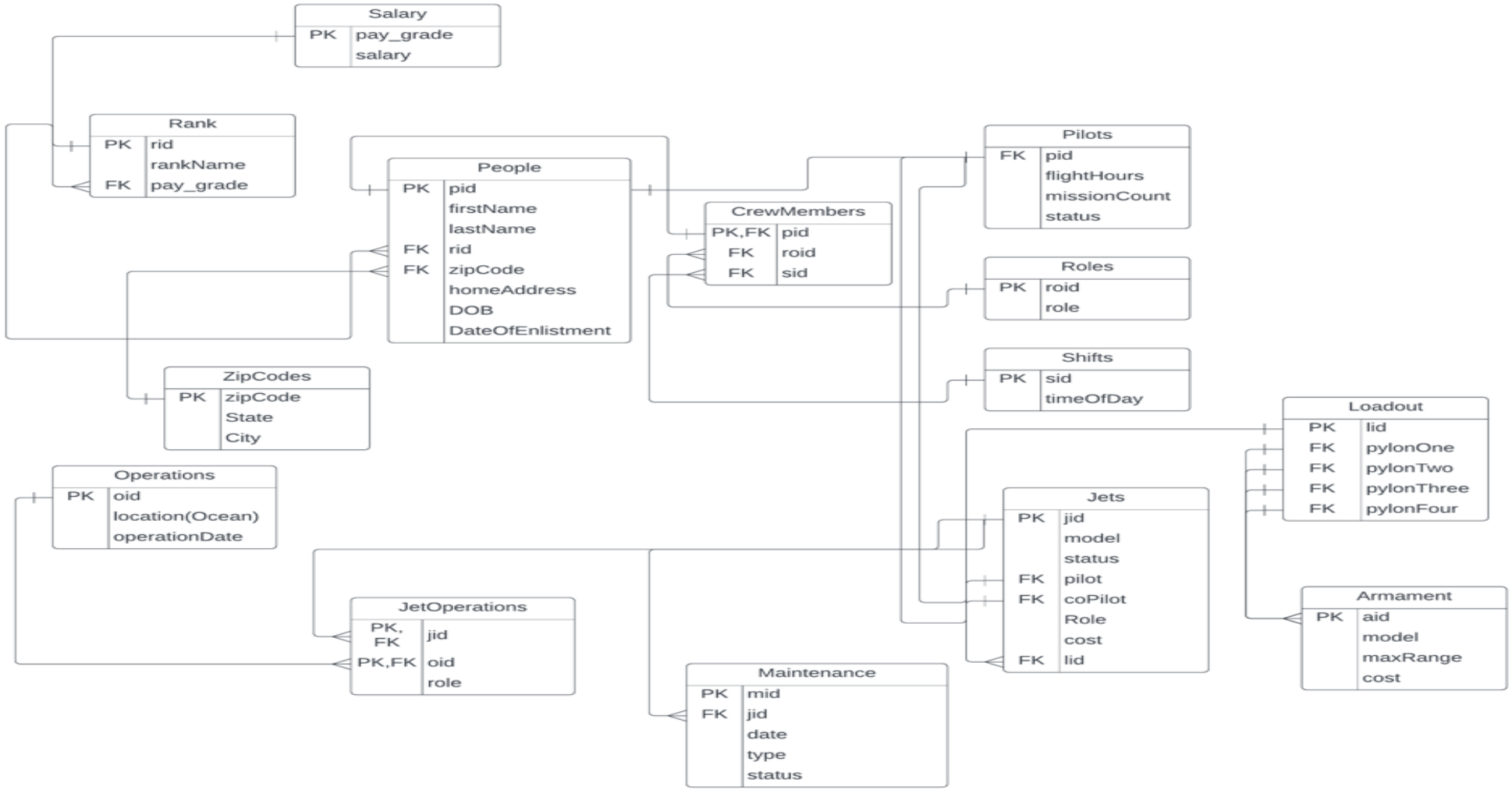
Table of Contents

Executive Summary

The Aircraft Carrier Operations Management System (ACOMS) is an advanced relational database designed for the next generation of modern aircraft carriers. ACOMS is engineered to enhance operational efficiency and streamline management across various critical dimensions of naval operations. This innovative system integrates comprehensive data management for personnel, equipment, and operational directives into a unified platform, optimizing resource utilization and strategic planning on aircraft carriers.

