DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

SRN :	Name :	Evaluation date and time:
PES1201800389	N Sanketh Reddy	29 - 5 - 2020

Functional Dependencies	2
Identifying Keys based on FDs	2
Normalization & testing for lossless join property	2+2
DDL: Table creation with all constraints	2+2
Triggers	2
SQL Queries	2
Viva / modifications(Unit III/ IV concepts)	2+2

Criminal Database management Systems

This is a project which helps in visualizing the police database and also has a python front-end for appending the data into the database and updating the data in the database.

The database has 6 main tables :

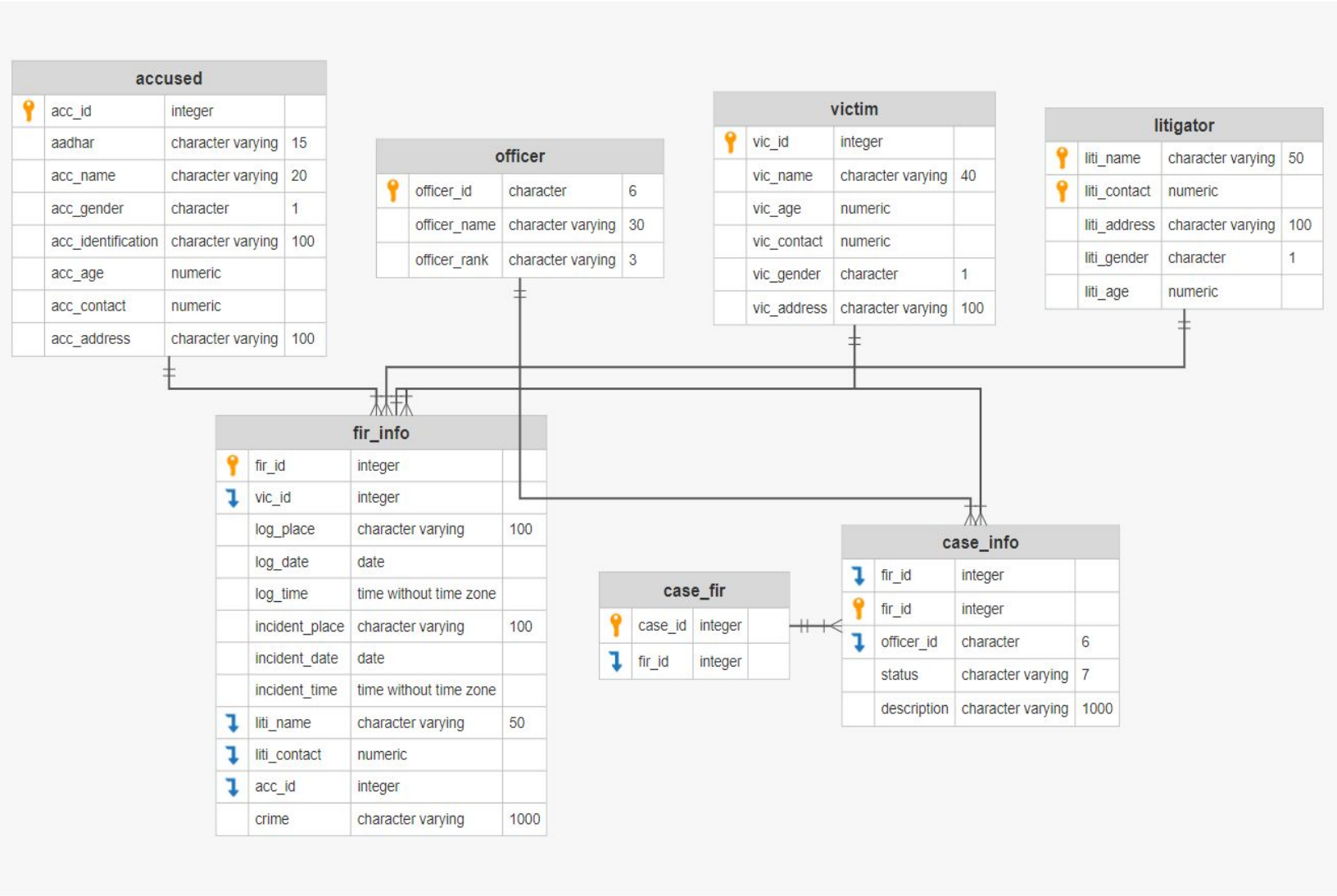
- Victim table: Stores the information of the victims.
- Accused table: Stores the information of the accused.
- Officer table: Stores the information of the police officers.
- Litigator table: Stores the information related to the Person filing the FIR.
- FIR_info: Stores the information of the FIRs
- Case_info: Stores the information related to the case.

