INSURANCE POLICY DATABASE PROJECT

We declare that we have completed this assignment completely and entirely on our own, without any consultation with others. We have read the UAB Academic Honor Code and understand that any breach of the Honor Code may result in severe penalties.

We also declare that the following percentage distribution ***faithfully*** represents individual group members' contributions to the completion of the assignment

Name	Overall Contribution (%)	Major work items completed by me	Signature or initials	Date
Leela Krishna Duddukuri	25%	ER Diagrams with explanation of assumptions and database relational schema and code execution	LKD	12/03/2021
Hima Sekhar Sakhamuri	25%	ER Diagrams with explanation of database constraints, with sample data and code execution	HSS	12/03/2021
Loka Hrishikesh Reddy	25%	ER Diagrams, indexes, creating three views of user and code execution	LHR	12/03/2021
Sai Vathsal Jammula	25%	ER Diagrams, three views of user, triggers and code execution	SVJ	12/03/2021

A1

Application background, requirements (use cases), and assumptions. Assumptions have to be reasonable and do not deviate too much from real world applications. Do NOT use people's names as Keys. The applications from the textbook are NOT qualified.

Application Background

We have built an insurance database model. This can be used by any system who wants to create insurance policies for customers, System user can create insurance plans for Automobile Insurance and Term Life Insurance. It also includes invoicing and claim management.

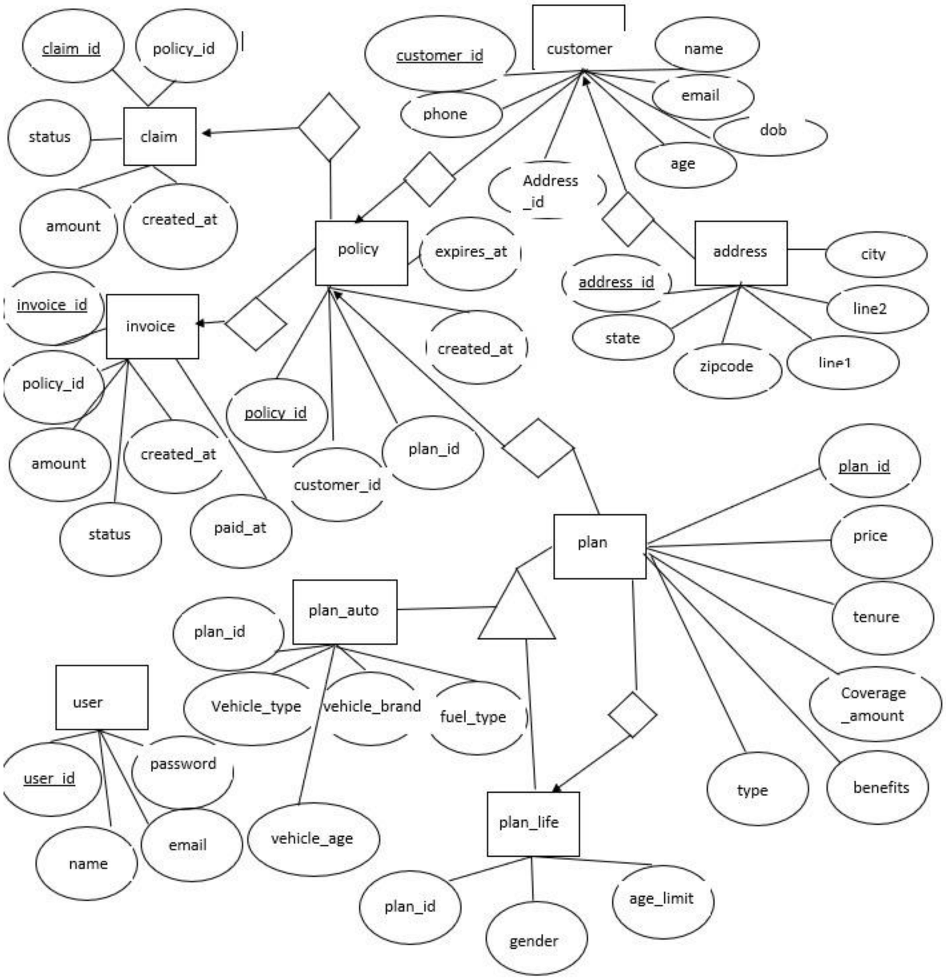
Use Cases :