The ACOMS database is built on PostgreSQL, chosen for its robustness and reliability. The schema includes multiple interconnected tables. The system contains two main sections: the operational and equipment information, information related to every personnel onboard the carrier, and detailed aspects of their job. Along with the complex people system to organize personnel onboard.

ER Diagram


Salary
PK pay_grade
salary
Rank
PK rid
rankName
FK pay_grade
People
PK pid
firstName
lastName
FK rid
FK zipCode
homeAddress
DOB
DateOfEnlistment
CrewMembers
PK,FK pid
FK roid
FK sid
Pilots
FK pid
flightHours
missionCount
status
Roles
PK roid
role
Shifts
PK sid
timeOfDay
ZipCodes
PK zipCode
State
City
Operations
PK oid
location(Ocean)
operationDate
Jets
PK jid
model
status
FK pilot
FK coPilot
Role
cost
FK lid
Loadout
PK lid
FK pylonOne
FK pylonTwo
FK pylonThree
FK pylonFour
Armament
PK aid
model
maxRange
cost
JetOperations
PK, FK jid
PK,FK oid
role
Maintenance
PK mid
FK jid
date
type
status

People Table

This table contains the information of every person aboard the carrier.

Create Statement:

```
create table people (
    pid int not null,
    firstname text,
    lastname text,
    rid int not null,
    zipcodes text,
    homeaddress text,
    dob date not null,
    dateofenlistment date not null,
    primary key(pid)
);
```

Sample Output:

pid [PK] integer	firstname text	lastname text	rid integer	zipcodes text	homeaddress text	dob date	dateofenlistment date
1	Alan	Laboseur	6	12601	3399 North Road	1990-06-17	2012-07-10
2	Lucy	Ripley	5	23510	321 Nostromo W...	1986-04-20	2006-03-14
3	Jack	Ryan	10	32202	456 Patriot Ave	1968-10-11	1990-09-01
4	Max	Doe	4	07928	789 Ocean Blvd	1995-08-30	2015-06-01
5	John	Rambo	3	96860	101 First Blood Dr	1971-07-06	1993-05-24
6	Sarah	Connor	2	29401	202 Sky Rd	1984-05-12	2004-10-26
7	Maverick	Mitchell	9	31547	303 Top Gun St	1969-05-27	1989-01-03
8	Tom	Cruise	8	06340	808 Book Ln	1989-11-16	2009-07-13

Functional Dependency:

pid → firstname, lastname, rid, zipcodes, homeaddress, dob, dateofenlistment

Rank Table

Table contains the rank ID (rid) for every rank available in the Navy, it contains the name of the rank and its paygrade.

Create Statement:

```
create table rank (
    rid int not null,
    rankname text not null,
    paygrade text not null,
    primary key(rid)
);
```

Sample Output:

rid [PK] integer	rankname text	paygrade text
1	Seaman Recruit	E1
2	Seaman Apprentice	E2
3	Seaman	E3
4	Petty Officer Third Class	E4
5	Petty Officer Second Class	E5
6	Ensign	O1
7	Lieutenant Junior Grade	O2
8	Lieutenant	O3

Functional Dependency:

rid → rankname, paygrade

Salary Table

This connects the paygrade from the Rank table to the Salary table which will give the paygrade a yearly salary dependent on the paygrade. E represents enlisted, and O represents officer.

Create Statement:

```
create table salary (
    paygrade text not null,
    salary int not null,
    primary key(paygrade)
);
```

Functional Dependency:

paygrade →salary

Sample Output:

paygrade [PK] text	salary integer
E1	12000
E2	14000
E3	17000
E4	21000
E5	26000
O1	68000
O2	75000
O3	91000

Zipcodes Table

Contains all the information about zipcodes of sailors onboard the carrier. It contains the state and city based on the zipcode number.

Create Statement:

```
create table zipcodes (
    zipcode text not null,
    state text not null,
    city text not null,
    primary key(zipcode)
);
```

Sample Output:

zipcode [PK] text	state text	city text
12601	NY	Poughkeep...
002	Virginia	Norfolk
003	Florida	Jacksonville
004	Hawaii	Pearl Harbor
005	Washington	Bremerton

Functional Dependency:

zipcode → state, city

Pilots Table

The Pilots Table contains information about the pilot, it used the People table to connect the pid to a pilot. The information provided is the flight hours, number of missions the pilot has completed, and status of the pilot.

Create Statement:

```
create table pilots (
    pid int not null,
    flighthours int,
    missioncount int,
    status text,
    primary key (pid)
);
```

Sample Output:

	pid [PK] integer	flighthours integer	missioncount integer	status text
1	1	1500	300	Active
2	2	1200	200	Active
3	3	900	150	Active
4	4	2100	250	Active
5	5	800	100	Active

Functional Dependency:

pid → flighthours, missioncount, status

Crew Members Table

Crew Members table contains information about everyone that isn't a pilot, it can vary from all non-pilot duties on the carrier. It takes the pid from People and connects the id of their role and scheduled shift time.

Create Statement:

```
create table crewmembers (
    pid int not null,
    roid int not null,
    sid int not null,
    primary key(pid)
);
```

Functional Dependency:

pid → roid, sid

Sample Output:

pid [PK] integer	roid integer	sid integer
6	1	1
7	2	2
8	3	3
9	4	4
10	1	5

Roles Table

Roles contains information about the available jobs that crew members can hold, ranging from ground crew to air traffic controllers.

Create Statement:

```
create table roles (
    roid int not null,
    role text,
    primary key(roid)
);
```

Sample Output:

roid [PK] integer	role text
1	Launch Technician
2	Aircraft Mechanic
3	Air Traffic Controller
4	Culinary Specialist

Functional Dependency:

roid → role

Shifts Table

This table contains information about the available shifts that can be assigned to crew members.

Create Statement:

```
create table shifts (
    sid int not null,
    timeOfDay text,
    primary key(sid)
);
```

Sample Output:

sid [PK] integer	timeofday text
1	Morning
2	Afternoon
3	Evening

Functional Dependency:

sid → timeOfDay

Operations Table

The operations table holds important information about previous operations and missions that were conducted the carrier and its crew. It stores the location of where it took place and the date.

Create Statement:

```
create table operations (
    oid int not null,
    location text not null,
    operationdate date not null,
    primary key (oid)
);
```

Sample Output:

oid [PK] integer	location text	operationdate date
1	Pacific Ocean	2024-05-01
2	Indian Ocean	2024-05-02
3	Atlantic Ocean	2024-05-03
4	Mediterranean Sea	2024-05-04
5	Arabian Sea	2024-05-05

Functional Dependency:

oid → location, operationdate

Jet Operations Table

This table builds upon the operations table by holding information about what jets(jid) were assigned to the operation that happened and what their role was during it.

Create Statement:

```
create table jetoperations (
    jid int not null,
    oid int not null,
    role text,
    primary key(jid, oid)
);
```

Sample Output:

jid [PK] integer	oid [PK] integer	role text
1	1	Reconnaissance
2	2	Surveillance
3	3	Combat
4	4	Patrol
5	5	Training

Functional Dependency:

jid, oid → role

Armaments Table

This table contains information about what armaments are stored on the carrier and information about the equipment such as max range and cost in USD.

Create Statement:

```
create table armaments (
    aid int not null,
    model text not null,
    maxrange int not null,
    costusd int not null,
    primary key(aid)
);
```

Sample Output:

aid [PK] integer	model text	maxrange integer	costusd integer
1	AIM-9 Sidewinder	18	60000
2	AIM-120 AMRAAM	180	400000
3	AGM-88 HARM	150	800000
4	AGM-65 Maverick	22	170000
5	Mark 84 General Purpose Bomb	0	20000

Functional Dependency:

aid → model, maxrange, costusd

Loadouts Table

This table uses the armaments stored on the carrier and says which pylons of the plane the armament can be equipped on. Planes have a lid foreign key where planes can equip a loadout to bring on operations. Planes can carry multiple of the same loadouts and armaments.

