

**CS2102**

**Project Report**

**Topic D – Car Pooling**

**Group 8**

1. Ang Wei Ming, A0168721B
2. Benjamin Chin Choon Kiat, A0168698B
3. Lee Yu Choy, A0177151H
4. Yeo Cheng Hong, A0168369L

6 November 2018

Table Of Content

[1](#_Toc529285204)

[1 General Architecture 3](#_Toc529285205)

[2 ER Diagram 3](#_Toc529285206)

[3 DDL 4](#_Toc529285207)

[3.1.1 User Schema 4](#_Toc529285208)

[3.1.2 Car Model Schema 4](#_Toc529285209)

[3.1.3 Car Schema 4](#_Toc529285210)

[3.1.4 Car Ride Schema 4](#_Toc529285211)

[3.1.5 ​Car Ride Bid Schema 5](#_Toc529285212)

[3.1.6 Transaction Log Schema 5](#_Toc529285213)

[3.2 Triggers and Functions 5](#_Toc529285214)

[3.2.1 ​After Approval Checks 5](#_Toc529285215)

[3.2.2 Constrain checks before bidding. 6](#_Toc529285216)

[3.2.3 Insert into audit table. 7](#_Toc529285217)

[4 Sample SQL 8](#_Toc529285218)

[4.1 Search Ride 8](#_Toc529285219)

[4.2 Rejection of ride bid 9](#_Toc529285220)

[4.2.1 Trigger behind assert 9](#_Toc529285221)

[4.2.2 Simple Insert Query 10](#_Toc529285222)

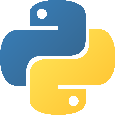
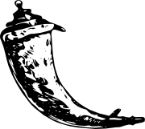
[4.3 Ride History 10](#_Toc529285223)

[4.3.1 Trigger 10](#_Toc529285224)

[4.3.2 Simple Update Query 11](#_Toc529285225)

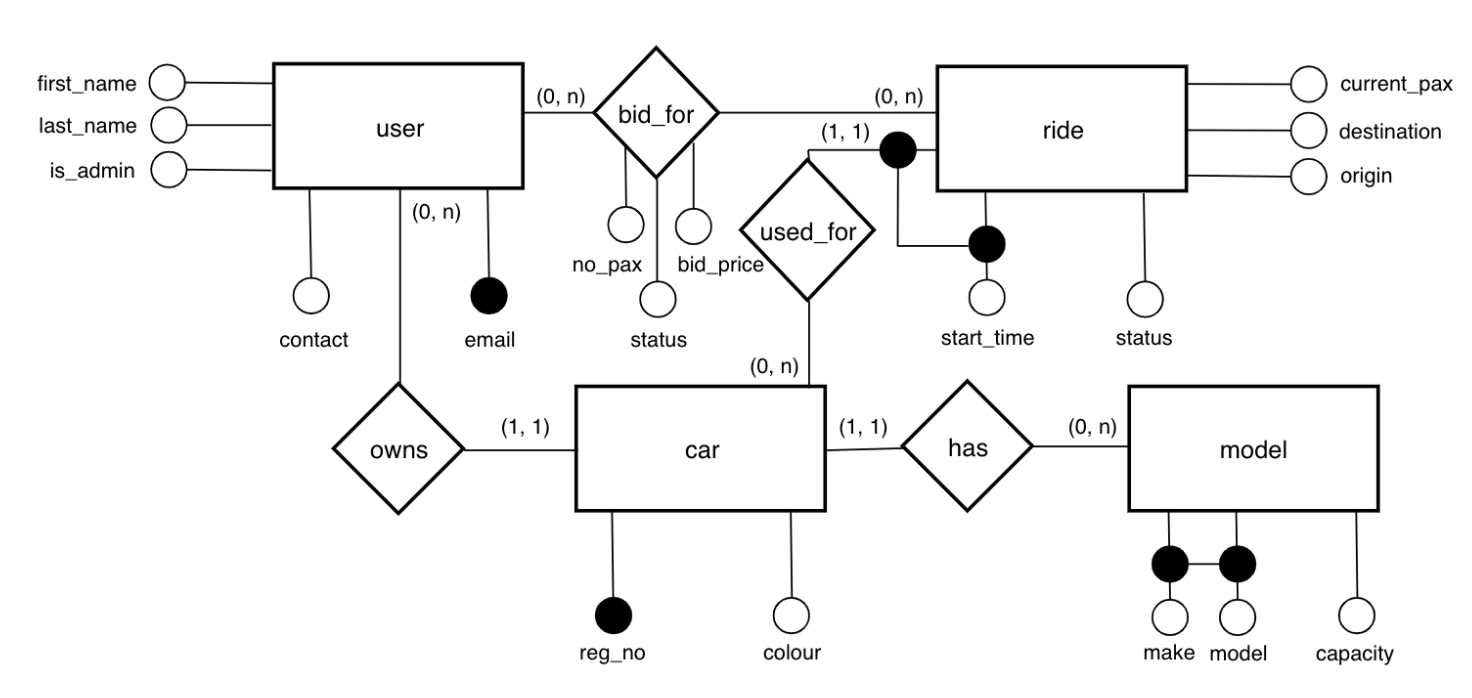
[5 Screen shots 11](#_Toc529285226)