After normalizing we get 7 tables because we had to split the case_info table into case_info and case_fir to satisfy the normalizing conditions.

The final list of tables:

- Victim
- Accused
- Officer
- Litigator
- Fir_info
- Case_info
- Case_fir

criminal



Final database schema.

Normalization:

- **1NF** : Removes repeating groups from the table. Create a seperate table for each set of related data. Identify each set of related data with a primary key.

We had no repeating values so the INF is satisfied.

- **2NF** : The tables in the database should not contain partial dependencies.(In case your primary key is single-valued, you can ignore this NF. In case it is multi-valued, Each non prime attribute has to depend on every value.) .
  The litigator table has a composite primary key and every other value in the table fully depends on both the attributes that make up the primary key, so the 2NF is satisfied.

- **3NF** : There should be no transitive dependency for non-prime attributes.
  The case_info table had a transitive relation:
  Case_id -> fir_id
  Fir_id -> all the other attributes of case_info
  Case_id -> all the other attributes of case_info
  So we split the table into 2 tables i.e case_fir and case_info.So the 3NF is satisfied.

- **3.5NF(BCNF)** : Every functional dependency A->B ,then A must be the Super Key of that particular table.
  The 3.5NF is already satisfied.

- **4NF** : The given relation may not contain more than one multi-valued attribute. The multi-valued dependency X→Y holds in a relation R if whenever we have two tuples of R that are same in all the attributes of X, then we can swap their Y components and get two new tuples that are also in R.
  We have no multivalued attributes, so 4NF is already satisfied.

- **5NF** : 5NF is satisfied when all tables are broken into as many tables as possible in order to avoid redundancy. Once it is in fifth normal

form it cannot be broken into smaller relations without changing the facts or the meaning.
5NF is already satisfied since it has passed all the previous normal forms.

Functional dependencies:

- **case_info** : fir_id -> {officer_id, status, description}
- **case_fir** : case_id -> fir_id
- **fir_info** : fir_id -> { vic_id,log_place, log_date, log_time, incident_place, incident_date, incident_time, liti_name, liti_contact, acc_id, crime}
- **accused** : acc_id -> {aadhar, acc_name, acc_gender, acc_identification, acc_age, acc_contact, acc_address}
- **litigator** : {liti_name , liti_contact} -> {liti_address, liti_gender, liti_age }
- **officer** : officer_id -> {officer_name , officer_rank}
- **victim** : vic_id ->{vic_name, vic_age, vic_contact, vic_gender, vic_adress}

Checks and Constraints:

- **officer**
  - CHECK(officer_id LIKE 'OF____')
  - PRIMARY KEY (officer_id)
- **litigator**
  - CHECK (liti_gender in ('M','F','m','f'))
  - CONSTRAINT litigator_pkey PRIMARY KEY (liti_name ,liti_contact)
- **victim**
  - vic_id integer NOT null
  - CHECK (vic_gender in ('M','F','m','f'))
  - CONSTRAINT victim_pkey PRIMARY KEY (v_id)

- **accused**
  - acc_id serial not null

    - check(aadhar ~ '^[0-9]*$')
    - CHECK (acc_gender in ('M','F','m','f'))
    - CONSTRAINT accused_pkey PRIMARY KEY (acc_id)

- **fir_info**
    - fir_id SERIAL PRIMARY KEY
    - log_date date DEFAULT current_date
    - log_time time default current_time
    - constraint fir_acc_fk FOREIGN KEY(acc_id) references accused(acc_id)
    - constraint fir_liti_fk FOREIGN KEY(liti_name,liti_contact) references litigator(liti_name,liti_contact)
    - constraint fir_vic_fk FOREIGN KEY(vic_id) references victim(vic_id))

- **case_info**
    - CHECK(status in ('OPEN','CLOSED'))
    - CONSTRAINT cd_fk1 FOREIGN KEY(fir_id) references fir_info(fir_id)
    - CONSTRAINT cd_fk2 FOREIGN KEY(officer_id) references officer(officer_id))

- **case_fir**
    - case_id SERIAL PRIMARY KEY
    - constraint case_fir_fk FOREIGN KEY(fir_id) references fir_vicid(fir_id))

DDL and Triggers:

Officer :

```
CREATE TABLE public.officer (
	officer_id bpchar(6) NOT NULL,
	officer_name varchar(30) NULL,
	officer_rank varchar(3) NULL,
	CONSTRAINT officer_officer_id_check CHECK ((officer_id ~~ 'OF____'::text)),
	CONSTRAINT officer_pk PRIMARY KEY (officer_id)
);
```

Victim :

```
-- Drop table

-- DROP TABLE public.victim;

CREATE TABLE public.victim (
	vic_id serial NOT NULL,
	vic_name varchar(40) NULL,
	vic_age numeric(2) NULL,
	vic_contact numeric(10) NULL,
	vic_gender bpchar(1) NULL,
	vic_address varchar(100) NULL,
	CONSTRAINT victim_pkey PRIMARY KEY (vic_id),
	CONSTRAINT victim_vic_gender_check CHECK ((vic_gender = ANY
(ARRAY['M'::bpchar, 'F'::bpchar, 'm'::bpchar, 'f'::bpchar])))
);

-- Table Triggers

-- DROP TRIGGER vic_triggger ON public.victim;

create trigger vic_triggger before
insert
   or
update
   on
   public.victim for each row execute function vic_trig();
```

Accused :

```
-- Drop table

-- DROP TABLE public.accused;

CREATE TABLE public.accused (
```

```
	acc_id serial NOT NULL,
	aadhar varchar(15) NULL,
	acc_name varchar(20) NULL,
	acc_gender bpchar(1) NULL,
	acc_identification varchar(100) NULL,
	acc_age numeric(2) NULL,
	acc_contact numeric(10) NULL,
	acc_address varchar(100) NULL,
	CONSTRAINT accused_aadhar_check CHECK (((aadhar)::text ~
'^[0-9]*$'::text)),
	CONSTRAINT accused_acc_gender_check CHECK ((acc_gender = ANY
(ARRAY['M'::bpchar, 'F'::bpchar, 'm'::bpchar, 'f'::bpchar]))),
	CONSTRAINT accused_pkey PRIMARY KEY (acc_id)
);

-- Table Triggers

-- DROP TRIGGER acc_trigger ON public.accused;

create trigger acc_trigger before
insert
   or
update
   on
   public.accused for each row execute function acc_trig();
```

Litigator:

```
-- Drop table

-- DROP TABLE public.litigator;

CREATE TABLE public.litigator (
	liti_name varchar(50) NOT NULL,
	liti_contact numeric(10) NOT NULL,
	liti_address varchar(100) NULL,
	liti_gender bpchar(1) NULL,
	liti_age numeric(2) NULL,
```

```
	CONSTRAINT litigator_liti_gender_check CHECK ((liti_gender = ANY
(ARRAY['M'::bpchar, 'F'::bpchar, 'm'::bpchar, 'f'::bpchar]))),
	CONSTRAINT litigator_pkey PRIMARY KEY (liti_name, liti_contact)
);
```

fir_info:

```
-- Drop table

-- DROP TABLE public.fir_info;

CREATE TABLE public.fir_info (
	fir_id serial NOT NULL,
	vic_id int4 NULL,
	log_place varchar(100) NULL,
	log_date date NULL DEFAULT CURRENT_DATE,
	log_time time NULL DEFAULT CURRENT_TIME,
	incident_place varchar(100) NULL,
	incident_date date NULL,
	incident_time time NULL,
	liti_name varchar(50) NULL,
	liti_contact numeric(10) NULL,
	acc_id int4 NULL,
	crime varchar(1000) NULL,
	CONSTRAINT fir_info_pkey PRIMARY KEY (fir_id),
	CONSTRAINT fir_acc_fk FOREIGN KEY (acc_id) REFERENCES
accused(acc_id),
	CONSTRAINT fir_liti_fk FOREIGN KEY (liti_name, liti_contact) REFERENCES
litigator(liti_name, liti_contact),
	CONSTRAINT fir_vic_fk FOREIGN KEY (vic_id) REFERENCES victim(vic_id)
ON DELETE CASCADE
);
```

Case_fir:

```
-- Drop table
```

```
-- DROP TABLE public.case_fir;

CREATE TABLE public.case_fir (
	case_id serial NOT NULL,
	fir_id int4 NULL,
	CONSTRAINT case_fir_pkey PRIMARY KEY (case_id),
	CONSTRAINT case_fir_fk FOREIGN KEY (fir_id) REFERENCES
case_info(fir_id)
);
```

case_info:

```
-- Drop table

-- DROP TABLE public.case_info;

CREATE TABLE public.case_info (
	fir_id int4 NOT NULL,
	officer_id bpchar(6) NULL,
	status varchar(7) NULL,
	description varchar(1000) NULL,
	CONSTRAINT case_info_pkey PRIMARY KEY (fir_id),
	CONSTRAINT case_info_status_check CHECK (((status)::text = ANY
((ARRAY['OPEN'::character varying, 'CLOSED'::character varying])::text[]))),
	CONSTRAINT cd_fk1 FOREIGN KEY (fir_id) REFERENCES fir_info(fir_id),
	CONSTRAINT cd_fk2 FOREIGN KEY (officer_id) REFERENCES
officer(officer_id)
);
```

Functions for triggers:

- **vic_trig() -> To avoid duplicate rows in Victim table**

```
CREATE OR REPLACE FUNCTION public.vic_trig()
 RETURNS trigger
```

```
 LANGUAGE plpgsql
AS $function$
DECLARE cnt_vic integer;
BEGIN
IF EXISTS (
select table_name from information_schema.tables where table_name='vic_temp')
THEN
DROP TABLE vic_temp;
END IF;
CREATE TEMP TABLE vic_temp AS
SELECT vic_id from  victim WHERE (NEW.vic_name = victim.vic_name AND
NEW.vic_contact=victim.vic_contact) ;
SELECT count(*) INTO cnt_vic from vic_temp;
if cnt_vic>=1
THEN
  RAISE EXCEPTION 'Victim details already exists';
  RETURN NULL;
END IF;
IF cnt_vic=0
THEN
RETURN NEW;
END IF;
RETURN NULL;
END;
$function$
;
```

- **fir_trig() -> To avoid duplicate FIRs**

```
CREATE OR REPLACE FUNCTION public.fir_trig()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
DECLARE cnt_fir integer;
BEGIN
IF EXISTS (
select table_name from information_schema.tables where table_name='fir_temp')
THEN
```

```
DROP TABLE fir_temp;
END IF;
CREATE TEMP TABLE FIR_table AS
SELECT fir_id from  fir_info WHERE (NEW.crime = fir_info.crime AND
NEW.vic_id=fir_info.vic_id AND NEW.incident_place=fir_info.incident_place AND

NEW.incident_date=fir_info.incident_date ) ;
SELECT count(*) INTO cnt_fir from  fir_temp;
if cnt_fir>=1
THEN
  RAISE EXCEPTION 'FIR already filed on this crime';
  RETURN NULL;
END IF;
IF cnt_fir=0
THEN
RETURN NEW;
END IF;
RETURN NULL;
END;
$function$
;
```

- **acc_trig() -> To avoid duplicate Accused details**

```
CREATE OR REPLACE FUNCTION public.acc_trig()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
declare cnt integer;
BEGIN
if exists(
select table_name from information_schema.tables where table_name = 'acc_temp')
then
drop table acc_temp;
end if;

create temp table acc_temp as
select acc_id from accused where new.aadhar = accused.aadhar;
```

```
SELECT count(*) INTO cnt from acc_temp;
if cnt>=1
THEN
  RAISE EXCEPTION 'Accused details already exists';
  RETURN NULL;
END IF;
IF cnt=0
THEN
RETURN NEW;
END IF;
RETURN NULL;
END;
$function$
;
```

Complex SQL Queries:

- **Retrive the officer's name who is incharge of murder cases**

select officer_name from officer where officer_id in (select officer_id from case_info where fir_id in (select fir_id from fir_info where (crime = 'MURDER') or (crime = 'murder')))

- **Retrieve accused names who were accused by litigators who are men and are of aged above 20 years.**

select acc_name from accused where acc_id in (select acc_id from fir_info where liti_name in (select liti_name from litigator where liti_gender = 'M' and liti_age > 20))

- **Retrieve the accused name whose age is above average accuRetrivesed age.**

select acc_name from accused where acc_age > (select avg(acc_age) from accused)

- **Retrieve the count of male and female victims**

select vic_gender, count(vic_gender) from victim group by vic_gender

- **Retrieve the the aadhar number of the accused for the case_id = 8**

select aadhar from (case_fir CF join fir_info F on CF.fir_id=F.fir_id join accused AC on AC.acc_id = F.acc_id)
where case_id=8

Python Frontend:

insert.py/ipynb :

This python script helps in filing the fir the litigator needs to enter needs to enter some details about himself and the incident the rest of the fields are automatically filled. And if all the fir details are typed in correctly ,the program continues and creates a case and allocates a police officer to deal with the case.

The program also creates an empty row in the accused and victim table related to the fir.

update.py/ipynb :

This python script helps in updating the victim and accused information which was earlier created ,if the victim or the accused already exists in the database its not gonna create a new value instead it is going to delete the new empty value that was created during the execution of insert.py and update the vic_id or acc_id of the already existing person in the case and fir information tables.