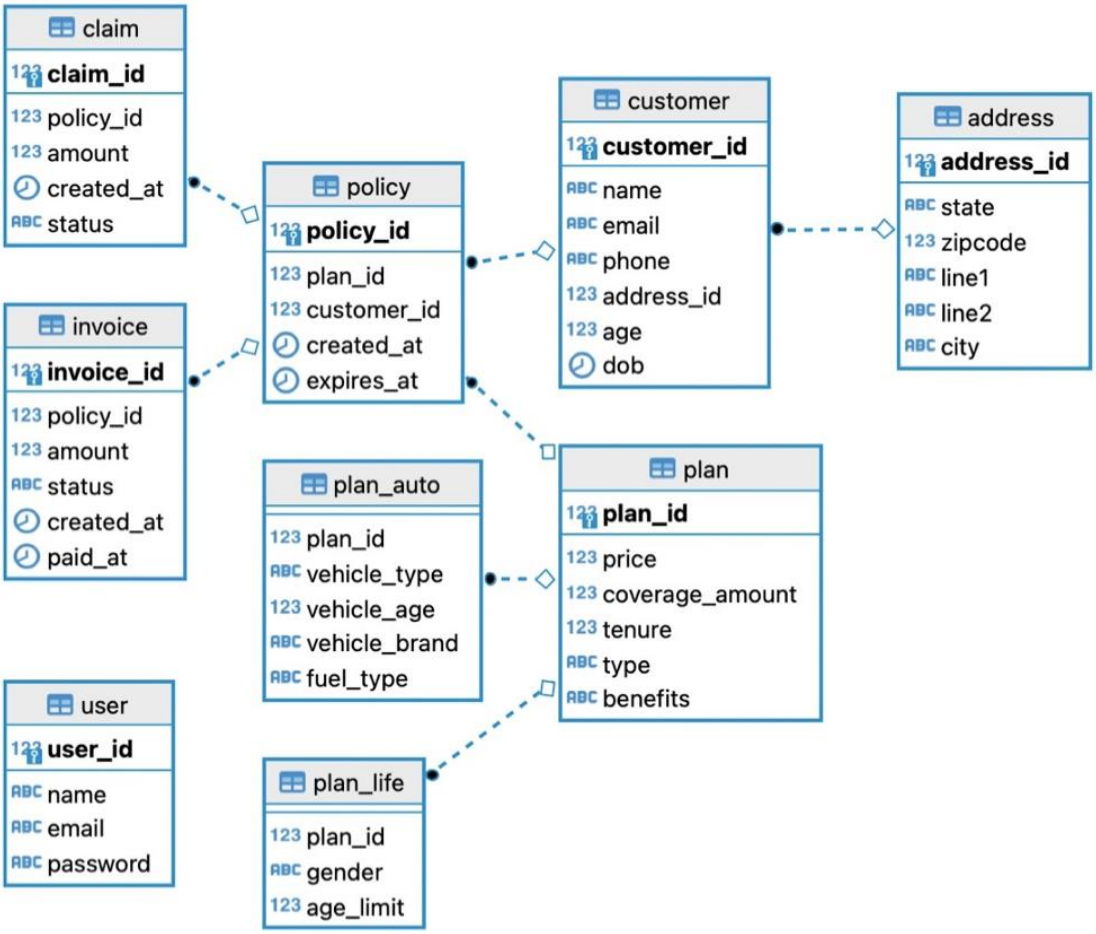
1. System user login details are stored in users table
2. Customers can be added with a separate table for Address
3. Plan table represents Insurance Plans. This table is the parent table for `plan_auto` which includes automobile specific details of each plan and `plan_life` which includes Term life insurance specific fields. Both `plan_auto` and `plan_life` table has **"IS-A" relationship** with parent `plan` table
4. Policy table contains policy specific details for each customer and is also linked with the plan table.
5. Invoice table contains invoicing details for each Policy
6. Claim table contains claim information for a Policy.

Assumption :

1. We have **not included** customers policy related information specific to each plan type i.e. customer vehicle information or customers life insurance related information. This can be considered as an extended use case for this system.