# General Architecture



* Server Language – Python 2.7
* Web Server - Python Flask 1.0.2
* Database – PostgreSQL 11.0

# ER Diagram



# DDL

### User Schema

create table if not exists "user" -- `"` used because PostgreSQL use 'user' as a keyword  
(  
email varchar(256) not null constraint user\_pkey primary key,  
contact numeric(8),  
first\_name varchar(50) not null,  
last\_name varchar(50) not null,  
is\_admin boolean default false not null,  
password varchar(512) not null  
)  
;  
​

### Car Model Schema

create table if not exists model  
(  
model varchar(256) not null,  
make varchar(256) not null,  
capacity integer not null constraint capacity\_min check (capacity > 0),  
constraint model\_pk primary key (model, make)  
)  
;  
​

### Car Schema

create table if not exists car  
(  
reg\_no varchar(8) not null constraint car\_pkey primary key,  
colour varchar(50),  
email varchar(256) not null constraint car\_email\_fkey references "user",  
make varchar(50) not null,  
model varchar(50) not null,  
constraint car\_make\_fkey foreign key (make, model) references model (make, model)  
)  
;  
​

### Car Ride Schema

create table if not exists ride  
(  
start\_time timestamp not null,  
status varchar(11) not null constraint ride\_status\_type  
check (((status)::text = 'in progress'::text) OR ((status)::text = 'completed'::text)),  
current\_pax integer not null,  
destination varchar(256) not null,  
origin varchar(256) not null,  
reg\_no varchar(8) not null constraint ride\_reg\_no\_fkey references car,  
constraint ride\_pkey primary key (start\_time, reg\_no)  
)  
;  
​

### ​Car Ride Bid Schema

create table if not exists ride\_bid  
(  
email varchar(256) not null constraint ride\_bid\_email\_fkey references "user",  
start\_time timestamp not null,  
reg\_no varchar(8) not null,  
no\_pax integer not null constraint min\_pax check (no\_pax > 0),  
bid\_price double precision check (bid\_price > 0),  
status varchar(13) default 'pending'::character varying not null constraint bid\_status\_type check (((status)::text = 'pending'::text) OR ((status)::text = 'successful'::text) OR ((status)::text = 'unsuccessful'::text)),  
constraint ride\_bid\_pkey primary key (email, start\_time, reg\_no),  
constraint ride\_bid\_start\_time\_fkey foreign key (start\_time, reg\_no) references ride

)  
;  
​

### Transaction Log Schema

create table if not exists audit\_log  
(  
start\_time timestamp not null,  
end\_time timestamp not null,  
status varchar(11) not null constraint ride\_status\_type check ((status)::text = 'completed'::text),  
current\_pax integer not null,  
destination varchar(256) not null,  
origin varchar(256) not null,  
reg\_no varchar(8) not null constraint ride\_reg\_no\_fk references car,  
constraint ride\_pk primary key (start\_time, reg\_no)  
)  
;

## Triggers and Functions

### ​After Approval Checks

This trigger will check if a bid have been changed from pending/unsuccessful to successful.

If this is true, it will update the ride current passenger to the number of passenger in the bid.

Which is follow by rejection all ‘pending’ bids which have their number of passenger greater than remaining seats

create or replace function on\_approval\_update\_pax() returns trigger  
language plpgsql  
as $$  
BEGIN  
IF NEW.status = 'successful' and OLD.STATUS <> 'successful'  
THEN  
UPDATE ride  
SET current\_pax = current\_pax + NEW.no\_pax  
WHERE reg\_no = NEW.reg\_no  
AND start\_time = NEW.start\_time;  
 UPDATE ride\_bid rb  
   SET status = 'unsuccessful'  
 FROM ride r, car c, model m  
 WHERE r.reg\_no = rb.reg\_no  
 AND r.start\_time = rb.start\_time  
 AND r.reg\_no = c.reg\_no  
 AND c.make = m.make  
 AND c.model = m.model  
 AND rb.reg\_no = NEW.reg\_no  
 AND rb.start\_time = NEW.start\_time  
 AND rb.status = 'pending'  
 AND rb.no\_pax > (m.capacity - r.current\_pax);  
END IF;  
​  
RETURN NULL;  
END  
$$  
;  
​  
​  
create trigger approval\_update  
after update  
on ride\_bid  
for each row  
execute procedure on\_approval\_update\_pax()  
;  
​

### Constrain checks before bidding.

This function will reject insertion/update if the bids have its numbers of passengers exceed the remaining seats.