Create Statement:

```
create table loadouts (
    lid int not null,
    pylonone int,
    pylontwo int,
    pylonthree int,
    pylonfour int,
    primary key (lid)
);
```

Sample Output:

lid [PK] integer	pylonone integer	pylontwo integer	pylonthree integer	pylonfour integer
1	1	2	3	4
2	2	2	3	5
3	1	1	4	4
4	3	3	5	5
5	4	4	1	2

Functional Dependency:

lid → pylonone, pylontwo, pylonthree, pylonfour

Jets Table

This table contains lots of information about the jets onboard the carrier. It says what model plane it is, the status of it and what pilots are assigned to it. Planes can have a pilot and co-pilot. The table also stores information about what loadout is currently on the plane, what role the plan serves, and the cost in USD.

Create Statement:

```
create table jets (
    jid int not null,
    model text not null,
    status text,
    pilot int,
    copilot int,
    role text,
    costUSD int,
    lid int,
    primary key(jid)
);
```

Sample Output:

jid [PK] integer	model text	status text	pilot integer	copilot integer	role text	costusd integer	lid integer
1	F/A-18 Hornet	Operational	1	[null]	Combat	70000000	1
2	F-35C Lightning II	Operational	3	[null]	Reconnaissance	95000000	2
3	EA-18G Growler	Maintenan...	5	6	Electronic Warfare	67000000	3
4	F-14 Tomcat	Operational	7	8	Interceptor	38000000	4
5	AV-8B Harrier II	Operational	9	[null]	Ground Attack	24000000	5

Functional Dependency:

jid → model, status, pilot, copilot, role, costUSD, lid

Maintenance Table

This table provides information about maintenance history of jets, it says what date the maintenance happened and what type it was and the status.

Create Statement:

```
create table maintenance (
    mid int not null,
    jid int not null,
    dateOf date,
    type text,
    status text,
    primary key(mid)
);
```

Sample Output:

mid [PK] integer	jid integer	dateof date	type text	status text
1	3	2024-04-10	Engine Overhaul	Completed
2	5	2024-04-15	Avionics Upgrade	In Progress
3	4	2024-04-20	Radar Calibration	Completed
4	2	2024-04-25	After Action Review	Scheduled
5	1	2024-04-30	Weapon Systems Check	Completed

Functional Dependency:

mid → jid, dateOf, type, status

View OperationOverview

This view provides an overview of the operations selected, it will show which pilots were at the operation and what jets were used along with how much the payload cost to use and what loadout was brought. It also contains information about the operation itself such as location, date and id.

```sql
create view OperationOverview as
select
    operations.oid,
    operations.location,
    operations.operationdate,
    jets.model,
    jets.status,
    people.firstname,
    people.lastname,
    rank.rankname as rank,
    loadouts.lid as loadout,
    coalesce((select costusd from armaments where aid = loadouts.pylonone), 0) +
    coalesce((select costusd from armaments where aid = loadouts.pylontwo), 0) +
    coalesce((select costusd from armaments where aid = loadouts.pylonthree), 0) +
    coalesce((select costusd from armaments where aid = loadouts.pylonfour), 0) as total_armament_cost
from operations
join jetoperations on operations.oid = jetoperations.oid
join jets on jetoperations.jid = jets.jid
join pilots on jets.pilot = pilots.pid
join people on pilots.pid = people.pid
join rank on people.rid = rank.rid
join loadouts on jets.lid = loadouts.lid
order by operations.operationdate desc;
```

View OperationOverview

Query 1: Provides view of operations between a given time frame

```
select *
from OperationOverview
where operationdate between '2024-05-01' and '2024-05-31';
```

oid integer	location text	operationdate date	model text	status text	firstname text	lastname text	rank text	loadout integer	total_armament_cost integer
3	Atlantic Ocean	2024-05-03	EA-18G Growler	Maintenan…	John	Rambo	Seaman	3	460000
2	Indian Ocean	2024-05-02	F-35C Lightning II	Operational	Jack	Ryan	Commander	2	1620000
1	Pacific Ocean	2024-05-01	F/A-18 Hornet	Operational	Alan	Laboseur	Ensign	1	1430000