A2
E-R diagram (with detailed explanation of assumptions and database constraints)





claim
claim_id
policy_id
amount
created_at
status

policy
policy_id
plan_id
customer_id
created_at
expires_at

customer
customer_id
name
email
phone
address_id
age
dob

address
address_id
state
zipcode
line1
line2
city

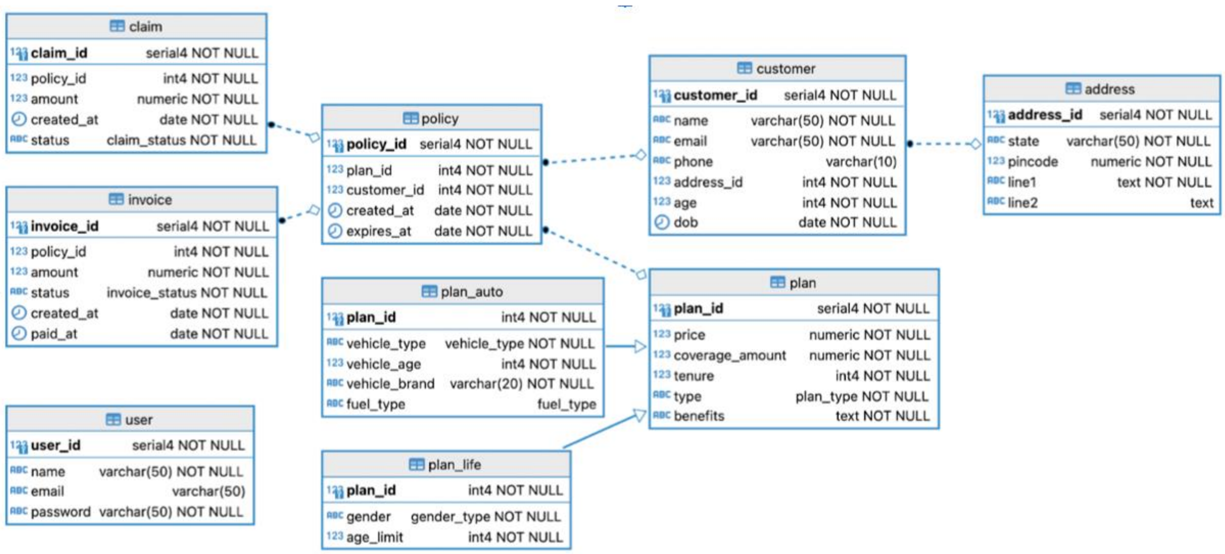
invoice
invoice_id
policy_id
amount
status
created_at
paid_at

plan_auto
plan_id
vehicle_type
vehicle_age
vehicle_brand
fuel_type

plan
plan_id
price
coverage_amount
tenure
type
benefits

user
user_id
name
email
password

plan_life
plan_id
gender
age_limit



claim	
claim_id	serial4 NOT NULL
policy_id	int4 NOT NULL
amount	numeric NOT NULL
created_at	date NOT NULL
status	claim_status NOT NULL

policy	
policy_id	serial4 NOT NULL
plan_id	int4 NOT NULL
customer_id	int4 NOT NULL
created_at	date NOT NULL
expires_at	date NOT NULL

customer	
customer_id	serial4 NOT NULL
name	varchar(50) NOT NULL
email	varchar(50) NOT NULL
phone	varchar(10)
address_id	int4 NOT NULL
age	int4 NOT NULL
dob	date NOT NULL

address	
address_id	serial4 NOT NULL
state	varchar(50) NOT NULL
pincode	numeric NOT NULL
line1	text NOT NULL
line2	text

invoice	
invoice_id	serial4 NOT NULL
policy_id	int4 NOT NULL
amount	numeric NOT NULL
status	invoice_status NOT NULL
created_at	date NOT NULL
paid_at	date NOT NULL

plan_auto	
plan_id	int4 NOT NULL
vehicle_type	vehicle_type NOT NULL
vehicle_age	int4 NOT NULL
vehicle_brand	varchar(20) NOT NULL
fuel_type	fuel_type

plan	
plan_id	serial4 NOT NULL
price	numeric NOT NULL
coverage_amount	numeric NOT NULL
tenure	int4 NOT NULL
type	plan_type NOT NULL
benefits	text NOT NULL

user	
user_id	serial4 NOT NULL
name	varchar(50) NOT NULL
email	varchar(50)
password	varchar(50) NOT NULL

plan_life	
plan_id	int4 NOT NULL
gender	gender_type NOT NULL
age_limit	int4 NOT NULL

B (15 points). **Relational Schema – must be** in 3NF (you need to formally prove it). The actual SQL create table commands are required. Include primary key and referential integrity constraints.