It will also reject insertion if the bids is bid by the driver himself.

create or replace function capacity\_checker()  
returns trigger  
language plpgsql  
as $$  
BEGIN  
IF (SELECT (r.current\_pax + NEW.no\_pax <= m.capacity)  
     FROM ride r  
            inner join car c on r.reg\_no = c.reg\_no  
            INNER JOIN model m on c.make = m.make and c.model = m.model  
                                    AND r.reg\_no = NEW.reg\_no  
                                    AND r.start\_time = NEW.start\_time)  
THEN  
 --  
 -- Do nothing  
ELSE  
  RAISE EXCEPTION 'Exceeded maximum capacity, please reduce your number of passenger';  
END IF;  
IF (SELECT (1)  
     FROM ride r  
            INNER JOIN car c on r.reg\_no = c.reg\_no  
     where r.reg\_no = NEW.reg\_no  
       AND r.start\_time = NEW.start\_time  
       AND c.email = NEW.email)  
THEN RAISE EXCEPTION 'Cannot Bid for Own Ride';  
ELSE  
  RETURN NEW;  
END IF;  
END  
$$;  
​  
create trigger cap\_check  
before insert OR update  
on ride\_bid  
for each row  
execute procedure capacity\_checker()  
;

### Insert into audit table.

This trigger is used to insert into a similar table which help in querying transaction history. The end time of the ride will also be updated upon completion.

This extra table although is redundant, but it can help to maintain integrity of the logs, as there will not be any updates in this table. (Only Insertion)

create or replace function audit() returns trigger  
language plpgsql  
as $$  
BEGIN  
IF NEW.status = 'completed' THEN  
 INSERT INTO audit\_log(start\_time,end\_time,status,current\_pax,destination,origin,reg\_no)  
 VALUES (OLD.start\_time,now(),NEW.status,OLD.current\_pax,OLD.destination,OLD.origin,OLD.reg\_no);  
END IF;  
​  
RETURN NEW;  
END;  
$$  
;  
​  
create trigger to\_audit  
before update  
on ride  
for each row  
execute procedure audit()  
;

## Sample Function and SQL

### Search Ride

If origin and destination is NULL, return all rides

SELECT u.first\_name,u.email,r.origin,r.destination,r.status,r.reg\_no, r.start\_time, r.current\_pax, (m.capacity - r.current\_pax) as pax\_left,  
EXISTS (SELECT c1.email FROM car c1 WHERE c1.reg\_no = c.reg\_no AND c1.email = %s) as is\_driver,  
EXISTS (SELECT rb.email FROM ride\_bid rb WHERE rb.reg\_no = c.reg\_no AND rb.email = %s AND rb.status = 'successful' AND rb.start\_time = r.start\_time) as has\_success\_bid,  
EXISTS (SELECT rb.email FROM ride\_bid rb WHERE rb.reg\_no = c.reg\_no AND rb.email = %s AND rb.status = 'unsuccessful' AND rb.start\_time = r.start\_time) as has\_unsuccessful\_bid,  
EXISTS (SELECT rb.email FROM ride\_bid rb WHERE rb.reg\_no = c.reg\_no AND rb.email = %s AND rb.status = 'pending' AND rb.start\_time = r.start\_time) as has\_pending\_bid  
FROM ride r, "user" u, car c,model m  
WHERE r.reg\_no = c.reg\_no  
and c.email = u.email  
and LOWER(r.origin) LIKE LOWER(%s) and LOWER(r.destination) like LOWER(%s)  
and r.status = 'in progress'  
and c.make = m.make  
and c.model = m.model  
ORDER BY r.start\_time ASC

### Login / Register

The following is the SQL for register and login respectively

SELECT REGISTER(%s,%s,%s,%s,%s)  
SELECT \* from login(%s,%s)

#### Stored Procedure for login and register

The stored procedure is used so that the password is hash on the database side before inserting into the table, which improve the security factor.