Query 2: Provides view of operations at specific location

```
select *
from OperationOverview
where location = 'Pacific Ocean';
```

oid integer	location text	operationdate date	model text	status text	firstname text	lastname text	rank text	loadout integer	total_armament_cost integer
1	Pacific Ocean	2024-05-01	F/A-18 Hornet	Operational	Alan	Laboseur	Ensign	1	1430000

View PilotSummary

This view provides information about pilots onboard the carrier. It provides all details including name, rank, home address and zip code. Their enlistment date, and specific details regarding their career in the navy such as flight hours, mission count, and their status onboard.

```
create view PilotSummary as
select
    people.pid,
    people.firstname,
    people.lastname,
    rank.rankname,
    people.zipcodes,
    people.homeaddress,
    people.dob,
    people.dateofenlistment,
    pilots.flighthours,
    pilots.missioncount,
    pilots.status
from
    people
join pilots
on people.pid = pilots.pid
join rank
on people.rid = rank.rid;
```

View PilotSummary

Query 1: Provides summary of all pilots who are of the rank of Ensign.

```
select
    pid,
    firstname,
    lastname,
    zipcodes,
    homeaddress,
    dob,
    dateofenlistment,
    flighthours,
    missioncount,
    status
from PilotSummary
where rankname = 'Ensign';
```

pid integer	firstname text	lastname text	zipcodes text	homeaddress text	dob date	dateofenlistment date	flighthours integer	missioncount integer	status text
1	Alan	Laboseur	12601	3399 North Road	1990-06-17	2012-07-10	1500	300	Active

View PilotSummary

Query 2: Provides summary of all pilots who have more than 1000 hours of flying.

```
select
    pid,
    firstname,
    lastname,
    zipcodes,
    homeaddress,
    dob,
    dateofenlistment,
    flighthours,
    missioncount,
    status
from PilotSummary
where flighthours > 1000;
```

pid integer	firstname text	lastname text	zipcodes text	homeaddress text	dob date	dateofenlistment date	flighthours integer	missioncount integer	status text
1	Alan	Laboseur	12601	3399 North Road	1990-06-17	2012-07-10	1500	300	Active
2	Lucy	Ripley	23510	321 Nostromo Way	1986-04-20	2006-03-14	1200	200	Active
4	Max	Doe	07928	789 Ocean Blvd	1995-08-30	2015-06-01	2100	250	Active

Reports

Report 1: This report shows the current jets and the pilots that occupy them

```
select
    jets.model,
    jets.status,
    people.firstname,
    people.lastname,
    pilots.flighthours
from jets
join pilots on jets.pilot = pilots.pid
join people on pilots.pid = people.pid
order by jets.status, pilots.flighthours desc;
```

model text	status text	firstname text	lastname text	flighthours integer
EA-18G Growler	Maintenan...	John	Rambo	800
F/A-18 Hornet	Operational	Alan	Laboseur	1500
F-35C Lightning II	Operational	Jack	Ryan	900

Report 2: This report shows members to their roles and respective shift times

```
select
    people.firstname,
    people.lastname,
    roles.role,
    shifts.timeOfDay
from crewmembers
join people
on crewmembers.pid = people.pid
join roles
on crewmembers.roid = roles.roid
join shifts
on crewmembers.sid = shifts.sid
order by roles.role, shifts.timeOfDay;
```

firstname text	lastname text	role text	timeofday text
Tom	Cruise	Air Traffic Controller	Evening
Maverick	Mitchell	Aircraft Mechanic	Afternoon
Sarah	Connor	Launch Technician	Morning

Procedures

Procedure 1: You can update the crew member roles and switch peoples jobs arounds by using the procedure. You have to put the pid of the person in the crewMemberID input, and then role ID in the newRoleID and it changes the job.

```
create or replace function UpdateCrewMemberRole(crewMemberId int, newRoleId int)
returns text as
$$
declare currentRole int;
begin
    select roid into currentRole from crewmembers where pid = crewMemberId;

    if currentRole != newRoleId then
        update crewmembers set roid = newRoleId where pid = crewMemberId;
        return 'Crew member role updated successfully.';
    end if;
end;
$$
language plpgsql;
```