All SQL create table commands are part of **final.sql** file which contains all the DDL commands.

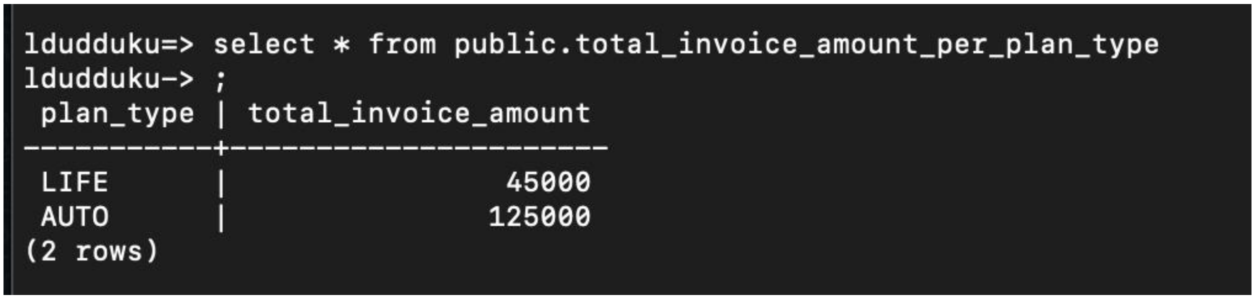
C (15 points). **Sample Data** Populate database with sample data. Insert at least 10 tuples per relation. Make sure that all queries from Deliverable D (below) have a non-empty answer.

Sample data is also part of **final.sql** file which contains all DML Insert Commands.

D (15 points). **Create Views** Create (write English descriptions and SQL syntax) at least 6 views that a user of the database would find useful. Do **NOT** include views that involve one single relation only. Attach screenshots of the query results from each created view, without them 0 points will be awarded.

View 1 : Total invoice amount per plan type
Description : We have 2 types of insurance plans in our current model Automobile Insurance Plan and Term Life Insurance Plan. This view contains total invoice amount generated for each type of plan

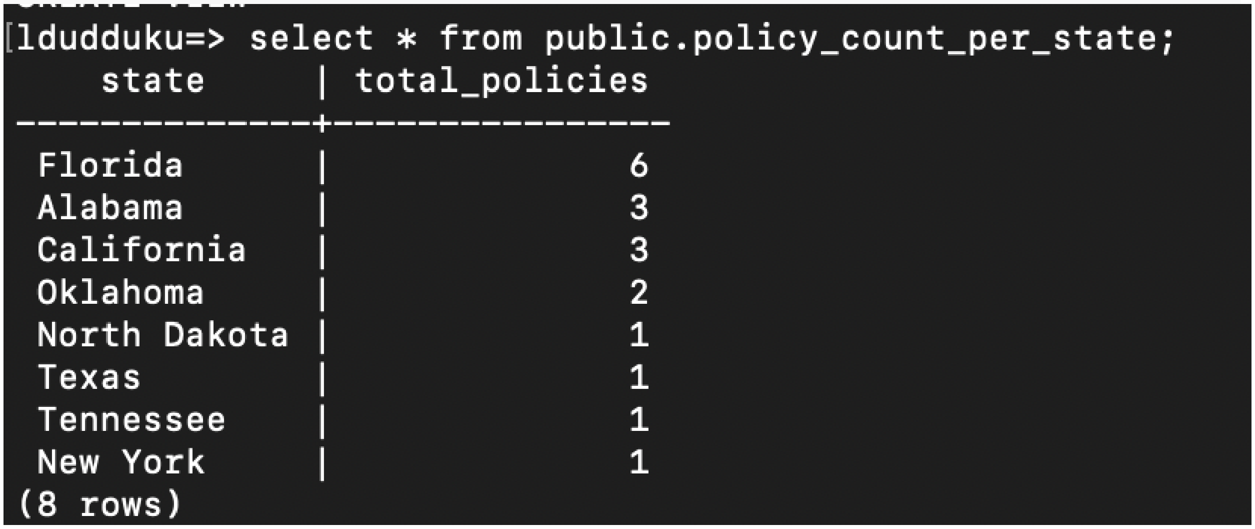
```
create or replace view total_invoice_amount_per_plan_type(plan_type,
total_invoice_amount) as
    SELECT plan.type    AS plan_type,
       sum(t2.psum) AS total_invoice_amount
FROM (SELECT policy.plan_id,
             t1.psum
      FROM (SELECT invoice.policy_id,
                   sum(invoice.amount) AS psum
            FROM invoice
            GROUP BY invoice.policy_id) t1
               JOIN policy ON policy.policy_id = t1.policy_id) t2
         JOIN plan ON plan.plan_id = t2.plan_id
GROUP BY plan.type;
```



```
ldudduku=> select * from public.total_invoice_amount_per_plan_type
ldudduku-> ;
 plan_type | total_invoice_amount
-----------+----------------------
 LIFE      |                45000
 AUTO      |               125000
(2 rows)
```

View 2 : Total policy count per region/state
Description : This view calculates the total number of policies created by customers present in each state.

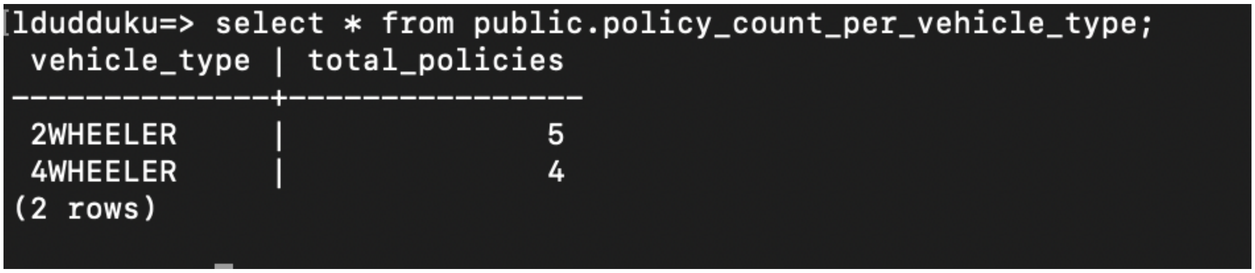
```
create or replace view policy_count_per_state(state, total_policies) as
    SELECT t1.state,
       count(*) AS total_policies
FROM (SELECT customer.customer_id,
             address.state
      FROM customer
               JOIN address ON address.address_id = customer.address_id) t1
         JOIN policy ON policy.customer_id = t1.customer_id
GROUP BY t1.state
ORDER BY (count(*)) DESC;
```



```
ldudduku=> select * from public.policy_count_per_state;
    state     | total_policies
--------------+----------------
 Florida      |              6
 Alabama      |              3
 California   |              3
 Oklahoma     |              2
 North Dakota |              1
 Texas        |              1
 Tennessee    |              1
 New York     |              1
(8 rows)
```

View 3 : Total Policy count per Vehicle type for Automobile Insurance
Description : For Automobile Insurance Plan we are considering 2 vehicle types `2WHEELER` and `4WHEELER`. This view shows the total number of policies created for each vehicle type.

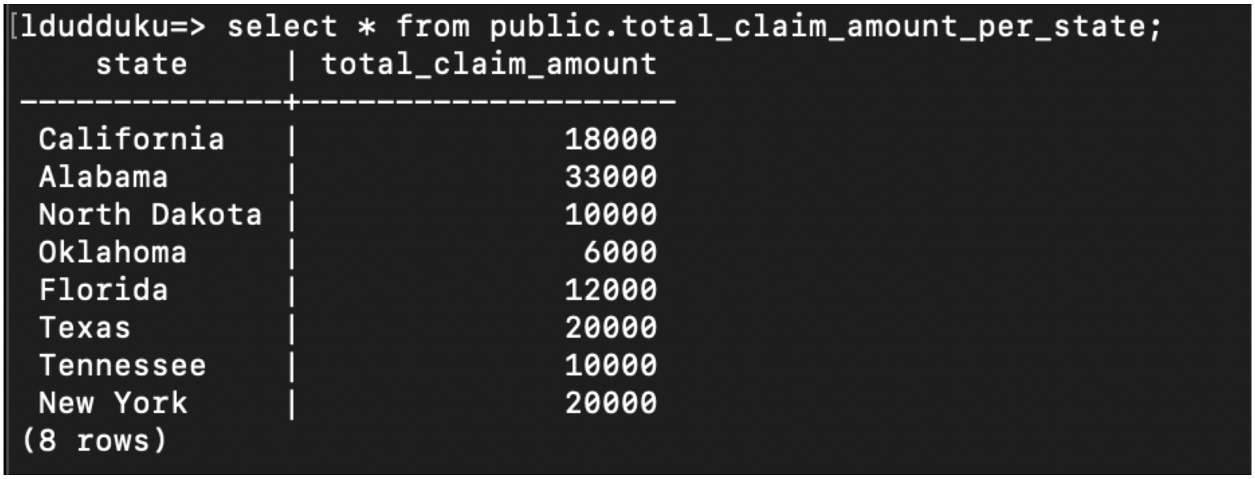
```
create or replace view policy_count_per_vehicle_type(vehicle_type,
total_policies) as
    SELECT plan_auto.vehicle_type,
       count(*) AS total_policies
FROM (SELECT policy.policy_id,
             plan.plan_id
      FROM policy
               JOIN plan ON plan.plan_id = policy.plan_id
      WHERE plan.type = 'AUTO'::plan_type) t1
         JOIN plan_auto ON plan_auto.plan_id = t1.plan_id
GROUP BY plan_auto.vehicle_type;
```



```
ldudduku=> select * from public.policy_count_per_vehicle_type;
 vehicle_type | total_policies
--------------+----------------
 2WHEELER     |              5
 4WHEELER     |              4
(2 rows)
```

View 4 : Total claim amount per region/state
Description : This view displays total claim amount claimed by customers from each state

```
create or replace view total_claim_amount_per_state(state,
total_claim_amount) as
    SELECT address.state,
       sum(t2.amount) AS total_claim_amount
FROM (SELECT customer.address_id,
             t1.amount
      FROM (SELECT policy.customer_id,
                   claim.amount
            FROM claim
                     JOIN policy ON policy.policy_id = claim.policy_id) t1
               JOIN customer ON customer.customer_id = t1.customer_id) t2
         JOIN address ON address.address_id = t2.address_id
GROUP BY address.state;
```

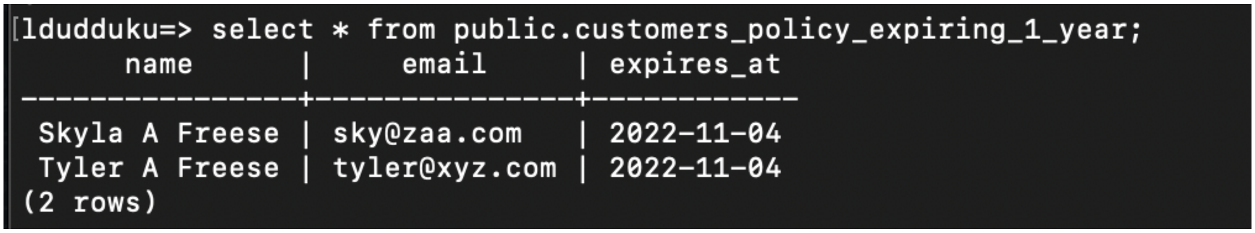


```
[ldudduku=> select * from public.total_claim_amount_per_state;
    state     | total_claim_amount
--------------+--------------------
 California   |              18000
 Alabama      |              33000
 North Dakota |              10000
 Oklahoma     |               6000
 Florida      |              12000
 Texas        |              20000
 Tennessee    |              10000
 New York     |              20000
(8 rows)
```

View 5 : Customer details whose policy is expiring within 1 year

Description : This view displays customers' information whose policies are going to expire in the next one year from the current date.

```
create or replace view customers_policy_expiring_1_year(name, email,
expires_at) as
    SELECT customer.name,
       customer.email,
       policy.expires_at
FROM policy
         JOIN customer ON customer.customer_id = policy.customer_id
WHERE policy.expires_at < (now() + '1 year'::interval);
```

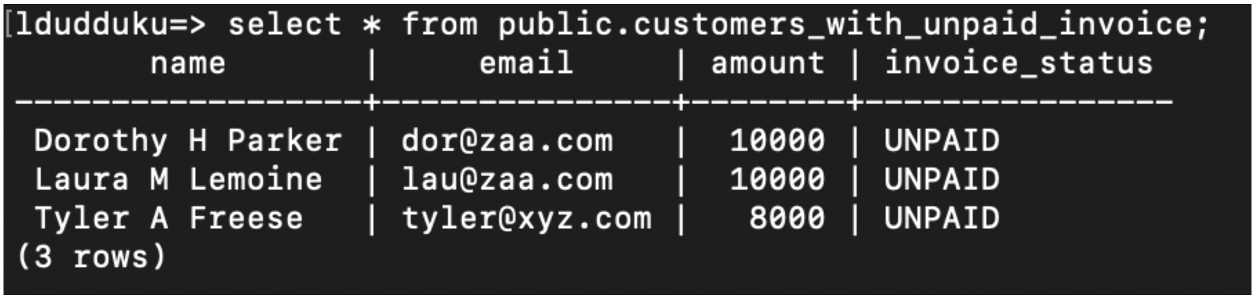


```
[ldudduku=> select * from public.customers_policy_expiring_1_year;
      name      |     email      | expires_at
----------------+----------------+------------
 Skyla A Freese | sky@zaa.com    | 2022-11-04
 Tyler A Freese | tyler@xyz.com  | 2022-11-04
(2 rows)
```

View 6 : Customer details with unpaid invoices

Description : This view displays details of those customers who has not paid their invoice yet.

```
create or replace view customers_with_unpaid_invoice(name, email, amount,
invoice_status) as
    SELECT customer.name,
       customer.email,
       t1.amount,
       t1.status AS invoice_status
FROM (SELECT p.customer_id,
             invoice.amount,
             invoice.status
      FROM invoice
               JOIN policy p ON invoice.policy_id = p.policy_id
      WHERE invoice.status = 'UNPAID'::invoice_status) t1
         JOIN customer ON t1.customer_id = customer.customer_id;
```



```
[ldudduku=> select * from public.customers_with_unpaid_invoice;
       name       |     email     | amount | invoice_status
------------------+---------------+--------+----------------
 Dorothy H Parker | dor@zaa.com   |  10000 | UNPAID
 Laura M Lemoine  | lau@zaa.com   |  10000 | UNPAID
 Tyler A Freese   | tyler@xyz.com |   8000 | UNPAID
(3 rows)
```

E (10 points). **Indexes** Create 3 indexes to support some of the queries (each view corresponds to a query) of Deliverable D. For each index, briefly explain (one sentence) how it will support a query (or queries). Do **NOT** create indexes for primary keys.

Index 1 : Index on plan_id in policy table

Description : We created this index for our **View 1 :** `total_invoice_amount_per_plan_type` As we are calculating invoice amount for each plan type , we need to fetch data from policy table which binds invoice and plan tables, to make this query fast we created index on plan_id

Index 2 : Index on customer_id in policy table

Description : We created this index for our **View 2 :** `policy_count_per_state`

As we are calculating policy count for each state, customer_id field is being used to detect the address and state information.

Index 3 : Index on policy_id in claim table
Description : We created this index for our **View 4 : `total_claim_amount_per_state`**
As we are calculating total claim amount for each state, the only field which binds claim to customer address is policy id inside the claim table .

F (10 points). **Constraints** Create at least 3 **non**-key/**non**-foreign-key constraints on the database. Do **not** create unique or not null constraints for the primary key.

Constraint 1 : email_check constraint on customer table's **email** field
Description : We want to verify if email entered by user is in correct format or not.

```
constraint email_check
            check ((email)::text ~ '^[A-Za-z0-9._%-]+@[A-Za-z0-9.-
]+[.][A-Za-z]+$'::text)
```

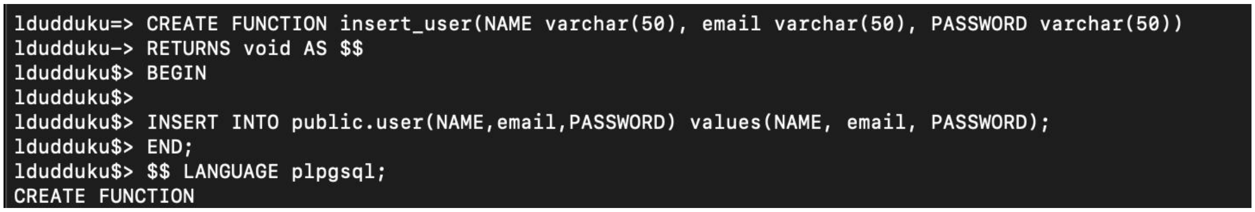
Constraint 2 : phone_check constraint on customer table's **phone** field
Description : We want to verify if phone number entered by user is 8 digit or 10 digit

```
constraint phone_check
            check ((phone)::text ~ '^(\d{8}|\d{10})$'::text)
```

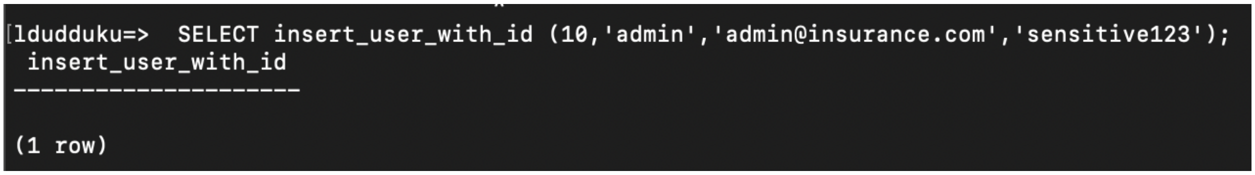
Constraint 3 : `expires_at_constraint` constraint on policy table's **expires_at** field
Description : We want to verify if expires_at is greater than created_at date of policy

```
constraint expires_at_constraint
        check (expires_at > created_at)
```

G (15 points). **Triggers** Create at least 2 Triggers on the database. Should you need to implement triggers to enforce constraints, you should choose to implement such trigger functions that cannot be readily replaced by existing constraint mechanisms provided by DBMS (e.g., do NOT create triggers that simply return an error message when there is a violation of the domain constraints, not null, unique, or a check constraint.) Include in the submission a statement of purpose for each trigger, the create trigger and trigger function statements, and the screenshots showing an actual triggering process as well as the results afterwards (you may also need to show the tuples in the table to show that the triggering condition is met.) At least 50% of points will be deducted if no such screenshots submitted.

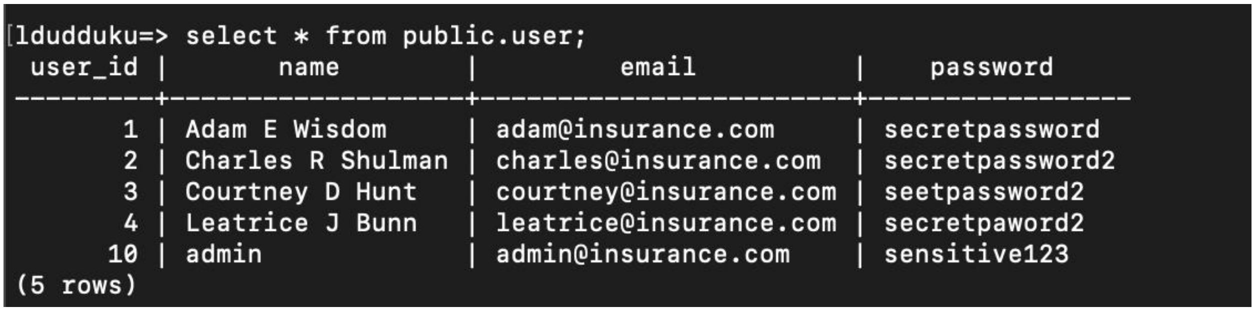


```
ldudduku=> CREATE FUNCTION insert_user(NAME varchar(50), email varchar(50), PASSWORD varchar(50))
ldudduku-> RETURNS void AS $$
ldudduku$> BEGIN
ldudduku$>
ldudduku$> INSERT INTO public.user(NAME,email,PASSWORD) values(NAME, email, PASSWORD);
ldudduku$> END;
ldudduku$> $$ LANGUAGE plpgsql;
CREATE FUNCTION
```



```
[ldudduku=>  SELECT insert_user_with_id (10,'admin','admin@insurance.com','sensitive123');
 insert_user_with_id
---------------------

(1 row)
```



```
[ldudduku=> select * from public.user;
 user_id |        name        |         email          |    password
---------+--------------------+------------------------+------------------
       1 | Adam E Wisdom      | adam@insurance.com     | secretpassword
       2 | Charles R Shulman  | charles@insurance.com  | secretpassword2
       3 | Courtney D Hunt    | courtney@insurance.com | seetpassword2
       4 | Leatrice J Bunn    | leatrice@insurance.com | secretpaword2
      10 | admin              | admin@insurance.com    | sensitive123
(5 rows)
```