CREATE OR REPLACE FUNCTION register(email varchar(256),contact numeric(8),first\_name varchar(50), last\_name varchar(50),password varchar(256))  
RETURNS BOOLEAN AS $$  
DECLARE  
success BOOLEAN;  
hashPassword varchar(512);  
BEGIN  
SELECT encode(digest($5, 'sha256'), 'hex') INTO hashPassword;  
INSERT INTO "user" VALUES ($1,$2,$3,$4,false ,hashPassword) Returning 1 into success;  
RETURN success;  
END;  
$$ LANGUAGE plpgsql;  
​  
CREATE OR REPLACE FUNCTION login(email varchar(256),password varchar(256))  
RETURNS setof "user" AS $$  
DECLARE  
hashPassword varchar(512);  
BEGIN  
SELECT encode(digest($2, 'sha256'), 'hex') INTO hashPassword;  
RETURN QUERY SELECT \* FROM "user" where "user".email = $1 and "user".password = hashPassword;  
END;  
$$ LANGUAGE plpgsql;

### Ride History

This allow us to keep track of when is a ride completed. With another table in place, it can help to prevent accidental update. As there are no other query except this trigger that is link to the table.

#### Trigger

create or replace function audit() returns trigger  
language plpgsql  
as $$  
BEGIN  
IF NEW.status = 'completed' THEN  
 INSERT INTO audit\_log(start\_time,end\_time,status,current\_pax,destination,origin,reg\_no)  
 VALUES (OLD.start\_time,now(),NEW.status,OLD.current\_pax,OLD.destination,OLD.origin,OLD.reg\_no);  
END IF;  
​  
RETURN NEW;  
END;  
$$  
;  
​  
create trigger to\_audit  
before update  
on ride  
for each row  
execute procedure audit()  
;

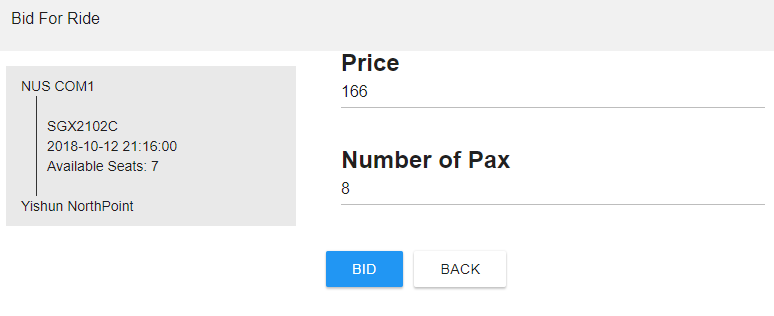
#### Simple Update Query

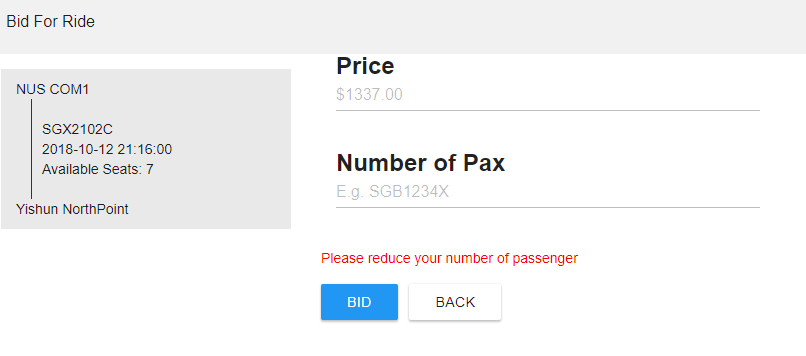
UPDATE ride  
   SET origin = %s, destination = %s, status = %s  
   WHERE reg\_no = %s AND start\_time = %s

# Assertion

## Rejection of bid

When the user is bidding for seats which is greater than current available seats, the system will reject the transaction via trigger





### Trigger behind

create or replace function capacity\_checker() returns trigger  
language plpgsql  
as $$  
BEGIN  
  IF ( SELECT (r.current\_pax + NEW.no\_pax <= m.capacity)  
   FROM ride r  
  inner join car c on r.reg\_no = c.reg\_no  
  INNER JOIN model m on c.make = m.make and c.model =m.model  
   AND r.reg\_no = NEW.reg\_no  
   AND r.start\_time = NEW.start\_time)  
    THEN  
    RETURN NEW;  
ELSE  
RAISE EXCEPTION 'Exceeded maximum capacity, please reduce your number of passenger';  
    END IF;  
END  
$$  
;  
create trigger cap\_check  
before insert  
on ride\_bid  
for each row  
execute procedure capacity\_checker()  
;

### Simple Insert Query

INSERT INTO ride\_bid (email,start\_time,reg\_no,no\_pax,bid\_price) VALUES (%s,%s,%s,%s,%s)

## Bid Approval

For every successful ride bid that is approved by the driver , there will be a trigger to update the current capacity of the ride and reject all pending bid which exceed the current available seats.