Procedures

Query: Shows how the procedure works, the example shown below shows the before the procedure. Then after the procedure and shows how the job role has changed.

```
select *
from crewmembers
```

pid [PK] integer	roid integer	sid integer
8	3	3
10	1	5
6	4	1
9	6	4
7	3	2

```
select UpdateCrewMemberRole(6, 7);
```

updatecrewmemberrole text
Crew member role updated successfully.

```
select *
from crewmembers
where pid = 6
```

pid [PK] integer	roid integer	sid integer
6	7	1

Procedures

Procedure 2: The jet status can be adjusted to what is necessary, can be changed to anything that is desired.

```
create or replace function updateJetStatus(jet_id int, new_status text)
returns void as $$
begin
    update jets
    set status = new_status
    where jid = jet_id;
end;
$$ language plpgsql;
```

Procedures

Query: Before and after, the plane is set to be operational but needs the status changed to needs repair. So, the function updateJetStatus is used, all that's needed is the planeID and status it needs to be changed to.

```
select * from jets where jid = 1;
```

jid [PK] integer	model text	status text	pilot integer	copilot integer	role text	costusd integer	lid integer
1	F/A-18 Hornet	Operational	1	[null]	Combat	70000000	1

```
select updateJetStatus(1, 'Needs Repair');
select * from jets where jid = 1;
```

jid [PK] integer	model text	status text	pilot integer	copilot integer	role text	costusd integer	lid integer
1	F/A-18 Hornet	Needs Repair	1	[null]	Combat	70000000	1

Triggers

Trigger: If the pilots hours get above 5000, it automatically retires them from the Navy and puts their status as retired in the database.

```
create or replace function retirePilot()
returns trigger as
$$
begin
    if new.flighthours > 5000 then
        update pilots
        set status = 'Retired'
        where pid = new.pid;
    end if;
    return new;
end;
$$
language plpgsql;
```

```
create trigger retirePilot
after update of flighthours on pilots
for each row
execute function retirePilot();
```

```
update pilots
set flighthours = flighthours + 3500
where pid = '001';

select pilots.pid, people.firstname, people.lastname, pilots.flighthours, pilots.status
from pilots
join people on pilots.pid = people.pid
where pilots.pid = '001';
```

pid integer	firstname text	lastname text	flighthours integer	status text
1	Alan	Laboseur	19000	Retired

Security

```
create role general;
grant all
on all tables in schema public
to general;
```

General Permissions: Can modify anything on the table and view anything. Have ultimate control over the database.

```
create role enlisted;
grant select
on all tables in schema public
to enlisted;
```

Enlisted Permissions: Can view any of the tables but cannot make changes to any of them.

```
create role officer;
grant select, insert, update
on all tables in schema public
to officer;
```

Officer Permissions: Can view and select, insert and update on any of the tables in the database.

Implementation Notes

The biggest weakness with this database is the human error component. Data must be entered accurately, or else problems could arise, it is vital that the information is checked over and made sure it can work with the database. For the most part, the procedures and functions can be used to change data around, but other than that there shouldn’t be any other way at least from a normal user of the database.

The database shown in the example is extremely small compared to what an actual carrier database would be shown. With more data comes greater opportunity to explore better researched information and ways to improve efficiency. It is very simple though to add new data to the database, and to change data is also simple.

Known Problems

As the database expands, so do the amount of ID numbers for all of the possible sources. Every single ID is listed as 001, it may be inefficient or difficult if you want to pinpoint where something is.
The ID names can be difficult to understand if reading the names without an alias, the majority of the ids all follow the same format, pid, mid, lid, etc.
There is the possibility of data duplication, an example of this could be a person serving multiple roles.
There are not many checks or return statements that explain what's going on. Many return statements when using a function or procedure return blank or nothing. The only way to confirm if there was a change is to select everything from that table and verify.

Future Enhancements

More normalization to ensure referential integrity and more checks in the functions and procedures.
More protection methods such as adding control gaps and levels of security and locking. The only protection implemented is limiting the amount of power certain roles have.
Increasing readability by using more aliases to show what ID's superficially and many other aspects of the database mean to someone who doesn't understand it on the complex level.
Allowing multi-role to the crew members who may do different jobs throughout the day. Same can go for pilots, pilots are assigned a jet and that's it. In the future it can be implemented to let them switch freely.