### Trigger as shown below

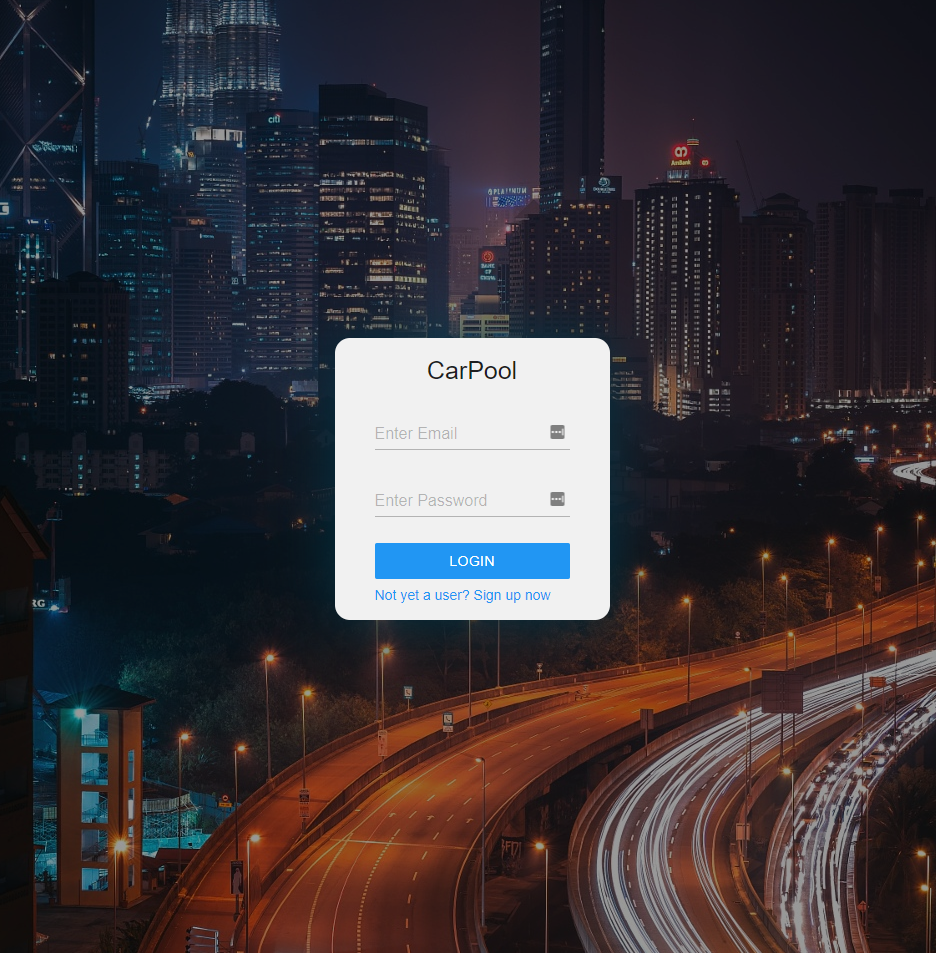
create trigger approval\_update  
after update  
on ride\_bid  
for each row  
execute procedure on\_approval\_update\_pax()  
;

### on\_approval\_update\_pax function

create or replace function on\_approval\_update\_pax() returns trigger  
language plpgsql  
as $$  
BEGIN  
IF NEW.status = 'successful' and OLD.STATUS <> 'successful'  
THEN  
UPDATE ride  
SET current\_pax = current\_pax + NEW.no\_pax  
WHERE reg\_no = NEW.reg\_no  
AND start\_time = NEW.start\_time;  
-- The following part update all current biddings which have their number of passenger above the current available seats  
-- for example total pax is 4, current pax is 2  
-- all bidding which is >2 will be automatically changed to unsuccessful  
 UPDATE ride\_bid rb  
   SET status = 'unsuccessful'  
 FROM ride r, car c, model m  
 WHERE r.reg\_no = rb.reg\_no  
 AND r.start\_time = rb.start\_time  
 AND r.reg\_no = c.reg\_no  
 AND c.make = m.make  
 AND c.model = m.model  
 AND rb.reg\_no = NEW.reg\_no  
 AND rb.start\_time = NEW.start\_time  
 AND rb.status = 'pending'  
 AND rb.no\_pax > (m.capacity - r.current\_pax);  
END IF;  
-- return NULL as trigger is called after update  
RETURN NULL;  
END  
$$  
;

The current\_pax attribute allow us to check for available seats easily.

# Screen shots

Login Page:

Index Page: